Sentiment Analysis for Product Reviews

By: Annet Nasimiyu Chebukati

Agenda

- ✓ Introduction
- ✓ Data Collection
- ✓ Data Cleaning and Preparation
- ✓ Text Vectorization
- ✓ Model Building
- ✓ Model Evaluation
- ✓ Insights and Recommendations

Introduction

Overview of the project, its objectives, and its importance

Sentiment Analysis is a powerful Natural Language Processing technique used to extract subjective information from text. It identifies and categorizes opinions expressed in a piece of text, especially to determine whether the writer's attitude towards a particular topic or product is positive, negative, or neutral.

In this project, I applied Sentiment Analysis to a dataset of product reviews from an e-commerce platform. The **goal** was to understand the sentiment behind each review based on its content and associated rating. These sentiments are valuable for various business strategies like product development, customer service improvement, and targeted marketing.

By analyzing patterns and customer sentiments, I aimed to unearth actionable insights that can drive strategic initiatives forward and enhance customer satisfaction.

Data Collection

- The first step was to prepare the data for analysis. I imported the dataset from **GitHub**, which consists of product reviews from Amazon e-commerce platform, into a Jupyter Notebook for processing and analysis.
- Ensuring the data was clean and structured was crucial at this stage, as it set the foundation for all the subsequent steps.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30846 entries, 0 to 30845
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   marketplace        30846 non-null  object
 1   customer_id        30846 non-null  int64
 2   review_id          30846 non-null  object
 3   product_id         30846 non-null  object
 4   product_parent     30846 non-null  int64
 5   product_title      30846 non-null  object
 6   product_category   30846 non-null  object
 7   star_rating        30846 non-null  int64
 8   helpful_votes      30846 non-null  int64
 9   total_votes        30846 non-null  int64
 10  vine               30846 non-null  object
 11  verified_purchase  30846 non-null  object
 12  review_headline    30844 non-null  object
 13  review_body        30842 non-null  object
 14  review_date        30846 non-null  object
 15  sentiment          30846 non-null  int64
dtypes: int64(6), object(10)
memory usage: 3.8+ MB
None
```

Data Cleaning and Preparation

- The raw text data from the reviews needed to be preprocessed to ensure it was in the optimal format for modeling. This involved cleaning the text (such as removing punctuation, URLs, HTML tags, etc.) and transforming it into a numerical format that machine learning models could interpret.

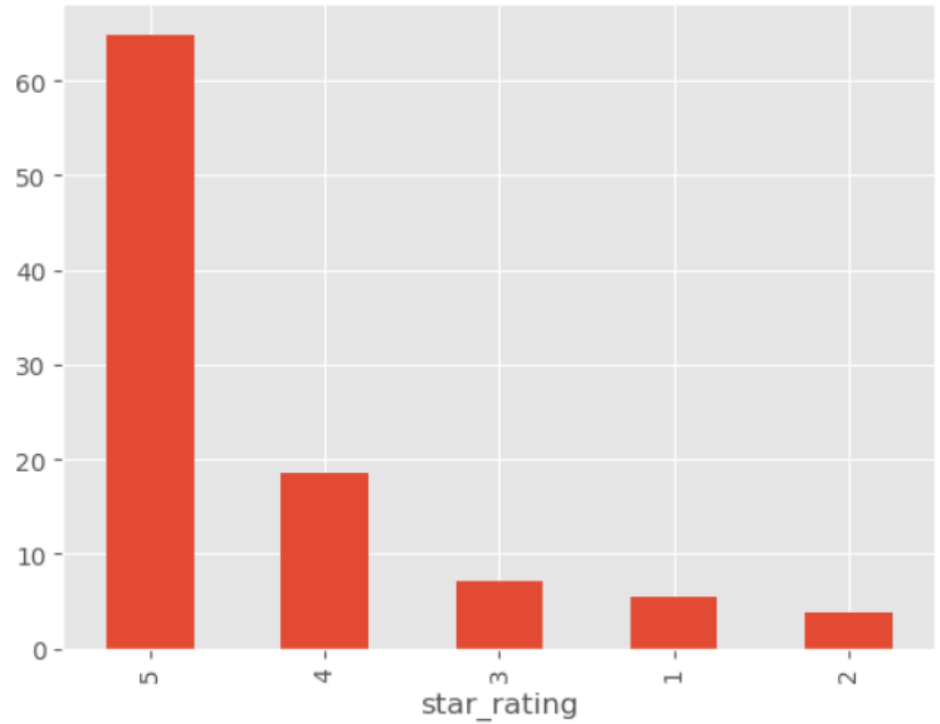
```
# Clean the text
def clean_text(Review):
    Review = str(Review).lower()  # convert to lowercase
    Review = re.sub('\[.*?\]', '', Review)
    Review = re.sub('https?://\S+|www\.\S+', '', Review)  # Remove URLs
    Review = re.sub('<.*?>+', '', Review)
    Review = re.sub(r'[^a-z0-9\s]', '', Review)  # Remove punctuation
    Review = re.sub('\n', '', Review)
    Review = re.sub('\w*\d\w*', '', Review)
    return Review

amz_df['review_body'] = amz_df['review_body'].apply(clean_text)
```

Exploratory Data Analysis

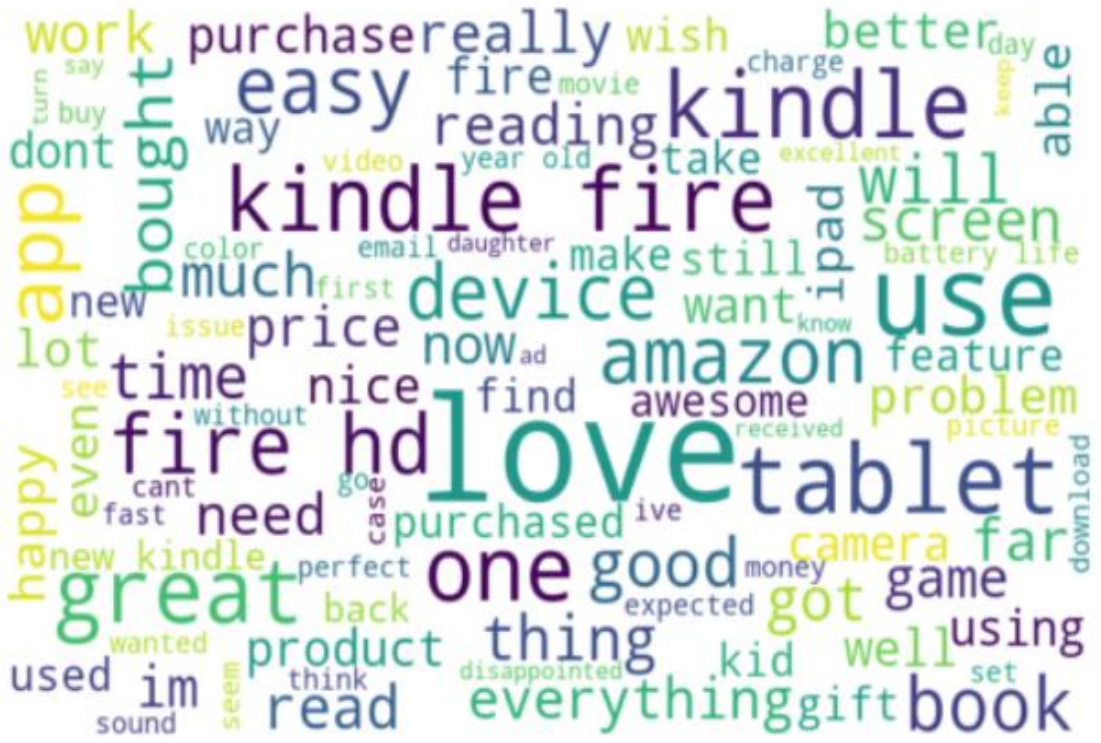
- Once the data was prepared, I conducted an initial exploration to understand the characteristics of the data.
- This step involved uncovering patterns, spotting anomalies, and forming hypotheses about the potential relationships between items.

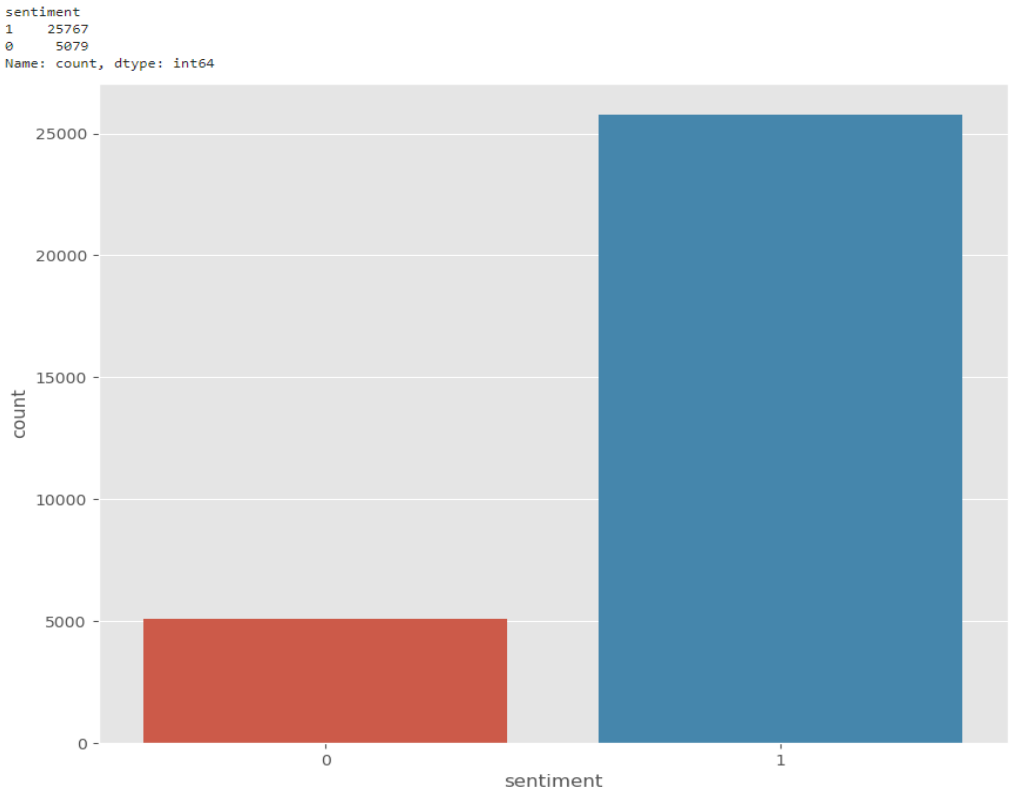
```
star_rating
5    64.899825
4    18.634507
3     7.184076
1     5.537185
2     3.744408
Name: count, dtype: float64
```



Exploratory Data Analysis

- I also checked for the most frequent words in the review text and the number of rows label positive – **1** (**25,767**) or negative sentiment – **0** (**5,079**).





Text Vectorization

- Transformation of the text data into a numerical format that machine learning models could interpret.

The text data was transformed into a numerical format using a technique called TF-IDF vectorization. This step converted the text data into a format that the machine learning model could interpret, paving the way for the next step - model building.

The **TfidfVectorizer()** from **sklearn.feature_extraction.text** is used to convert the text data into a matrix of TF-IDF features.

Model Building

- Construction of a machine learning model to classify the reviews' sentiment.

- A machine learning model was constructed to classify the sentiment of the reviews. The model was trained on the preprocessed and vectorized text data, learning to recognize patterns associated with positive, negative, and neutral sentiments.

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(amz_df['reviews_text'], amz_df['sentiment'], random_state=42, test_size=0.20)

# Print the shapes of the train and test sets
print(X_train.shape, X_test.shape, y_train.shape)
```

```
(24676,) (6170,) (24676,)
```

Model Training & Evaluation

- The performance of the model was rigorously assessed using appropriate evaluation metrics. This step was crucial in understanding the strengths and weaknesses of the model, and in identifying areas for improvement.

```
# Define the resampling method
method = SMOTE(random_state=42)

# Vectorize your text data
vectorizer = TfidfVectorizer(stop_words="english")
X = vectorizer.fit_transform(amz_df['reviews_text'])

# Create the resampled feature set
X_resampled, y_resampled = method.fit_resample(X, amz_df['sentiment'])

# Now you can split your resampled data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, random_state=42, test_size=0.20)

# Then you can continue with your model as before
clf = RandomForestClassifier(n_estimators=100, random_state=42)

fit_model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('Training accuracy:', fit_model.score(X_train, y_train))
print('Test accuracy:', fit_model.score(X_test, y_test))
print('AUC-ROC:', roc_auc_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Model Interpretation

Training accuracy is approximately **99.44%**, which means that the model correctly classified about 99.44% of the training data.

Test accuracy is approximately **93.09%**, which means that the model correctly classified about 93.09% of the test data.

The AUC-ROC is approximately **0.93**. This score is a measure of the model's ability to distinguish between the classes. A score of 1 represents a perfect classifier, so **0.93** is a very good score.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate. For both classes, the precision is quite high (**0.90** for class **0** and **0.96** for class **1**), indicating that the model has a low rate of false positives.

Recall (Sensitivity) is the ratio of correctly predicted positive observations to the all observations in actual class. The recall is also quite high for both classes (**0.97** for class **0** and **0.89** for class **1**), indicating that the model can detect the positive instances at a high rate.

F1-score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. The F1-score is quite high for both classes (**0.93**), indicating that the model is robust.

Model Interpretation

Confusion matrix provides a visual representation of the model's performance. It shows the true positives, true negatives, false positives, and false negatives.

	Predicted Positive	Predicted Negative
Actual Positive	5007	173
Actual Negative	539	4588

True Positives (TP): model correctly predicted **5007** positive reviews. These are cases where both the actual class was positive, and the model also predicted the positive class.

True Negatives (TN): model correctly predicted **4588** negative reviews. These are cases where both the actual class was negative, and the model also predicted the negative class.

False Positives (FP): model incorrectly predicted **539** reviews as positive. These are cases where the actual class was negative, but the model predicted them as positive. This type of error is also known as a "Type I error" or "false alarm".

False Negatives (FN): model incorrectly predicted **173** reviews as negative. These are cases where the actual class was positive, but the model predicted them as negative. This type of error is also known as a "Type II error" or "miss".

Insights

- **High Accuracy**: The model has approximately **99.44%** and a test accuracy of approximately **93.09%.** Indicates that the model is performing well in classifying reviews as positive or negative.
- **Good Performance Metrics**: High AUC-ROC score of approximately **0.93**, indicating a good balance between sensitivity and specificity. The precision, recall, and F1-score are also high for both classes, indicating that the model is performing well on both positive and negative reviews.
- **Confusion Matrix**: The confusion matrix shows that the model has more true positives and true negatives than false positives and false negatives. This indicates that the model is correctly classifying most of the reviews.

Recommendations

- ✓ **Monitor Overfitting**: The noticeable difference between the training and test accuracy, could be a sign of overfitting. It's recommended to monitor the model's performance over time to ensure it continues to generalize well to new data.
- ✓ **Consider Business Context**: Depending on the specific requirements of your task, you might want to aim for reducing either false positives or false negatives. For instance, if it's more costly to misclassify a negative review as positive (false positive), you should aim for reducing false positives even if it means increasing false negatives, and vice versa.
- ✓ **Iterative Improvement**: Building a machine learning model is an iterative process. It's recommended to continuously improve the model based on new data and feedback.
- ✓ **Leverage Insights**: The insights gained from the model can be used to improve products and services. For example, by understanding what aspects customers are unhappy about (from negative reviews), efforts can be made to improve those areas.

Thank you

annetnasimiyuchebukati@gmail.com

Linkedin

Github

Portfolio

Phone: +254 719406701