**Natural Language processing and recommender systems – Amazon fashion Dataset**

#### Data exploration & conclusions

## Phase 1

A screenshot of a computer

Description automatically generated

The Amazon Fashion dataset contains a total of 3176 rows.

A screenshot of a computer

Description automatically generated

The following columns has missing values: style: 69 ; reviewText: 16 ; vote: 2879 ; image: 3070



Dropping rows where ‘reviewText’ columns has NaN values

A graph of red and blue dots

Description automatically generated A diagram of a box plot

Description automatically generated

Identifying outliers using 25-75% percentile and using z-score

A graph with a box plot

Description automatically generated

A close up of words

Description automatically generated

Word cloud shows the most frequent words

# **Pre-processing steps**

Below are the appropriate columns chosen for the sentiment analyzer

* reviewText: This column contains the actual customer review text, which is the primary source of sentiment about the product. Analyzing this text will enable the sentiment analysis model to determine whether the sentiment towards the product is positive, negative, or neutral. It's the most crucial column for performing sentiment analysis.
* overall: This column represents the rating given by the customer, ranging from 1 to 5. It provides a quantitative measure of the customer's sentiment towards the product, which can be used to label the data for supervised learning approaches or to evaluate the performance of the sentiment analysis models by comparing the predicted sentiment with the actual rating.
* summary: Although not as detailed as the review text, the summary often encapsulates the overall sentiment of the review in a concise manner. This column can complement the reviewText analysis by providing additional insights into the customer's sentiment, especially in cases where the summary might capture sentiment not explicitly stated in the reviewText.
* verified: Indicates whether the purchase was verified. While not directly related to sentiment analysis, this column can be useful for filtering the dataset to include only verified reviews, ensuring that the analysis is based on genuine customer feedback and potentially improving the model's accuracy and reliability.

The selection of these columns is driven by their relevance to sentiment analysis:

* reviewText is directly analyzed to extract sentiment, making it indispensable for the task.
* overall provides a straightforward, quantifiable indication of sentiment, useful for both labeling and model evaluation.
* summary offers a brief yet potentially insightful expression of sentiment, serving as a useful supplement to the detailed review text.
* verified helps in ensuring the authenticity of the reviews being analyzed, which can contribute to the overall quality and trustworthiness of the sentiment analysis.
* These selections align with the project's objectives to leverage customer textual reviews for sentiment analysis, ensuring that the analysis is based on relevant, genuine, and comprehensive data.

## Phase 1

**Duplicate Removal:** This step involves removing duplicate entries to avoid any bias or skewness in the sentiment analysis. In the provided code, duplicates are identified based on the combination of 'reviewerID', 'asin', and 'reviewText', ensuring that each review is unique to a specific product and user. This is crucial to maintain the integrity of the dataset.

A computer code with black text

Description automatically generated with medium confidence

**Outlier Removal:** Outliers can significantly affect the performance of sentiment analysis models. The code indicates that outliers were removed based on review length, although the specific method is not shown. Typically, reviews that are too short may not contain enough information for analysis, while excessively long reviews might include irrelevant content. Removing these helps focus on reviews that are more likely to contain meaningful sentiment.

**Data Labeling:** The sentiment labels are derived from the 'overall' rating scores, with ratings of 4-5 marked as 'Positive', 3 as 'Neutral', and 1-2 as 'Negative'. This labeling is foundational for supervised learning models and helps in evaluating the performance of lexicon-based sentiment analysis.

**Removal of Unverified Reviews:** Unverified reviews might be less trustworthy, so the code filters to include only those reviews marked as verified. This increases the likelihood that the sentiment analysis is performed on authentic customer feedback.

**Column Selection:** Only the essential columns are retained for the sentiment analysis, 'reviewText' for the actual review content, and 'label' for the sentiment label. This streamlines the dataset to include only the data necessary for the task at hand.



A screenshot of a computer code

Description automatically generated



**Text Pre-processing steps**

* Converting text to lowercase to ensure uniformity.
* Removing punctuations, incomplete words, and white spaces to clean the text and prepare it for analysis.
* Tokenization, which splits the text into individual words or tokens.
* Removal of stopwords, which are common words that typically don't carry sentiment.
* Stemming and lemmatization, which reduce words to their base or root form, improving the model's ability to associate different forms of the same word with the same sentiment.

A screen shot of a computer code

Description automatically generated



# **Models**

## Assumptions/Heuristics/algorithms/packages used

**Vader Analysis**

* **Assumptions**: Vader is specifically attuned to sentiments expressed in social media. It assumes that word order matters for sentiment intensity and directly captures polarity (positive/negative) and intensity (strength) of emotions.
* **Heuristics**: Employs a combination of qualitative and quantitative methods to capture lexical features and incorporates intensifiers (e.g., "very") to adjust sentiment intensity.
* **Algorithms**: Uses a rule-based model that applies grammatical and syntactical rules to analyze text sentiments.
* **Packages Used**: vaderSentiment Python package.

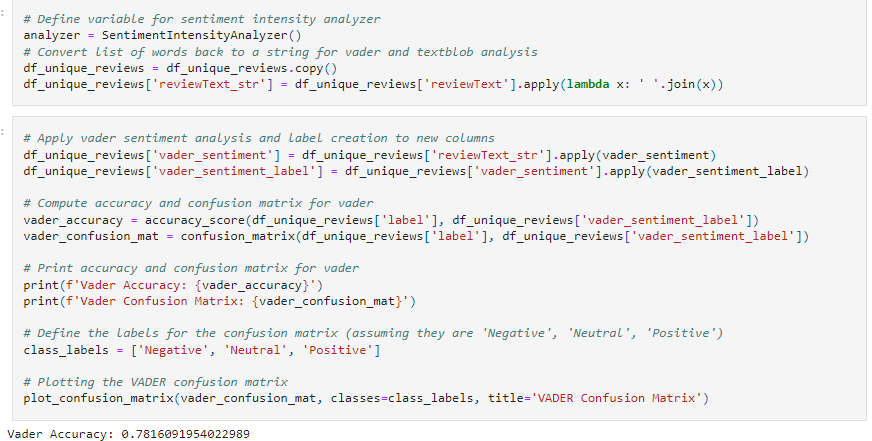
**TextBlob Analysis**

* **Assumptions**: TextBlob assumes that sentiment extraction can be effectively achieved through a straightforward API by providing access to common text-processing operations through a familiar interface.
* **Heuristics**: Utilizes pattern analysis and a Naive Bayes classifier trained on a movie reviews dataset for sentiment analysis.
* **Algorithms**: Employs both rule-based and machine learning techniques to assign polarity scores and subjectivity scores.
* **Packages Used**: textblob Python package.

## Model explanations and how it works

### **Vader Analysis**

Vader (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. It determines the sentiment of a text by analyzing the text's lexical features for positivity or negativity while considering intensifiers and grammatical constructs that may influence the sentiment intensity. For each piece of text, Vader returns a compound score that aggregates the computed sentiment scores to classify the sentiment as positive, neutral, or negative.



A diagram of a blue and white box

Description automatically generated with medium confidence

### **TextBlob Analysis**

TextBlob simplifies text processing in Python. For sentiment analysis, it relies on the pattern library and Naive Bayes algorithms to offer polarity and subjectivity scores. Polarity measures how positive or negative the sentiment is, ranging from -1 (very negative) to 1 (very positive). Subjectivity quantifies the amount of personal opinion and factual information contained in the text, ranging from 0 (very objective) to 1 (very subjective).

A screenshot of a computer code

Description automatically generated

A chart with blue squares

Description automatically generated

## Fine tunning steps

### Vader Analysis

* Pre-processing: Minimal pre-processing is required, as Vader handles negations and punctuation inherently. However, converting emoticons to their word equivalents can enhance performance.
* Parameter Tuning: Adjust the compound score thresholds to better categorize sentiments as positive, neutral, or negative based on the context or domain-specific needs.

### TextBlob Analysis

* Pre-processing: Basic text cleaning including removing unnecessary symbols and numbers can improve the quality of the analysis.
* Classifier Training: For more tailored sentiment analysis, train the TextBlob Naive Bayes classifier with domain-specific datasets to enhance accuracy.

# **Testing results summary**

### Vader Sentiment Analysis Model:

* Accuracy: The Vader model achieved an accuracy of approximately 0.7816.
* Confusion Matrix:
  + True Negatives (TN): 4
  + False Positives (FP): 7
  + False Negatives (FN): 14
  + True Positives (TP): 260

This confusion matrix indicates that the Vader model was quite proficient at identifying positive reviews but struggled more with negative and neutral ones, as evidenced by the lower TN and higher FN counts.

### TextBlob Sentiment Analysis Model:

* Accuracy: The TextBlob model achieved an accuracy of approximately 0.7787.
* Confusion Matrix:
  + True Negatives (TN): 11
  + False Positives (FP): 3
  + False Negatives (FN): 11
  + True Positives (TP): 258

The TextBlob confusion matrix suggests a more balanced performance across different sentiment classes, with similar numbers for FN and TN when compared to the Vader model. However, it had a slightly lower accuracy and TP count, which indicates that while it was better at distinguishing negative sentiments, it might have missed some positive ones.

# **Final Conclusions**

* **Data Quality and Cleaning:**
* The Amazon Fashion dataset initially contained 3,176 reviews with some missing data, specifically in the 'style', 'reviewText', 'vote', and 'image' columns. This necessitated data cleaning steps, including removing rows with missing 'reviewText' to ensure the quality of sentiment analysis.
* **Review Distribution:**
* The exploration of the data revealed a varied number of reviews per product and per user, indicating a wide range of engagement among customers. Some products attracted more reviews, which could suggest higher popularity or provoke more feedback from customers.
* **Outlier Handling:**
* Outliers were identified and treated accordingly to minimize their potential impact on the sentiment analysis models. The details of this process were not explicitly provided, but such actions are critical in maintaining a high-quality dataset for analysis.
* **Sentiment Analysis Models:**
  + The sentiment analysis using the Vader and TextBlob models showed that Vader achieved an accuracy of approximately 0.7816, while TextBlob achieved slightly less at approximately 0.7787.
  + The confusion matrices for both models showed that while they were proficient at identifying positive sentiments, they had limitations in accurately classifying negative and neutral sentiments. Specifically, the Vader model identified 260 true positives and had some misclassifications across negative and neutral sentiments, and the TextBlob model identified 258 true positives with a more balanced performance but also misclassified some sentiments.
* **Word Cloud Insights**: A word cloud generated from the reviews highlighted positive words like 'comfortable', 'love', 'fit', 'perfect', and 'shoe' as the most prominent, suggesting overall positive sentiment in the dataset.
* **Conclusion**:
* The sentiment analysis models applied to the Amazon Fashion dataset reveal valuable insights into customer sentiments, with a general leaning towards positive reviews. The analysis underscores the necessity of thorough data cleaning and preprocessing to improve model accuracy. Despite the challenges in identifying less frequent sentiments, the models demonstrate a strong ability to capture the majority sentiment, which is positive in this dataset. These findings can inform strategies to address customer feedback and enhance product recommendations.

# **References**

Bajaj, A. (2023, April 19). *Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/>

Mishra, D. (2024, January 12). *Building a recommendation system for Amazon fashion products using deep reinforcement learning...* Medium. <https://medium.com/@vibhu12345/building-a-recommendation-system-for-amazon-fashion-products-part-1-6469b58bd427>

Shah, P. (2021, December 15). Sentiment Analysis using TextBlob - Towards Data Science. *Medium*. <https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524>

Sharma, S. (2022, August 20). Sentiment analysis using the SentiWordNet Lexicon - Srishti Sharma - Medium. *Medium*. <https://srish6.medium.com/sentiment-analysis-using-the-sentiwordnet-lexicon-1a3d8d856a10>

**Phase 2:**

**Feature Extraction:**

We have dropped the following columns: ('vote','style','image','reviewTime','reviewerID','asin','reviewerName','unixReviewTime') because there were some columns containing null values, empty values, and mainly most columns were irrelevant because it was not contextual based on the recommendation, and it was not useful columns that can contribute to the review-rating recommendations.

We have dropped the columns the verified, review\_length, and the overall columns has values (1, 2, 3, 4, 5) which were basically ratings and that we converted to label columns which contains string values (negative for values 1 and 2, neutral for value 3, and positive for values 4 and 5).

So the columns we took finally were reviewText which contains the review text and the label column which contains the positive, negative and neutral values in column.

Then we removed the stopwords from the reviewText and dropped the duplicates from the reviewText columns and our dataframe shape was (657920, 2)

Then we applied the basic preprocess function and then printed the value counts from the label column and we found that there were more positive values which was imbalanced dataset and then we balanced the dataset with total labels of around 2000 for training in which all the values were equally distributed. We also created another subset which samples 2000 values from the original filtered dataset which is also balanced.

**Machine Learning Modelling with Word2Vec Text Representation Technique**

From the subset sample which is balanced we tokenized and then created a function which will take the tokenized reviews and the model and create the Word2Vec representations of the tokenized reviews, and we are creating a stack and converting nan values to numeric values that is (NaN to 0) and then y with the labels.

The shape of X an y:

((2000, 100), (2000,))

Creating a train-test split of 90% training and rest 10% testing and also shuffling according to the labels (y) and then creating 4 models from sklearn library:

1. Logistic regression – Accuracy – 68.5%
2. Support Vector Machine (SVM) - Accuracy – 68.5%
3. Gradient Boosting Classifier (GBC) - 70%
4. Multi-layer Perceptron Classifier (MLP) – 68.5%

The accuracies were quite low and not as expected. Only Gradient Boosting Classifier worked well among all.

**Machine Learning Modelling with TF-IDF Text Representation Technique**

Initializing the TF-IDF Vectorizer and then calling the fit\_transform method on the reviewText column which will create the text representation of the reviewText and then using the train-test split with 90% of training data and 10% of testing data. Then we created the 5 machine learning models:

1. Logistic Regression – Accuracy – 77.5%
2. Support Vector Machine (SVM) – Accuracy – 74.5%
3. Multinomial Naïve Bayes (MNB) - Accuracy – 70.5%
4. Gradient Boosting Classifier (GBC) - Accuracy – 76%
5. Multi-layer Perceptron Classifier (MLP) - Accuracy - 74.5%

Saving the models to pickle files for further testing.

**Creating balanced dataset for apple-to-apple comparison**

Creating a df\_comparison for testing the models with balanced dataset of label columns containing total 1000 values for testing.

Loading the csv file into a new python file and below are the value counts:

A screenshot of a computer

Description automatically generated

Calling the Vader and Text Blob Sentiment function used same in phase 1 to check and test and it worked well with around 80% with vader sentiment.

Then we loaded all the models and applied the text representation technique (TF-IDF) and called the predict function on the model on the testing dataset (X) and calculated the accuracy, precision, recall, F1-score of the models.

A black and white screen with numbers and lines

Description automatically generated

Among all the SVM and the MLP worked and outperformed well with 80% and 94% accuracy.

**Recommendations System – Using LDA Approach**

We loaded the original big dataset and below are the columns of the original dataset:

A screenshot of a computer program

Description automatically generated

We only considered the columns overall, reviewText, summary, asin columns as it was helpful for building the recommendation system because the overall columns gave the original rating, reviewText is the review of the product, the summary is the summary of the product review and the asin is the product id.

We combined the reviewText and the summary into the new column named review.

So now the data frame shape is (883636, 3).

We dropped the duplicates and the new shape is (873740, 3)

Then we applied the text preprocessing (like lowering the text, removing punctuations, removing spaces) and then removing the stopwords and then tokenizing the words.

A screenshot of a computer

Description automatically generated

We also removed the reviews which were empty.

The final data frame shape is (873740, 3).

Calculating the value counts in the asin column:

A screenshot of a computer screen

Description automatically generated

Then we are taking the index of the of top 20 asin value counts and for each asin value we are only taking 200 reviews in other words we took the top 20 products, and each product has minimum of 200 reviews. Below is the attached picture of the asin and its value counts:

A screen shot of a computer

Description automatically generated

We tokenized each review and then appended the tokens into a list and created a dictionary using corpora and creating a corpus which will create a BOW (Bag of Words) from the document using doc2bow function and for every text in the text\_data list.

Then creating the LDA model which will take the corpus the num\_topics to consider and the dictionary which we created earlier which maps words.

Then printing the topics from the LDA model.

Then getting the item from the corpus and then passing to the get\_document\_topics function from LDA model and getting the topics and then the topics are:

A screenshot of a computer

Description automatically generated

Then creating dominant function which will select the top 1 topic with the highest probability value and then applying the function to the topics column and created the new column dominant\_topic column with the highest probability:

A screen shot of a computer

Description automatically generated

Calculating the average ratings of each topic by grouping all the dominant\_topic values together and then calculating the mean of the overall column.

Then the taking the average of the overall column and dominant\_topic column by mapping it into the average rating of the topic and then rounding it to 2 decimal places by adding new column named enhanced\_rating:

A screen shot of a computer

Description automatically generated

**Recommendation System - Using Cosine Similarity Approach**

Getting the data frame used same in LDA approach and then initializing the TF-TDF and then calling the fit\_transform on the reviewText column and creating the tfidf\_matrix and then calculating the cosine similarity of the matrix.

TF-IDF Matrix shape: 4000x5293

Cosine Similarity shape: (4000, 4000))

Then getting the indices of the top similar reviews for each reviews then mapping it into the overall column and then printing the cosine similarity of all rows according to the top similar reviews index.

Computing the weighted ratings based on the cosine similarity with the following formula:

weighted\_ratings = np.sum(similar\_ratings \* cosine\_similarity[:, top\_similar\_indices], axis=1) / np.sum(cosine\_similarity[:, top\_similar\_indices], axis=1)

Then using np.where function which will return 0 if there are any NaN values or else it will return the computed weighted\_ratings.

The shape of the weighted\_ratings is (4000, 5) that is each review has 5 rating values:

A black screen with white numbers

Description automatically generated

Getting the maximum values among all 5 from whole weighted ratings and then rounding it to 2 decimal places and then creating a new column named computed\_rating\_tfidf with these computed ratings.

**Printing the original overall rating and the computed rating using weighted\_ratings:**

A screenshot of a computer screen

Description automatically generated

**Comparison of Overall Ratings, Ratings calculated using LDA Approach, Ratings calculated using TF-IDF Approach**

A screen shot of a computer

Description automatically generated

**Plotting the comparison Plots:**

A graph with different colored lines

Description automatically generated A graph with different colored lines

Description automatically generated

There is not major difference between LDA approach and TF-IDF approach for the above 10 classes.

**Recommendations – Using LLM Approach**

Loading the original big dataset and taking the overall and reviewText column and then selecting the rows whose reviews length is greater than 100 and then splitting it.

Then the Review Count length vs the Number of Reviews histogram plot is plotted below:

A graph with blue squares

Description automatically generated

**Using T5 Model:**

Calling the summarization pipeline from the transformers library and then loading the T5 base model and then setting the minimum and the maximum length of 50 and getting the summary as the output from the function.

The output of the review from the T5 base model is:

"i've had Nike's before and have always been pleased with the comfort, performance, and quality . but this was the first time i ordered a pair online without trying them on first, so I was nervous ."

Then the creating 2 column one contains the review summary from the T5 model and another storing the length of the review.

**Using BART Model:**

Calling the BART tokenizer and the model from transformers and using both returning the summary with minimum and the maximum length of the summary.

Getting the review summary from the BART base model and then creating 2 columns one with bart generated review summary and another with the length of the generated review summary.

**Testing Phase:**

Taking the reviews that have question mark in the review and dropping the duplicates and the shape of question reviews is (3, 2).

Then calling the text generation pipeline from transformers model using GPT-2 model then generating a function with the prompt as “As an Amazon service representative, answer the following question from a customer.” Then passing the prompt with max\_length of 100 and getting the response.

For every review we are getting the response from the model.

Then formatting the answers and then adding the column named answers to the question\_reviews data frame.

Selecting the custom review and then getting the response from the model

Question is: is this product available in different colors?

The answer the model generated is:

'Amazon.com is currently in stock on all available Amazon.ca products.\n\nWhat is your experience as an Amazon customer?\n\nAs an Amazon customer, you will receive a confirmation email through your email address. You will then be directed to Amazon.com and received an updated product listing.\n\nThere is a 5-'