Predicting the Helpfulness of Amazon Reviews

Lina Battikha, Kurumi Kaneko, Nicole Reardon, Candus Shi

Abstract

In this project, we attempt to predict the helpfulness of Amazon reviews from a subset of the 2014 Amazon Product Reviews dataset, which includes features such as the overall rating, the time of review, the review text, and the number of helpful votes a review received. Since our goal is to predict how helpful a review is in terms of the proportion of voters, we started by implementing a Linear Regression model. We then find that a XGBoost Regressor model with the following parameters—300 estimators, a learning rate of 0.1, and max depth of 5–had the best performance on this task.

1. Introduction

How does one decide whether or not to purchase a product on Amazon? One common metric is to read the reviews of a product. However, some products may have thousands of reviews, and most customers may not read them all. It may be beneficial to display more helpful reviews higher up on the website for a better user experience.

Each row in the dataset (He and McAuley, 2016) (McAuley et al., 2014) is a user's review of an item. Features include:

- reviewerID: ID of the reviewer
- asin: ID of the product
- reviewerName: name of the reviewer
- helpful: helpfulness rating of the review
- reviewText: text of the review
- overall: rating of the product (1-5 stars)
- summary: summary of the review
- unixReviewTime: time of the review as a unix timestamp
- reviewTime: time of the review in MM DD, YYYY format

The dataset encodes the helpful feature as a list of two values, e.g. [3, 15] where the first value is the total number of helpful votes and the second value is the total number of votes for the review.

The entire dataset includes reviews from May 1996 - July 2014, resulting in a total of 142.8 million reviews. To reduce runtime in training and to maximize the number of helpful/not helpful votes per review, we focus on products only in the Electronics category and use the 5-core version of the dataset. This is a more dense subset of the overall dataset as it only includes users and items that have at least 5 reviews each. We also looked at other categories that we presumed would attract more users to vote on whether a review was helpful, such as Home and Kitchen, but found that the Electronics category had a higher proportion of reviews with at least one vote.

Additionally, we only consider the subset of the data where the reviews had votes. 963,227 reviews did not have any votes, i.e. the helpful feature was [0, 0]. Since these types of reviews give us no information on whether a review was helpful or how helpful a review was, we remove them. For example, if we feature engineer the proportion of helpful votes, a review with helpful feature [0, 10] is not the same as a review with helpful feature [0, 0]. The former represents a completely unhelpful review while the latter does not give us any information on helpfulness. The remaining dataset includes 725,961 reviews.

2. EDA

First, we conduct some exploratory data analysis. Figure 1 shows the distribution of the overall ratings of the reviews with votes, and it illustrates that most reviews were positive (4 or 5 stars). Figure 2 shows the distribution of the percentage of helpful votes, and it shows that most reviews had a high percentage of helpfulness. There are also a handful of unhelpful reviews and neutral reviews (approximately half of votes indicate helpfulness). Overall, our outcome variable (how helpful a review is) is not normally distributed. This skewness may violate the assumptions of models such as linear regression, particularly the normally distributed residuals and heteroscedasticity.

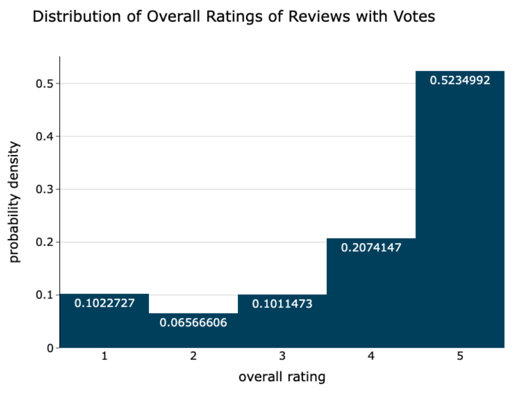


Figure 1: Distribution of Overall Ratings of Reviews with Votes

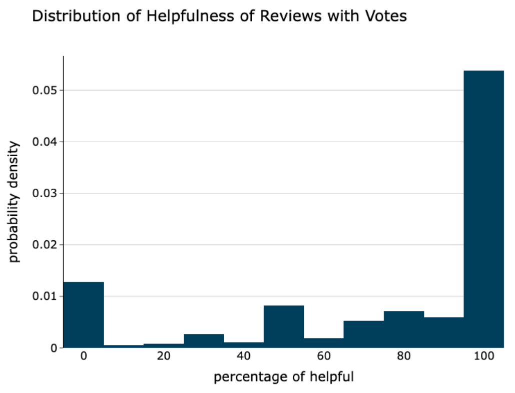


Figure 2: Distribution of Helpfulness of Reviews with Votes

We also want to leverage the temporal features of the dataset. For different products, we can see temporal patterns in how helpful a review was. For example, Figure 3 shows reviews for a product that had many reviews early on that gradually got more helpful as time passed. Figure 4 shows reviews for a product that had many reviews later on that had a wider range of helpfulness but in general increased in helpfulness as time passed.

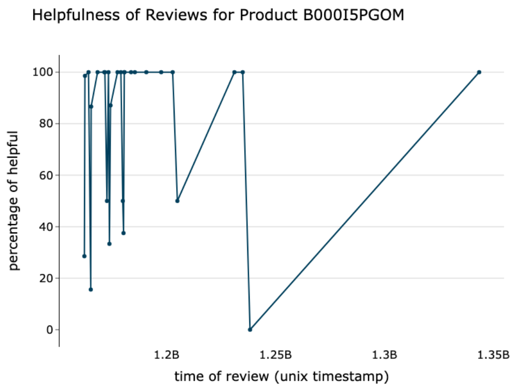


Figure 3: Helpfulness of Reviews for a Product with Frequent Early Reviews

Finally, we take a look at the number of positive votes vs negative votes for each category of the overall rating. As we can see from Figure 5, positive reviews garnered more helpful votes; while from Figure 6, we can observe a bimodal shape, where polarized ratings (1 star and 5 stars) attracted more negative votes compared to the other star ratings.

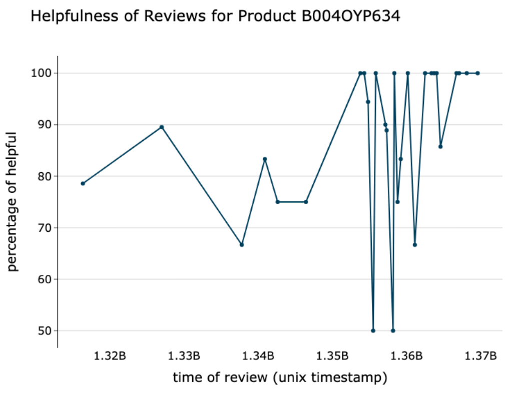


Figure 4: Helpfulness of Reviews for a Product with Frequent Later Reviews

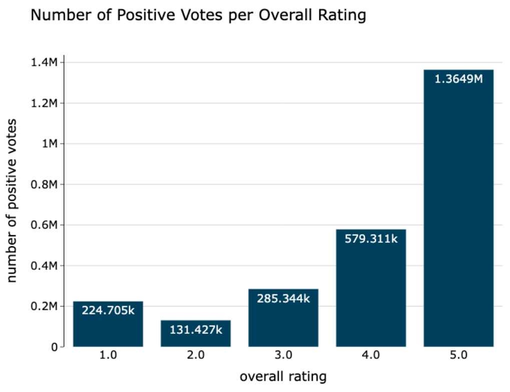


Figure 5: Number of Positive Votes per Overall Rating

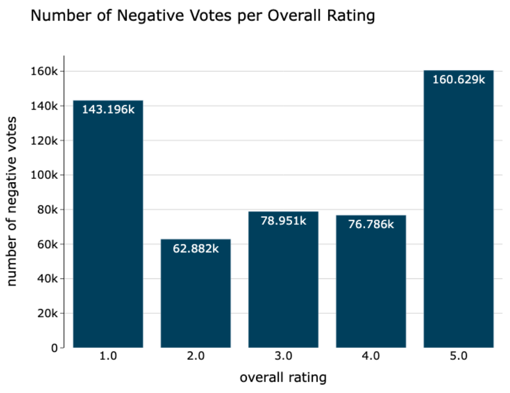


Figure 6: Number of Negative Votes per Overall Rating

3. Predictive Task

Predictive task: The predictive task that we decided on is to predict the helpfulness of a review. We classified helpfulness as the proportion of helpful reviews to overall reviews. This was done by using the helpful column that was already included in the data set. We decided on this predictive task as we hypothesized that the attributes that were included in the dataset would provide a good indication of whether a review is likely to be helpful or not.

Features + Processing: For this predictive task, the features we decided to use were ”reviewerID”, ”asin”, overall rating (”overall”), ”reviewText”, and ”date”. Our feature selection was based on what we believed would most influence the helpfulness of a review. We chose to use ”reviewerID” as the popularity of the reviewer can influence whether the review was helpful. Similarly, we used ”asin” because the item itself can influence how helpful the review was - if an item is more popular, more likely that the reviews for that item are more helpful. The use of overall rating is fairly straight forward. The use of ”reviewText” is necessary to understand the semantics of the review, which would be good indication of how helpful the review is. Finally, we used ”date” as a feature because it is more likely the more recent reviews are going to more helpful than later reviews. Here is a summary of how we processed the features that we selected.

- ”reviewerID” and ”asin”: For these feature, we decided to label encode them. By using a LabelEncoder, we are able to pass in these string values into our model. This way of transforming the data is similar to one-hot-encoding, however, it simpler, more space efficient, and has less space dimensionality. Since there are 59437 unique asin values and 164861, it was better to use this type of transformation for these features so our model does not crash from all additional columns that would have been added with a one-hot encoder.
- ”reviewText”: For text review, we performed two transformations on this feature before we passed it into the model. The first thing we did was transform the text using a TF-IDF. However, we found that leaving the transformation to just TF-IDF left our data to be too high dimensional. To address this problem, we performed decomposition on the vectorized text using TruncatedSVD in order to decrease the dimensionality. The number of components that we used was 500.
- ”overall”: We left overall rating as a numeric feature, passing it directly into the model as is.
- ”datetime”: We extracted this feature from the unixReviewTime by converting it into into DateTime object and passed that directly into the model without extracting any additonal features.

Baseline: We had two baseline models which were both using Linear Regression. The first baseline model had the features of ”overall” and ”reviewerID” (after it was label encoded). The mean squared error was 0.116. We chose these features as they could be a good indication of how helpful a review was if there was information about how reviewed that item as well as the the rating of the item itself was also included. The second baseline mode was similar, except this time the two features were ”overall” and ”unixReviewTime”. The mean squared error of this baseline model was 0.115. We chose to also use these features as a baseline as the timing of the reviewing, whether more recent or older, as well as the rating of the item could also be good indication of the helpfulness of the review.

Validity: We tested multiple models, looking for the one that would give us the best results. For all the models that we tried, we used mean squared error as our evaluation and comparison metric. The different models that we compared were Linear Regression and a XGBoost Regressor.

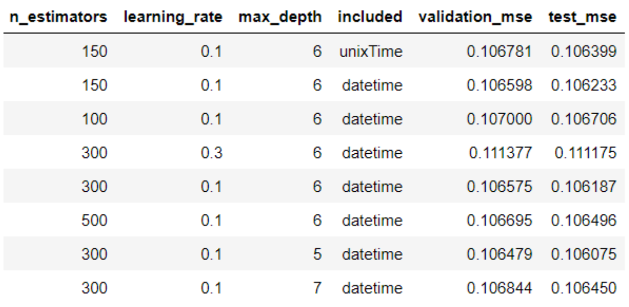
4. Models

Our predictive model utilizes XGBoost Regressor, a robust ensemble learning algorithm, to predict product review helpfulness. The choice of XGBoost was due to the several advantages of gradient boosting trees (GBTs). As GBTs are an ensemble method of decision trees, the XGBRegressor model allows it to capture non-linear patterns effectively, which can be crucial in understanding the sentiments expressed in text data. In addition, given the high variability in the number of reviews of each user and product in our dataset, this model was used because of its ability to handle overfitting and outliers.

We employed a two-step approach to preprocess the review text (“reviewText”). First, TfidfVectorizer was used to convert the raw text into numerical features, focusing on the top 5000 features. Subsequently, TruncatedSVD was applied to reduce dimensionality to 500 components, addressing the issues of high dimensionality. The remaining categorical data (“reveiwerID” and “asin”) were converted into numerical data using sklearn’s LabelEncoder. The resulting reduced features were combined with other relevant features, including overall rating (“overall”) and review timestamp (“unixReviewTime” or “datetime”).

Following this, the dataset was split into training, and validation testing sets (60% training, 20% validation, and 20% testing) to facilitate model evaluation. The XGBoost model was optimized using a predefined set of hyperparameters, including n_estimators, max_depth, and learning_rate. The choice of these hyperparameters aimed to balance between the model’s complexity and predictive accuracy. However, we encountered scalability issues during training due to the dataset’s large size (1,689,188 rows). To address this, we removed product reviews that did not have any votes resulting in a smaller but more manageable training dataset (362,980 rows). This trade-off enabled us to mitigate computational challenges while still preserving the integrity of the analysis. In addition, as we were predicting the helpfulness percentage of the review, the removal of zero-vote reviews allowed us to avoid NaN values, and having a percentage of 0% indicated the rating was not helpful at all rather than the helpfulness percentage not existing. Though the reduction in the size of the dataset allowed it to be more manageable when training the models, hyperparameter tuning the model using grid search proved to be difficult. Manual optimization was performed instead, adjusting one parameter of the model at a time and recording the validation and testing MSEs for comparison. See Figure 7 below for a complete breakdown of hyperparameter testing.

The XGBoost Regressor offers several advantages, one of which is its high predictive accuracy, attributed to its ensemble learning approach that combines the predictions of multiple decision trees. Regularization techniques embedded in XGBoost, such as L1 and L2 regularization, contribute to preventing overfitting, and the model’s ability to handle missing data reduces the need for extensive preprocessing. However, XGBoost comes with computational complexity, particularly in large datasets, and parameter tuning can be an iterative process, impacting its ease of use. The ensemble nature of XGBoost



n_estimators	learning_rate	max_depth	included	validation_mse	test_mse
150	0.1	6	unixTime	0.106781	0.106399
150	0.1	6	datetime	0.106598	0.106233
100	0.1	6	datetime	0.107000	0.106706
300	0.3	6	datetime	0.111377	0.111175
300	0.1	6	datetime	0.106575	0.106187
500	0.1	6	datetime	0.106695	0.106496
300	0.1	5	datetime	0.106479	0.106075
300	0.1	7	datetime	0.106844	0.106450

Figure 7: Hyperparameter Tuning for XGBoost Regressor

can also present challenges in terms of model interpretability compared to linear regression.

Alternatively, linear regression offers simplicity and interpretability as key advantages. The coefficients of each feature provide clear insight into the relationships between independent and dependent variables. The computational efficiency of linear regression makes it a suitable choice for tasks where resources are limited or datasets are large. However, the simplicity of linear regression can also be a limitation when dealing with complex, non-linear relationships in the data. The linear regression model assumes linearity and may struggle to capture intricate patterns, especially compared to the flexibility offered by ensemble methods like XGBoost. Linear regression is also sensitive to outliers, which can impact the model's coefficients and predictions greatly.

5. Literature Review

Numerous studies have delved into the prediction of review helpfulness, adopting approaches similar to ours by incorporating different features alongside the review text. Kim's research (Kim et al., 2006), for instance, analyzed various feature classes encompassing structural, lexical, syntactic, semantic, and metadata dimensions. Employing the SVM regression method, the study identified key features such as review length, unigrams, and product rating as the most influential. These findings align closely with the conclusions drawn from our model.

In another study conducted by Zeng (Zeng et al., 2020), the focus was on determining the impact of sentiment features on helpfulness. Employing gradient boosting and random forest methods for classification, the study evaluated performance metrics such as recall, precision, and F-measure. Similar to this study, our group used gradient boosting, however, we evaluated our model's performance based on a different metric; we used the mean squared error (MSE).

The two studies mentioned above developed models based on hand-crafted features which are features created by human intuition and knowledge. However, recent studies have utilized deep neural models for more precise predictions. Chen's study (Chen et al., 2018), for instance, employed a convolution neural network (CNN) model that leveraged both word-level and character-based representations. Their experimentation of this CNN-based approach outperforms existing models, including the one developed by our group. While studies relying on hand-crafted features have yielded conclusions similar to ours, recent research highlights the effectiveness of more advanced models that incorporate better features for making more accurate predictions about the helpfulness of a review.

6. Results and Conclusions

Mean Squared Error (MSE) was used as the evaluation metric and the final model, using XGBoost Regressor, tuned with parameters of n_estimators - 300, learning_rate - 0.1, max_depth - 5, and using “datetime” as opposed to “unixReviewTime” had an MSE of 0.10648 for the validation set and 0.10608 for the testing set. To provide context, we compared the XGBoost regression model against two baseline Linear Regression models (one using label-encoded ”reviewerID” and ”overall” and the second using ”unixReviewTime” and ”overall”) had MSEs of 0.116 and 0.115, respectively. An alternative model we employed was a Linear Regression model that contained the same features and preprocessing as the XGBoost Regressor, i.e. vectorized and truncated “reviewText”, label-encoded “reviewerID” and “asin”, and “overall”, “year”, and “month” as features, resulted in a validation MSE of 0.10790 and testing MSE of 0.10717.

An interesting feature we could potentially add is the rank of a review for a particular item based on its time. For example, once a customer lands on a product's page, older reviews may not be as helpful as newer reviews. Due to the constraint of a big dataset, this idea may be ambitious and unfeasible. Regardless, we were able to reduce the MSE from 0.116 in our baseline model to 0.10608 in our final XGBoost model with the provided features.

Note: While we achieved better results with the XGBoost Regressor, they were not too far off from the results that we were getting from the Linear Regression models (roughly 0.001 difference). Therefore, if the interpretability of the model was a priority, we would reevaluate our use of XGBoost Regressor.

References

Cen Chen, Yinfei Yang, Jun Zhou, Xiaolong Li, and Forrest Sheng Bao. Cross-domain review helpfulness prediction based on convolutional neural networks with auxiliary domain discriminators. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2:602–607, 2018.

R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *WWW*, 2016.

Soo-Min Kim, Patrick Pantel, Tim Chklovski, and Marco Pennacchiotti. Automatically assessing review helpfulness. *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 423–430, 2006.

J. McAuley, C. Targett, J. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. *SIGIR*, 2014.

Z. Zeng, Z. Zhou, and X. Mu. User review helpfulness assessment based on sentiment analysis. *The Electronic Library*, 38(2):337–351, 2020. doi: https://doi.org/10.1108/EL-08-2019-0200.