**Semester Project Final Report**

Team: T-10

Team members: Tianran Huang, Jialu Liu, Lisha Xia,

Xiaochen Yan, Shiqin Zeng, Zhongyan Zheng

Project Title: Amazon QA Conversation Analysis

[**1. Problem Formulation and Definition 3**](#_ky6vihpbcib)

[1.1 Motivation and Problem Scope 3](#_uv45eegwz8zo)

[1.2 Objective 3](#_1wkju7ldq7b3)

[1.3 Layer of Modeling (Health and Personal Care Topic) 4](#_c7zxn8ix0t6p)

[1.4 Literature Review 5](#_vlsu2puq2sy)

[1.4.1 Sentiment Analysis 5](#_7ylrzrgcn1g8)

[1.4.2 Topic Modeling 7](#_csszjr58bqp)

[1.5 Proposed Method 9](#_vxnf9qt629xt)

[1.5.1 Data Preparation and Preprocessing 10](#_qd039l6xw0ow)

[1.5.2 Visualization for Insightful Analysis 10](#_nc91kwunezz1)

[1.5.3 Handling Imbalanced Data 10](#_2nef29p45521)

[1.5.4 Model Development and Training 10](#_4jdw1x33qafu)

[1.5.5 Model Evaluation 10](#_gp3s47oh2fgw)

[1.5.6 Advanced Model Configuration 11](#_1gnvzocxkhm0)

[1.5.7 Results Interpretation and Reporting 11](#_ppig59550fes)

[**2. Data Collection 11**](#_unlzar8jpdy0)

[2.1 Amazon question/answer data 11](#_ab1ikx593org)

[2.2 Amazon product data 12](#_de7ovsngks10)

[**3. Data Mining Strategies 13**](#_9eo7bfwp26ny)

[3.1 Exploratory Data Analysis 13](#_wdsx02g75bjw)

[3.1.1 Reviews Distribution Based on Date 14](#_crjza416diei)

[3.1.2 Word Characteristics 15](#_geq56jk687hu)

[(a) Sentiment Polarity Distribution 15](#_gxszv04weguy)

[(b) Average Sentiment Polarity per Day 15](#_p1z5h6rkpjaj)

[(c) Topic Modeling Using Review Texts 15](#_b3on5151ffk)

[3.1.3 Numerical Data Characteristics 16](#_fy5rg3zaby4n)

[3.1.4 Response Variables 16](#_ay5pg5ublu4u)

[3.2 Feature Engineering 16](#_s7pzi4kbmfw5)

[3.3 Topic Modeling 17](#_dteyyhenlpsa)

[3.4 Data Cleaning & Export 17](#_6vspw2bllel6)

[3.5 Data Partition 18](#_ubjv6oih7gib)

[3.6 Normalization 18](#_6kvpt4i4ms8n)

[3.7 Oversampling for Handling Imbalanced 18](#_skazqwjbjdln)

[3.8 Grid Search with Cross-Validation 20](#_5km3xyrv1nas)

[**4. Machine Learning Tools and Model Selection 21**](#_kncdt5fssgx5)

[4.1 Sentiment Analysis 21](#_e16s9aegq9eu)

[4.2 Topic Modeling 22](#_l07xd46ukhif)

[4.3 Classification Models 24](#_vtwjduzcgkt2)

[4.3.1 SVC 24](#_edyqoajuswbv)

[4.3.2 Adaboosting 25](#_24nk75zhrg3p)

[4.3.3 Gradient Boosting 25](#_e7bidxj75z36)

[4.3.4 Random Forest 26](#_e3cnj98hyb96)

[4.4 Model Evaluation 26](#_jq140d1fncpq)

[4.4.1 Confusion Matrix 26](#_bvt6wjpjjvup)

[4.4.2 Accuracy 27](#_qvrbjo1rm46a)

[4.4.3 Precision 27](#_urfksoj3iqof)

[4.4.4 Recall (Sensitivity) 28](#_x4dv36t52vs2)

[4.4.5 F1 Score 28](#_mmwyvbdctbra)

[**5. Results and Property Assessment 28**](#_vzb5vjgr2h57)

[5.1 SVC 28](#_y63l89nfaghf)

[5.2 Adaboosting 30](#_p0xy6rk7usvs)

[5.3 Gradient Boosting 31](#_f5l9iwlbsxhf)

[5.4 Random Forest 32](#_2ugk4cmvo0ic)

[5.5 Comparison 35](#_lk81xzgdqy24)

[5.6 Overall Result 35](#_frpyh47e0uds)

[**6. Conclusion 35**](#_x9hhipoc4h00)

[6.1 Conclusion 35](#_lc81r5haxkvl)

[6.2 Enterprise Discussion 36](#_95zplpj2tzxt)

[6.3 Future Plan 36](#_48xpe5s4p0t)

[**Reference 37**](#_qsgohbmj1uxb)

[**Appendix 41**](#_ink19sutr71k)

# 1. Problem Formulation and Definition

## 1.1 Motivation and Problem Scope

Analyzing questions and answers (QA) on Amazon involves understanding customer queries and the corresponding responses from other customers or sellers. Amazon allows customers to ask questions about products, and other users or sellers can respond. This fosters engagement and helps potential buyers make informed decisions. Customers rely on QA sections to gather additional information about products that may not be available in the product description. Customers may use the QA section to report problems or seek clarification about certain aspects of a product. Sellers can directly interact with customers through the QA section, addressing concerns, providing additional information, or clarifying doubts. Revealing hidden patterns and insights within these chats can translate to better customer engagement strategies, improve business product feedback loops, and ensure accurate information. For academic purposes, chat analysis can illuminate intricate social dynamics, varied communication styles, and evolving linguistic trends, providing valuable insights for research and exploration. Our project focuses on one specific category of products on Amazon, which is “*Health and Personal Care Topic*”. The reason we choose this specific domain in Amazon is that Health and personal care products are often in high demand as they cater to essential needs as individuals are generally inclined to invest in items that enhance their overall well-being. Therefore, the dataset of the Health and Personal Care category covers a massive amount of product information and reviews, and the diversity and size of the dataset provide a rational scope for conducting the research project. Additionally, the model's pipeline is scalable for various products, given that other product data on Amazon adhere to the same format and can follow similar state-of-the-art techniques of the proposed models.

## 1.2 **Objective**

The project involves two main stages: data mining of textual product information and using classification models to predict feedback scores. In the data mining phase, we conduct sentiment analysis and topic modeling on customer conversations to gain insights into emotions and discussion themes within product-related QA sections. This helps identify recurring topics and understand the nature of customer inquiries and their resolutions.

In the classification phase, we convert conversations into quantitative feature vectors to train models such as Random Forest, SVC, and AdaBoost. These models are tasked with predicting overall feedback scores, which help identify the most useful responses from historical data and evaluate the quality of current responses. This streamlined approach enhances our ability to understand and predict customer feedback efficiently.

## 1.3 **Layer of Modeling (Health and Personal Care Topic)**

1. QA Data & Reviewer Data
   1. Question Type
      1. Yes/No (Like “I wonder if/whether…” or “Does the item…?”)
      2. Open-ended (What/Which/How/Why)
   2. Answer Type
      1. Yes - Positive
      2. No - Negative
      3. The polarity of the answer could not be predicted
   3. Review of Customers
      1. Overall: Evaluation by Customers, a number ranged from 1.0 to 5.0
      2. Positive (great/Quickly/love/five stars…)
      3. Negative (limited/one star/fake…)
      4. Helpfulness rating of the view ranged from 0 to 1
   4. Content
      1. Subject – Topic Modeling
      2. Keywords Extraction
   5. Analysis
      1. Sentimental Analysis
      2. Word Frequency Analysis
      3. Topic Analysis
      4. Uncertainty Analysis
   6. Applications
      1. Sentiments of Customer Reviews
      2. Reputation Management
      3. Experience Analysis
2. Product Information
   1. ID (asin)
   2. Name
   3. Price (in US dollars)
   4. Category
   5. Brand name
   6. Image of the product
   7. Related products (also bought, reviewed, bought together, buy after viewing)
   8. Sales rank

## 1.4 **Literature Review**

### 1.4.1 **Sentiment Analysis**

Sentiment analysis, also known as opinion mining, is a Natural Language Processing technique that identifies and classifies opinions in text data to determine the sentiment toward specific subjects, being positive, negative, or neutral. Sentiment analysis in the field of business intelligence is useful for not only product producers but also consumers to make more informed purchase decisions (Xia et al., 2019). Sentiment analysis helps businesses in numerous ways. To illustrate, companies can leverage sentiment analysis outcomes to refine products, analyze customer opinions, or adjust marketing strategies. By studying Amazon service food reviews, Bose et al. have shown that sentiment analysis is capable of distinguishing customer behaviors and rising customer satisfaction.

With e-commerce businesses rapidly blooming, the corresponding need for product and review analysis analysis has also been steadily growing. Sentiment analysis – in particular, phrase level or aspect level (Schouten and Frasincar 2015) – is especially popular for product reviews. Sentiment analysis can be conducted on several levels: Document Level, Sentence Level, Phrase Level, and Aspect Level (Figure 1). Document-level sentiment analysis is conducted on a whole document, thus irrelevant to our topic. The sentence-level analysis assesses the sentiment polarity of individual sentences. It is useful for documents with varied sentiments and can aggregate these to understand overall sentiment or address challenges like ambiguous statements. The phrase-level sentiment analysis focuses on extracting sentiments from phrases. It considers the impact of word choice on sentiment, influenced by demographic and psychological factors. The aspect level targets specific aspects within sentences to assign polarity, allowing for more detailed sentiment analysis by examining and aggregating the sentiment of each aspect mentioned. Our project will use a combination of sentence, phrase, and aspect levels of sentiment analysis to investigate the objective.

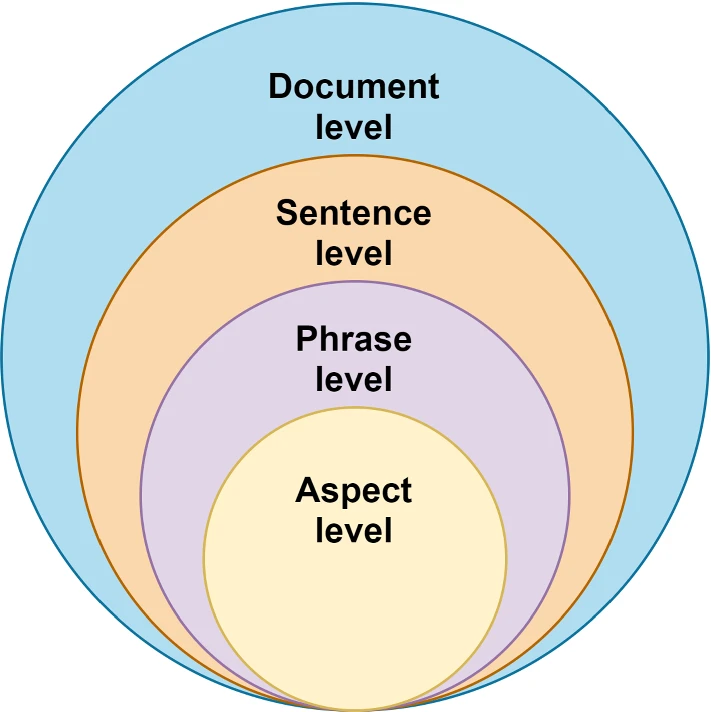


Figure 1: Level of sentiment analysis (Wankhade et al. 2022)

In addition, sentiment analysis is useful for user-oriented question-answering (QA) text pair data. However, QA style data is difficult to directly apply traditional sentiment analysis due to its sequential nature and complex intra-relations among sentences. Facing those challenges, Shen et al. (2018) innovated a hierarchical matching network model that builds upon traditional sentiment classification methods. This model is specifically designed for QA-style data analysis and has demonstrated remarkable efficiency, outperforming other baseline approaches.

### 1.4.2 **Topic Modeling**

Topic Modeling (TM) is a natural language processing (NLP) technique that aims to automatically identify topics present in a collection of texts. The goal is to reveal hidden thematic structures in documents without explicit labeling or supervision.

The article *Using Topic Modeling Methods for Short-Text Data: A Comparative Analysis* focuses on applying machine learning, natural language processing, and topic modeling techniques to short-text data. It studies the challenge of extracting useful information from the large volume of user-generated textual content on social networks. The study also evaluates five common topic modeling methods and finds that Latent Dirichlet Allocation and Non-negative Matrix Factorization perform well in extracting meaningful topics from online social data.

TM techniques, including probabilistic latent semantic analysis (PLSA), latent semantic analysis (LSA), and latent Dirichlet allocation (LDA), can automatically extract topics from both short texts (Cheng et al., 2014) and standard long-text data (Xie and Xing, 2013). However, challenges arise in TM for short texts, particularly in online social network (OSN) platforms, due to issues like slang, data sparsity, errors, and unstructured data (Albalawi et al., 2020). Researchers have solved these challenges using techniques, such as word embedding models and specific short-text TM methods like the biterm topic model (BTM) (Yan et al., 2013). The fundamental steps of text mining areas shown in Figure 2.

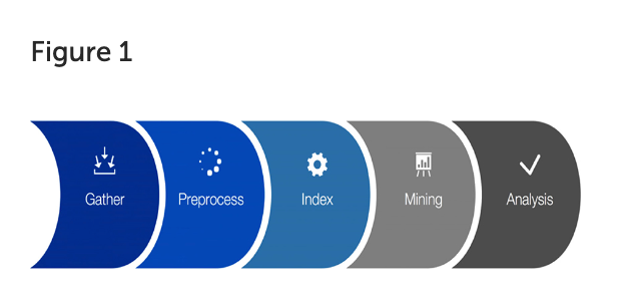


Figure 2: The steps involved in a text mining process (Kaur and Singh, 2019).

The following five TM methods are discussed in the article.

Latent Semantic Analysis (LSA):

LSA, proposed by Deerwester et al. (1990), is a distributional semantics method using Singular Value Decomposition (SVD) to represent raw text in a vector space. It is useful for advanced topic modeling, identifying latent topics by decomposing the term-document matrix. The Figure 3 below shows SVD of the LSA TM method.

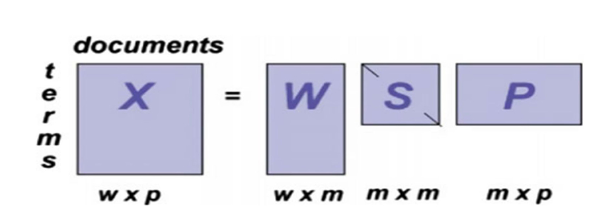


Figure 3: SVD of the LSA topic modeling method (Neogi et al., 2020).

Latent Dirichlet Allocation (LDA):

Introduced by Blei et al. (2003), LDA is a popular probabilistic model in TM. It organizes a corpus as a mixture of latent topics, extracting accurate topics from document collections and assumes each document consists of various topics (Albalawi et al., 2020). Also, as Albalawi et al. (2020) summarized one of the most significant advantages of LDA is that topics can be inferred without input data. The schematic diagram of the model is shown below (Figure 4).

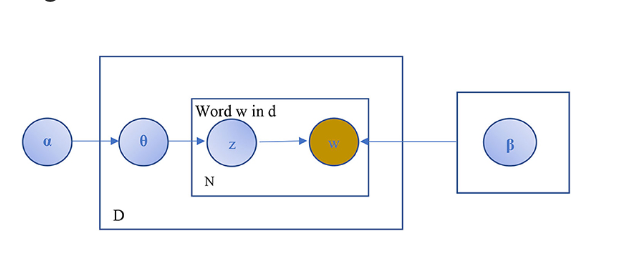


Figure 4: A schematic diagram of the LDA (Albalawi et al., 2020)

Non-negative matrix factorization(NMF):

NMF, an unsupervised matrix factorization method, performs dimension reduction and clustering simultaneously (Berry and Browne, 2005; Kim et al., 2014). Although it is widely used in TM, it is not commonly used in short-text analysis (Albalawi et al., 2020). Yan et al. (2013) and Chen et al. (2019) proposed NMF models for short-text topics through different methods. NMF works well in many areas, including image processing and text analysis, even videos (Albalawi et al., 2020).

Principal Component Analysis( PCA)

PCA is a useful tool in text processing since the early 1990s (Jolliffe, 1986; Slonim and Tishby, 2000; Gomez et al., 2012). It reduces high-dimensional feature vectors while retaining essential information, but it is computationally expensive for large text datasets (Albalawi et al., 2020).

Random Projection (RP)

RP is used in machine learning tasks like classification and clustering (Wang and McCallum, 2006; Ramage et al., 2011). It uses a random matrix to reduce dimensionality (Dasgupta, 2000). One of the significant advantages of RP is its accuracy is not related with data, but it may yield sparse results and high distortion (Albalawi et al., 2020).

Besides, Albalawi et al. (2020) also shared their data preprocessing experiments, which are focused on English text data collected from different sources. Preprocessing steps, including stemming, lemmatizing, and tokenization, were performed using Python and Natural Language Toolkit (NLTK) (Bird et al., 2009). Then the TF-IDF term-weighting method was used to score word importance based on frequency and relevance in the corpus (Albalawi et al., 2020).

## 1.5 Proposed Method

In our project, we aim to analyze customer feedback across various product categories using a combination of classification models. Here's how we propose to implement and evaluate our methods:

### 1.5.1 Data Preparation and Preprocessing

We begin by importing essential Python libraries and loading the Amazon review dataset into a Pandas DataFrame. Our initial data preparation includes removing unnecessary identifiers and separating the target variable 'overall' which indicates the product ratings. We proceed with data normalization using the `MinMaxScaler` to standardize the features, ensuring that our model inputs have consistent scales.

### 1.5.2 Visualization for Insightful Analysis

To understand the distribution of ratings within the dataset, we employ matplotlib and seaborn to create visual representations. Pie charts are used to visualize the proportion of each rating category, providing a clear view of the dataset's imbalance. We also plot the frequency distribution of ratings to further analyze the data characteristics.

### 1.5.3 Handling Imbalanced Data

Given the skewed distribution of ratings, we utilize the SMOTETomek technique to balance our dataset. This method combines the Synthetic Minority Over-sampling Technique (SMOTE) with Tomek links to both augment the minority classes and clean overlapping samples between classes. The balanced data is visualized to confirm the effectiveness of this approach.

### 1.5.4 Model Development and Training

We employ a variety of classifiers including Random Forest, SVC (Support Vector Classifier), Decision Tree, and ensemble methods such as AdaBoost and Gradient Boosting. Each model is trained on the balanced dataset, and we employ hierarchical classification techniques to differentiate between multiple rating levels based on predefined thresholds.

### 1.5.5 Model Evaluation

For each model, we use cross-validation to assess performance robustly and prevent overfitting. Accuracy, precision, recall, and F1-score are calculated to measure the effectiveness of each classifier. We also perform grid searches to fine-tune model parameters, aiming to optimize their performance.

### 1.5.6 Advanced Model Configuration

We implement GridSearchCV for hyperparameter tuning, particularly focusing on the number of trees in Random Forest and depth of trees in Gradient Boosting models. This step ensures that each model operates at its optimal parameters.

### 1.5.7 Results Interpretation and Reporting

The performance of each model is analyzed through confusion matrices, providing detailed insights into the classification accuracy across different classes. We generate heatmaps from these confusion matrices to visualize the prediction accuracy and misclassifications, aiding in intuitive understanding and presentation of the results.

# 2. Data Collection

## 2.1 Amazon question/answer data

The dataset is available at this link: [Amazon question/answer data](https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/qa/). This dataset's description originates from a collection assembled by McAuley and colleagues, who gathered the data through web scraping of questions and answers on Amazon. It contains about 1.4 million entries and features a wide variety of Question and Answer pairs from Amazon, spanning 21 product categories including automotive, electronics, home and kitchen, among others. We have selected the "Health and Personal Care" subcategory as our research data. The variables relevant to our analysis are as follows:

* ‘asin’ refers to the product ID.
* ‘questionType’ indicates the type of question.
* ‘answerType’ signifies the type of answer, with possible values being ‘Y’ for yes, ‘N’ for no, or ‘?’ for when the answer's polarity is unpredictable.
* ‘answerTime’ represents the raw answer timestamp.
* ‘unixTime’ denotes the answer timestamp converted to Unix time.
* ‘question’ is the question text.
* ‘answer’ is the answer text.

## 2.2 Amazon product data

The dataset is accessible at this link: [Amazon review data](https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/links.html). This dataset comprises reviews (including ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (graphs for products also viewed and also bought) from Amazon. It encompasses 142.8 million reviews collected from May 1996 to July 2014.

The variables from the reviews dataset are as follows:

* ‘reviewerID’ represents the ID of the reviewer.
* ‘asin’ indicates the ID of the product.
* ‘reviewerName’ denotes the name of the reviewer.
* ‘helpful’ refers to the helpfulness rating of the review.
* ‘reviewText’ describes the text of the review.
* ‘overall’ signifies the rating of the product.
* ‘summary’ stands for the summary of the review.
* ‘unixReviewTime’ represents the time of the review in Unix time.
* ‘reviewTime’ indicates the raw time of the review.

The variables from the product metadata are as follows:

* ‘asin’ represents the ID of the product.
* ‘title’ represents the name of the product.
* ‘price’ represents the price in US dollars at the time of the crawl.
* ‘imUrl’ represents the URL of the product image.
* ‘related’ represents related products, including ‘also bought’, ‘also viewed’, ‘bought together’, and ‘buy after viewing’.
* ‘salesRank’ represents sales rank information.
* ‘brand’ represents the brand name.
* ‘categories’ represents the list of categories the product belongs to.

We will merge Amazon question/answer data and Amazon product data for our research by matching the "ASINs" field.

# Data Mining Strategies

## 3.1 **Exploratory Data Analysis**

One product was explored to explain the data characteristics as shown in Figure 5:

# 

Figure 5: Product instance in Amazon

## 

Using an electronic scale product as an example, one entry from the raw data is presented as follows:

{

"reviewerID": "AND3GQSC4A06R",

"asin": "B001KXZ808",

"reviewerName": "NaN",

"helpful": [0, 0],

"reviewText": "We have had this scale for almost a year and a half; it is working really well overall. It is considerably more consistent and our favorite out of all of the scales that we have owned. Sometimes it does like to fluctuate the poundage on you and you have to weigh a few times to get a consistent reading. (Reason for the 4 not 5 stars.)My wife also says that one of her main downsides is that she feels like it is a little delicate with it being glass with our kids. But in saying that, with our kids, this scale gets rough-housed and it is holding up very well.",

"overall": 4.0,

"summary": "Overall Great Scale",

"unixReviewTime": 1405987200,

"reviewTime": "2014-07-22"

}

### 3.1.1 Reviews Distribution Based on Date

We sorted and selected the most recent 1,000 entries from the dataset. The data distribution based on date is described in Figure 6, and sentiment analysis is shown in Figure 7.

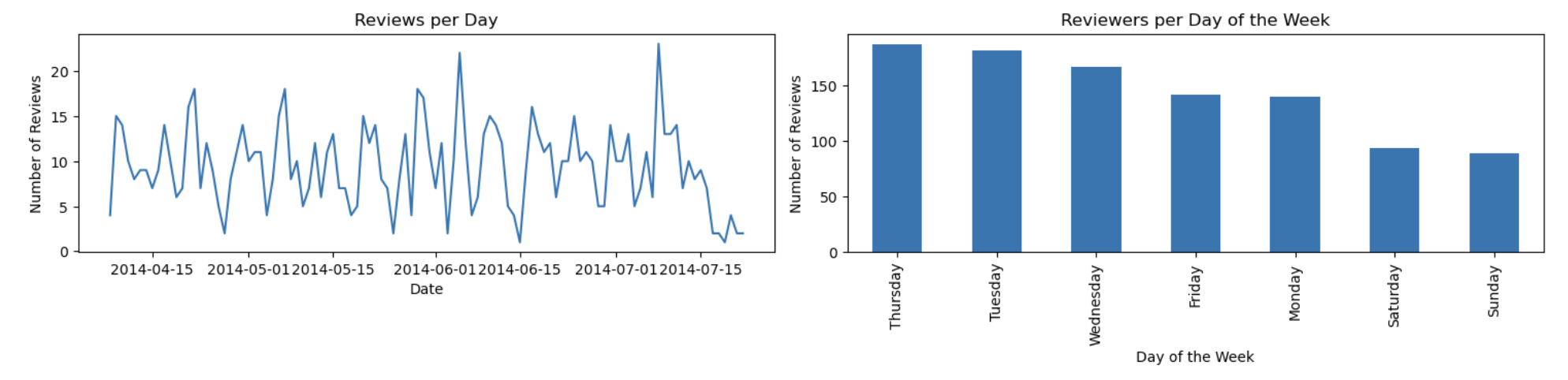


Figure 6: Data distribution based on date

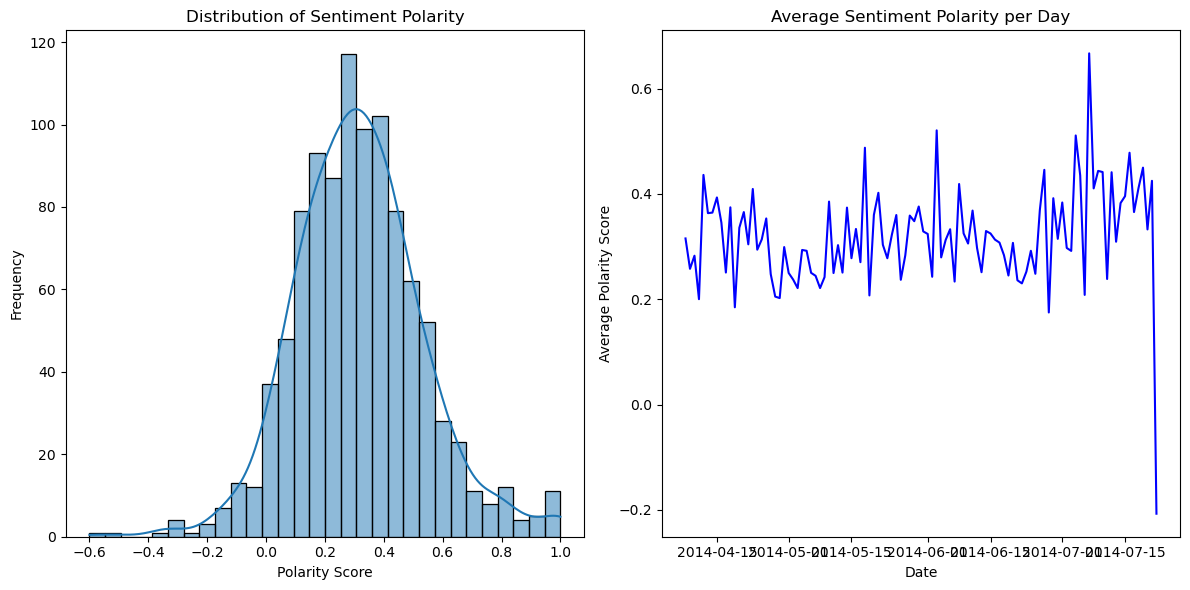


Figure 7: Sentiment analysis distribution

### 3.1.2 Word Characteristics

#### (a) Sentiment Polarity Distribution

The histogram illustrates the distribution of sentiment polarity scores found in all the reviews. These scores vary from -1, representing a highly negative sentiment, to +1, denoting a highly positive sentiment. Scores close to 0 signify a neutral sentiment. Analyzing the shape of this distribution helps in understanding the general sentiment towards the product, whether it leans more toward positive or negative.

#### (b) Average Sentiment Polarity per Day

This line graph represents the daily average sentiment polarity. Variations in the graph reflect daily shifts in the product's overall reviews. High points signal days marked by predominantly positive discussions, whereas low points reveal days with more negative sentiments. Such visualizations are crucial for grasping the product's review trends over time. They become especially revealing when linked to particular issues or subjects addressed in the QA section, providing deep insights into the dynamics of product feedback.

#### (c) Topic Modeling Using Review Texts

For each review, the topic model provides a distribution over topics (Figure 4), indicating the presence and proportion of each topic within the review. Transform these topic distributions into a vector of topic weights for each review, and each element in this vector represents the weight or contribution of a topic to the review, effectively summarizing its thematic content. Finally, we can extract the reviews-topic distribution as a feature set.

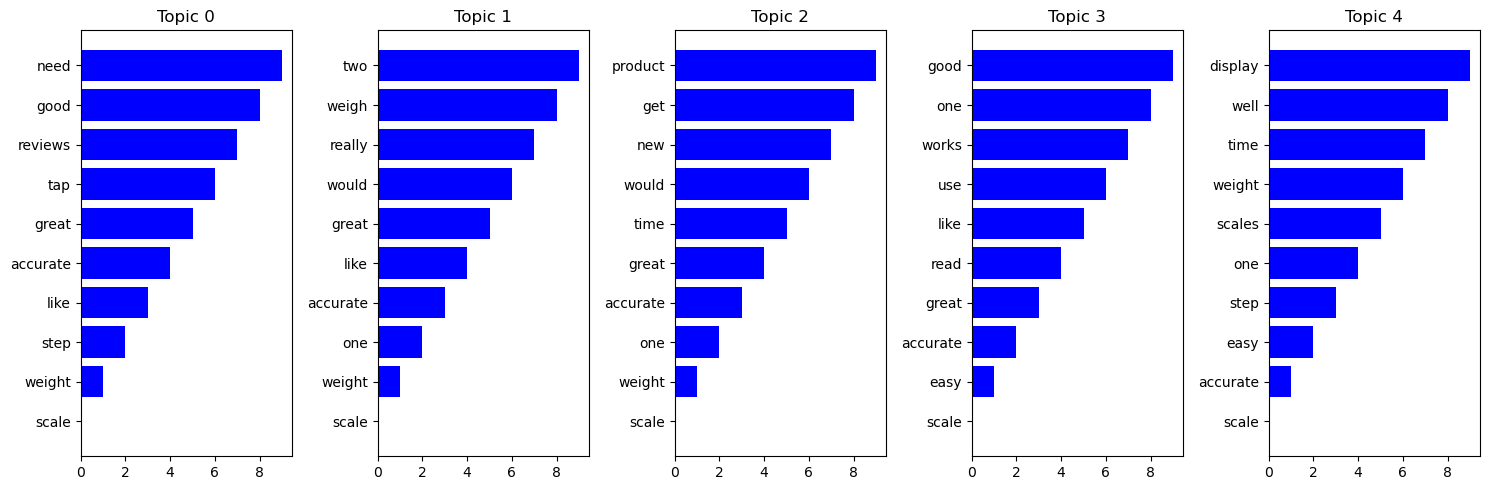


Figure 8: Distribution of topics based on topic modeling

## 

### 3.1.3 Numerical Data Characteristics

Helpful: [a,b] transfer the list to the total number of the responses (b) and a ratio to present the helpful ratio (a/b) respectively.

Product price: a numerical number of the price.

Review date: transfer the date to the year, month, day, and week of the day.

### 3.1.4 Response Variables

Our response variable is the overall score: [1.0, 2.0, 3.0, 4.0, 5.0].

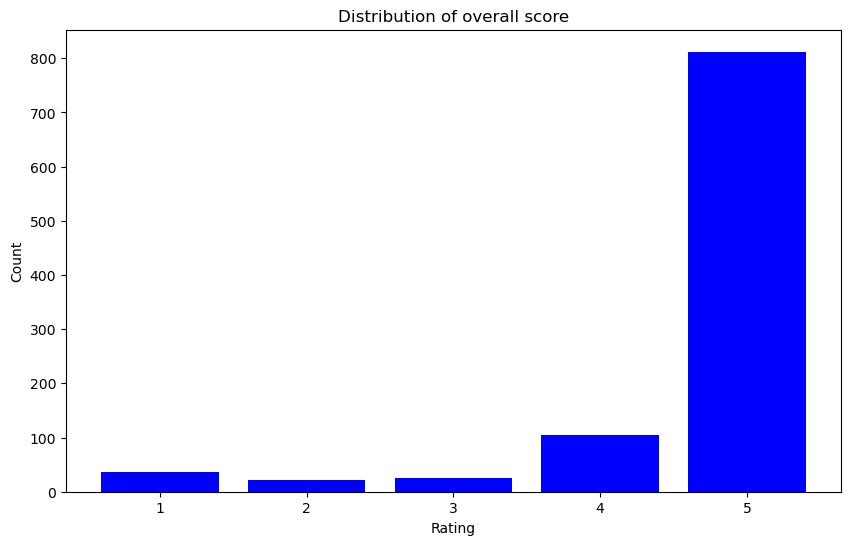


Figure 9: Distribution of overall score

The result (Figure 9) shows there is a significant disparity in overall rating distribution. The most defining characteristic is the large disparity in the number of instances among different rating scores. Over 90% of the data belongs to the rating 5 class. Several resampling methods will be considered to the original dataset, or use ensemble learning techniques that combine multiple models to improve the prediction of minority class examples.

## 3.2 Feature Engineering

1. **Datetime Conversion**: Convert relevant fields to datetime format to facilitate time series analysis, enhancing the precision of trend assessments and forecasts.
2. **Helpfulness Metric**: Transform the "Helpful" attribute from text to a numeric format. This quantification will allow for a systematic evaluation of the quality and relevance of each response within the dataset.
3. **Sentiment Analysis:** Implement sentiment analysis to assign a polarity score to each comment. This process categorizes the comments into various sentiment classes based on their positivity or negativity, thus providing deeper insights into customer perceptions and feedback trends.

## 3.3 Topic Modeling

1. **Text Preprocessing:** Clean and normalize text data to prepare for effective modeling. This step involves removing noise and standardizing text to ensure the data is uniform and analytically viable.
2. **Dictionary and Corpus Creation:** Construct the necessary data structures for topic modeling. This includes creating a dictionary of terms and a corpus that represents the documents in a structured format suitable for analysis.
3. **LDA Model Application:** Employ the Latent Dirichlet Allocation (LDA) model to extract significant topics from the text data. This method helps in identifying the underlying themes or subjects discussed across the documents, enabling strategic content analysis and decision-making.
4. **Mapping High-Frequency Words to Numerical Codes：**For each product, we identify the top five most frequent words. We then assign a unique number to each word across the entire vocabulary, allowing us to represent each product by a set of five numerical codes. These codes serve as one of the exploratory variables in our analysis.

## 3.4 Data Cleaning & Export

1. **Column Selection:** Identify and select subsets of relevant columns from the complete dataset that are critical for the analysis.
2. **Data Extraction:** We mainly consider the data related to five products: “B0000U1OCI”, “B00B5H5BGA”, “B0095PZHPE”, “B0032TNPOE”, “B001KXZ808” and their features. Save the refined dataset to a new file for subsequent analyses or sharing. This step ensures that cleaned and processed data is easily accessible for future use without reprocessing.

## 3.5 Data Partition

**Training and Testing Split:** Utilize the train\_test\_split function to randomly divide the dataset into training and testing subsets with the ratio of 80:20. This separation is crucial for validating the predictive power of models and for minimizing overfitting.

## 3.6 Normalization

**MinMaxScaler Implementation:** we apply MinMaxScaler to ensure all numerical features have consistent scales across the dataset. Normalization is essential for optimizing model performance as it levels the playing field for all features, ensuring no single attribute unduly influences the model’s outcomes.

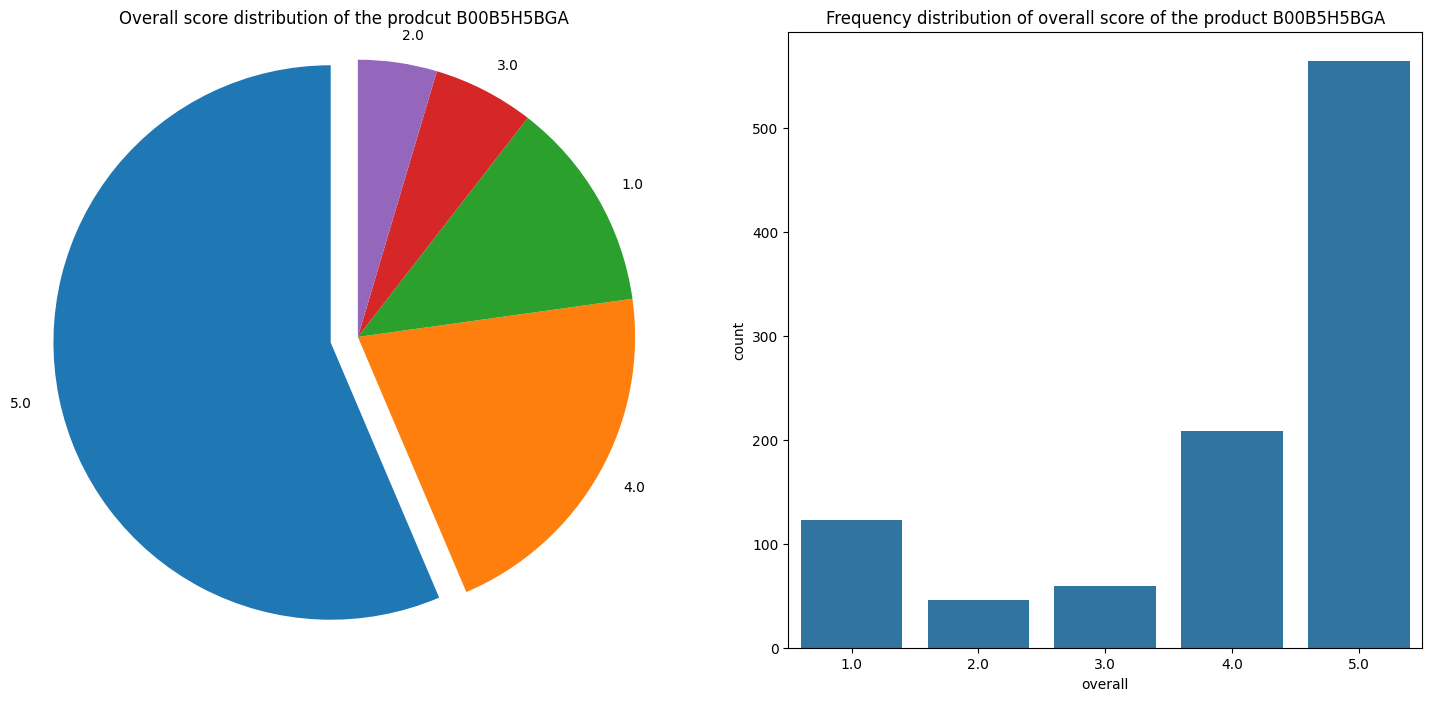
## 3.7 Oversampling for Handling Imbalanced

We use SMOTETomek Algorithm to handle imbalanced data:

1. **Synthetic Sample Generation:** Generate synthetic instances for under-represented classes to balance the dataset, enhancing model fairness and accuracy.
2. **Overlap Removal:** Remove overlapping samples between synthetic and majority class samples. This step is aimed at refining the quality of the training data by ensuring clear class boundaries, which helps in reducing classification errors.

Taking Product B00B5H5BGA as an example shown in Figure 10, the following graphics demonstrate the comparison before and after applying the SMOTETomek technique. It's evident that after employing the SMOTETomek algorithm, our data has become significantly more balanced.

Distribution before applying SMOTETomek:



Distribution after applying SMOTETomek:

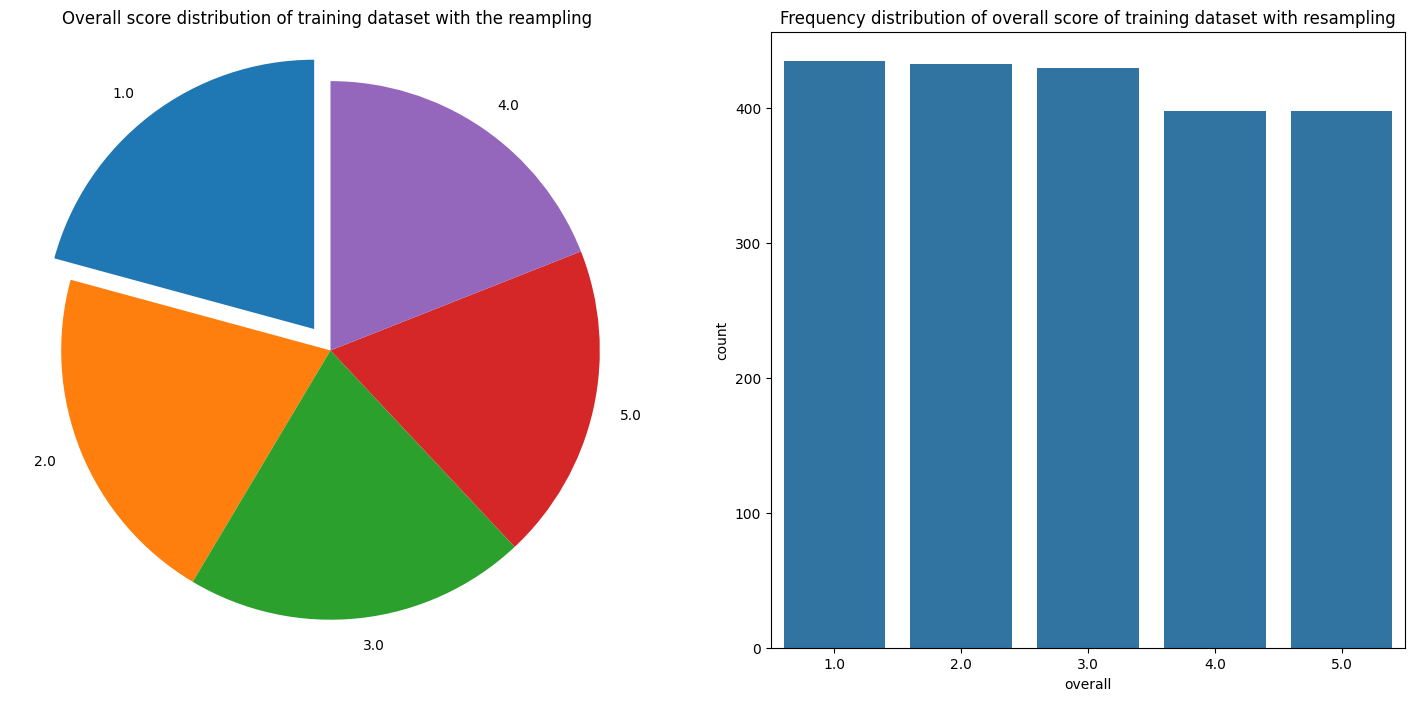


Figure 10: Distribution before and after applying SMOTETomek

## 3.8 Grid Search with Cross-Validation

**3.8.1 Grid Search and Model Configuration:**

We conduct a systematic grid search with cross-validation to fine-tune the hyperparameters of the model configurations. This exhaustive search allows for the identification of the optimal hyperparameters that yield the best model performance, facilitating efficient resource utilization and improving predictive accuracy.

**3.8.2 Robustness through Cross-Validation:**

We employ k-fold cross-validation to gauge the model's ability to generalize to unseen data reliably. The model is trained and assessed across different data subsets, ensuring that its performance remains consistent and robust.

**3.8.3 Benefits of Cross-Validation:**

Cross-validation offers a more solid evaluation of model performance than a singular train-test split, as it examines the model across multiple data scenarios.

* Cross-validation provides a more robust evaluation of a model's performance compared to a single train-test split.
* Cross-validation simulates the model's performance on different subsets of the data, giving insights into how well it generalizes to new, unseen samples.
* Cross-validation helps prevent data leakage, where information from the test set unintentionally influences the training process. Each fold is treated as an independent evaluation, reducing the risk of biased performance estimates.
* It provides a more comprehensive assessment of a model, which helps identify whether the model's performance is consistent or varies significantly based on the data it encounters.

# Machine Learning Tools and Model Selection

## 4.1 Sentiment Analysis

Sentiment analysis can help understand customer satisfaction, product strengths and weaknesses, and consumer trends. There are primarily three types of sentiment analysis:

1. Lexicon-based (unsupervised): Relies on a predefined lexicon (or dictionary) of words that have been assigned positive, negative, or neutral sentiments. This method calculates sentiment scores based on the presence and combination of these words in the text.
2. Machine Learning-based (supervised): Involves training a model on a dataset where the correct sentiments are known to predict sentiments on new, unseen data.
3. Deep Learning: Uses neural networks to capture deeper linguistic patterns and context. This approach often results in higher accuracy but requires more computational resources and a larger dataset for training.

Given that the scores corresponding to the reviews are known and considering the nuances and complexities of human language, we choose to conduct the Machine Learning-based approach for our analysis.

Steps to Conduct Sentiment Analysis using Machine Learning:

1. Data Preprocessing:
2. Cleaning: Remove special characters and extra white spaces.
3. Tokenization: Break down each text piece into individual words or tokens.
4. Normalization: Convert all words to lowercase to ensure consistency.
5. Removing Stopwords: Remove words such as "and" and "the" that do not contribute meaning to the sentiment.
6. Stemming/Lemmatization: Reduce words to their base or root form.
7. Feature Extraction:
8. Convert text data into numerical features that machine learning models can process. Common methods include BOW, TF-IDF as in the Topic Modeling session above, and word embeddings (e.g., Word2Vec, GloVe).
9. Model Selection and Implementation:
   1. Choose a suitable package or pretrained machine learning model suitable for sentiment analysis such as Natural Language Toolkit ( NLTK ) or Bidirectional Encoder Representations from Transformers (BERT) models .
   2. Apply the pre-trained model to new product reviews to predict sentiment.
   3. Analyze the sentiment results to extract actionable insights and interpretations.
   4. Identify common themes in positive and negative reviews to help inform product improvements, marketing strategies, and customer service approaches.

## 4.2 Topic Modeling

This process begins by preparing the text data through several steps: data cleaning, lowercasing, tokenization, stopwords removal, lemmatization, and stemming. It then extracts the text features using the Bag Of Word (BOW) method and the Term Frequency-Inverse Document Frequency (TF-IDF) and constructs a dictionary and a corpus essential for employing Latent Dirichlet Allocation (LDA) for topic modeling. The LDA model is executed to unearth the underlying topics within the review data. It highlights the keywords tied to each topic. The outcome is a collection of topics, where each is delineated by words that hold the most relevance to it. This approach offers insights into the primary themes discussed for each product. Here are the detailed investigation steps:

1. Data Pre-Processing
   1. Data Cleaning
      1. For missing values and duplicate reviews, we will remove the entire row of the product review.
      2. Remove punctuation and special characters.
   2. Lowercasing
      1. Convert all text to lowercase ensures consistency and reduces the complexity of the data.
   3. Tokenization
      1. Break down the reviews into individual words or tokens.
   4. Stopwords Removal
      1. Remove common stopwords like "and", "the", "is" and words with less than 3 characters.
   5. Lemmatization
      1. Change words to their base or root form (e.g., "walking" to "walk").
   6. Stemming
      1. Removes prefixes and suffixes to reduce words to their stems (e.g., "walking" to "walk").
2. Feature Extraction
   1. Use the Bag Of Word (BOW) method to represent text data.
   2. Use the Term Frequency-Inverse Document Frequency (TF-IDF) to evaluate the relevance of a word within a document in a collection of documents.
3. Modeling
   1. Use Latent Dirichlet allocation (LDA) to find the dominant topic with two different inputs, one from BOW, and another one from TF-IDF.
   2. After getting the dominant topic, merge the keyword and the dominant keyword as the feature data, with the category as the y data.
   3. Divide the data with 0.1 as the test size and 0.9 as our train size.
   4. Classify the reviews based on X to the right categories via algorithms such as logistic Regression and Multi-Layer Perceptron.

## 4.3 Classification Models

### 4.3.1 SVC

SVC (Support Vector Classifier) is a classification algorithm based on SVM (Support Vector Machine) that addresses both binary and multiclass problems. This algorithm operates by identifying an optimal hyperplane in high-dimensional space to differentiate between categories. The working principle of SVC involves finding a hyperplane that maximizes the classification margin, using kernel techniques to project the data into a higher-dimensional space where it becomes linearly separable. The training process of SVM involves minimizing a quadratic function relative to the weights. SVC is well-suited for complex classification tasks and performs exceptionally well when the dimensionality of the data exceeds the number of samples. Due to its strategy of maximizing the margin, SVC exhibits strong generalization capabilities. However, its drawbacks include being time-consuming, dependent on careful parameter tuning, and relatively difficult to interpret compared to other models, such as decision trees. For our model, we utilized the SVC package from scikit-learn.

### 4.3.2 Adaboosting

AdaBoost is an ensemble learning algorithm that combines multiple weak classifiers to create a strong classifier. Initially, AdaBoost assigns equal weights to all samples in the training set. In each iteration, it trains a weak classifier using the current weights to minimize the weighted error. The algorithm then increases the weights of misclassified samples and decreases the weights of correctly classified samples, which shifts its focus towards the more challenging cases. Ultimately, AdaBoost combines all the weak classifiers into a single strong classifier. This algorithm is straightforward and effective but is generally sensitive to outliers. For our model, we used the AdaBoostClassifier package from scikit-learn.

### 4.3.3 Gradient Boosting

GBDT (Gradient Boosting Decision Tree) is a type of ensemble learning method commonly used for regression and classification problems. The core idea of gradient boosting is to incrementally build a model by iteratively adding decision trees to correct errors from previous models, ultimately creating a highly accurate prediction model. The process begins by initializing a base learner that makes an initial prediction on the training data. It then calculates the residuals for each sample. In each iteration, GBDT trains a new decision tree to fit these residuals. The newly trained trees are added to the existing model in proportion to a learning rate, and each tree's output is weighted according to its effectiveness in reducing the overall error. The process terminates when the number of iterations reaches a preset number of trees or when improvements to the model are no longer significant. GBDT is known for its high predictive accuracy and ability to handle complex non-linear relationships, offering strong generalization capabilities. However, its drawbacks include lengthy training times, complex parameter tuning, and sensitivity to outliers. For our model, we utilized the GradientBoostingClassifier package from scikit-learn.

### 4.3.4 Random Forest

Random Forest is an ensemble learning method that enhances overall prediction accuracy and robustness by combining the predictions from multiple decision trees. The process works by randomly sampling from the training data to generate a training subset for each tree. At each node split, a subset of features is randomly selected, and multiple decision trees are independently constructed based on these randomly drawn samples and features. The final classification result is determined through a voting mechanism among the trees. Random Forest can handle high-dimensional data and, due to the introduction of randomness in both samples and features, it is less prone to overfitting compared to a single decision tree. It provides important scores for each feature in the decision-making process, which is crucial for understanding the significance of data features. The main drawbacks of Random Forest are its high model complexity, significant computational resource requirements, and its challenging interpretability due to the complexity of the model structure. For our model, we used the RandomForestClassifier package from scikit-learn.

### 

### 4.3.4 Hierarchy Classification

We created a hierarchical structure to organize the outputs of different classifiers. The reason for building the hierarchy tree is to assign priority to cases with a high probability of a rating score under 4. This hierarchy decision tree can serve as a supplementary model to optimize the response service from the merchant. Initially, the original rating categories are mapped to a new hierarchical structure. For example, ratings >=4 are mapped to the parent category '>=4', while ratings of 5.0 are mapped to the child category '=5'. Then, a classifier is trained for each parent category in the hierarchy. For example, a decision tree classifier is trained for the parent category '>=4'. Similarly, a classifier is trained for each child category in the hierarchy. For instance, a decision tree classifier is trained for the child category '5.0'. Eventually, using this hierarchical classifier system, predictions are made for the test data, and predictions are processed iteratively based on the hierarchy to obtain the overall prediction result and help decision making.

## 

Figure 11: Hierarchy Graph for Classification

## 4.4 Model Evaluation

We will evaluate the models using classification metrics such as confusion matrix, accuracy, precision, recall, and F1 scores to ensure that they effectively categorize comments.

By analyzing the categorization results, we can understand which features (topics, sentiment scores, keyword frequency) are most predictive of review categorization and will highlight product aspects that influence customer feedback.

### 4.4.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model and shows the counts of true positives, false positives, true negatives, and false negatives. It is important to understand the types of errors the model is making. It is used to identify a detailed breakdown of the model's predictions and provides insights into the specific types of errors (e.g., false positives, false negatives) made by the model, helping in targeted improvements.

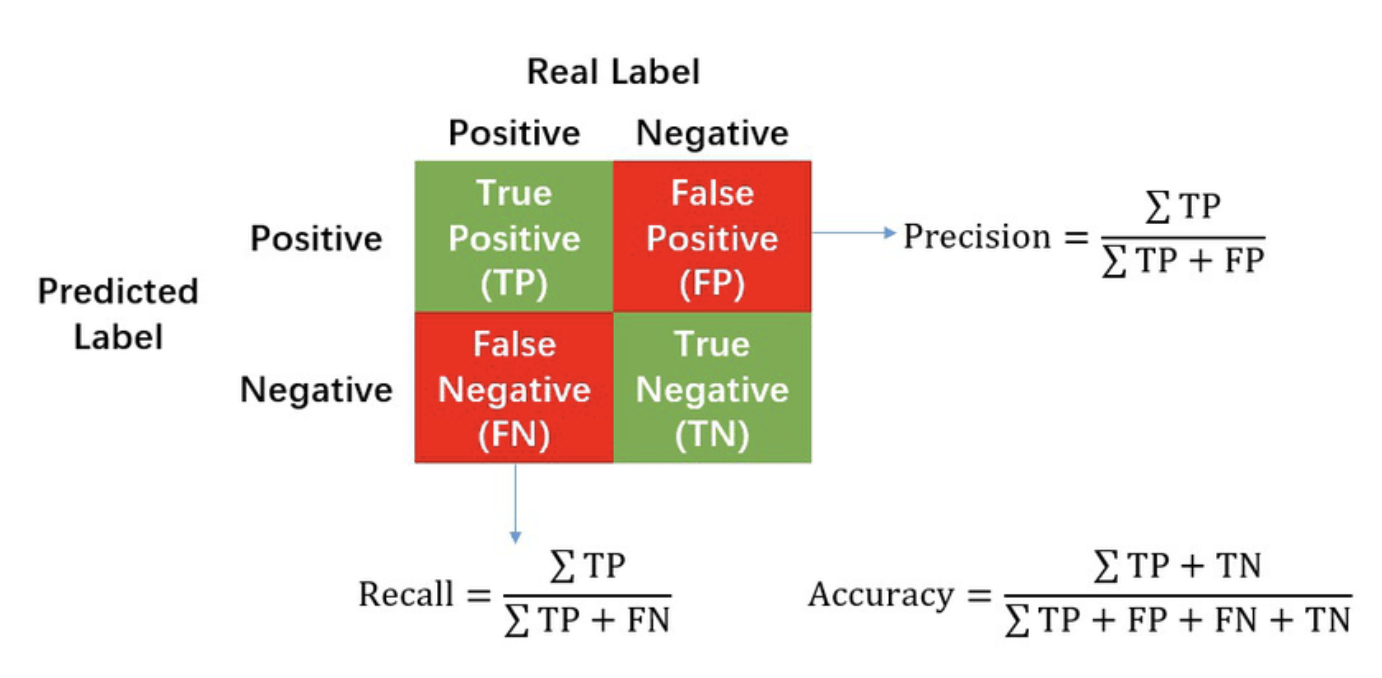


Figure 12. Calculation of Precision, Recall and Accuracy in the confusion matrix.

### 4.4.2 Accuracy

Accuracy is the ratio of correctly predicted instances to the total instances in the dataset, providing a basic measure of the model's correctness. It is important for a general assessment of the model's performance, especially when classes are balanced and the cost of false positives and false negatives is similar. Accuracy is easy to understand and calculate, providing a high-level view of model performance compared to other metrics.

#### 

### 4.4.3 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations, measuring the model's exactness, especially when the cost of false positives is high. It ensures that positive predictions made by the model are accurate, making it crucial in scenarios where false positives are costly.

### 4.4.4 Recall (Sensitivity)

Recall is the ratio of correctly predicted positive observations to all observations in the actual class, measuring the model's ability to capture all positive instances. F1 score is important when there is an imbalance between the classes or when both false positives and false negatives are costly, providing a balanced measure of precision and recall.

### 4.4.5 F1 Score

The F1 score is the weighted average of Precision and Recall. It provides a balance between precision and recall, making it useful when there is an uneven class distribution or when both false positives and false negatives are costly. Thus it is useful in imbalanced datasets, providing a single metric that balances precision and recall.

# 5. Results and Property Assessment

## 5.1 SVC

For the confusion matrix and performance metrics of the support vector machine model, we can observe the following. The precision, recall, and F1 score for class 1, 2, 3, and 4 are all 0, which means that the model does not correctly classify samples from any of these classes. Class 5 performs relatively well, with a precision of 0.585 and a recall of 1, which means that the model succeeds in identifying all samples that are true to class 5, but there are some samples from other classes that are incorrectly predicted The F1 score is 0.738, which indicates a relatively good performance in recognizing class 5.

The overall accuracy of the model is 58.5%, and since the metrics for classes 1 through 4 are all 0, we know that this accuracy only reflects the model's performance on class 5. Therefore the Macro Average metrics are very low (Macro Average) metrics are very low. The Weighted Average metric is slightly higher than the Macro Average due to the fact that it takes into account the number of samples in each class and Class 5 has more samples. Taken together, the model performs poorly on the vast majority of classes and is clearly over-biased towards one class.

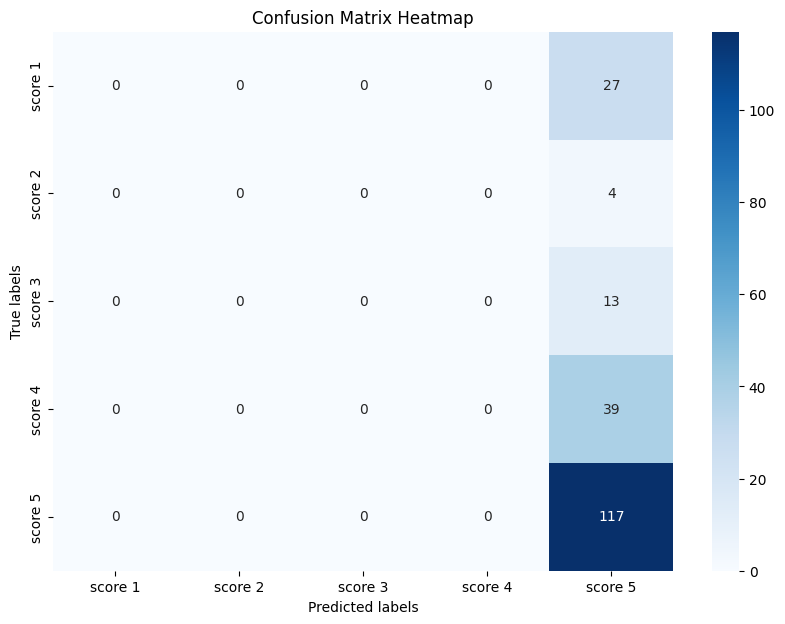


Figure 13: Confusion Matrix of SVC

Table 1. SVC Results

| Metrics | Result | Class | Precision | Recall | F1 Score |
| --- | --- | --- | --- | --- | --- |
| Weighted Avg Precision | 0.3422 | Class 1 | 0.0 | 0.0 | 0.0 |
| Weighted Avg Recall | 0.585 | Class 2 | 0.0 | 0.0 | 0.0 |
| Weighted Avg F1-Score | 0.4318 | Class 3 | 0.0 | 0.0 | 0.0 |
| Macro Avg F1-Score | 0.1476 | Class 4 | 0.0 | 0.0 | 0.0 |
| Macro average precision | 0.1170 | Class 5 | 0.585 | 1.0 | 0.738 |
| Macro average recall | 0.2 |  |  |  |  |
| Overall accuracy | 0.585 |  |  |  |  |

## 5.2 Adaboosting

The model class 1 has a relatively good precision of 0.577 and a recall of 0.556, which indicates that the model has some ability to recognize this class.The F1 score of 0.566 points out that the precision and recall are more balanced. The model classes 2, 3, and 4 are underperforming in terms of precision, recall, and F1 scores, but are gradually improving. Class 5 is the best performing class with a precision of 0.702, recall of 0.624 and F1 score of 0.661. Despite this, the model still produces a certain number of false positives when recognizing this class.

The overall accuracy is 51.5%. The performance problem of this model may be related to several factors, such as sample imbalance, poor selection of features, or the need for further adjustment of the model parameters.

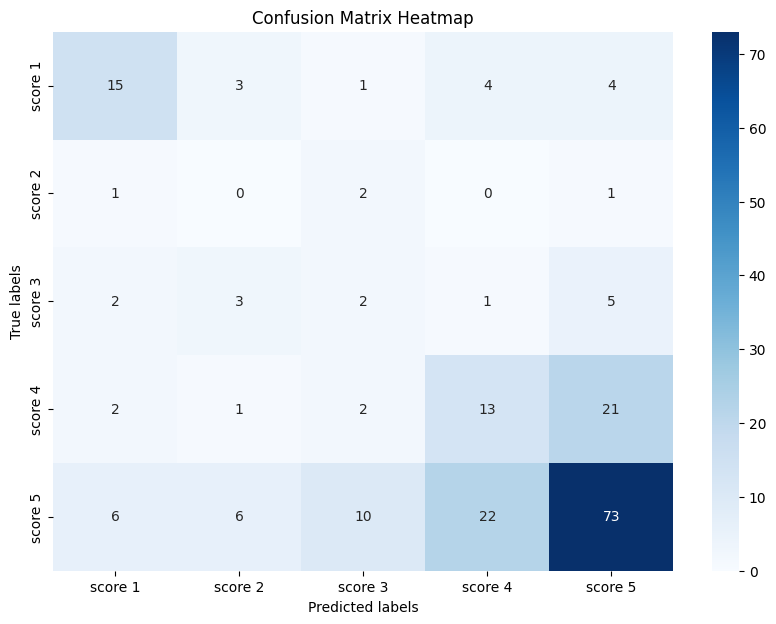


Figure 14: Confusion Matrix of Adaboosting

Table 2. Adaboosting Results

| Metrics | Result | Class | Precision | Recall | F1 Score |
| --- | --- | --- | --- | --- | --- |
| Weighted Avg Precision | 0.5595 | Class 1 | 0.5769 | 0.5555 | 0.5660 |
| Weighted Avg Recall | 0.515 | Class 2 | 0.0 | 0.0 | 0.0 |
| Weighted Avg F1-Score | 0.5357 | Class 3 | 0.1176 | 0.1538 | 0.1333 |
| Macro Avg F1-Score | 0.3378 | Class 4 | 0.325 | 0.3333 | 0.3291 |
| Macro average precision | 0.3443 | Class 5 | 0.7019 | 0.6239 | 0.6606 |
| Macro average recall | 0.3333 |  |  |  |  |
| Overall accuracy | 0.515 |  |  |  |  |

## 5.3 Gradient Boosting

Class 1 has a precision of 0.6129, a recall of 0.7037, and an F1 score of 0.6552, which indicates that the model performs relatively well on this class.

Class 2 has a 0 for all three metrics, indicating that the model fails to correctly identify any class 2 samples. Class 3 has a precision of 0.3333, a recall of only 0.0769, and an F1 score of 0.125, which indicates that the model is weak in recognizing class 3 and is prone to miss detection.

Class 4 has a precision of 0.1818, a recall of 0.0513, and an F1 score of 0.08, which further indicates that the model has poor performance in recognizing class 4. Class 5 has a precision of 0.6863, a recall of 0.8974, and an F1 score of 0.7778, which is the best performance of all the classes, indicating that the model has a strong recognition capability on this class.

The Macro average is 0.3629, 0.3459, and 0.3276, respectively, and these relatively low values indicate that the model's overall performance is average across all classes. The weighted precision of 0.5413 and the weighted recall and weighted F1 scores of 0.635 and 0.5672, respectively, are relatively high, mainly due to the fact that the high performance of the model on class 5 increases the weighted average. The confusion matrix heatmap shows that the model is relatively accurate in its predictions for classes 1 and 5, but has large errors in its predictions for the other classes.

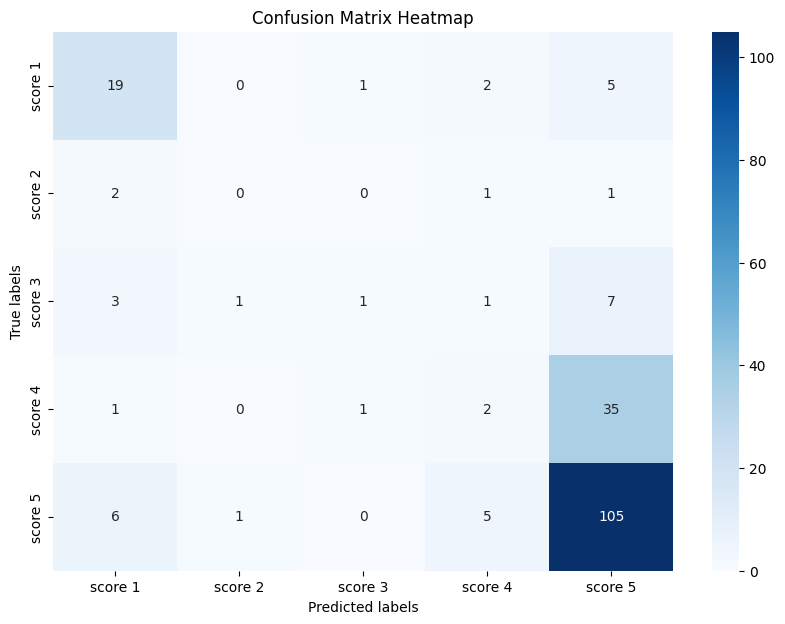


Figure 15: Confusion Matrix of Gradient Boosting

Table 3. Gradient Boosting Results

| Metrics | Result | Class | Precision | Recall | F1 Score |
| --- | --- | --- | --- | --- | --- |
| Weighted Avg Precision | 0.5413 | Class 1 | 0.6129 | 0.7037 | 0.6552 |
| Weighted Avg Recall | 0.635 | Class 2 | 0.0 | 0.0 | 0.0 |
| Weighted Avg F1-Score | 0.5672 | Class 3 | 0.3333 | 0.0769 | 0.125 |
| Macro Avg F1-Score | 0.3276 | Class 4 | 0.1818 | 0.0513 | 0.08 |
| Macro average precision | 0.3628 | Class 5 | 0.6863 | 0.8974 | 0.7777 |
| Macro average recall | 0.3458 |  |  |  |  |
| Overall accuracy | 0.635 |  |  |  |  |

## 5.4 Random Forest

Class 1 performs very well with a precision of 0.8438, a recall of 0.9, and an F1 score of 0.8710, which suggests that the model is able to identify and classify samples in this class very accurately. Class 2 performs significantly lower than class 1.0 with a precision of 0.3333, a recall of 0.4, and an F1 score of 0.3636, which may indicate that samples in this class are more difficult to distinguish or are underrepresented in the training data.

Class 3 has a precision of 0.5, a recall of 0.7778, and an F1 score of 0.6087, which suggests that the model has a high underdetection rate for this class, but that samples classified into this class are usually accurate. Class 4 has a precision and recall of 0.7561 and an F1 score of 0.7561, which shows the robust performance of the model on this class. Class 5 has a very high precision of 0.9533, a recall of 0.8869, and an F1 score of 0.9189, which means that the model performs well on this class as well, even though some true class 5 samples were not identified.

Accuracy per class is consistent with recall, with 0.9, 0.4, 0.7777, 0.7561, and 0.8869 for classes 1 through 5. The overall accuracy of 0.845 indicates that the majority of the samples were correctly classified. Macro average values are lower than the precision or recall of individual classes, with a precision of 0.6772, a recall of 0.7441, and an F1 score of 0.7036. They are unaffected by the high performance of any of the classes, and provide an overall balanced performance measure for all classes.

The Weighted Average values are higher, with a precision of 0.8605, a recall of 0.845, and an F1 score of 0.8505. Since they take into account the number of samples in each class, this means that the model classifies the majority of samples well. The confusion matrix heatmap shows the misclassification between classes. It can be seen that most of the misclassification occurs when the prediction is for class 5, which may be due to some similarity in features with other classes. Overall, the overall performance of the model is good.

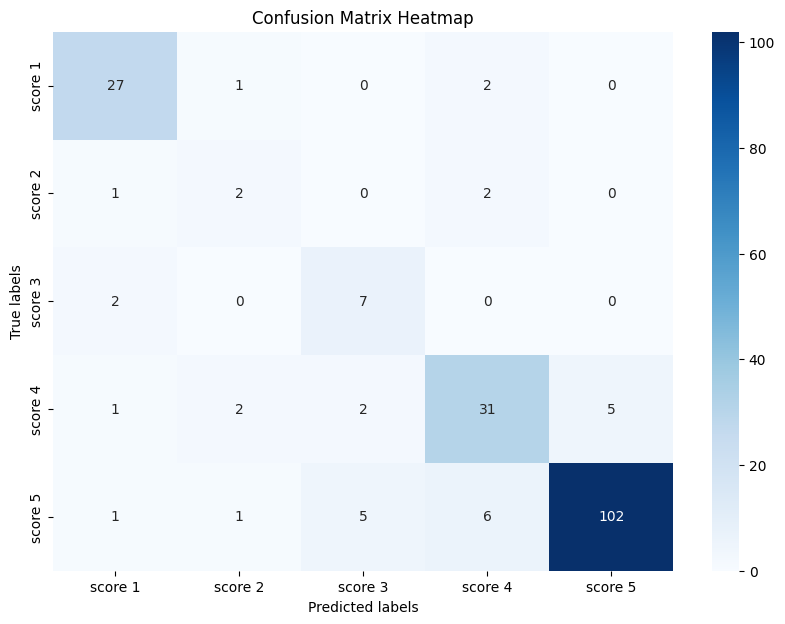


Figure 16: Confusion Matrix of Random Forest

Table 4. Random Forest

| Metrics | Result | Class | Precision | Recall | F1 Score |
| --- | --- | --- | --- | --- | --- |
| Weighted Avg Precision | 0.8605 | Class 1 | 0.8438 | 0.9 | 0.8710 |
| Weighted Avg Recall | 0.845 | Class 2 | 0.3333 | 0.4 | 0.3636 |
| Weighted Avg F1-Score | 0.8505 | Class 3 | 0.5 | 0.7778 | 0.6087 |
| Macro Avg F1-Score | 0.7037 | Class 4 | 0.7561 | 0.7561 | 0.7561 |
| Macro average precision | 0.6773 | Class 5 | 0.9533 | 0.8870 | 0.9189 |
| Macro average recall | 0.7442 |  |  |  |  |
| Overall accuracy | 0.845 |  |  |  |  |

## 5.5 Comparison

SVC: This model performs relatively well on class 5, but underperforms on the other classes. SVC usually performs well when the data has clear boundaries. Moreover, if the data is unbalanced, SVC may tend to favor the class with more samples. These reasons might explain why SVC has precision, recall, and F1 scores of 0 for most classes, but shows signs of overfitting on class 5.

AdaBoost: The performance of AdaBoost has improved over SVC in classes 1 and 5, which may be because AdaBoost iteratively strengthens the classification boundaries, thus allowing it to better recognize these classes. However, since AdaBoost relies on a series of weak classifiers, if these weak classifiers perform poorly on certain classes, or if the data features are insufficient to distinguish all classes, its performance will be limited.

GBDT: It performs well on classes 1 and 5, but its predictive ability is limited for other classes, which may be due to the gradient boosting model focusing more on those cases that are difficult to classify during continuous iterations.

RF: Random Forest performs best among all the models, which is because Random Forest builds multiple trees and introduces randomness to prevent overfitting, enhancing the model's generalization ability across different classes. Additionally, Random Forest is relatively robust against outliers and unbalanced data, which enables it to achieve more balanced performance across most classes.

## 

## 5.6 Hierarchy decision tree results

Table 5. Overall Result

| Class | <2 | <3 | <4 | >=4 | =5 |
| --- | --- | --- | --- | --- | --- |
| Accuracy | 0.895 | 1.0 | 0.97 | 0.925 | 1.0 |
| Mean Cross-Validation Accuracy | 0.9200 | 1.0 | 0.92875 | 1.0 | 0.755 |

The table above presents the performance of a classifier with varying levels of accuracy across different classes. Classes "<3" and "≥4" achieve perfect accuracy and cross-validation scores, indicating reliable classification. The class "<2" shows high consistency with accuracy above 0.89 and the highest cross-validation score among the non-perfect classes. Contrastingly, class "=5" exhibits a significant discrepancy with perfect accuracy but much lower cross-validation accuracy, hinting at possible overfitting or data variability issues. Overall, the classifier performs well, but the "=5" class requires further investigation to enhance model reliability.

# 6. Conclusion

## 6.1 Conclusion

In conclusion, this project successfully classifies the five different products using different models and data mining strategies, demonstrating the effectiveness of various machine learning models in analyzing customer conversations and predicting product feedback scores in the customer reviews of Amazon's health and personal care products. The Random Forest model is the top performer compared to Support Vector Machine, GBDT, and AdaBoost. The RF model's macro average F1-score of 0.7037 indicates its ability to handle imbalanced classes effectively. Moreover, the overall result score indicates that the Random Forest model excels in predicting the 5-star rating categories, especially with perfect accuracy for ratings less than 3 and equal to 5. The mean cross-validation accuracy confirms the model's robustness and generalization ability. The project shows that using sentiment analysis and topic modeling, we can gain a deeper understanding of customer emotions, concerns, and the key themes that drive product discussions. We also successfully used the result of topic modeling to classify the products based on different topic distributions. The development of different machine learning models to predict product feedback scores provides us a quantitative measure of customer satisfaction and helps us identify the most influential factors.

## 6.2 Enterprise Discussion

The project holds significant value for enterprises in the e-commerce domain, particularly those dealing with health and personal care products. By leveraging advanced data mining techniques and regression models, businesses can gain insights into customer sentiments, concerns, and overall satisfaction with their products. This information can be crucial in shaping product development strategies, improving customer support service, and enhancing the overall user experience. By understanding the key topics and issues that customers discuss in the QA sections, enterprises can proactively address common concerns, improve product descriptions, and provide targeted support to customers. Moreover, the model’s ability to predict product feedback scores can help businesses identify areas for improvement and prioritize their efforts accordingly.

## 6.3 Future Plan

Building upon the foundation laid by this project, future work could involve the development of a chatbot that leverages the insights gained from data mining techniques, text analysis, and predictive machine learning models. The chatbot could be designed to provide personalized responses to customer questions based on the identified topics and sentiments. By integrating the chatbot with the existing QA system on Amazon, businesses are able to generate instant and accurate responses to customer inquiries, improving the overall user experience and reducing the workload on the business side. Additionally, the chatbot could be trained to proactively suggest relevant products or provide customized recommendations based on detected customer preferences and past interactions. Another enhancement to the existing model could be incorporating more advanced natural language processing techniques, such as deep learning-based models, to improve the accuracy and contextual understanding of customer conversations. In summary, the development of a chatbot based on the insights from this project could result in a more efficient, personalized, and satisfactory experience for e-commerce customers.

# 

# Reference

[1] Albalawi, R., Yeap, T. H., & Benyoucef, M. (2020). Using topic modeling methods for short-text data: A comparative analysis. *Frontiers in Artificial Intelligence*, *3*. https://doi.org/10.3389/frai.2020.00042

[2] Berry, M. W., and Browne, M. (2005). Email surveillance using non-negative matrix factorization. *Compute Math Organize Theory* 11, 249–264. https://doi.org/10.1007/s10588-005-5380-5

[3] Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python*. O’Reilly.

[4] Blei, D.M., Ng, A.Y. and Jordan, M.I. (2003) Latent Dirichlet Allocation. J Machine Learning Research Archive, 3, 993-1022.

[5] Bose, R., Dey, R. K., Roy, S., & Sarddar, D. (2019). Sentiment analysis on online product reviews. Information and Communication Technology for Sustainable Development, 559–569. https://doi.org/10.1007/978-981-13-7166-0\_56

[6] Chen, Y., Zhang, H., Li, R., Z, Y., & Lin, J. (2019). Experimental explorations on short text topic mining between LDA and NMF based Schemes. Knowledge-Based Systems, 163, 1–13. https://doi.org/10.1016/j.knosys.2018.08.011

[7] Cheng, X., Yan, X., Lan, Y., & Guo, J. (2014). BTM: Topic Modeling over Short Texts. *IEEE Transactions on Knowledge and Data Engineering*, *26*(12), 2928–2941. https://doi.org/10.1109/tkde.2014.2313872

[8] Dasgupta, S. (2000). Experiments with random projection. *arXiv (Cornell University)*, 143–151. https://arxiv.org/pdf/1301.3849

[9] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407. https://doi.org/10.1002/(sici)1097-4571(199009)41:6&lt;391::aid-asi1&gt;3.0.co;2-9

[10] Gómez, J. C., Boiy, E., & Moens, M. (2011). Highly discriminative statistical features for email classification. Knowledge and Information Systems, 31(1), 23–53. https://doi.org/10.1007/s10115-011-0403-7

[11] Jolliffe, I. T. (1986). Principal component analysis. In Springer series in statistics. https://doi.org/10.1007/978-1-4757-1904-8

[12] Kaur, A., and Singh, R. (2019). Implementing sentiment analysis in relevance with Indian Elections. *J. Emerg. Technol. Innov. Res.* 6, 454–460. <http://www.jetir.org/papers/JETIR1905073.pdf>

[13] Kim, J., He, Y., and Park, H. (2014). algorithms for non-negative matrix and tensor factorizations: a unified view based on block coordinate descent framework. *J. Glob. Optim.* 58, 285–319. https://doi.org/10.1007/s10898-013-0035-4

[14] Neogi, P. P., Das, A. K., Goswami, S., & Mustafi, J. (2019). Topic modeling for text classification. *Advances in Intelligent Systems and Computing*, 395–407. https://doi.org/10.1007/978-981-13-7403-6\_36

[15] Ramage, D., Manning, C. D., & Dumais, S. T. (n.d.). Partially labeled topic models for interpretable text mining. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD’11*. https://doi.org/10.1145/2020408.2020481

[16] Schouten, K., & Frasincar, F. (2016). Survey on aspect-level sentiment analysis. IEEE Transactions on Knowledge and Data Engineering, 28(3), 813–830. https://doi.org/10.1109/tkde.2015.2485209

[17] Shen, C., Sun, C., Wang, J., Kang, Y., Li, S., Liu, X., Si, L., Zhang, M., & Zhou, G. (2018). Sentiment classification towards question-answering with hierarchical matching network.

[18] Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. https://doi.org/10.18653/v1/d18-1401

[19] Slonim, N., and Tishby, N. (2000). “Document clustering using word clusters via the information bottleneck method,” in *The 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY: ACM), 208–215. https://doi.org/10.1145/345508.345578

[20] Wang, X., and McCallum, A. (2006). *Topics Over Time: A Non-Markov Continuous-Time Model of Topical Trends. Philadelphia*, PA: ACM SIGKDD. https://doi.org/10.1145/1150402.1150450

[21] Wankhade, M., Rao, A. C., & Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. Artificial Intelligence Review, 55(7), 5731–5780. https://doi.org/10.1007/s10462-022-10144-1

[22] Xia, H., Yang, Y., Pan, X., Zhang, Z., & An, W. (2019, May 13). Sentiment analysis for online reviews using conditional random fields and support vector machines - electronic commerce research. SpringerLink. https://link.springer.com/article/10.1007/s10660-019-09354-7

[23] Xie, P., and Xing, E. P. (2013). “Integrating document clustering and topic modeling,” in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence* (Bellevue, WA), 694–703.

[24] Yan, X., Guo, J., Lan, Y., & Cheng, X. (2013). A biterm topic model for short texts. *Proceedings of the 22nd International Conference on World Wide Web*. https://doi.org/10.1145/2488388.2488514

# 

# Appendix

#Preprocessing

import pandas as pd

import gzip

def parse(path):

g = gzip.open(path, 'rb')

for l in g:

yield eval(l)

def getDF(path):

i = 0

df = {}

for d in parse(path):

df[i] = d

i += 1

return pd.DataFrame.from\_dict(df, orient='index')

df = getDF('reviews\_Health\_and\_Personal\_Care.json.gz')

df

df[df["asin"] == "B001KXZ808"].overall.value\_counts()

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from datetime import datetime

df\_instance = df[df["asin"] == "B0000U1OCI"]

# Convert timestamp strings to datetime objects for analysis

df\_instance['reviewTime'] = pd.to\_datetime(df\_instance['reviewTime'], format='%m %d, %Y',errors='coerce')

# Extract date, time, day of week, and hour for further analysis

df\_instance['date'] =df\_instance['reviewTime'].dt.date

df\_instance['year'] = df\_instance['reviewTime'].dt.year

df\_instance['month'] = df\_instance['reviewTime'].dt.month

df\_instance['day'] = df\_instance['reviewTime'].dt.day

df\_instance['time'] = df\_instance['reviewTime'].dt.time

df\_instance['day\_of\_week'] = df\_instance['reviewTime'].dt.day\_name()

df\_instance['numeric\_day'] = df\_instance['reviewTime'].dt.dayofweek

df\_instance['numeric\_day'] = df\_instance['numeric\_day'] + 1

df\_instance['helpful\_first\_number'] = df\_instance['helpful'].apply(lambda x: x[0])

df\_instance['helpful\_second\_number'] = df\_instance['helpful'].apply(lambda x: x[1])

# Handle division by zero by using a conditional statement

df\_instance['helpful\_division'] = df\_instance['helpful'].apply(lambda x: x[0] / x[1] if x[0] != 0 else 0)

df\_instance['helpful\_division'].sort\_values(axis=0, ascending=True)

# Message count analysis

# Count of messages per day

#df\_instance = df[df["asin"] == "B001KXZ808"]

sorted\_df =df\_instance.sort\_values(by='date', ascending=False)

top\_1000 = sorted\_df.head(1000)

pd.set\_option('display.max\_colwidth', 1000)

#print(top\_1000.head(1)['reviewText'])

df\_instance

reviewers\_per\_day = top\_1000['date'].value\_counts().sort\_index()

# Count of messages per sender

reviewer\_per\_sender = top\_1000['reviewerName'].value\_counts()

# Count of messages per day of the week

reviewers\_per\_day\_of\_week = top\_1000['day\_of\_week'].value\_counts()

# Plotting the results for visual analysis

plt.figure(figsize=(15, 6))

# Messages per day

plt.subplot(2, 2, 1)

reviewers\_per\_day.plot(kind='line')

plt.title('Reviews per Day')

plt.xlabel('Date')

plt.ylabel('Number of Reviews')

# # Messages per sender

# plt.subplot(2, 2, 2)

# reviewer\_per\_sender.plot(kind='bar')

# plt.title('Messages per Sender')

# plt.xlabel('Sender')

# plt.ylabel('Number of Messages')

# plt.xticks(rotation=45)

# Messages per day of the week

plt.subplot(2, 2, 2)

reviewers\_per\_day\_of\_week.plot(kind='bar')

plt.title('Reviewers per Day of the Week')

plt.xlabel('Day of the Week')

plt.ylabel('Number of Reviews')

plt.tight\_layout()

plt.show()

from textblob import TextBlob

# Sentiment Analysis Function

def analyze\_sentiment(message):

return TextBlob(message).sentiment

# Apply sentiment analysis to each message

top\_1000['sentiment'] = top\_1000['reviewText'].apply(lambda x: analyze\_sentiment(x))

# Extracting sentiment polarity and subjectivity

top\_1000['polarity'] = top\_1000['sentiment'].apply(lambda x: x.polarity)

top\_1000['subjectivity'] = top\_1000['sentiment'].apply(lambda x: x.subjectivity)

#Plotting

import matplotlib.pyplot as plt

import seaborn as sns

# Sentiment Polarity Distribution

plt.figure(figsize=(12, 6))

# Distribution of Polarity Scores

plt.subplot(1, 2, 1)

sns.histplot(top\_1000['polarity'], kde=True, bins=30)

plt.title('Distribution of Sentiment Polarity')

plt.xlabel('Polarity Score')

plt.ylabel('Frequency')

def classify\_polarity(polarity):

if polarity < -0.2:

return -1

elif abs(polarity) <= 0.2:

return 0

else:

return 1

top\_1000['polarity\_class'] = top\_1000['polarity'].apply(classify\_polarity)

# Average Sentiment Polarity per Day

avg\_polarity\_per\_day = top\_1000.groupby('date')['polarity'].mean()

plt.subplot(1, 2, 2)

avg\_polarity\_per\_day.plot(kind='line', color='blue')

plt.title('Average Sentiment Polarity per Day')

plt.xlabel('Date')

plt.ylabel('Average Polarity Score')

plt.tight\_layout()

plt.show()

selected\_columns = ['asin','overall','year','month','day','numeric\_day','helpful\_first\_number','helpful\_second\_number', 'helpful\_division','polarity', 'polarity\_class']

new\_df = top\_1000[selected\_columns].copy()

new\_df.to\_csv("output.csv", index=False, encoding='utf-8-sig')

import pandas as pd

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from gensim import corpora

from gensim.models.ldamodel import LdaModel

# Download necessary NLTK data

nltk.download('punkt')

nltk.download('stopwords')

# Define a function for text preprocessing

def preprocess\_text(text):

tokens = word\_tokenize(text.lower())

tokens = [word for word in tokens if word.isalpha()] # Remove non-alphabetic tokens

tokens = [word for word in tokens if word not in stopwords.words('english')] # Remove stopwords

return tokens

# Apply preprocessing to chat messages

top\_1000['processed\_message'] = top\_1000['reviewText'].apply(preprocess\_text)

# Creating a dictionary and corpus needed for topic modeling

dictionary = corpora.Dictionary(top\_1000['processed\_message'])

corpus = [dictionary.doc2bow(text) for text in top\_1000['processed\_message']]

# Running LDA model

lda\_model = LdaModel(corpus, num\_topics=5, id2word=dictionary, passes=10)

# Displaying the topics

topics = lda\_model.print\_topics(num\_words=5)

for topic in topics:

print(topic)

import matplotlib.pyplot as plt

import pandas as pd

# Number of words to display per topic

num\_words = 10

# Retrieve the words for each topic

topics = {i: [word for word, \_ in lda\_model.show\_topic(i, topn=num\_words)] for i in range(lda\_model.num\_topics)}

# Convert to DataFrame for easier plotting

topics\_df = pd.DataFrame(topics)

# Plotting

plt.figure(figsize=(15, 5))

for i in topics\_df.columns:

plt.subplot(1, len(topics\_df.columns), i+1)

plt.barh(topics\_df[i], range(len(topics\_df[i])), color='blue')

plt.title(f'Topic {i}')

plt.yticks(range(len(topics\_df[i])), topics\_df[i])

plt.tight\_layout()

plt.show()

#Classification

#install packages

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC,SVR

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score,mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn import preprocessing

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

#deal with unbalanced dataset

from imblearn.combine import SMOTETomek

from imblearn.under\_sampling import NearMiss

#visualization

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

# prepare dataset

df = pd.read\_csv('B00B5H5BGA.csv')

df.drop(columns='asin',inplace=True)

y= df.overall

y = pd.DataFrame(y, columns=['overall'])

df.drop(columns='overall',inplace=True)

X = df

# normalize the data

scaler = MinMaxScaler()

scaler.fit(X)

X\_scaled = scaler.transform(X)

X\_scaled = pd.DataFrame(X\_scaled, columns=X.columns)

#resample the data

# Create a subplot grid of 1 row by 2 columns, and set the figure size

f, ax = plt.subplots(1, 2, figsize=(18, 8))

# Calculate the quality counts

quality\_counts = y['overall'].value\_counts()

explode = [0.1 if i == quality\_counts.idxmax() else 0 for i in quality\_counts.index]

# Plot the pie chart in the first subplot

ax[0].pie(quality\_counts, labels=quality\_counts.index, startangle=90, shadow=False, explode=explode)

ax[0].set\_title('Overall score distribution of the prodcut B00B5H5BGA') # Set title for the first subplot

ax[0].axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle

# Correct the count plot: change "wine quality" to the correct column name 'quality'

# and plot it in the second subplot

ax[1] = sns.countplot(x="overall", data=y, ax=ax[1])

ax[1].set\_title("Frequency distribution of overall score of the product B00B5H5BGA") # Set title for the second subplot

# Display the plot

plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Implementing Oversampling for Handling Imbalanced

smk = SMOTETomek(random\_state=101)

X\_res,y\_res=smk.fit\_resample(X\_train, y\_train)

# Create a subplot grid of 1 row by 2 columns, and set the figure size

f, ax = plt.subplots(1, 2, figsize=(18, 8))

# Calculate the quality counts

quality\_counts = y\_res['overall'].value\_counts()

explode = [0.1 if i == quality\_counts.idxmax() else 0 for i in quality\_counts.index]

# Plot the pie chart in the first subplot

ax[0].pie(quality\_counts, labels=quality\_counts.index, startangle=90, shadow=False, explode=explode)

ax[0].set\_title('Overall score distribution of training dataset with the reampling' ) # Set title for the first subplot

ax[0].axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle

# Correct the count plot: change "wine quality" to the correct column name 'quality'

# and plot it in the second subplot

ax[1] = sns.countplot(x="overall", data=y\_res, ax=ax[1])

ax[1].set\_title("Frequency distribution of overall score of training dataset with resampling") # Set title for the second subplot

# Display the plot

plt.show()

hierarchy = {

'>=4': ['5.0', '4.0'],

'=5': ['5.0'],

'< 3': ['2.0', '1.0'],

'< 4' : ['3.0','2.0','1.0'],

'< 2': ['1.0']

}

# Map the original labels to the new hierarchy labels

y\_train\_hierarchy = y\_train.replace({

'5.0': '=5, >=4', '4.0': '>=4', '3.0': '< 4', '2.0': '< 3, < 4', '1.0': '< 2'

})

classifiers = {}

for parent\_class, child\_classes in hierarchy.items():

y\_train\_parent = y\_train\_hierarchy.isin([parent\_class])

y\_train\_child = y\_train\_hierarchy[y\_train\_hierarchy.isin(child\_classes)]

dt = DecisionTreeClassifier(random\_state=101)

dt.fit(X\_train, y\_train\_parent)

classifiers[parent\_class] = dt

# Calculate accuracy for each level of the hierarchy

accuracies = {}

for parent\_class, clf in classifiers.items():

y\_pred\_parent = clf.predict(X\_test)

# print(y\_pred\_parent)

accuracies[parent\_class] = accuracy\_score(y\_test.isin([parent\_class]), y\_pred\_parent)

for child\_class in hierarchy[parent\_class]:

y\_pred\_child = clf.predict\_proba(X\_test)

# print(y\_pred\_child)

# Use the predicted probabilities for further processing or analysis

# Display the accuracies

for parent\_class, acc in accuracies.items():

print(f"Accuracy for {parent\_class}: {acc}")

from sklearn.ensemble import RandomForestClassifier

# Convert 'y' DataFrame to string

y\_train = y\_train.astype(str)

hierarchy = {

'>=4': ['5.0', '4.0'],

'=5': ['5.0'],

'< 4' : ['3.0','2.0','1.0'],

'< 3': ['2.0', '1.0'],

'< 2': ['1.0']

}

# Map the original labels to the new hierarchy labels

y\_train\_hierarchy = y\_train.replace({

'5.0': '=5, >=4', '4.0': '>=4', '3.0': '< 4', '2.0': '< 3, < 4', '1.0': '< 2'

})

# Train a classifier for each level of the hierarchy

classifiers = {}

for parent\_class, child\_classes in hierarchy.items():

y\_train\_parent = y\_train\_hierarchy.isin([parent\_class])

y\_train\_child = y\_train\_hierarchy[y\_train\_hierarchy.isin(child\_classes)]

rf = RandomForestClassifier(random\_state=42)

# Perform cross-validation

accuracies = cross\_val\_score(rf, X\_train, y\_train\_parent, cv=5)

print(f"Mean cross-validation accuracy for {parent\_class}: {accuracies.mean()}")

# Fit the classifier

rf.fit(X\_train, y\_train\_parent)

classifiers[parent\_class] = rf

# Calculate accuracy for each level of the hierarchy

accuracies = {}

for parent\_class, clf in classifiers.items():

y\_pred\_parent = clf.predict(X\_test)

accuracies[parent\_class] = accuracy\_score(y\_test.isin([parent\_class]), y\_pred\_parent)

for child\_class in hierarchy[parent\_class]:

y\_pred\_child = clf.predict\_proba(X\_test)

# Use the predicted probabilities for further processing or analysis

# Display the accuracies

for parent\_class, acc in accuracies.items():

print(f"Accuracy for {parent\_class}: {acc}")

print(y\_test)

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support

import pandas as pd

# Convert 'y' DataFrame to string

y = y.astype(str)

y\_res = y\_res.astype(str)

# Assuming your original dataset is 'X' and 'y'

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=101)

hierarchy = {

'>=3': ['5.0', '4.0', '3.0'],

'>=4': ['5.0', '4.0'],

'=5': ['5.0'],

'< 3': ['2.0', '1.0'],

'< 2': ['1.0']

}

# Map the original labels to the new hierarchy labels

y\_train\_hierarchy = y\_res.replace({

'5.0': '=5, >=4', '4.0': '>=4', '3.0': '>=3', '2.0': '< 3', '1.0': '< 2'

})

# Train a classifier for each level of the hierarchy

classifiers = {}

for parent\_class, child\_classes in hierarchy.items():

y\_train\_parent = y\_train\_hierarchy.isin([parent\_class])

y\_train\_child = y\_train\_hierarchy[y\_train\_hierarchy.isin(child\_classes)]

rf = RandomForestClassifier(random\_state=101)

# Perform cross-validation

accuracies = cross\_val\_score(rf, X\_res, y\_train\_parent, cv=5)

print(f"Mean cross-validation accuracy for {parent\_class}: {accuracies.mean()}")

# Fit the classifier

rf.fit(X\_res, y\_train\_parent)

classifiers[parent\_class] = rf

# Predict and print the labels for each class in the hierarchy

for parent\_class, clf in classifiers.items():

print(f"Predictions for {parent\_class}:")

y\_pred\_parent = clf.predict(X\_test)

y\_pred\_child = clf.predict\_proba(X\_test)

# Print final predictions along with actual labels

for idx, (actual, final) in enumerate(zip(y\_test, y\_pred\_parent)):

print(f"Actual: {actual[idx]}, Predicted: {final}, Predicted Probabilities: {y\_pred\_child[idx]}")

# Calculate precision, recall, and F1-score

for parent\_class, clf in classifiers.items():

y\_pred\_parent = clf.predict(X\_test)

precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred\_parent, average='macro')

print(f"Precision for {parent\_class}: {precision}")

print(f"Recall for {parent\_class}: {recall}")

print(f"F1-score for {parent\_class}: {f1}")

print(y\_test)

accuracies = {}

for parent\_class, clf in classifiers.items():

# print(parent\_class)

y\_pred\_parent = clf.predict(X\_test)

accuracies[parent\_class] = accuracy\_score(y\_test.isin([parent\_class]), y\_pred\_parent)

for child\_class in hierarchy[parent\_class]:

print(child\_class)

y\_pred\_child = clf.predict\_proba(X\_test)

# Use the predicted probabilities for further processing or analysis

print(y\_pred\_child)

break

# Display the accuracies

for parent\_class, acc in accuracies.items():

print(f"Accuracy for {parent\_class}: {acc}")

# Initialize predictions with the test data

predictions = pd.DataFrame(index=X\_test.index)

print(predictions)

# Iterate over each classifier (level of the hierarchy

# For each level of the hierarchy

for parent\_class, clf in classifiers.items():

# Predict probabilities for each class at this level

probabilities = clf.predict\_proba(X\_test)

print(probabilities)

# Store the probabilities or class labels in the predictions dataframe

predictions[parent\_class] = probabilities

# At this point, predictions dataframe will have the probabilities or class labels at each level of the hierarchy

# You can then use these predictions to determine the final predicted value based on your hierarchy logic

import networkx as nx

import matplotlib.pyplot as plt

# Create a directed graph

G = nx.DiGraph()

# Add nodes and edges to the graph

G.add\_node('Review Data', subset='default')

G.add\_node('score < 4', subset='level1')

G.add\_node('score >=4', subset='level1')

G.add\_node('score < 2', subset='level2')

G.add\_node('score < 3', subset='level2')

G.add\_node('score = 5', subset='level2')

G.add\_node('score >= 4', subset='level2')

G.add\_edge('Review Data', 'score < 4')

G.add\_edge('Review Data', 'score >=4')

G.add\_edge('score < 4', 'score < 2')

G.add\_edge('score < 4', 'score < 3')

G.add\_edge('score >=4', 'score = 5')

G.add\_edge('score >=4', 'score >= 4')

# Draw the graph

pos = nx.multipartite\_layout(G, subset\_key='subset')

nx.draw(G, pos, with\_labels=True, node\_size=2000, node\_color='skyblue', font\_size=10, font\_weight='bold', edge\_color='gray', arrowsize=20)

plt.title('Hierarchy Graph for Classfication')

plt.show()

# Classification model

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

svc = SVC(random\_state = 101)

accuracies = cross\_val\_score(svc, X\_train, y\_train, cv=5)

svc.fit(X\_train,y\_train)

y\_pred\_svc = svc.predict(X\_test)

print("Train Score:",np.mean(accuracies))

print("Test Score:",svc.score(X\_test,y\_test))

# # Calculate the quality counts

f, ax = plt.subplots(1, 2, figsize=(18, 8))

quality\_counts = y\_test['overall'].value\_counts()

print(quality\_counts)

explode = [0.1 if i == quality\_counts.idxmax() else 0 for i in quality\_counts.index]

# Plot the pie chart in the first subplot

ax[0].pie(quality\_counts, labels=quality\_counts.index, startangle=90, shadow=False, explode=explode)

ax[0].set\_title('Overall score distribution of test') # Set title for the first subplot

ax[0].axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle

# Correct the count plot: change "wine quality" to the correct column name 'quality'

# and plot it in the second subplot

ax[1] = sns.countplot(x="overall", data=y\_test, ax=ax[1])

ax[1].set\_title("Frequency distribution of overall score of test") # Set title for the second subplot

# Display the plot

plt.show()

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

# Classification model

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Implementing Oversampling for Handling Imbalanced

smk = SMOTETomek(random\_state=101)

X\_res,y\_res=smk.fit\_resample(X\_train, y\_train)

dt = DecisionTreeClassifier(random\_state=101)

ada\_boost = AdaBoostClassifier(base\_estimator=dt, random\_state=101)

accuracies = cross\_val\_score(ada\_boost, X\_res, y\_res, cv=5)

ada\_boost.fit(X\_res, y\_res)

y\_pred\_ada = ada\_boost.predict(X\_test)

print("Train Score:", np.mean(accuracies))

print("Test Score:", ada\_boost.score(X\_test, y\_test))

from sklearn.ensemble import GradientBoostingClassifier

# Classification model

#X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

gb = GradientBoostingClassifier(random\_state=42)

accuracies = cross\_val\_score(gb, X\_res, y\_res, cv=5)

gb.fit(X\_train, y\_train)

y\_pred\_gb = gb.predict(X\_test)

print("Train Score:", np.mean(accuracies))

print("Test Score:", gb.score(X\_test, y\_test))

from sklearn.ensemble import RandomForestClassifier

# Classification model

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

rf = RandomForestClassifier(random\_state=42)

accuracies = cross\_val\_score(rf, X\_res, y\_res, cv=5)

rf.fit(X\_res, y\_res)

y\_pred = rf.predict(X\_test)

print("Train Score:", np.mean(accuracies))

print("Test Score:", rf.score(X\_test, y\_test))

param\_grid = {

'n\_estimators': [100, 200], # Number of trees in the forest

'max\_depth': [None, 10, 20, 30], # Maximum depth of the tree

'min\_samples\_split': [2, 5, 10], # Minimum number of samples required to split an internal node

'min\_samples\_leaf': [1, 2, 4], # Minimum number of samples required to be at a leaf node

'bootstrap': [True, False], # Method of selecting samples for training each tree

}

# Initialize a RandomForestClassifier

rf = RandomForestClassifier(random\_state=42)

# Initialize the GridSearchCV object

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, n\_jobs=-1, verbose=2)

# Fit the grid search to the data

grid\_search.fit(X\_res, y\_res)

# Predict using the best model

best\_model\_rf = grid\_search.best\_estimator\_

y\_pred\_rf = best\_model\_rf.predict(X\_test)

report = classification\_report(y\_test, y\_pred\_rf, output\_dict=True)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)

accuracy = report['accuracy']

print(f'Overall accuracy: {accuracy}')

# Macro average metrics

macro\_avg\_precision = report['macro avg']['precision']

macro\_avg\_recall = report['macro avg']['recall']

macro\_avg\_f1 = report['macro avg']['f1-score']

print(f'Macro average precision: {macro\_avg\_precision}')

print(f'Macro average recall: {macro\_avg\_recall}')

print(f'Macro average F1-score: {macro\_avg\_f1}')

# Weighted average metrics

weighted\_avg\_precision = report['weighted avg']['precision']

weighted\_avg\_recall = report['weighted avg']['recall']

weighted\_avg\_f1 = report['weighted avg']['f1-score']

print(f'Weighted average precision: {weighted\_avg\_precision}')

print(f'Weighted average recall: {weighted\_avg\_recall}')

print(f'Weighted average F1-score: {weighted\_avg\_f1}')

# To store results of models

from sklearn.metrics import classification\_report

report = classification\_report(y\_test, y\_pred\_gb, output\_dict=True)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_gb)

# Print accuracy for each class

for label, metrics in report.items():

if label not in ('accuracy', 'macro avg', 'weighted avg'):

print(f'\nClass {label}:')

print(f'Precision: {metrics["precision"]}')

print(f'Recall: {metrics["recall"]}')

print(f'F1 Score: {metrics["f1-score"]}')

# Calculating per-class accuracy

print("\nPer-class accuracy:")

class\_accuracies = conf\_matrix.diagonal() / conf\_matrix.sum(axis=1)

for i, accuracy in enumerate(class\_accuracies):

print(f'Class {i} accuracy: {accuracy}')

accuracy = report['accuracy']

print(f'Overall accuracy: {accuracy}')

# Macro average metrics

macro\_avg\_precision = report['macro avg']['precision']

macro\_avg\_recall = report['macro avg']['recall']

macro\_avg\_f1 = report['macro avg']['f1-score']

print(f'Macro average precision: {macro\_avg\_precision}')

print(f'Macro average recall: {macro\_avg\_recall}')

print(f'Macro average F1-score: {macro\_avg\_f1}')

# Weighted average metrics

weighted\_avg\_precision = report['weighted avg']['precision']

weighted\_avg\_recall = report['weighted avg']['recall']

weighted\_avg\_f1 = report['weighted avg']['f1-score']

print(f'Weighted average precision: {weighted\_avg\_precision}')

print(f'Weighted average recall: {weighted\_avg\_recall}')

print(f'Weighted average F1-score: {weighted\_avg\_f1}')

cm = confusion\_matrix(y\_test, y\_pred\_svc)

# Create a heatmap from the confusion matrix

plt.figure(figsize=(10, 7)) # Set the figure size

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=['score 1', 'score 2','score 3','score 4', 'score 5'],

yticklabels=['score 1', 'score 2','score 3','score 4', 'score 5']) # Change class names as needed

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix Heatmap')

plt.show()