Final Project Report

Sentiment-Driven Classification of Footwear Reviews on Amazon

Submitted to:

Professor Jennifer Boyce

Prepared By:

Taraka Paruchuru

Sentiment-Driven Classification of Footwear Reviews on Amazon

In the competitive landscape of e-commerce, understanding customer sentiments and preferences is paramount for tailoring product recommendations and enhancing customer satisfaction. This project aims to address the specific domain of footwear sold on Amazon, a leading e-commerce platform, by performing sentiment analysis and opinion mining on customer reviews. The core objective is to develop a sophisticated analysis framework that can accurately classify customer reviews into three distinct sentiment categories: good, bad, and neutral. By achieving a granular understanding of customer sentiments associated with each product, this project seeks to contribute significantly to the development of advanced recommender systems. These systems will leverage the insights derived from the sentiment analysis to offer personalized footwear product suggestions to customers visiting Amazon. This endeavor not only promises to elevate the shopping experience by aligning product recommendations more closely with individual preferences but also aims to drive sales growth by effectively matching products with the right customers. Through this project, we aspire to harness the power of sentiment analysis to unlock new dimensions of customer engagement and satisfaction in the e-commerce domain, ultimately fostering a more intuitive and responsive online shopping environment.

Data

The dataset, amazon_uk_shoes_products_dataset_2021_12.csv was obtained from data.world which was actually shared by http://crawlfeeds.com.

This dataset contains 11 columns and 6823 rows.

Column Name	Data Type
url	Qualitative (Nominal)
product_name	Text (Product Categorization)
reviewer_name	Nominal
review_title	Text (Sentiment Analysis)
review_text	Text (Review Analysis)
review_rating	Ordinal
verified_purchase	Boolean
review_date	Quantitative (Discrete)
helpful_count	Discrete Quantitative
uniq_id	Nominal
scraped_at	Quantitative (Discrete)

In our project, we aim to delve into the sentiment analysis of product reviews hosted on the Amazon UK website. Our focus is to classify these reviews into distinct sentiment categories utilizing a select set of features that are pivotal for our analysis.

To our sentiment analysis-based classification project, we have carefully chosen a subset of features that are instrumental to achieving our objectives. These features include:

Selected Features for Analysis:

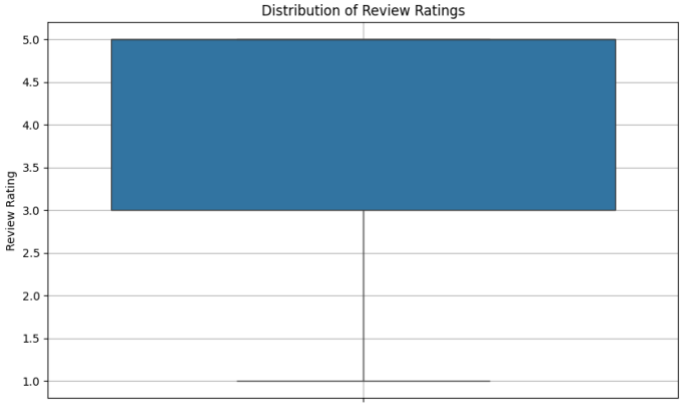
Product Name: To identify the product being reviewed.
Review Text: The main content of the review, which is critical for sentiment analysis.
Helpful Count: The number of people who found the review helpful, indicating its relevance and impact.
Verified Purchase: A flag indicating whether the reviewer purchased the product, which can influence the authenticity and perspective of the review.
Review Rating: Acts as a critical indicator of sentiment, providing an immediate gauge of customer satisfaction. Review ratings offer substantial support to the textual content of reviews in determining the sentiment towards a product.

I have strategically decided to exclude certain attributes that do not directly contribute to our sentiment analysis or classification algorithm. These excluded features are:

Excluded Features:

URL, Reviewer Name, Review Title, Unique ID: Although informative, these do not offer direct value for sentiment analysis.
Review Date, Scraped At: These features, while potentially useful for trend analysis or tracking over time, are not currently within the scope of our project focused on sentiment analysis.

Data Distribution based on Ratings:



The box plot shows the distribution of review ratings for a product on Amazon UK specific, As per the box plot horizontal line in the middle of the box represents the median rating. This is the value that separates the higher half of the ratings from the lower half. The median rating is around 5.0, indicating that most reviews are positive. Regarding the spread The box shows the interquartile range (IQR), which represents the middle 50% of the ratings. The height of the box tells you how spread out these middle ratings are. In this case, the box is relatively short, suggesting that the ratings are clustered closely around the median, the whiskers are relatively short, indicating that there are few ratings that fall outside the IQR.
Outliers are not applicable and the same are evident in the box plot , There are no outliers visible in this plot, suggesting that most ratings fall within a typical range which is by default option provided to customers from 1 to 5. So, in overall the dataset has more positive reviews compared to negative and neutral.

Data Preparation and Feature Exploration:

During data preparation, I have analyzed the dataset to understand the distribution and range of values for each feature. This included examining the number of unique values to identify potential limitations or inconsistencies. I have focused on several key features for our sentiment analysis task:

Feature Type	Feature Name	Data Type
Text Data	Product Name	String
Text Data	Review Text	String
Supporting Data	Review Rating	Numerical
Supporting Data	Verified Purchase	Boolean
Supporting Data	Helpful Count	Numerical

Language Filtering

While our data originated from Amazon UK, we identified reviews in multiple languages beyond English. To ensure accurate sentiment analysis, we filtered the data to include only English language reviews.

Feature Engineering (Feature extraction**)**
I utilized a Python IDE for the exploratory data analysis of data presented in a .csv file. After filtering the dataset to retain only English reviews, we were left with 4,003 reviews. Upon careful examination of the **'rating'** feature, which consists of ordinal values ranging from 1.0 to 5.0, I engaged in feature engineering. This involved creating a categorical feature, **'rating_category'**, which was segmented into three **discrete categories**—good, bad, and average—based on logical conditions applied to an existing numeric feature, **'review_rating'**.

I have refrained from further proceeding with one-hot encoding for the categories of good, bad, and average.

Additionally, I extracted numerical values from a text column, '**helpful_count'**, that contains mixed data types (text and numbers), to create a new feature, '**helpful_numbers**'. This feature quantitatively represents the original information in a numerical format that is more suitable for analysis or modeling. This transformation is particularly useful since the original data was not in a format directly usable by machine learning algorithms such as the Naive Bayes classifier, which require numerical input when combined with TF-IDF features generated from the **'review_text'**.

Data Cleaning:

In the "**review_text**" column, we have observed the presence of null values, which have been replaced with empty strings. This decision was based on the minimal occurrence of these null values, accounting for less than 1% of the data, making the impact of replacing them with empty strings negligible. Additionally, I have converted the "**review_text**" column to the string data type. This step is essential before proceeding with our sentiment analysis of text data, as our sentiment analysis algorithms are sensitive to string data. Furthermore, the dataset does not contain duplicate values, which is to be expected since customer reviews are inherently unique, with each text review providing a distinct perspective on each product.

Pre-Processing specific to Sentiment Analysis:
we have developed a function named "**preprocess_text**", which is applied to each row of the **"review_text"** column using the "**apply**" method. This function processes the text data through different stages of processing, utilizing the built-in libraries provided by NLTK.

Tokenize the Text: Splits the text into individual words or tokens.
Tokenization is the first step in turning unstructured text into a structured form that can be analyzed. Converting the text to lowercase ensures that the same words are recognized as identical, regardless of case. Here we used **word_tokenize** function from the NLTK library on the lowercase version of the text.
Remove Stop Words: Filters out common words that are usually irrelevant in analyzing text's meaning like this, is, was…
Stop words are typically removed because they occur frequently in the language and usually don't contribute to the overall meaning of the text, allowing the focus to be on the more meaningful words. We used **stopwords.words('english')** a list of English stop words provided by the NLTK library for our project.
Lemmatize the Tokens: Converts words to their base or dictionary form.
Lemmatization helps in reducing the inflectional forms of each word into a common base or root. Unlike stemming, lemmatization considers the context and converts the word to its meaningful base form, which helps in grouping together the inflected forms of a word so they can be analyzed as a single item.

We imported **WordNetLemmatizer** from NLTK and applies it to each token.

Join the Tokens Back into a String: Reconstructing the processed tokens back into a single string.

After preprocessing, the tokens need to be reassembled into a form that can be used by downstream processes or models. This step transforms the list of words back into a text string, making it possible to store the processed text in a Data Frame or use it in text analysis tasks. **join** method is used concatenate the list of lemmatized tokens into a single string, with spaces between each token.
Finally, this function transformed the original "**review_text**" column by cleaning and normalizing the text according to the steps defined in the function, thereby preparing the text data for further analysis

Sentiment analysis:

SentimentIntensityAnalyzer from NLTK's **Vader** module is initialized, VADER is a lexicon and rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media or similar texts. It's pre-trained and can provide a sentiment score out of the box without additional training data. I have used a function **classify_sentiment.**
To process the **review_text** data using analyzer. polarity_scores(text) which results four categories: **neg** (negative), **neu** (neutral), **pos** (positive), and **compound** (an aggregated score that sums up the valence scores of each word in the lexicon, normalized between -1 (most extreme negative) and +1 (most extreme positive)).
We have commented out the categories as we would focus on **compound** score for further processing.

Scaling of Sentiment scores:
The MinMaxScaler is initialized and then fit to the **sentiment** scores, which scales these scores to a range between 0 and 1. This transformation is performed because the compound scores generated by VADER can range from -1 to 1, and scaling is required for classification machine learning models where input values are expected to be not non-negative values.

Vectorizing Text with TF-IDF
Initialize TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer that will be used to convert the text data into a format that's usable for machine learning models.
Transforms the **review_text** column into a sparse matrix of TF-IDF features.

Preparing Additional Features:
sentiment, **helpful_numbers**, and **verified_purchase** columns from the Data Frame into NumPy arrays and reshape them into column vectors (**-1, 1** indicates that NumPy will automatically adjust the number of rows and use 1 column):

Combining Features:
We employ horizontally stacks for the TF-IDF features matrix with the **sentiment**, **helpful_numbers** and **verified_purchase** arrays. The **hstack** function is used for horizontal stacking because **tfidf_features** are a sparse matrix, and this operation efficiently concatenates these features while maintaining the sparse structure.

Train and Test Data Split:
The dataset is split into training and testing subsets using the **train_test_split** function from **sklearn. model_selection**. The **combined_features** matrix serves as the input features (**X**), and **data ['rating category']** is the target variable (**y**). The dataset is divided into 80% for training and 20% for testing, controlled by the **test_size=0.2** parameter. The **random state = 'n'**parameter ensures that the split is reproducible, meaning you'll get the same train-test split every time you run this code.

Modeling:

For the classification of products based on sentiment analysis of customer reviews, we have selected three distinct classification models: Naive Bayes (specifically Multinomial Naive Bayes), Random Forest, and Support Vector Machines (SVM).
Initially I have tried Naive Bayes classifiers which are particularly well-suited for text classification tasks due to their foundation in Bayes' Theorem, which leverages the probabilities of words to make predictions. The Multinomial Naive Bayes variant is specifically designed for features that represent counts or frequency data, making it ideal for use with word counts or TF-IDF features derived from text, It is computationally efficient, making it a good choice for large datasets common in sentiment analysis projects Naive Bayes can often achieve high accuracy, especially in the context of natural language processing tasks. Given its simplicity and effectiveness, Multinomial Naive Bayes serves as an excellent baseline model. It allows for quick experimentation and can provide a benchmark for comparing other complex models.

Results and Evaluation:

Naive Bayes Classification Report

	precision	recall	f1-score	support
average	0.46	0.07	0.12	89
bad	0.63	0.26	0.37	147
good	0.76	0.98	0.85	565
accuracy			0.74	801
macro avg	0.62	0.43	0.45	801
weighted avg	0.70	0.74	0.68	801

Naive Bayes offers competitive accuracy and serves as a strong baseline, particularly given its computational efficiency. While it does not excel in any category, it provides consistent performance across the board.
The overall accuracy of the model is 0.74, meaning it correctly predicts the sentiment of reviews 74% of the time across all categories.
Looking at the models Precision and Recall values of Good, Bad, Average values the models excel in identifying the good sentiment but has difficulty distinguishing for the Bad and average sentiments accurately.

Random forest: This is next classification model we choose based on some of the benefits of Random Forest's ensemble learning method capable of handling the high dimensionality of text data without feature selection. This is beneficial for sentiment analysis tasks where the feature space can be extremely large due to the vast vocabulary of natural language, robustness, and ability to avoid overfitting.

Random forest Classification Report

	precision	recall	f1-score	support
average	0.57	0.04	0.08	89
bad	0.63	0.48	0.54	147
good	0.80	0.96	0.87	565
accuracy			0.77	801
macro avg	0.67	0.50	0.50	801
weighted avg	0.74	0.77	0.72	801

The models Recall and Precision for Average is 4% and 57% which indicates it predicted only 4% average responses out of which only 57% predicted are accurate which is not pretty good. Whereas the Recall and Precision for Bad sentiment is 48% and 63% which is better than Navies and for the good sentiment the Recall and Precision is 96% and 80% which stands better than the Naïve Bayes Classifier.

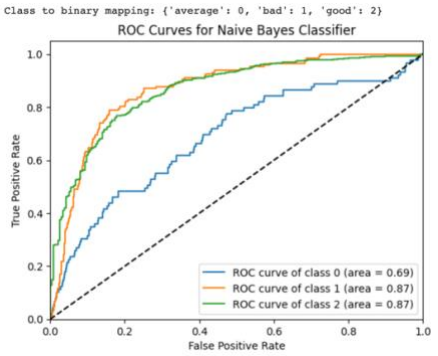
Support Vector Machines (SVM): This is the third algorithim we choose for model building and evaluating the results based on the popularity of text classification problems where each word or n-gram can be considered a dimension. SVM focuses on finding the optimal hyperplane that separates the data into classes, which is beneficial for complex sentiment classification tasks. Also, SVM versatile for various kinds of data, including linear and non-linear relationships between features.

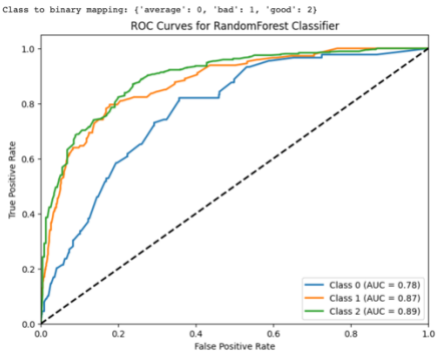
SVM Classification Report

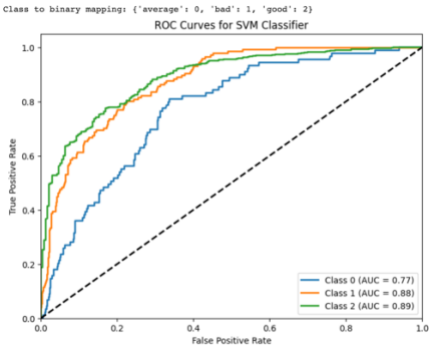
	precision	recall	f1-score	support
average	0.00	0.00	0.00	89
bad	0.67	0.04	0.08	147
good	0.71	1.00	0.83	565
accuracy			0.71	801
macro avg	0.46	0.35	0.30	801
weighted avg	0.62	0.71	0.60	801

The models ‘Average’ category Recall and Precision values are 0 which are to our surprise, also the Recall and Precision of Good and Bad Categories are not a good indication.
The SVM classification model demonstrates a significant bias towards predicting the 'good' sentiment, as indicated by its perfect recall and highest precision and F1-score in this category entirely failing to recognize 'average' and 'bad' sentiments correctly.

<u>ROC (Receiver Operating Characteristic) curve Analysis</u>:







All three classifiers show strong performance in distinguishing 'good' sentiment, with the Random Forest Classifier being slightly superior in this regard. When it comes to 'average' sentiment, SVM holds a narrow edge over the Random Forest Classifier, and both are better

than Naive Bayes. For 'bad' sentiment, SVM and Random Forest show the same level of performance and are better than Naive Bayes.
The Random Forest Classifier has the most consistent performance across all classes, closely followed by the SVM Classifier.

Conclusion:

All models struggle with the 'average' sentiment category, with SVM not correctly identifying any 'average' reviews. Random Forest shows a slightly better precision than Naive Bayes but fails significantly in recall. For ‘Bad’ category Random Forest outperforms the other models in balancing precision and recall for the 'bad' sentiment, achieving the highest F1-score. For ‘Good’ category All models excel in identifying 'good' sentiment, with Random Forest and SVM showing high precision and recall. SVM, notably, has a perfect recall but at the expense of misclassifying other sentiments as 'good.

Random Forest emerges as the most robust model for this sentiment analysis task, demonstrating the highest accuracy and a good balance between precision and recall across categories. Its superior performance in the 'bad' sentiment category and strong results for 'good' sentiment make it particularly valuable.
Naive Bayes offers competitive accuracy and serves as a strong baseline, particularly given its computational efficiency. While it does not excel in any category, it provides consistent performance across the board.
SVM shows a strong bias towards predicting 'good' sentiments, reflected in its perfect recall for this category. However, this comes at a significant cost to its ability to recognize 'average' and 'bad' sentiments, as evidenced by its lower overall accuracy and poor recall for these categories.

Future Improvement:
As observed from the initial data analysis and three ML classification model results
1)Addressing the class imbalance could potentially improve model sensitivity towards the less represented categories ('average' and 'bad'). But we can’t suggest using artificial methods like SMOTE because Interpolating between text samples does not guarantee that the generated synthetic text samples will retain meaningful semantics.
2)we can try other large NLP libraries or frameworks like BERT, Text Blob for better sentiment analysis supporting the classification.