Kaggle Competition Report

Joseph Aoun[1]

[1]Boston Univeristy

October 29, 2024

Abstract

In this report, I will discuss the strategies I employed to achieve a 64.2% accuracy in a kaggle competition. The project utilized an Amazon dataset containing approximately 1.6 million text reviews from customers about a wide range of products available on the platform.

1 Introduction

Before delving into the methods I employed, it is essential to explore and understand the dataset's structure. Upon importing the dataset using pandas, I identified nine primary features:

1. **Id:** A unique identifier for each review entry.
2. **ProductId:** The identifier of the product being reviewed.
3. **UserId:** The identifier of the user who authored the review.
4. **HelpfulnessNumerator:** The number of users who found the review helpful.
5. **HelpfulnessDenominator:** The total number of users who evaluated the review's helpfulness.
6. **Time:** The timestamp indicating when the review was submitted.
7. **Summary:** The title or headline of the review.
8. **Text:** The main content of the review.
9. **Score:** The rating given by the user, ranging from 1 to 5 stars.

At first glance it seems that only the text review and summary are the most important features of the dataset, however as we will see later, it turns out that there is a surprising amount of features that seem to help with the accuracy.

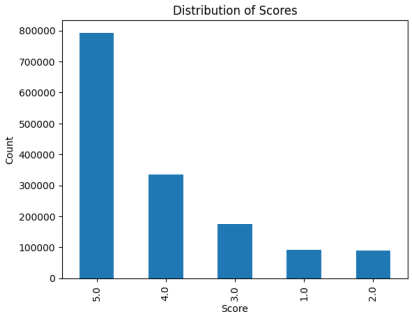

Figure 1: Word distribution

2 First Model Assumptions

The first step I performed was visualized the distribution of the ratings in the data set, surprisingly, the majority of the reviews were skewed towards a 5 star rating, followed by 4 stars, 3 stars, 1 star and 2 stars (Figure 1). This indicated that accuracy alone might not be enough to evaluate model success, especially for intermediate ratings

Textual Analysis with Word Clouds: To understand common language associated with different ratings, I created word clouds for each rating category. Reviews with lower ratings (1 or 2 stars) commonly included words like "disappointed" or "poor," while higher ratings featured words like "love" and "perfect." (See Figure 2)

My main objective was to first try and build a model utilizing only the text,"the:" summary, and helpfulness features of the dataset. To achieve this, I began by cleaning these features, removing unnecessary punctuations – commas, dots, etc. The next step was "normalizing" the text field, which entailed converting all characters to lower case. This standardization mainly reduces the number of unique words by treating "I Love" and "i love" as the same token. Additionally, I removed "filler" words such as "the" or "so" that don't add much contribution to the review. Finally, I concatenated the text and summary fields into a single comprehensive

Table 1: Caption

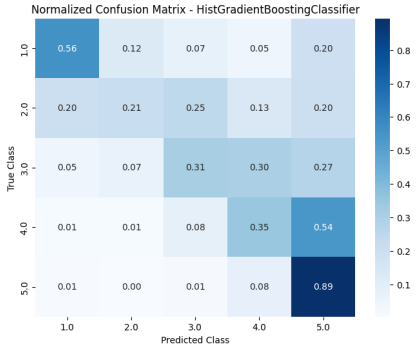

Figure 2: Confusion matrix

feature. The reason is that both features highly very correlated and it would be better to amalgamate them into one feature to reduce redundancy in the dataset.

After cleaning the text input, I utilized the TFIDF(Term Frequency-Inverse Document Frequency) method learned in class to vectorize the tokens – creating an embedding space from the words. I set maxfeatures to 3000 and the n-gram range to be from 1 to 2 (uni-gram and bi-gram could be helpful in detecting not only the current word, "happy", but also the previous two tokens "not happy" which could add important meaning to the review). My intuition was that certain words such as "hate" would be prevalent in lower-rated reviews while words such as "love" would be more ubiquitous in positive reviews. In addition, I added a helpfulness ratio (dividing HelpfulnessNumerator / HelpfulnessDenominator). The rationale behind it was that these ratios could further accentuate a product's rating. For instance, if a product received a low rating but garnered many up-votes for its reviews, it might indicate inherent issues with the product. Consequently, this could make subsequent reviews more likely to be negative as well. After conducting thorough research, I developed an ensemble model that combines **Lasso-regularized Logistic Regression** with a **Random Forest** classifier. Upon training the ensemble on the dataset, it achieved an accuracy of **56%**.

3 Second Iteration

After visualizing the output of my model via a confusion matrix, it became evident that the model effectively distinguished between 5-star and 1-star reviews. However, the classification of intermediate ratings was less accurate. Specifically, 4-star reviews were frequently misclassified as 5 stars, and 2-star reviews tended to be incorrectly labeled as 1 star. (Figure 2)This pattern suggests that the model struggles to accurately capture the nuanced sentiments present in the middle ratings, leading to a tendency to favor the extreme star categories. To alleviate this issue, I tried adding extra features to my dataset to help the model better discriminate the classes. The first feature I added was a sentiment analyzer using python's TextBlob library. This Natural Language Processing (NLP) tool assigns a sentiment score within the continuous range of [-1, 1], where **-1** indicates very negative sentiment, **0** represents neutral sentiment, and **1** signifies very positive sentiment. The next feature I added was "!" count, . The rationale behind this is that individuals expressing strong emotions—whether positive or negative—are likely to use multiple exclamation marks in their reviews. Similarly, I introduced a **capitalized words counter**, operating under the assumption that users conveying anger or excitement tend to capitalize words more frequently. In addition, I created a length measure feature, which is the length of the review. My hypothesis was that longer reviews might correlate with extreme ratings, either very low or very high, as users tend to elaborate more when they have strong opinions about a product. It is important to note that I also experimented with stemming — to reduce words to their root or base form — but it did perform worse since it was very naive in its performance, for instance the word "caring" would become "car" and thus loose its meaning. Furthermore, I experimented with incorporating **Part of Speech (POS) tagging** to provide grammatical context to the words in the reviews. While POS tagging can enhance understanding by identifying the role of each word (e.g. noun, verb, adjective), the implementation proved to be time-consuming and computationally expensive. Additionally, the inclusion of POS tags did not contribute to any significant improvement in the model's performance, making it an inefficient addition to the feature set. I kept the same models and was able to achieve a modest 60% accuracy.

4 Final Iteration

After meticulous research I first changed the model I was using (ensemble) to a histgradient-

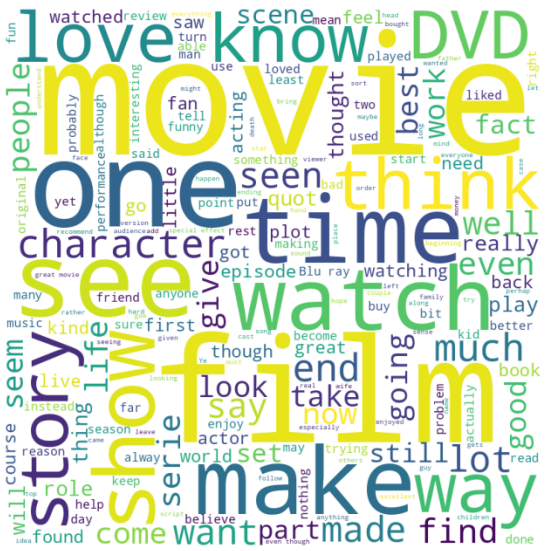

Figure 3: word cloud

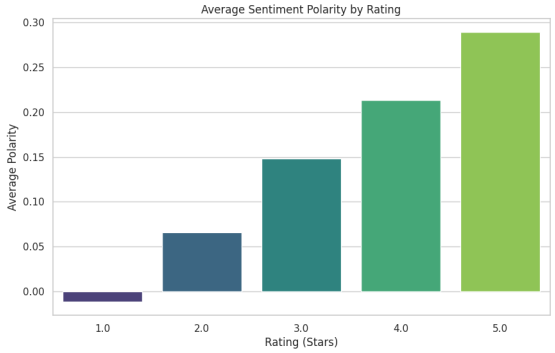

Figure 4: Average Sentiment by rating

boostingregressor model. It is basically an advanced implementation of the Gradient Boosting algorithm provided by the Scikit-Learn library. In addition, I added a subjectivity metric which outputs a float from [0, 1] 0 being an objective sentence and 1 being a subjective sentence. One big change which affected my accuracy was choosing to incorporate aggregate features mainly for the products and for users. Since I noticed that there was a strong correlation between a product's average review and a user's review on the product. In addition, certain users tend to give high / low ratings, and bu incorporating these features, I was able to capture this information in my dataset. I also added some more features such as sentiment shifting and negative word counting (to help the model distinguish a bit more negative reviews). And after successfully fine tuning the model, I was able to get to a 64.2% accuracy.

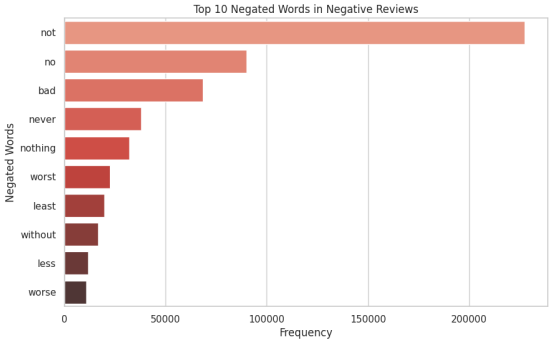

Figure 5: top 10 negative words

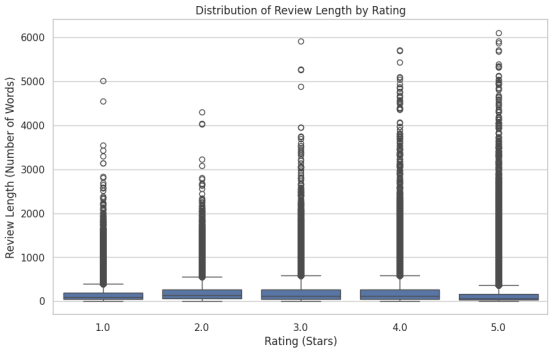

Figure 6: Review length / rating

Discussion

I have tried many features as outlined above some of which where not only computationally expensive, but also didn't improve the accuracy that much. What I noticed was that having a good data was much more important that than what model I was using. In addition visualizing the dataset was an invaluable resource to deeply understanding the dataset at hand. Finally, I used parallelization using pandas which ran the notebook much faster.

References

[1] https://www.geeksforgeeks.org/logistic-regression-vs-random-forest-classifier/
[2] https://medium.com/@apappascs/how-to-perform-sentiment-analysis-in-python-using-textblob-dc64f262e9ea
[3] https://medium.com/data-science-lab-spring-2021/amazon-review-rating-prediction-with-nlp-28a4acdd4352
[4] https://arxiv.org/abs/1605.05362
[5] https://matheo.uliege.be/handle/2268.2/2707