Predicting Amazon Review Helpfulness Ratio

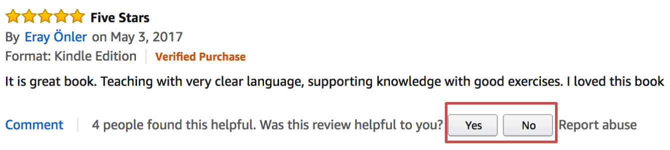
Kyle M. Shannon
Department of Computer Science & Engineering
University of California, San Diego
kshannon@ucsd.edu
June, 12, 2017

***Abstract*— Amazon, one of the largest online retailers with millions of products and an active online community, seeks to provide customers with the most helpful reviews for a product of interest. Reviews can be voted on by online shoppers as either *helpful* or *not helpful*. Most reviews have never been voted on and the reviews that have received vote(s), most votes tend to be for *helpful*. I present a model for predicting the ratio of positive votes out of total votes a review receives. Due to the heavy-tailed reality of Amazon review votes, my model partitions the data into separate areas, specifically the tail, body and the area where the number of total votes are 0 or 1. My model was evaluated using mean absolute error and on unseen test data performs adequately with a score of 0.1753, compared to a baseline model, which scored 0.261. The baseline model's sole predictor variable was the unique user's average helpfulness ratio. Lastly, I discuss the challenges of performing this machine learning task and future work using additional natural language processing techniques.**

I. Introduction

Predicting the helpfulness of an online product review serves both the customer and online retailer. The customer is provided with the most helpful and relevant reviews about a product of interest without having to mine for good reviews. Additionally, online retailers benefit from customers on the fence about purchasing an item. Due to the competitive nature of online retail, where users have at their fingertips an enormous amount of options, businesses like Amazon benefit from providing customers with good quality information about the product, as well as reviews from other customers who have purchased the same product. When a product has only a few reviews, it is easy enough for a customer to read them all; however, when faced by 1000s of reviews a customer is likely not to have the time nor patience to filter for good content.

To alleviate this issue, Amazon introduced a simple voting mechanism whereby customers can either vote a review as helpful or unhelpful.[1] Each customer gets one vote for each unique review on Amazon, as shown in Fig. 1.

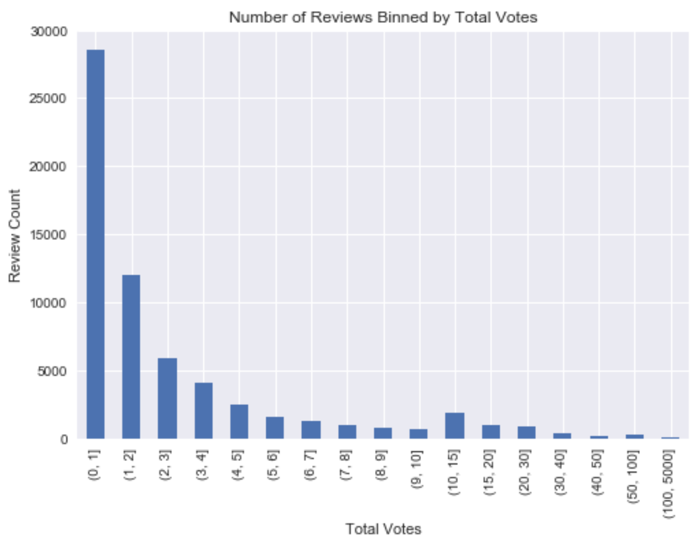

★★★★★ Five Stars
By Eray Önler on May 3, 2017
Format: Kindle Edition | Verified Purchase
It is great book. Teaching with very clear language, supporting knowledge with good exercises. I loved this book
Comment | 4 people found this helpful. Was this review helpful to you? Yes No Report abuse

1. Amazon review, showing the option to vote helpful or unhelpful

Two interesting characteristics of Amazon reviews are that most reviews simply do not have any votes. Most customers either never read most reviews and therefore do not vote on them, or perhaps they just read the reviews with the largest number of votes and accordingly cast a vote to that review. This creates a landscape where many products have a vast amount of zero vote reviews and a large skew of voted reviews. Additionally, reviews with at least one or more vote are for the most part voted helpful. Perhaps this phenomena occurs because reviews tend to be on average more helpful than not. Alternatively, customers may simply ignore down voting poor reviews and favor up voting good reviews. The underlying pattern of this phenomena is what I am trying to model and predict.

II. Exploratory Data Analysis

The training data set consisted of 200,000 observations and 12 features. The test set has 14,000 observations and will be used as a final holdout set to evaluate the model. Approximately 68% of the training data consisted of reviews which had 0 votes. Fig. 2 shows the heavy tail distribution of the data set.



2. Binned Amazon reviews by number of total votes

There are two types of data represented in these sets, (i) Metadata which consists of a review's userID, itemID, rating in stars 1-5, and so on. (ii) There is also review data, which is the actual text of the review, written by the user. These two avenues provide various ways to look at the data. There is the high level approach where you look at the overall review data and try to find patterns concerning the metadata. For example do reviews with more votes tend to have 5 or 1 stars? Additionally, the text of the review provides insight into the makeup of the actual review. What type of reviews are typically voted highly? Reviews with long text bodies, or maybe reviews with the words 'pro' and 'con'. The following tables provide more high level findings.

Value Count of Ratings

Star	***Count***
5 Stars	116034
4 Stars	42136
3 Stars	22343
2 Stars	11488
1 Star	7999

a. Ratings are given on a 1-5 star scale. Where 5 star is the best rating

Value Count of Category Type

Category	***Count***
0 - e.g. woman's clothing	141410
1	51398
4	2938
2	2346
3	1908

a. Categories were encoded into one of 5 bins, this was done before I received the data

The only feature to have missing data was 'price'. Approximately 63% of that data was missing, so I decided to not use this feature or impute the missing data. There were 39,249 unique users and 19,913 unique items in the training data set. I performed more data analysis by splitting the data set by reviews with at least one vote and reviews with no votes. There were approximately 68,000 reviews with at least one vote. This smaller subset of data was the data I used to build features and train/validate models. As I went through my analysis some of the guiding questions I choose to explore included:

- Are there power/poor users, e.g. users whose reviews are always highly voted as helpful or unhelpful?
- Do reviews have seasonality? Are reviews more likely to be voted if they were written on weekends or during holidays?
- Is there a delta time component to a unique item and when it was first reviewed?
- What is the best place to split the body and tail distribution?
- Can I quantify the *quality* a review? Readability?
- Is a product's rating indicative of its review?
- Can an analysis of the data's top eigenvectors from PCA provide any insight?

III. Data Cleaning and Shaping

I did not spend too much time detecting and dealing with outliers because the data set provided was already fairly clean. Each review had a hash which was set as an index for auditing purposes. Categorical data was *one hot encoded* to the original column was removed. All text data was extracted and saved onto disk in JSON format with the review hash as the key and text as value. Text data comprised of almost the entire size (~100 Mbs). The text was processed later during the feature engineering phase, discussed later. The column 'helpful' was a dictionary of total votes ('outOf') and the number of helpful votes received ('nHelpful'). The data was portioned into 4 separate pieces based on the value of the 'outOf' column. The following mapping was used:

{'outOF' : 0, 'outOF' : 1, 'outOF' : 2-10, 'outOF' : 11-n}

The purpose for this split was to the break the data into partitions, create features for each partition and also train models. Lastly, I transformed the response variable from being the number of helpful votes received to predicting the ratio of helpful. For the final submission, I transformed the predicted ratio back into number of reviews by multiplying the 'outOF' integer throughout.

IV. Data Pipeline

The data outline I built to preprocess the data, build features, train evaluate and validate models, and finally build a test prediction .txt file is outlined as follows:

- Created three separate python modules to import into jupyter notebooks
    - *shape.py* - data cleaning and shaping
    - *engineer.py* - feature engineering functions
    - *invest.py* - bringing raw .json.gz data files into jupyter notebook and pickling/ loading pickle files
- Created six different jupyter notebooks
    - *EDA* - exploratory analysis, very unstructured
    - *Report* - along the lines of a lab journal with findings and questions of interest to tackle
    - *outOf_1* - notebook to work with just data where the counts of review = 1
    - *outOf_2_10* - notebook for outOf partition
    - *outOf_11_n* - notebook for outOf partition
    - *Build_Kaggle_Submission* - notebook which takes in pickled models and data and builds a .txt file to submit for final testing

The outOf notebooks were responsible for pulling in a partition of data, calling functions from the three modules I wrote, and finally choosing, validating and tuning a model to export for use in the Build notebook.

V. Feature Engineering and selection

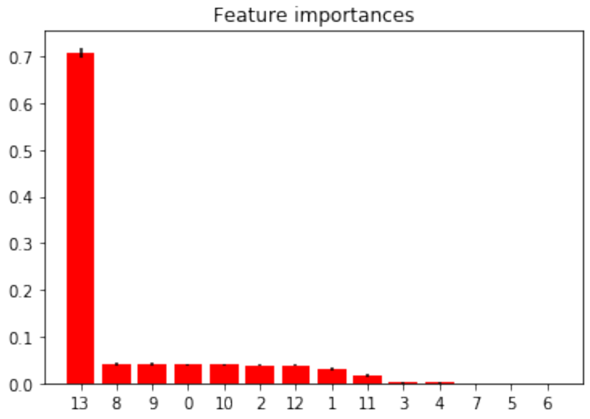
Feature Engineering and selection is one of the most critical steps in the machine learning process.[2,3] The model's performance is heavily dependent upon the quality of data provided. I engineered several new features that I believed would help tease out the important structure and patterns within the data. From the review text data some useful features I found were:

{ 'review_length', 'readability_index', 'flesch_readability', 'sentiment_polarity', 'sentiment_subjectivity' , 'count_caps'}

Other engineered features pertaining to the meta data include:

{ 'negative_rating', 'positive_rating', 'item_avg_rating', 'item_rating_delta', 'review_time_delta' }

I created several more features, but determined from correlation studies that there was either multicollinearity issues, or the feature simply did not delineate with the response variable very much: $corr(feature, response) < 0.10$. Looking at the kurtosis and skew of each individual feature provided insight into which features I decided to try log transforming. Finally, I performed two feature selection methods (i) an ablation study for my regression models to determine the features that had the least or greatest impact on the scoring criteria, *MAE*. (ii) for the classifier model I built a Extra Trees Classifier to determine the most important features. By far the most important feature was {13: 'user_helpfulness'}



This feature does make intuitive sense that it is the most important, however this feature will also not be the best feature for a model that generalizes. This is because users that are not in the training set, who are in the test set, their helpfulness score will be the imputed global average. There were about 50 such unique users to the test set.

VI. Model Selection and Tuning

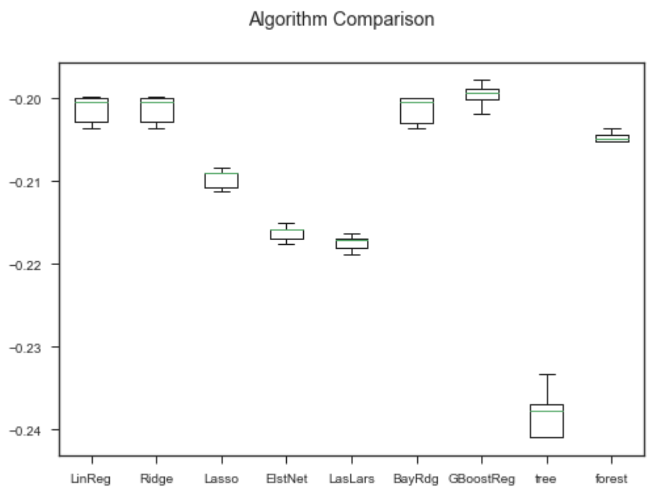
I created three distinct models for three of the data set partitions.

A. *Binary Classifier for 'outOf' = 1*

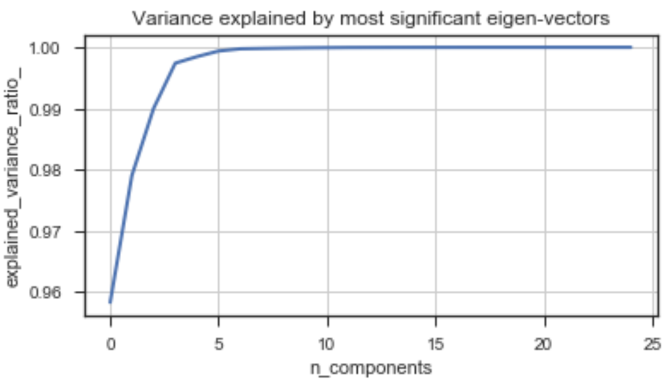
Both stock logistic regression [4] (LR) and decision trees [5] (DTs) models performed well on this type of problem. I decided on LR and used K-fold cross validation [6] (CV) to evaluate and tune the model. The parameters used were: {C = 0.1, penalty = 'l2', max_iter = 100}. The results from the CV were an MAE of 0.0928 with a standard deviation = 0.00132.

B. *Gradient Boosting Regressor for 'outOf' = 2-10*

Model selection for this partition of the data was chosen by using cross validation to test several stock models:



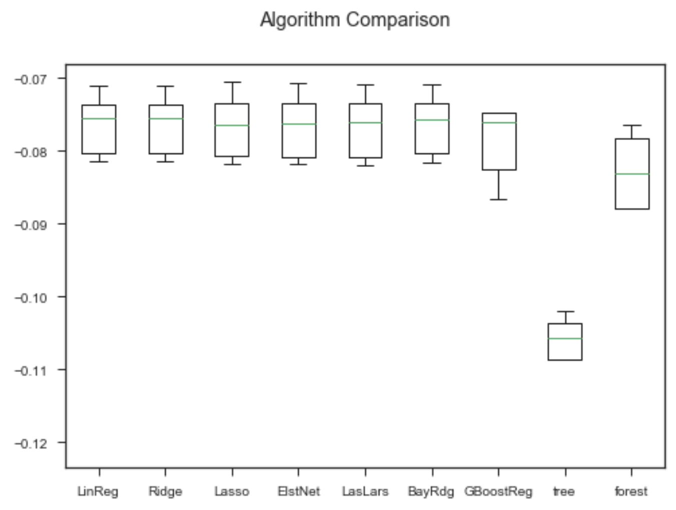
Stock Gradient Boosting Regressor[7] performed best during model selection with an MAE of 0.1995 and a standard deviation of 0.00138. I performed a PCA analysis[8] to reduce the feature set into an embedded subspace of principal components.



The first six eigenvectors capture nearly 100% of the variance explained. And looking at the first eigenvector, almost 97% of the variance can be explained. I suspect that this much of the variance being captured in the first eigenvector will lead my model to perhaps overfitting if some of the information captured by my features are not the same generalized patterns found in the test set, but perhaps are noisy patterns inherent to the training set. Final model tuning was performed using K-fold cross validation with sklearn's GridSearchCV class. Final parameters selected were: {n_estimators=400, loss='ls', learning_rate= 0.1, max_depth=4, min_samples_split=3, min_samples_leaf=3}

C. *Elastic Net Regression for 'outOf' = 11-n*

Following the same search criteria as the previous model, Elastic Net[9] performed the best when comparing stock models. With an MAE of 0.07661 and a standard deviation of 0.00427. Elastic Net was ultimately chosen because some of my data had multicollinearity and ElasticNet's additional L2 term acts as a stabilizer by introducing convexity to help with this issue[7].



Final model tuning was performed as with the other models, using K-fold CV. The selected parameters were: {alpha=0.1, l1_ratio=0.01, max_iter=1000}.

VII. Validating and Testing

During each model training phase, I took the partitioned data set and split it up into 80% training and a 20% holdout set to test the final tuned model. Using sklearn's[10] TestTrainSplit class I divided the data, ensuring that a small number of unique users in the holdout set were not in the training set. The purpose being to build a good proxy of the real holdout test data set for local testing. The training set was then used in CV studies to select and tune the best model. After the final model was selected, I refit the model on the entire partition's dataset. It is important to note that my models only made predictions where the observation had an outOf value greater than 0. If the outOf value was 0, I simply predicted 0 as a review with no votes can not have any positive or negative votes. All of the scoring criteria used was the Mean Absolute Error rate[11], given as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i | = \frac{1}{n}\sum_{i=1}^{n} | e_i |$$

Where

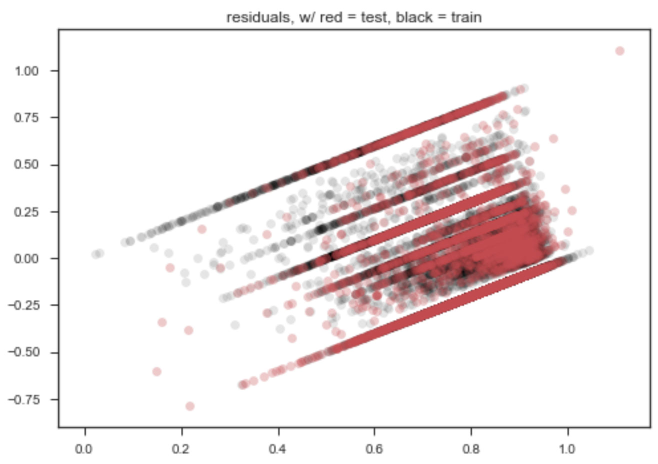
$$AE = |e_i| = |y_i - \hat{y}_i|$$

$$Actual = y_i$$

$$Predicted = \hat{y}_i$$

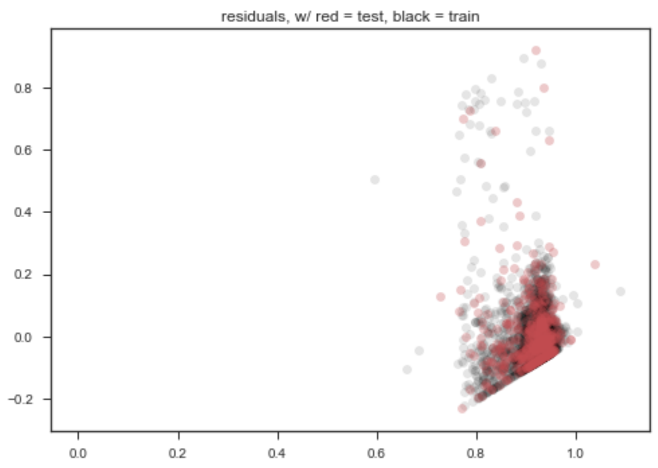
The test data, comprised of 14,000 observations were split into two holdout groups. One public where the final models could be evaluated against, and one private group where models would be evaluated against only once after the competition end date. On the public test set, my combined models performed with an MAE score of 0.16957, and my final private board performance was 0.17529. The difference being approximately 0.0057. This was a good indicator that my model was not overfitting very much and had the ability, on this test dataset, to generalize fairly well.

A look at the residuals vs predicted values plots from both regression models shows some interesting findings. The first plot from outOf = 2-10 shows residuals that are fairly randomized and not correlated; however, there are these long streaks residuals are sub areas where the dependent variable is completely linearly dependent on the predictors or perhaps the dependent variable is a constant [12]. A QQ-plot could help to examine the issue further by looking at the normalcy of the data. These streaks may indicate that the data is non normal, if this is true then perhaps some of the models chosen may not be the best and a polynomial interaction term or squaring a feature



such as the item's rating might have helped with a model like Random Forest Regressor.

Examining the residual plot from the outOf model = 11-n shows that there potentially is slight heteroskasdicity. That is the variance of the dependent variable are not uniform across the values of the predictor variables that predict the dependent



variable. The straight cut line at the bottom is due to the count integer type of data, and the fact that a count cannot be less than 0.

VIII. Discussion and Future Work

One issue I had with my model is the overall performance, I believe that trimming more of the features away either through additional ablation studies, 'add one'/'take one out' [7] analysis, or attempting more preprocessing steps e.g. MinMaxScaler, Normalization, Standardization, might have benefitted performance.

Most of the features that I used and which were helpful seemed to come from the review's meta data. This is an interesting find, because when a person envisions a model of what it means to predict a ratio of a helpful review, that person might reasonably guess that it is the review's text which is most important. But with the limited NLP performed, most of the predictive power came from the previous helpfulness of a user, which is certainly relevant, and the meta data of the review. Which seems less important. I suppose the model presented here is more of a model of the underlying patterns in the metadata versus a good presentation of the quality of a review given how many votes it has.

Moving forward, heavy NLP of the text data might be more useful for building such a model, it may also bring the model closer to what the true prediction is hoping to capture.

References

1. http://www.techtimes.com/articles/62330/20150620/amazon-new-customer-reviews-system-more-helpful-gain-trust-heres.htm
2. Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78-87.
3. Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, *97*(1), 245-271.
4. Cramer, J. S. (2002). The origins of logistic regression.
5. Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106.
6. Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, No. 2, pp. 1137-1145).
7. T. Hastie, R. Tibshirani and J. Friedman. Elements of Statistical Learning Ed. 2, Springer, 2009.
8. Smith, L. I. (2002). A tutorial on principal components analysis. *Cornell University, USA*, *51*(52), 65.
9. https://web.stanford.edu/~hastie/TALKS/enet_talk.pdf
10. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
11. Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research*, *30*(1), 79-82.
12. Searle, S. R. (1988). Parallel lines in residual plots. *The American Statistician*, *42*(3), 211-211.