Heriot-Watt University

F20AA Coursework Report

F20AA - Applied Text Analytics Coursework 2 Report

Name: Devesh Pansare
HW ID: H00353155
HW username: dp2004

Name: Hana Khan
HW ID: H00379151
HW username: hsk2001

Name: Neil Patrao
HW ID: H00385850
HW username: nrp2000

Due Monday, 1st April (Week 12)

1 Data Exploration and Visualization

In this section we discuss how we explored and visualized the dataset

1.1 Missing Values

We begin by loading the train and test datasets and proceed to search for missing values and, not so surprisingly, find that we do, in fact, have them. We see that we have 8 'NaN' values in the train set and 4 in the test set. These missing values pose a problem to our upcoming processing step and will be addressed there.

1.2 Class Imbalance

We proceed with some simple exploratory data analysis (EDA). We immediately notice a significant class imbalance:

Overall	Review
5	283158
4	45831
3	21470
1	10772
2	9624

Table 1: Class Imbalance

We further visualize this class imbalance with the help of a countplot 1 and a funnel chart 2:

1.3 Duplicate Reviews

Duplicate values are undesirable as they lead to analysis bias and overfitting along with an increase in computational costs. For these reasons, these duplicate reviews must be taken care of. We visualize these duplicates with the help of a horizontal line plot in 3. The fully zoomed out plot on the right looks like it's all duplicates but this is only due to the horizontal lines having a minimum thickness.

1.4 Word Count

Lastly, we visualize the word-count distribution over all the reviews. We visualize this using kernel distributions in 4 and 5

1.5 Word Cloud

Finally, as part of visualization, we define two functions generate_word_cloud and categorize_rating.

The generate_word_cloud function creates a word cloud visualization for a given set of reviews, showing the most common words in the text. The size of each word in the cloud represents its frequency in the reviews. The function takes a list of reviews and a sentiment label as input, concatenates the reviews into a single string, generates the word cloud, and displays it using Matplotlib.

The categorize_rating function categorizes ratings into three groups: 'Positive', 'Neutral', and 'Negative'. Ratings greater than 3 are classified as 'Positive', ratings equal to 3 are classified as 'Neutral', and ratings less than 3 are classified as 'Negative'.

After defining these functions, the code filters the reviews based on their ratings into three categories: positive, neutral, and negative. It then generates word clouds for each sentiment category using the generate_word_cloud function. These word clouds provide a visual summary of the most common words associated with each sentiment, helping to understand the overall sentiment of the reviews.

We display the word clouds for each sentiment in 6, 7 and 8

2 Text Processing and Normalization

In this section, we discuss the text processing and normalization steps used to transform text into standard format, making it consistent and easier to process.

2.1 Steps

1. **Standardization:** The text was converted to a standard format, i.e., to lower case, so that different variations of the same word can be treated the same. Lowercasing reduces the vocabulary size where words with different cases represents the same word.

2. **Removing Missing values:** This is done so that further preprocessing can be applied to the text.

3. **Dropping duplicates:** The given dataset contained duplicates which can potentially introduce bias to the text classification task. Removing duplicates helps us focus on unique instances of the text data and reduces the chances of overfitting.

4. **Removing special characters:** The text was cleaned to ensure that special characters like whitespace characters, emoji and punctuations were removed or replaced.

5. **Tokenization:** Text is broken down into meaningful words for further analysis. Here only tokens with alphabetic characters are retrieved.

6. **Removing stop words:** Words that don't contain any significant meaning like "this", "and", etc were removed. This helps us to remove noise from our text and helps reducing dimensionality of the feature space.

7. **Stemming and lemmatization:** These were used to reduce words to their root form.

    - **Stemming:** This reduces related words to the same stem even if it's not a dictionary word thus reducing the vocabulary size. We used NLTK's Porter algorithm for stemming. Upon applying stemming, we saw that some of the words don't represent real English words. For example, the sentence, "It works well with my machine. I use mostly cones on it.", became, "[work, well, machin, use, mostli, cone]", where "machin" and "mostly" aren't actual words.

- **Lemmatization:** For a more accurate approach we used NLTK's WordNet lemmatizer. This ensures that the root word belongs in the dictionary. For the above example sentence after lemmatization became, "[work, well, machine, use, mostly, cone]", all of these represent a word in the English dictionary.

3 Vector Space Model and Feature Representation

To represent text into a format which ML model supports, we implemented various vectorization techniques. We discuss them in this section

3.1 Bag of Words Representation

We created a sparse matrix from the frequency of vocabulary words where each row in the matrix is a sentence vector. We used Scikit's CountVectorizer to implement this representation. We found that there are 39090 unique vocabulary words in our train dataset and each sentence vector length is equal to the vocabulary size.

3.2 TF-IDF Vectorization

Term Frequency–Inverse Document Frequency is another frequency-based method that considers the importance of a word in a document. It is an improvement over Bag of Words. In BOW, all the words contribute equal importance even prepositions, conjunctions, and articles, which is not desired. In TF-IDF, frequent words don't overcome the less frequent but important words. We made use of Scikit's TfidfVectorizer. According to TF-IDF, the term 'work', represented by 'wo', achieves the highest score, making it the most significant feature. In contrast, numerous words that might have been considered for feature construction in Bag of Words contribute little to no value here, represented by a score of 0.

3.3 Word2Vec

Every word is represented as an n-dimensional vector, i.e., all words are assigned to an n-dimensional space in a way that words with similar meanings are located near each other in this high-dimensional space. We used the Gensim library and trained the Word2Vec model on our train dataset. Unlike Bag of Words and TF-IDF, where it ignored semantics completely, Word2Vec was more contextually aware.

3.4 Comparing Bag of words, TF-IDF and Word2Vec

We applied BOW, TF-IDF and Word2Vec with random forest classifier on default parameters to check the performance of each. We saw that td-idf performs well compared to others with a train accuracy of 78.8 and test accuracy of 75.6%. This could be because the documents in the dataset were relatively short. Word2Vec performed poorly with a train accuracy of 98.8% and test accuracy of 73.6%. It is clearly seen that it overfit on the train dataset. This is because the training process requires large amount of data and computational resources.

3.5 Comparing n-gram Features

We compared different n-gram hyperparameters of TF-IDF vectorizer while keeping the default hyperparameters of Gradient Boosting to check the performance for each. We found that unigrams together with bigrams performed the best.

4 Model Training, Selection and Hyperparameter Tuning and Evaluation

4.1 Model 1: Random forest classifier with TF-IDF Vectorization

Keeping the default hyperparameters of TF-IDF vectorization using unigrams, we tuned the hyperparameters of random forest with the help of hyperopt on the processed and lemmatised train dataset. Cross validation was used to calculate the score of Hyperopt because it provides a more robust and reliable estimate of model performance compared to using a single train-test split. It also prevents overfitting to a single train-test split. Averaging the evaluation metric across multiple folds of the dataset reduce the variance in the performance estimation.

4.1.1 Hyperparameter Tuning

- **n_estimators:** [50, 100, 1000]
- **max_depth:** [5, 10, None]
- **min_samples_split:** [2, 5, 10]
- **min_samples_leaf:** [1,2, 5]

The best parameters found by hyperopt were 'max_depth': 2, 'min_samples_leaf': 0, 'min_samples_split': 1 and 'n_estimators': 0. These are the indices of the best parameter values from the given choices list.

4.1.2 Results

We got a train accuracy of 97.7% and a test accuracy of 74.7%. This shows that the model overfits on the train dataset. To address overfitting and improve the generalization performance of our model, we manually tuned max_depth parameter:

- **max_depth:** [100, 200, 250, 500]

Limiting the depth prevents trees from becoming too complex and overfitting to the training data. The model with max_depth = 200 gave the best results with a train accuracy of 89% and a test accuracy of 74.5%.

4.2 Model 2: Logistic Regression with TF-IDF Vectorization

Keeping the default hyperparameters of TF-IDF vectorization using unigrams, we tuned the hyperparameters of logistic regression with the help of hyperopt on the processed and lemmatised train dataset.

4.2.1 Hyperparameter Tuning

- **C:** [0.001, 0.1, 1.0, 10.0]
- **penalty:** ['l1', 'l2']
- **class_weight:** ['balanced', None]
- **solver:** ['liblinear', 'saga']

The best parameters found by hyperopt were 'C': 2, 'class_weight': 1, 'penalty': 0 and 'solver': 1.

4.2.2 Results

We got a train accuracy of 78.1% and a test accuracy of 77.0%. Taking the best hyperparameters, we ran it on a balanced dataset, and we saw that the accuracy on the test dataset reduced drastically. Train accuracy of 67.7% and a test accuracy of 61.2%.

4.3 Model 3: Gradient Boosting with TF-IDF Vectorization using Unigrams

Keeping the default hyperparameters of TF-IDF vectorization using unigrams, we tuned the hyperparameters of Gradient Boosting classifier manually on the processed and lemmatised train dataset.

4.3.1 Hyperparameter Tuning

- **learning_rate:** [0.01, 0.05, 1.0]
- **subsample:** [0.6, 0.8, 1]
- **n_estimators:** [50, 200, 400, 500]
- **max_depth:** [5, 7]

The best parameters we found were 'learning_rate': 0.05, 'subsample': 0.6, 'n_estimators': 400 and 'max_depth': 5.

4.3.2 Results

We got a train accuracy of 78.3% and a test accuracy of 76.2%.

4.4 Model 4: Gradient Boosting with Tuned TF-IDF Vectorization

We tuned the hyperparameters of TF-IDF vectorizer manually while keeping the default hyperparameters of Gradient Boosting classifier on the processed and lemmatised train dataset.

4.4.1 Hyperparameter Tuning

- **ngram_range:** [(1,1), (1, 2), (2,2), (1,3), (3,3)]
- **max_df:** [0.5, 0.7, 0.9]
- **min_df:** [2, 3, 5]

The best parameters we found were 'ngram_range': (1,2) and 'max_df': 0.5.

4.4.2 Results

We got a train accuracy of 76.1% and a test accuracy of 75.6%.

5 Modelling Text as a Sequence

In NLP, modelling text as a sequence involves treating text data as a sequence of words, where the order of the elements is important. Modelling text helped us understand that the context and meaning of words in a sequence is crucial. NLP models learned the relationships between words and their context, enabling them to make more accurate predictions in tasks like sentiment analysis, text classification, and machine translation.

5.1 Model Selection

In this section, unlike Part D, we trained various types of deep-learning models for text-classification, including LSTM, Stacked LSTM, CNN, Bidirectional LSTM, GRU, and BERT. Models like LSTM, Stacked-LSTM and Bi-directional LSTM were chosen for their ability to capture long-term dependencies in sequential data, making them well-suited for analyzing text sequences. On the other hand, CNNs are effective for extracting local features in sequential data, making them suitable for tasks like text classification where important features may be localized within the text. By using the convolutional layers, the model learns to detect patterns such as word sequences or phrases that are indicative of sentiment.

Moreover, bidirectional LSTMs allows the model to capture dependencies from both past and future contexts. This is beneficial for tasks where the sentiment of a word is influenced by both preceding and succeeding words, improving the model's understanding of the overall context. GRU was also added to compare the performance with the other models given that it is computationally more efficient.

Finally, the BERT was chosen for its ability to capture contextual information from the entire input sequence. Unlike traditional RNN-based models, BERT uses a transformer architecture that allows it to consider dependencies between words regardless of their distance in the sequence. This makes it particularly effective for tasks requiring a deep understanding of context, such as sentiment analysis in long texts.

5.2 Data Preprocessing

We processed and lemmatized the datasets as seen in part B and part C to get X_train and X_test. To further preprocess the datasets, several key processes were applied to prepare the text data for training the models. We started off with Tokenizing where the text is split into individual words aka tokens performed using the Tokenizer class from Keras. This mapping allows the neural network to represent words as integers, which can be used as input.

Then, the text data is converted into sequences of integers using the texts_to_sequences method. Each word in the text is replaced with its corresponding integer value from the tokenizer's vocabulary. This step essentially transforms the text data into a format that the neural network can understand and process.

After applying padding to all sequences to ensure that all the sequences have a uniform length. This is necessary because neural networks require fixed-dimensional input. The target labels (y_train and y_test) are adjusted to start from 0 instead of 1, required for the sparse_categorical_crossentropy loss function, which expects labels to be in the range [0, num_classes-1].

5.3 Model Training and Hyperparameter Tuning

We made use of the helper function train_model() that provides a flexible way to train different types of models for text classification, including the complex BERT model. This function uses the best hyperparameters obtained from the optimization process to train all the models.

Moreover, we also performed hyperparameter tuning using grid search for all the models to find the best hyper-parameters. The models were tuned on the following parameters:

- **'batch_size':** [64, 128]
- **'epochs':** [3,5,7,10]
- **'units':** [30, 50]
- **'dropout':** [0.2, 0.5]
- **'recurrent_dropout':** [0.2, 0.5]
- **'activation':** ['softmax', 'relu']

5.4 Evaluation Metrics

In evaluating the performance of the models for text classification, several standard metrics such as accuracy, precision, recall, and F1-score are used. These metrics are appropriate for text classification because they provide a comprehensive evaluation of the model's performance, considering both correct and incorrect predictions. By analyzing these metrics, we gained insights into how well the model is performing and identify areas for improvement.

5.5 Results and Comparison

The results obtained from each of the models, with the best hyperparameters after tuning, were as follows:

Model	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
LSTM	81.34%	76.79%	76.64%
Stacked-LSTM	78.93%	77.26%	79.78%
CNN	94.61%	72.39%	72.35%
Bidirectional-LSTM	81.22%	76.28%	76.19%
GRU	75.85%	74.22%	74.07%

Table 2: Model Accuracies

Overall, the Stacked-LSTM model was the best-performing model after hyper-parameter tuning each of the models, achieving the highest accuracy on the test set.

Due to computational limitations, we were unable to train, and hyper-parameter tune the BERT model for our text classification task. BERT requires significant computational resources and memory, which exceeded our available resources. As a result, we focused on training and tuning other models, including LSTM, Stacked LSTM, CNN, Bidirectional LSTM, and GRU, which were more feasible given our constraints.

6 Topic Modelling of High and Low Ratings

The purpose of topic modeling in analyzing reviews, particularly in the context of high and low ratings, is to uncover underlying themes or topics that are prevalent in customer feedback. By applying topic modeling techniques such as Latent Dirichlet Allocation (LDA), we aim to extract meaningful insights from the reviews and understand the key factors influencing customer sentiment.

In this section, we identify common themes or topics in reviews with high ratings to understand what aspects of the product or service customers appreciate the most. Additionally, we also identify common themes or topics in reviews with low ratings to pinpoint areas for improvement and address potential issues that may be causing dissatisfaction among customers.

By achieving these goals, we can gain valuable insights into customer opinions and preferences, which can inform decision-making processes and help improve the overall customer experience.

In the implementation, we first began with an initial analysis to get an idea of the complete dataset as compared to the High-rated and the Low-rated reviews separately.

6.1 Methodology

We made use of Latent Dirichlet Allocation (LDA) to identify common topics in the reviews. The choice of parameters included selecting 20 topics and considering the top 10 words for each topic due to it being a common choice of approach in topic modelling, particularly with LDA. 20 topics provides a balance between having enough topics to capture diverse aspects of the data while avoiding too many topics,

which can lead to overly specific or redundant themes. On the other hand, choosing the top 10 words for each topic provides a concise representation of the main keywords associated with each topic. We tested out with more top words, but the specify of the topics diluted a bit, whereas with fewer than 10 words, we felt a lack of context from the words that were output.

The dataset used for topic modelling was a lemmatized dataset. This helped us improve the quality of the topics extracted from the text. This helped us a ton in consolidating similar topics and reducing noise in the model. Moreover, lemmatization also helped us improve the interpretability of the topics by reducing the complexity of the vocabulary used in the text data.

Moreover, we also perform Sentiment analysis over the top 20 topics over the whole dataset (lemmatized), High-Rated reviews and the low rated-reviews.

These preprocessing steps were essential to ensure that the text data was clean and ready for analysis with LDA. By tokenizing the text and removing stop words, the model could focus on meaningful words and phrases to identify relevant topics in the reviews.

6.2 Results

From the Sentiment analysis we notice that the total dataset gave 10 topics as positive and 10 topics as negative out of the top 20 topics. This tells us that the dataset that we used was generalized well as far as sentiments are concerned. Moreover, for the high-rated reviews dataset, we received 13 positive topics and 7 negative comments. Lastly, for the low-rated reviews, we had 0 positive topics and 20 negative comments from the top 20 topics. This shows us that the High Rated reviews are clearly positive sentiment, low-rated as negative sentiment and the whole dataset is generalized well based on sentiments.

We display bar charts that show the frequency of each topic in the respective datasets. For example, 11 the frequency of each topic in the High-rated reviews dataset.

10 shows the frequency of each topic in the Low-rated reviews dataset.

Additionally, we also visualize common topics between 1-star and 5-star reviews topics. We then print out a bar chart that displays the frequency of the common words in 12.

This helps us identify and visualize the common words that are prominent in both positive and negative reviews, providing insights into the key factors that contribute to the sentiment expressed in the reviews.

6.3 Discussion

The analysis conducted in this section revealed common themes in reviews with high and low ratings, such as product quality, delivery time, and packaging.

The topic modeling analysis of high and low ratings in customer reviews provides valuable insights into customer sentiments and preferences. The identification of common themes in high-rated reviews highlights the aspects of a product or service that customers appreciate the most.

Conversely, the analysis of low-rated reviews helps identify areas for improvement and areas of dissatisfaction among customers. By understanding these pain points, businesses can take proactive measures to address these issues and improve their products or services.

The sentiment analysis conducted on the top 20 topics in each dataset further reinforces the findings from the topic modeling. The distribution of positive and negative topics in the datasets aligns with the reviews' overall sentiment, indicating that the topic modeling effectively captured the underlying sentiment in the reviews.

6.4 Conclusion

In conclusion, the topic modeling analysis of high and low ratings in customer reviews has provided valuable insights into customer sentiments and preferences. The analysis showed the effectiveness of using topic modeling techniques like Latent Dirichlet Allocation (LDA) to extract meaningful insights from text data.

Overall, the findings from this analysis can serve as a valuable resource for businesses looking to understand customer sentiments and improve their products or services based on customer feedback.

7 Conclusion

In this analysis, we explored various aspects of analyzing product reviews using advanced machine learning techniques. We began with data exploration and visualization, gaining insights into rating distributions and review lengths. This exploration helped us understand the dataset and identify areas for further analysis.

Next, we focused on text processing and normalization, applying tokenization, stemming, and lemmatization to prepare the text data. These steps were crucial for cleaning the data and making it suitable for modeling.

Moving on to the vector space model and feature representation, we used TF-IDF and Word2Vec to convert text into numerical forms. This transformation enabled us to apply machine learning algorithms and extract meaningful insights.

In the model training, selection, hyperparameter tuning, and evaluation phase, we trained LSTM and XGBoost models for stock price prediction and sentiment analysis. Through tuning and cross-validation, we improved model performance and selected the best-performing models.

In modeling text as a sequence, we explored LSTM models for sequence classification, discussing hyperparameter tuning strategies for improved performance.

Lastly, in the topic modeling of high and low ratings section, we used LDA to uncover common themes in customer reviews. This analysis provided valuable insights into customer sentiments and preferences, aiding in business decisions and enhancing customer satisfaction.

Overall, this analysis demonstrates the efficacy of advanced machine learning techniques in extracting insights from text data. By combining data exploration, text processing, modeling, and evaluation, we gain a comprehensive understanding of customer feedback, empowering us to make informed decisions for product and service enhancement.

8 Appendix A

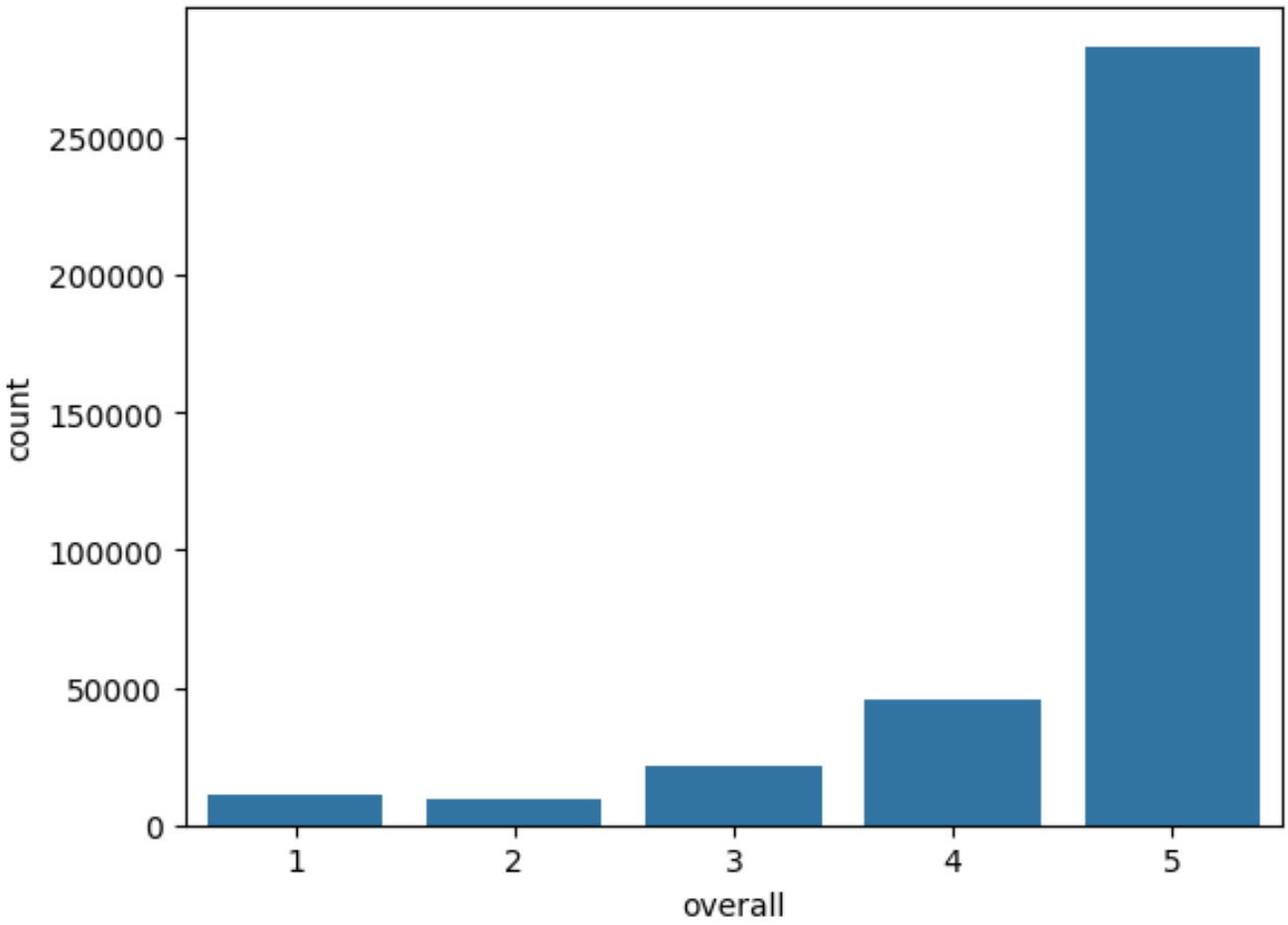

Figure 1: Countplot

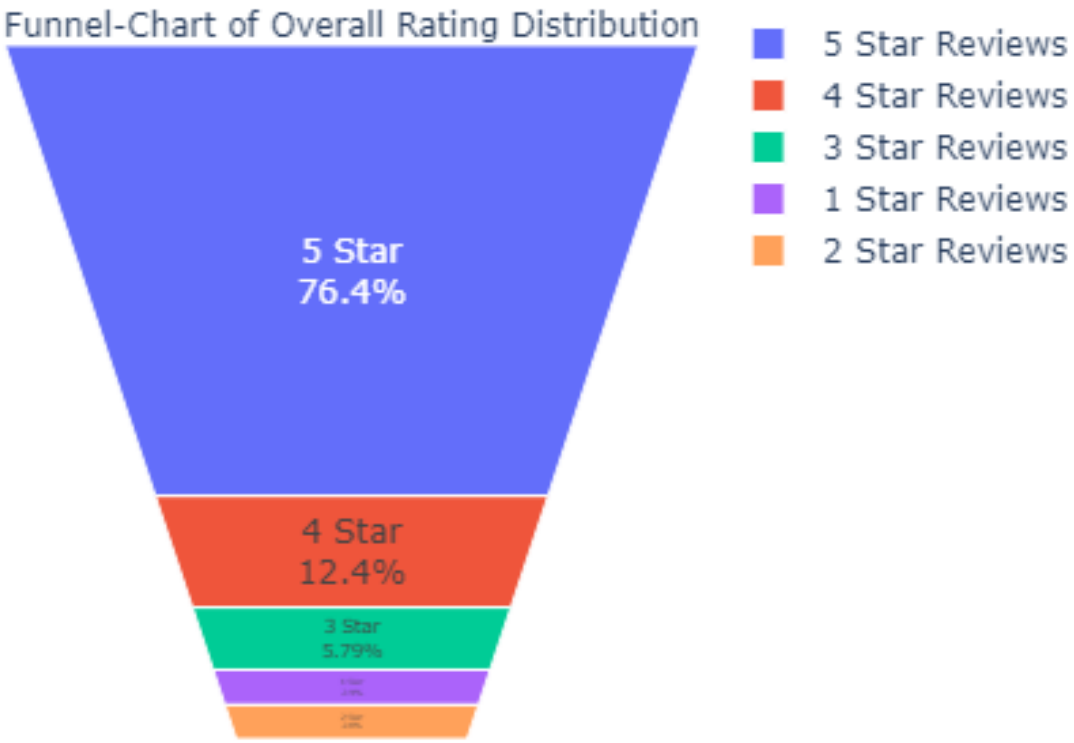

Figure 2: Funnel Chart

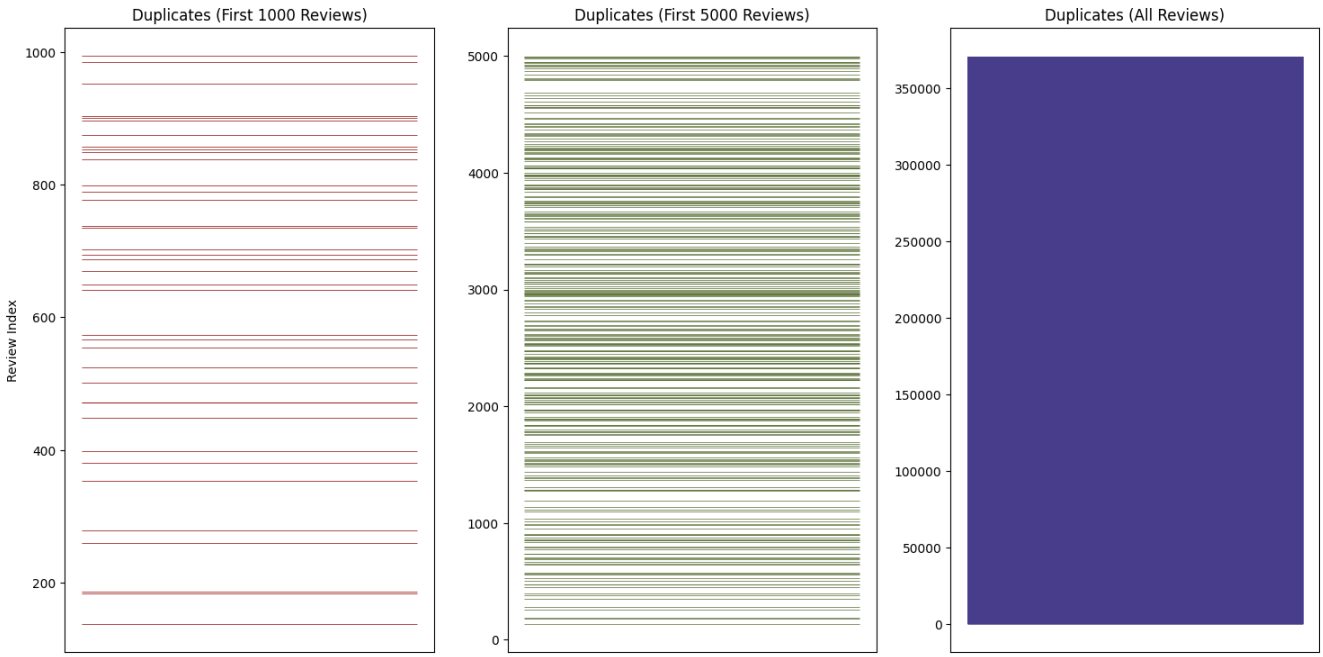

Figure 3: Duplicate Reviews

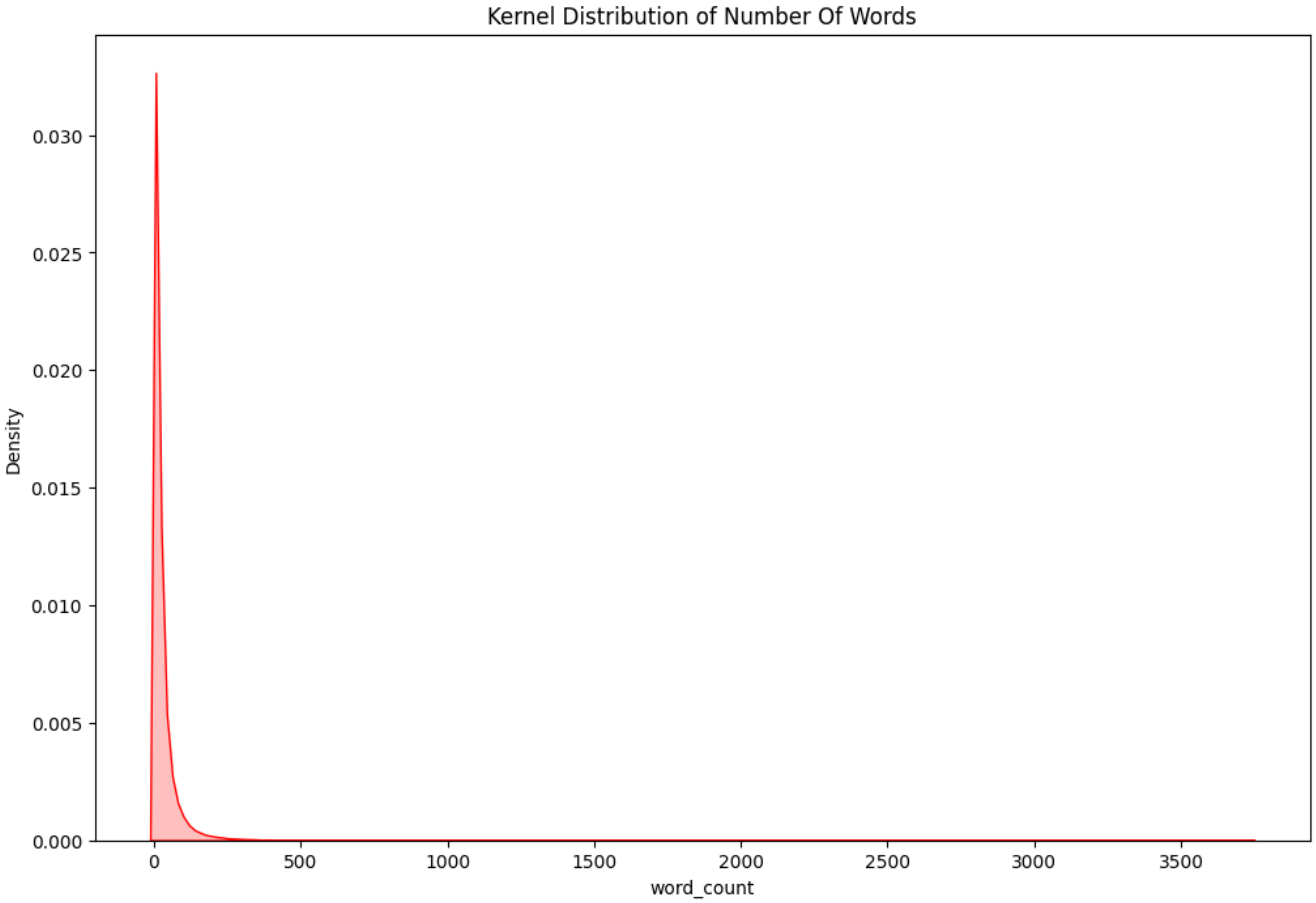

Figure 4: Kernel Distribution

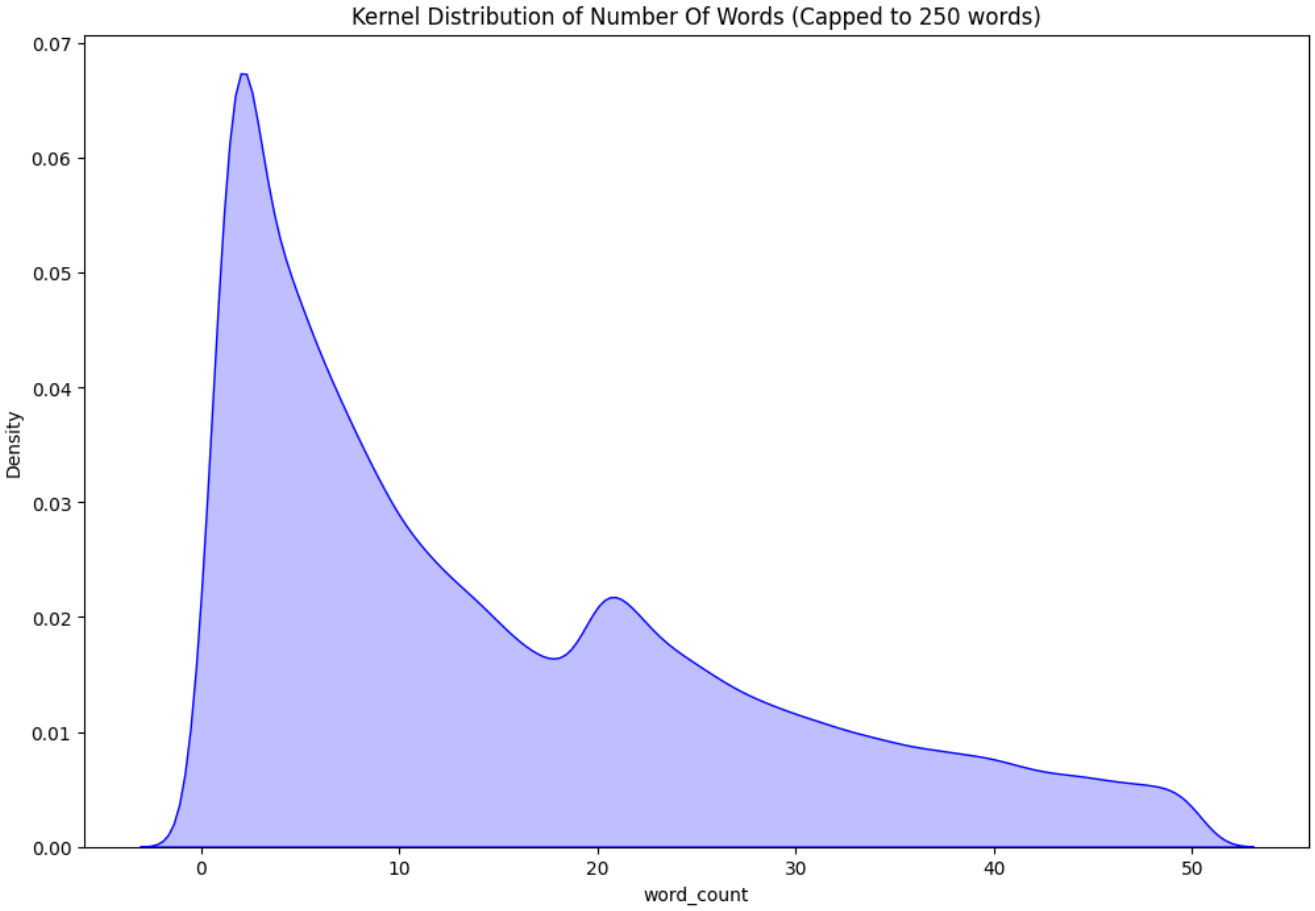

Figure 5: Kernel Distribution (Capped)

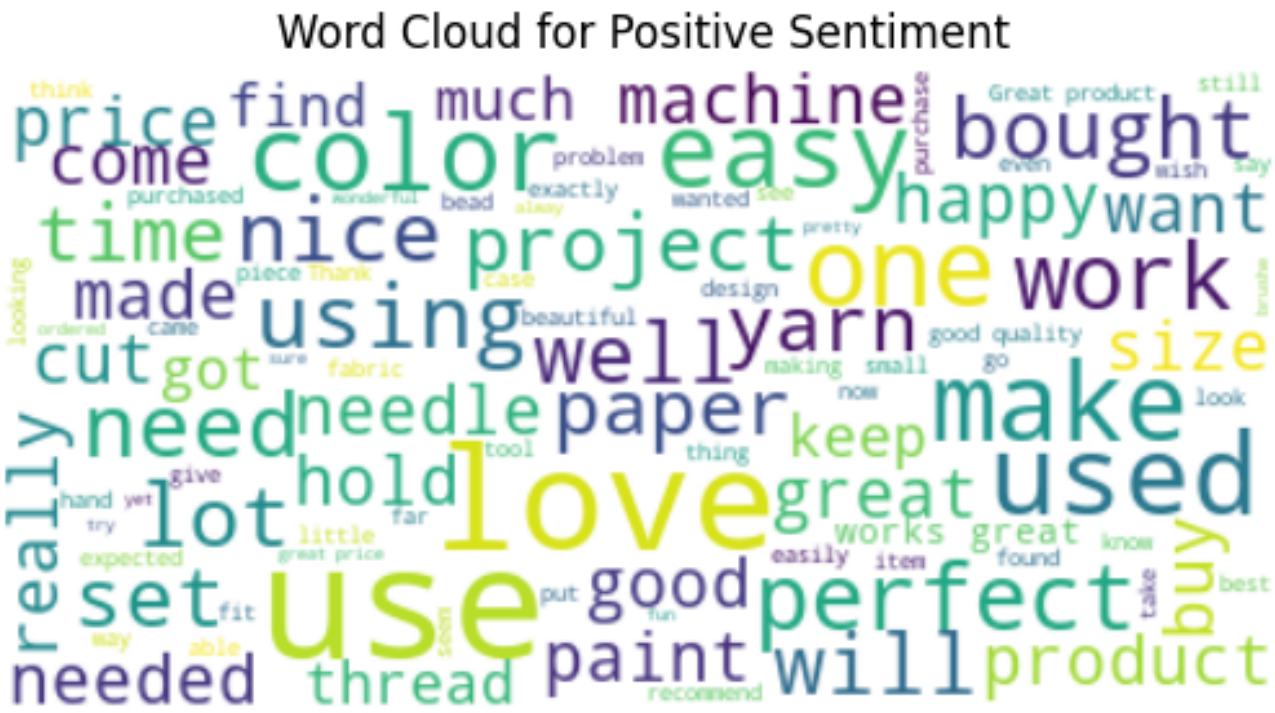

Figure 6: Word Cloud for Positive Sentiment

Figure 7: Word Cloud for Neutral Sentiment

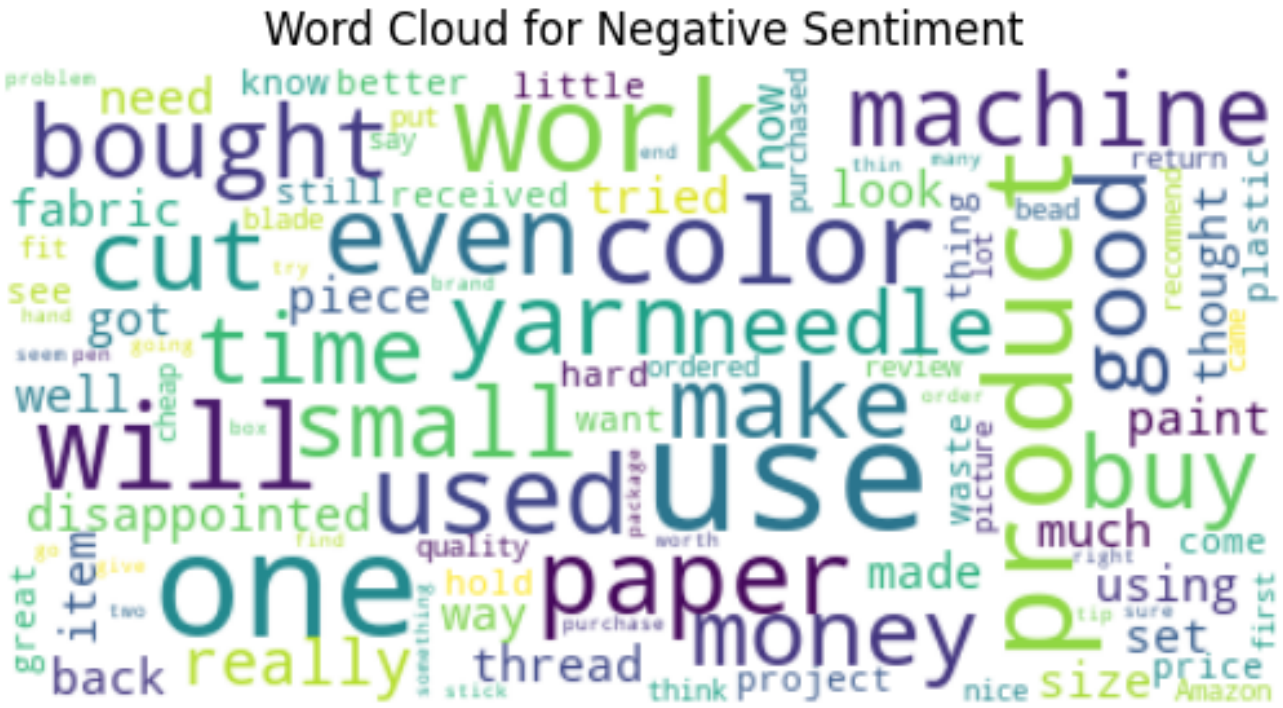

Figure 8: Word Cloud for Negative Sentiment

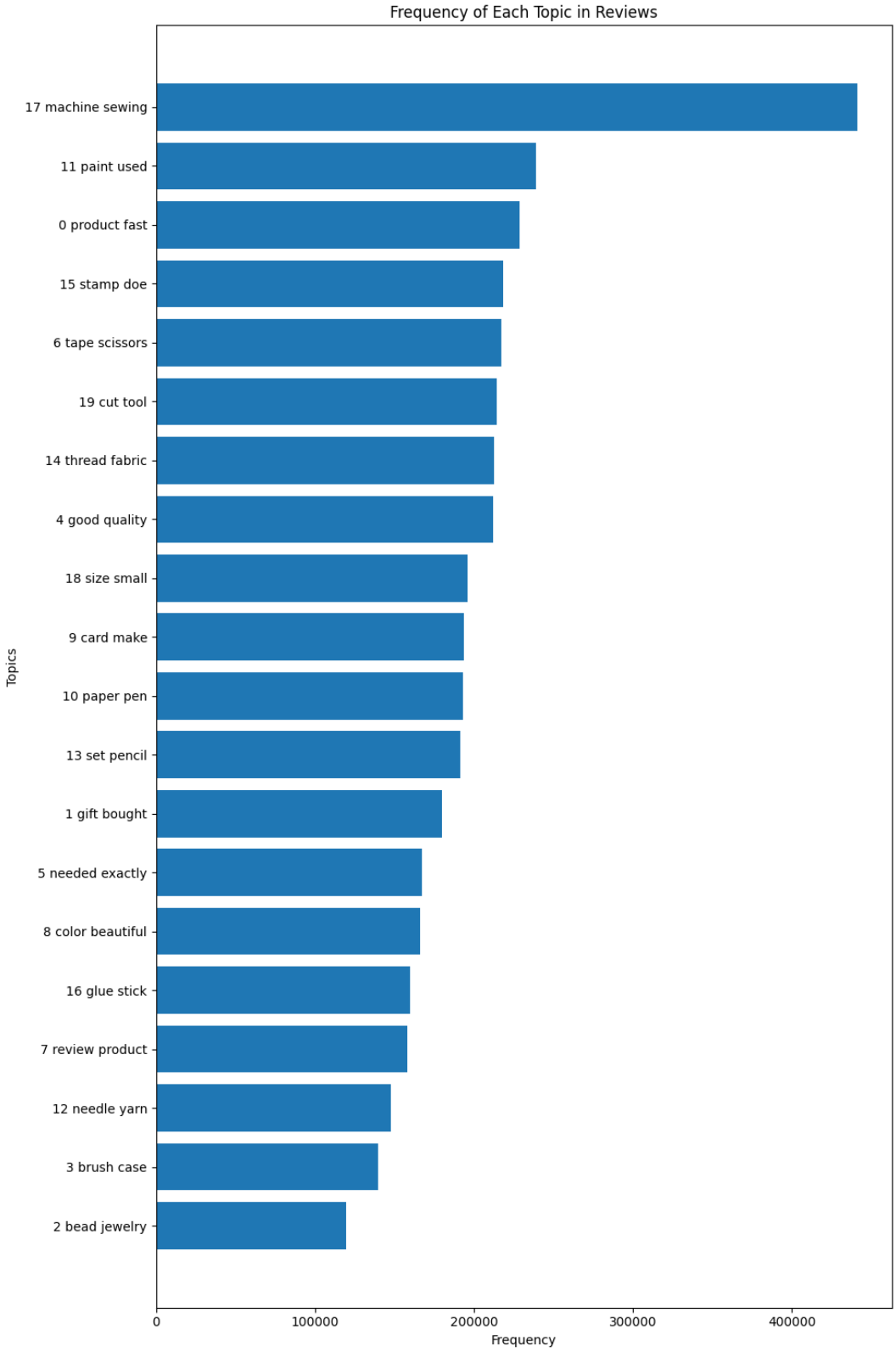

Figure 9: Topic Frequency (Complete Dataset)

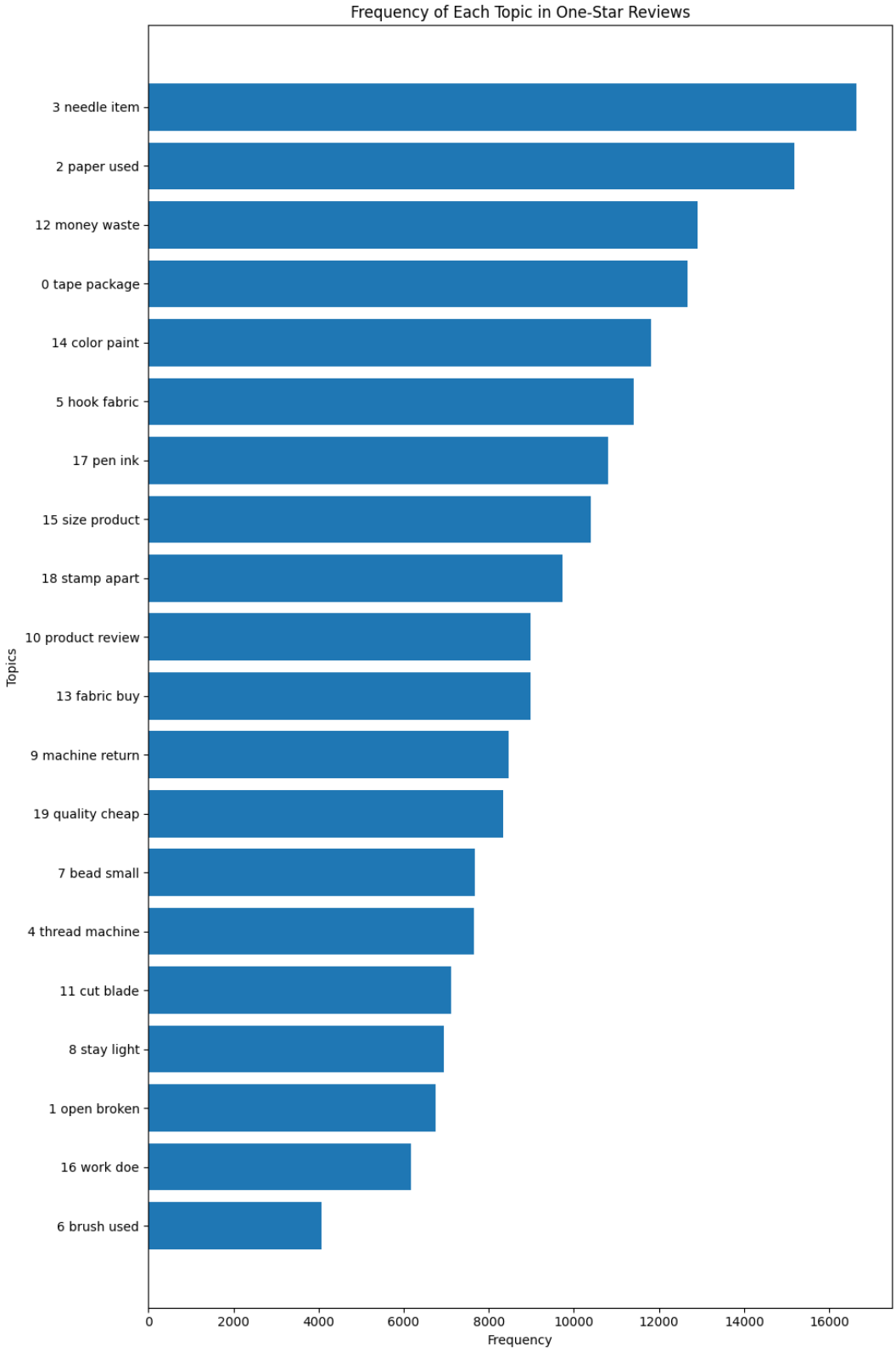

Figure 10: Topic Frequency (1-Star)

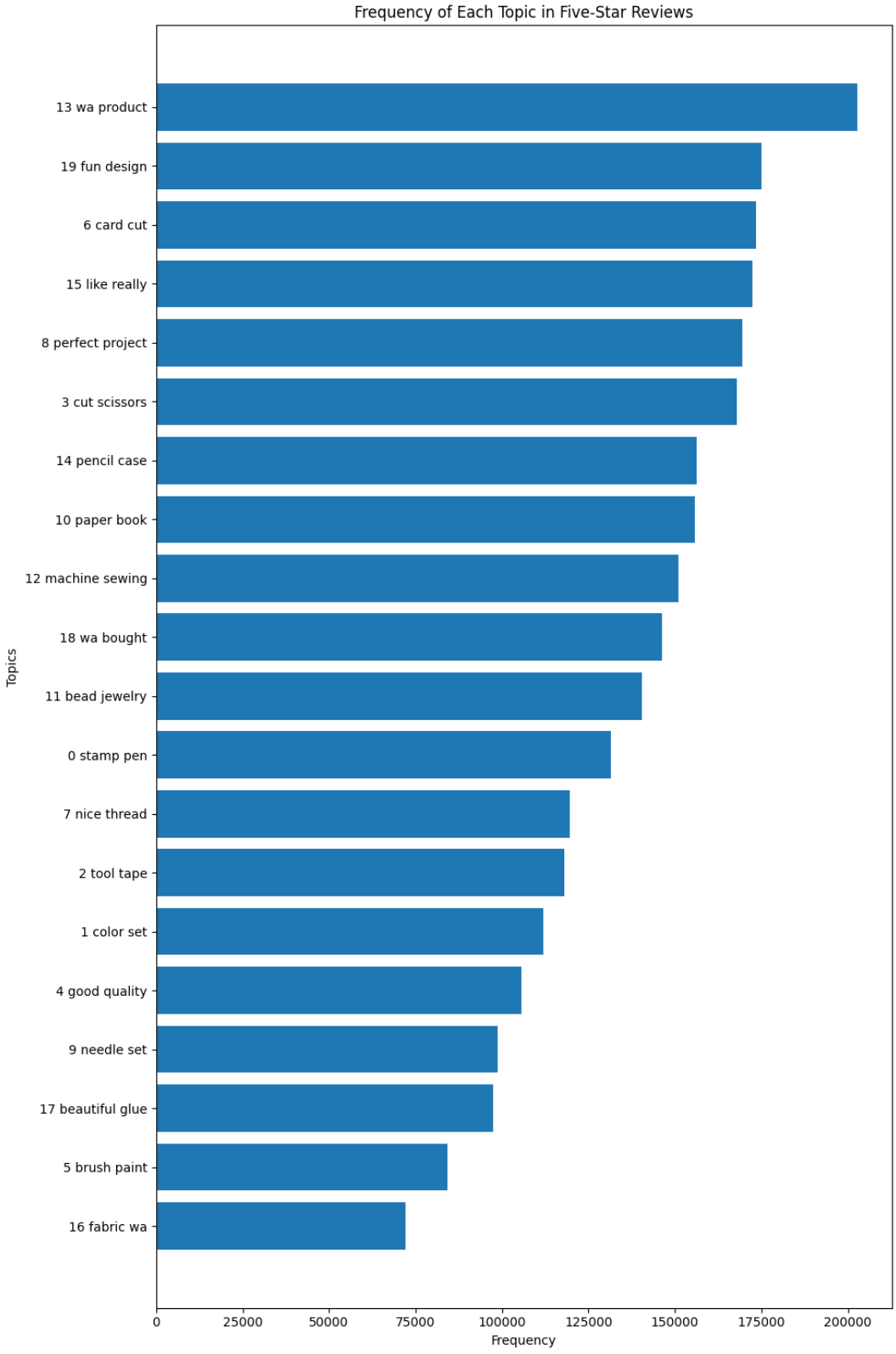

Figure 11: Topic Frequency (5-Star)

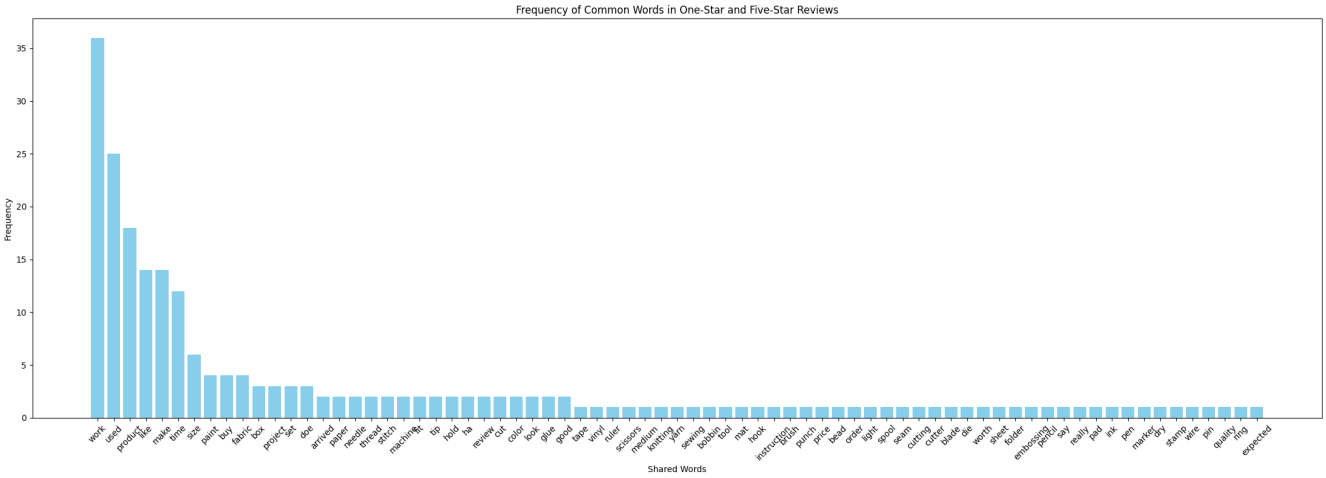

Figure 12: Common Words Frequency

9 Appendix B

9.1 Task Distribution per group member

- **Devesh:** Modelling Text as a Sequence, Topic Modelling of High and Low Ratings
- **Neil:** Data Exploration and Visualization, Model Training, report
- **Hana:** Text Processing and Normalization, Vector Space Model and Feature Representation, Model Training, Selection and Hyperparameter Tuning and Evaluation

9.2 Link to Google Colab

`https://drive.google.com/drive/folders/1yIabw-T_OhYtulY99F-hvJbIVuuy7dlg?usp=sharing`