Sentiment Analysis to Predict Helpfulness of Amazon Reviews

Hiroki Hoshida

University of California, San Diego
December 7, 2020

Abstract

Apart from the one to five star rating one can give to a product in an Amazon review, each review has its own 'Helpful' score other users give to the review itself. This paper focuses on creating a classification model to predict whether a review is 'Helpful' or not using various features found in the Amazon provided review dataset on products in the 'Video Games' category. By creating features based on review text length as well as features based on sentiment analysis of the text and the comparisons to the star rating given and the mean star rating for each product, a random forest based classifier model can be created that can fairly accurately predict if a review would be 'Helpful' or 'Not Helpful'. Amazon, and other similar online retailers would be able to use this information to provide high quality reviews, even before votes are placed, which may incentivise customers to buy more products they may not have considered before.

Keywords: Machine Learning, Sentiment Analysis, Review Helpfulness

Introduction

Amazon, like many other online shopping websites, use a rating system that customers can use to determine a product's quality. These reviews consist of a five star rating system (one star is worst, five is best), and usually also come with some sort of text the reviewer includes. Amazon also includes a 'Helpful' feature, where other users can rate the review itself by voting the review 'Helpful' or 'Not Helpful.' This allows for the ranking of the reviews themselves, and Amazon raises the top rated reviews to the top and pushes the worst rated reviews to the bottom. This means that if users find a review helpful, a review can rise to the top even if its star rating does not align with the average star rating of the product. One example of this is the 'Super Mario 3D All-Stars' game for the Nintendo Switch. Despite the five star average rating, the top rated review (and is thus shown first) is a rather critical three star review (Figure 1). However, one thing to note is the high quality text within the review. Even though the review may not align with the general populations' opinions, certain features of this review made people vote 'Helpful', even if they may not necessarily agree with it.

Figure 1: Top rated review for 'Super Mario 3D All-Stars, ' a five-star product according to Amazon.

Clearly, there are many factors in what makes a review 'Helpful' or 'Not Helpful', and thus this project aims to create a model that could help describe and predict 'Helpful' reviews.

A model like this one is not just for educational purposes, however. Amazon and other online retailers could easily use models like these to prepare potentially 'Helpful' reviews in advance, and rank reviews on products with low view counts and low 'Helpful' votes across its reviews. The model created in this project classifies reviews into 'Helpful' or 'Not Helpful' using various features, some provided (such as star rating) and others engineered.

Data

Dataset

Amazon provides a massive dataset of reviews on its site categorized into the major product categories available on the site. These datasets are available as tab-delimited files and can be downloaded and explored freely [1]. For this project, I decided to use reviews in the 'Video Games' category on Amazon. This was for various reasons. One is that gamers tend to be more tech savvy than the average consumer, which I assumed would lead to more interactions in the reviews. Second is that the sample size was fairly large (1780268 reviews) which meant that even with some filtering and engineering it would leave me with a sizable but still easy to manipulate dataset.

The dataset contains 15 fields, as shown in Table 1.

Field	Description
marketplace	2 letter country code of the marketplace where the review was written.
customer_id	Random identifier that can be used to aggregate reviews written by a single author.
review_id	The unique ID of the review.
product_id	The unique Product ID the review pertains to.
product_parent	Random identifier that can be used to aggregate reviews for the same product.
product_title	Title of the product.
product_category	Broad product category that can be used to group reviews.
star_rating	The 1-5 star rating of the review.
helpful_votes	Number of helpful votes.

total_votes	Number of total votes the review received.
vine	Review was written as part of the Vine program.
verified_purchase	The review is on a verified purchase.
review_headline	The title of the review.
review_body	The review text.
review_date	The date the review was written.

Table 1: Fields found in the reviews dataset.

There were some fields that could be dropped immediately: marketplace (all reviews are from the US), product_category (all reviews are for products in the 'Video Games' category) and product_parent (product_id was used for product identification). This left 12 fields for exploration.

Exploratory Analysis

The first analysis I did was the value counts for each of the categorical features: star_rating, vine, and verified_purchase. Plotting star_rating showed that a majority of the reviews were five stars, though none of the star counts were insignificant (Figure 2).

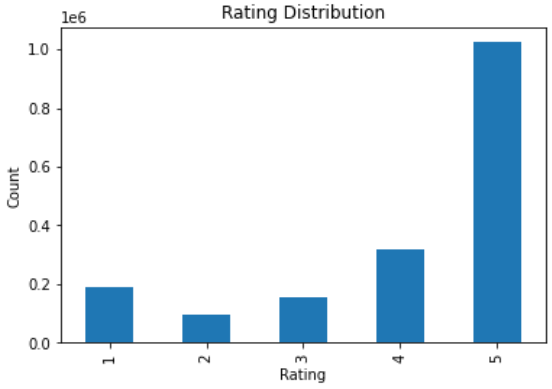


Figure 2: Star rating distribution found in dataset.

Next, I counted the values in vine and verified_purchase (Table 2).

.

Feature	Yes	No
vine	4284 (0.24%)	1775984
verified_purchase	1164785 (65.42%)	615483

Table 2: Fields found in the reviews dataset.

The number of 'Yes' in vine was extremely small, so I decided to just drop the column. However, verified_purchase had a much more even split, so I one-hot encoded the feature to use for analysis.

Each review had a helpful_votes and total_votes, which are the number of 'Helpful' votes the review received and the total number of votes the review received, respectively. By using this, I created a helpful_ratio column by dividing helpful_votes by total_votes. Some reviews, however, had a very low number of votes or none at all, which created some heavy skew in the data. So, I decided to remove all reviews with less than 5 votes, which left the dataset with 282533 reviews. Out of these reviews, the mean helpful_ratio was 0.577 with a standard deviation of 0.312 (Figure 3). The average helpful_ratio had a slight positive correlation with the star_rating (Figure 4).

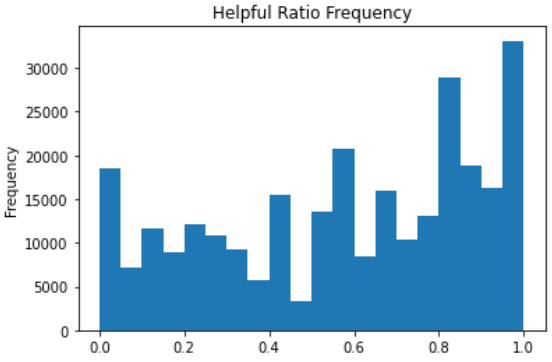


Figure 3: Helpful ratio frequency of dataset

Furthermore, I created three new columns in the dataset: reviews_by_user, which gave the total number of reviews by the user from the unfiltered original dataset, average_rating, the average rating of a product by all of its reviews, and .word_count, the number of words found in review_body for each review.

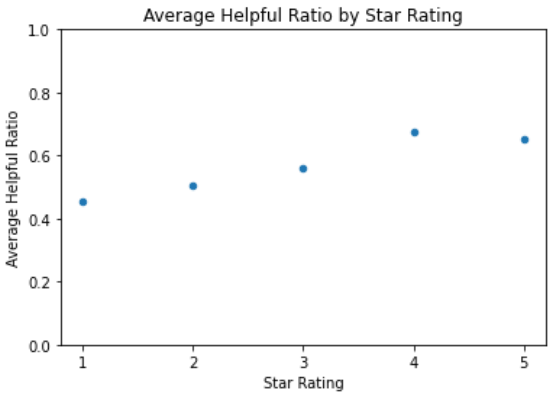


Figure 4: Average helpful ratio by star ranking.

These columns allowed for the creation of another column, rating_deviation, which was created by subtracting the average_rating of the product from the star_rating of the particular review.

In order to allow classification, I separated all the reviews available to 'Helpful' or 'Not Helpful', with a threshold of 0.75. Out of the filtered reviews, 39.0% of the reviews were classified as 'Helpful' and 60.9% were classified as 'Not Helpful'.

Predictive Task

Classification

As stated in the Introduction, my main goal with this project was to be able to classify reviews as 'Helpful' or 'Not Helpful'. In order to create the best model, I decided to test various popular classification models, such as Logistic Regression, Ridge Regression (used as a classifier), K-Nearest Neighbor, and Random Forest. In order to test these models, I used sklearn's accuracy_score, which computes subset accuracy for classification models. As the two classifications are fairly equal in size in the dataset, accuracy_score should return a fairly accurate score value.

Baseline Models

Before creating the classification models however, I created two baselines I would be able to test my models against. Both are based on the data available in the dataset.

1. 'Helpful' if the star_rating is within 1 star range from the average_rating.
2. 'Helpful' if the total review count of the user is above the mean.

I chose the first baseline model because it seems logical that most of the highly voted reviews would fairly closely align with the general consensus. I chose the second model as someone who writes many reviews may be more likely to write quality reviews, which would lead to potentially higher voted reviews. The first baseline model gave a subset accuracy of 0.668. The average number of reviews written was 1.38, and the second baseline model gave a subset accuracy of 0.517.

Further Features

In order to beat the baselines, I decided to create more features I could train my model on. The dataset does not contain any information on the products themselves apart from the name (such as price), so I focused on the text content of each review instead. I explored various processing methods and decided to use sentiment analysis to determine the *subjectivity* and *polarity* of each review. I first used the Natural Language Toolkit's (NLTK), word lemmatizer, then I used TextBlob, an open source Python library for textual data analysis. TextBlob uses pattern matching with scores from SentiWordNet, an opinion mining lexical resource.

With this, I created two more columns: subjectivity, a float column ranging from 0.0 to 1.0 where 0.0 are the most objective reviews and 1.0 are the most subjective, and polarity, a float column ranging from -1.0 to 1.0 where -1.0 is most negative and 1.0 is most positive. I then scaled star_reviews and average_rating between -1.0 and 1.0, and calculated the polarity_deviation, which is the difference between the average_rating and polarity of the review text (Figure 5). I chose to include this because of examples like the 'Mario' review above

(Figure 1). Despite the average star score, the review contains some positive words (masterpiece etc.) that could have led to a better 'Helpful' score.

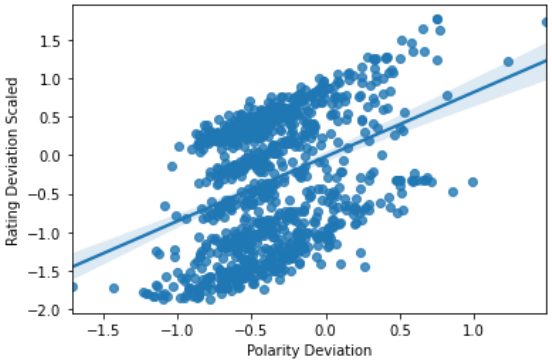


Figure 5: Correlation between polarity deviation of average rating and rating deviation. The two are positively correlated

Model Selection

I tested the four classification models above, each with the default parameters provided in scikit-learn, using the new features I created. I used a train/test split of 0.6/0.4 over the entire dataset. Out of the four, Random Forest gave me the best accuracy scores (Table 3). In fact, the other three models gave a worse score than baseline number 1.

Model	Accuracy Score
Logistic Regression	0.63015
Ridge Classification	0.627425
Random Forest Classification	0.70275
K-Nearest Neighbor	0.66075

Table 3: Accuracy scores of the models using default parameters.

I believe Random Forest worked the best out of the four models because of a few reasons. One is that perhaps the solution is non-linear, which would result in the logistic regression and ridge regression models to underperform. Another is that the data is not independently and identically distributed, which would also mean the two linear models would underperform. Finally, the data may contain defining outliers that are missed by k-nearest neighbors, resulting in random forest to be the best performing out of the four. Overall, I felt that random forest gave me the most flexible model to work with, especially with its parallelizability, meaning I would be able to run the model and test variants quickly.

Model

Random Forest

With my earlier exploration of the four models, I decided to further focus on the random forest model. This was because of multiple reasons. One is the parralizability mentioned earlier, which would become helpful when tuning hyperparameters later. Random forest is also robust to outliers and non-linear data, weaknesses of the k-nearest neighbor and linear classifiers, respectively. Furthermore, using a random forest classifier helps to reduce bias and potential overfitting. The random forest model is called so because of its structure of multiple decision trees. By using combinations of different available features, the random forest model can sort features by importance; that is, the features that have the largest impact on the regression or classification prediction result are rated higher. This allows for feature selection, which is done later in the paper. Though it is said that random forest gets slower as features increase, the ability to run many threads simultaneously as well as the fact that the dataset is fairly sparse on features that would multiple in size a la one-hot encoded features that the scalability of the model was not a large issue at all. Furthermore, the property of random forest models itself is that they are resistant to overfitting compared to other classification and regression models.

Bad Features

In order to further improve the model before tuning the hyperparameters, I tested a wide variety of other features I could create from the text of the reviews. I browsed the reviews manually to potentially find

patterns. One thing I noticed was the excessive usage of symbols and capital letters in some reviews. One example of such a review is this:

> '!!!!!!!!!!!!!!!!!!!!THIS CONTROLLER IS NOT WORTH A DIME. SELLERS WONT CONTACT YOU BACK AND CONTROLLER STOPS WORKING. NOT WORTH BUYING.!!!!!!!!!!!!!!!!Great controller. It very fun to use but the rapid fire turns off once in a while. I will give it one more week of testing and ill fully rate it in my option but for now 3 stars. Update. It stops working while in mid game. I sent the seller and email like 2 weeks ago and never replied. Not worth buying'

Because of reviews like this, I assumed that usage of symbols and capital letters were a good sign of a passionate, and thus quality review. So, I created two features: exclamation to count the number of exclamation points and capital_letters to count the ratio of capital letters to total characters. However, using the features in the random forest model led to no improvement in accuracy, and with certain train/test splits led to worse performance than without. So, I removed these two features again from the feature set.

Feature Selection

As I currently had multiple redundant features (scaled_star_reviews and star_reviews, etc.) I decided to select features using the random forest model, which is another reason I decided to use random forest. Using scikit-learn's SelectFromModel module, I found the importance scores of each feature. By selecting the features with scores above the mean, I created the final list of features to build my model on: word_count, average_rating_scaled, rating_deviation_scaled, polarity_deviation, and subjectivity.

Using these features and scikit-learn's default random forest parameters, the model gave an accuracy score of 0.720925, an improvement of over 0.02.

Hyperparameters

To improve upon the model, I used a randomized search then a grid search to find the best possible parameters. In my initial randomized search, I used the combinations seen below. The random forest model's ability to run multiple trees simultaneously came in handy, as it allowed me to test a large number of hyperparameters in the fraction of the time it would have taken to test other models.

Parameter	Tested Values
n_estimators	10, 50, 100, 150
max_depth	20, 50, 100, 200, 300
min_samples_split	2, 3, 5, 10, 20
min_samples_leaf	1, 2, 3
bootstrap	True, False

Table 4: Hyperparameters tested with random search.

With 200 random combinations of these parameters tested with 5-fold cross validation, the best parameters were:

Parameter	Value
n_estimators	150
max_depth	300
min_samples_split	20
min_samples_leaf	1
bootstrap	True

Table 5: Best parameters found with randomized search.

I then used grid search to further fine tune these parameters.

Parameter	Tested Values
n_estimators	140, 150, 160
max_depth	250, 275, 300, 325, 350
min_samples_split	19,20,21
min_samples_leaf	1
bootstrap	True

Table 6: Hyperparameters tested with random search.

Again, with 5-fold cross validation, the best parameters were:

Parameter	Value
n_estimators	160
max_depth	300
min_samples_split	21

Table 7: Best parameters found with grid search.

Based on this, my final model had the parameters:

Parameter	Value
n_estimators	160
max_depth	300
min_samples_split	21
min_samples_leaf	1
bootstrap	True

Table 8: Final parameters of model.

The random forest model with these parameters had an accuracy score of 0.72855 on the test dataset, a minor improvement of 0.008 over the pre-hyper parameterized model.

Literature

Existing Work

The dataset was found on the Amazon review dataset webpage [1], and as such is extremely popular in analyses. This same dataset has been used by many researchers, both public and private, to create accurate recommender systems. One example is Fang and Zhan's 2015 paper on sentiment analysis [2], more specifically sentiment polarity categorization. Sentiment polarity was a huge part of making my model accurate, and my methods are fairly similar to those from this paper. In the paper, sentiment scores were calculated with a similar pattern matching formula with a sentiment index.

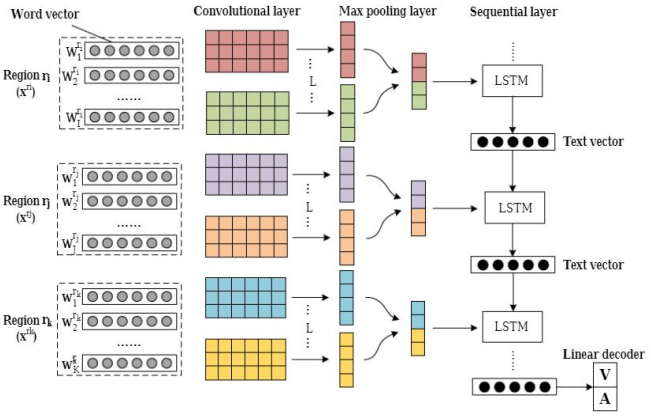
According to Mumdabi and Schuff [3], reviews with extreme ratings are usually less helpful than those with more moderate ratings. Furthermore, review depth is also important for helpful reviews. The conclusions from this study are similar to mine, as my model relied heavily on rating deviations from the average as well as review length.

State-of-the-Art Systems

More modern papers, however, contain far more advanced methods of sentiment analysis. One method is shown in Jin Wang et.al.'s paper on using a convolutional neural network - long short-term memory (CNN-LSTM) model for sentiment analysis [4]. This method uses both CNN's abilities to extract local sentiment information (sentences etc.) but also applies LSTM's methods of modelling texts across sentences for long-distance sentiment information (paragraph, document, etc.). This model creates convoluted layers in each region of the full document and creates n-grams locally, the layers which output max-pooling subsamples. LSTM is then used to traverse through all the regions created from CNN and create the final representation of the text with sentiment analysis (Figure 6).

This clearly differs from my methods as my model does not use neural networks at all, and does not use n-grams, which may have resulted in a higher score.

Figure 6: Architecture of the CNN-LSTM model..



Results

Comparison to Alternatives

Compared to models with neural networks implemented, my model is weaker. For example, the CNN-LSTM model in Jin Wang et.al. 's paper provided a 0.77 Pearson correlation coefficient using the Stanford Sentiment Treebank.

The results of my model show that there is a correlation between review length and review helpfulness as well as a correlation between review deviation from the mean and review helpfulness. However, the term 'helpfulness' in the context of Amazon reviews may be misleading. This is because the review deviation further from the mean resulting in lower helpfulness scores can mean that users are using the 'Helpful' button as an 'Agree' button instead. However, it does not change the fact that the 'Helpful' reviews are what the customers want to see, and thus Amazon should be pushing these potentially 'Helpful' reviews up, even before significant voting takes place. The final parameters used, word_count, average_rating_scaled, rating_deviation_scaled, polarity_deviation, and subjectivity, all are based around the rating and text.

Model	Accuracy Score
Logistic Regression	0.63015
Ridge Classification	0.627425
Random Forest Classification (Hyperparameterized and feature selected)	0.72855
K-Nearest Neighbor	0.66075
Baseline 1	0.668
Baseline 2	0.517

Features such as verified purchaser were removed by feature selection, so this may imply that review readers do not necessarily care if a review has the 'Verified Purchase; badge on a review before voting. If the dataset contained more information, such as the price of the product, more granular product categories, or even the brand or publisher of the product, further analysis based around the product itself could have been done. Regardless, I believe the results of my random forest model given the available data has a satisfactory accuracy. The other models I tested had a worse accuracy than the fairly simple baseline 1 model, and my fully trained random forest classification beats the baseline handily.

Future analyses can be done on the other categorical data Amazon offers. However, I believe that my model will still stay generally accurate, as I do not implement any video game specific features in my model.

References

- [1] Amazon Reviews PDS. (n.d.). Retrieved from https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt
- [2] Fang, X., & Zhan, J. (2015). Sentiment analysis using product review data. *Journal of Big Data*, 2(5). doi:10.1186/s40537-015-0015-2
- [3] Mudambi, S. M., & Schuff, D. (2010). Research Note: What Makes a Helpful Online Review? A Study of Customer Reviews on Amazon.com. *MIS Quarterly*, 34(1), 185-200. doi:10.2307/20721420
- [4] Wang, J., Yu, L., Kai, R. K., & Zhang, X. (2016). Dimensional Sentiment Analysis Using a Regional CNN-LSTM Model. *Association for Computational Linguistics*, 225-230. doi:10.18653/v1/P16-2037