Amazon Reviews - Prediction of Rating and Helpfulness (an NLP Use Case)

Raghavendra Prasad Savada

Table of Contents

Introduction

Reviews written by the customers who have already purchased the products help other customers make informed decision about their purchases. In fact, these reviews are perceived to be more reliable than paid brand endorsements. Additionally, these reviews provide wealth of information that can be used to derive actionable insights.

Since 1995, Amazon have encouraged its customers to write reviews on its platform about the products they purchased. In this period, they have gathered millions of reviews about thousands of products. Amazon made this dataset of over 150 million reviews made publicly available. This dataset with a huge corpus of text data is considered to be a treasure trove by those who are involved in Data Science and Machine Learning research. This dataset is particularly useful in the area of Natural Language Processing.

Amazon dataset along with its review text has a few interesting features. Each review is associated with a star rating on the scale of 1 to 5 as rated by reviewers. When other customers read a review, Amazon allows them to vote the review as either helpful or not helpful in making their purchase decision. The dataset also has information about number of total votes and helpful votes each review received.
Here, I have used this dataset to train models that can predict sentiment expressed in a review as positive, negative or neural, and helpfulness of a review using the review text as a predictor. This model has a few applications.

Model Applications

1. The very purpose of any machine learning prediction is to be able to predict the labels of future examples which do not come with labels. However, in the case of Amazon reviews, even the future data comes with a label. Because, even in the future, when a customer writes a review on the Amazon website, they label it in the form of star rating. Despite this fact, I see at least two different applications for this model.

    a. We should be able to use this model to predict the sentiment of the unlabeled reviews outside of Amazon platform. For instance, opinion expressed on various other platforms like twitter, blogger etc. This model can also be used to predict sentiments in reviews or opinions about non-Amazon products.
    b. On Amazon platform, when customer write a review, occasionally their ratings do not match the opinion expressed. For instance, they have expressed a highly positive opinion, but by mistake chose one star as rating. The above model can be used to spot such discrepancies. Then Amazon can automatically send an email to the customer pointing out this discrepancy and asking if the customer is willing to update their star rating to reflect the opinion expressed in the review.

2. The model that predicts helpfulness of a review has the following application. One of the metric Amazon uses to order the reviews is the number of helpful votes each review receives. Then most helpful reviews are displayed at the top of their product page. However, the rate at which helpful votes accumulate is generally lower than the rate at which reviews are written. So, during the initial period of product release, Amazon may not have reliable metric to order their reviews. It should also be noted that more than 75% of the reviews on Amazon do not receive any votes (see below). So, the model I am going to build can be used to predict helpfulness of reviews so that Amazon can utilize this predicted score to order the reviews during a period when they have reviews that are not voted yet.

Data Gathering

Dataset

Amazon review dataset with 150,962,278 reviews is spread across 46 tsv files corresponding to 43 product categories. This dataset is made available in amazon S3 bucket *amazon-reviews-pds*. Also, there are 5 additional multilingual datasets, which I am not going to use in this project. The dataset has the following features.

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 customer.
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: Rating on the scale of 1 to 5.
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.

All the features in the dataset are self-explanatory except vine. Y in vine column indicates that review is written as a part of Amazon vine program. In this program Amazon invites the most trusted reviewers on Amazon to post opinions about new and pre-release items to help other customers to make informed purchase decisions (https://www.amazon.com/gp/vine/help).

Missing Values

There are 13 rows which do not have the primary key (review_id). These rows have null in all the columns. This is probably due to addition of blank lines at the end of some tsv files. 18,788 rows have null in review_body column. Since there is no way to impute missing review_body, these rows will be deleted. Similarly, ~2,000 rows have null values in column like star_rating, helpful_votes, total_votes, review_headline. Since we have more than 100 million data points, deleting these rows would be an appropriate choice of handling missing values.

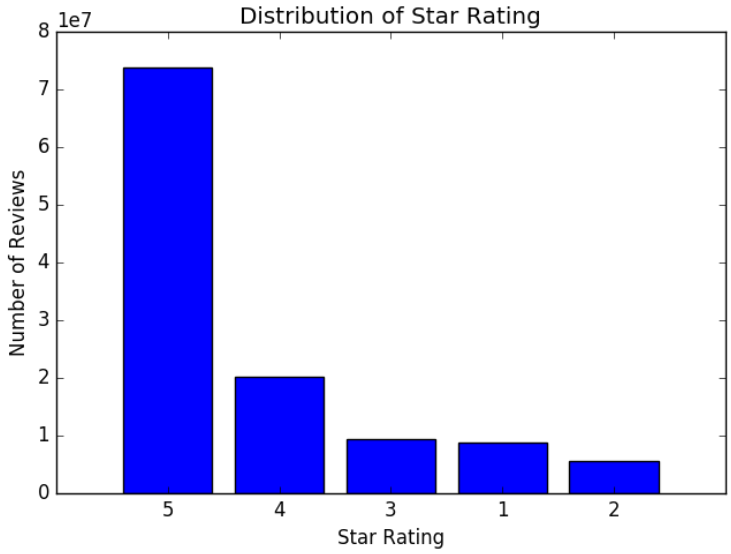
Dataset with Verified Purchase Only

On Amazon platform anyone can leave a review regardless of whether purchased a product or not. More than 22% of the reviews have been written by customers who did not have a verified purchase. One of the major problems in online review platforms like Amazon is fake reviews; positive reviews written by vendors to promote themselves or negative reviews by competitors. The probability of a review being fake significantly increases when it is written by someone without a verified purchase. Therefore, I have decided to include only reviews labelled as verified purchase for further analysis. After removing missing values and reviews from the customers without a verified purchase, there was 117,698,232 reviews.

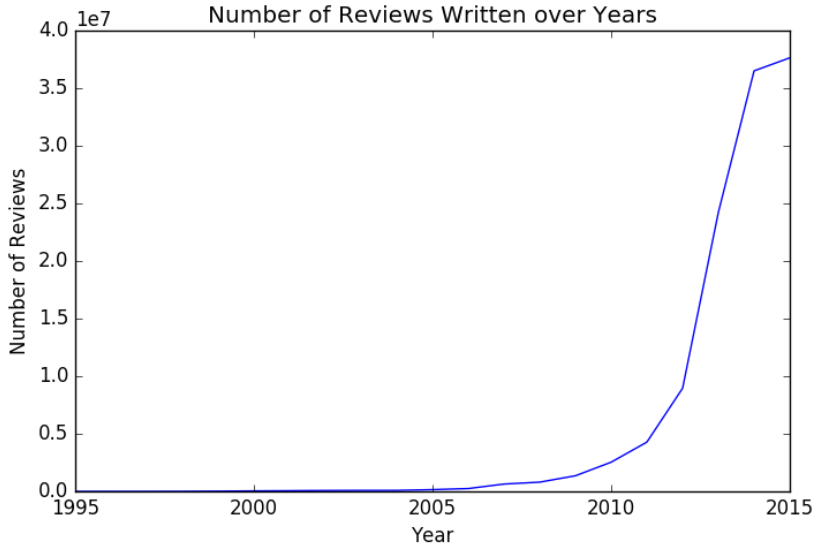
Exploratory Data Analysis

The following are the observations made from Exploratory Data Analysis. You can find all the charts in the notebook *exploratory_data_analysis.ipynb*.

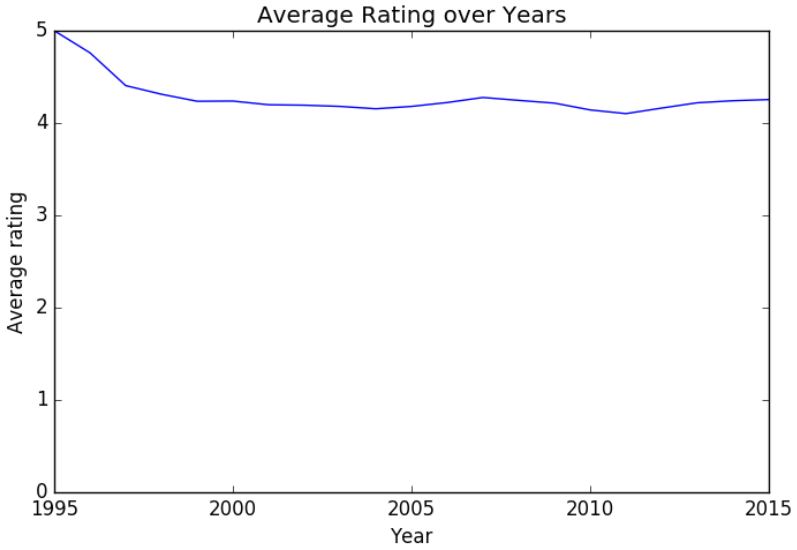
1. Over the years Amazon products have an average rating of 4.2.
2. More than 62% of the reviews on Amazon has a rating of 5. Only 12% of the reviews has a star rating of 1 or 2.



3. The product category *Digital E-books* has the highest number of reviews.
4. Since around year 2010, the number of reviews increased exponentially. This may be due to increase in smart phone usage and social media growth.

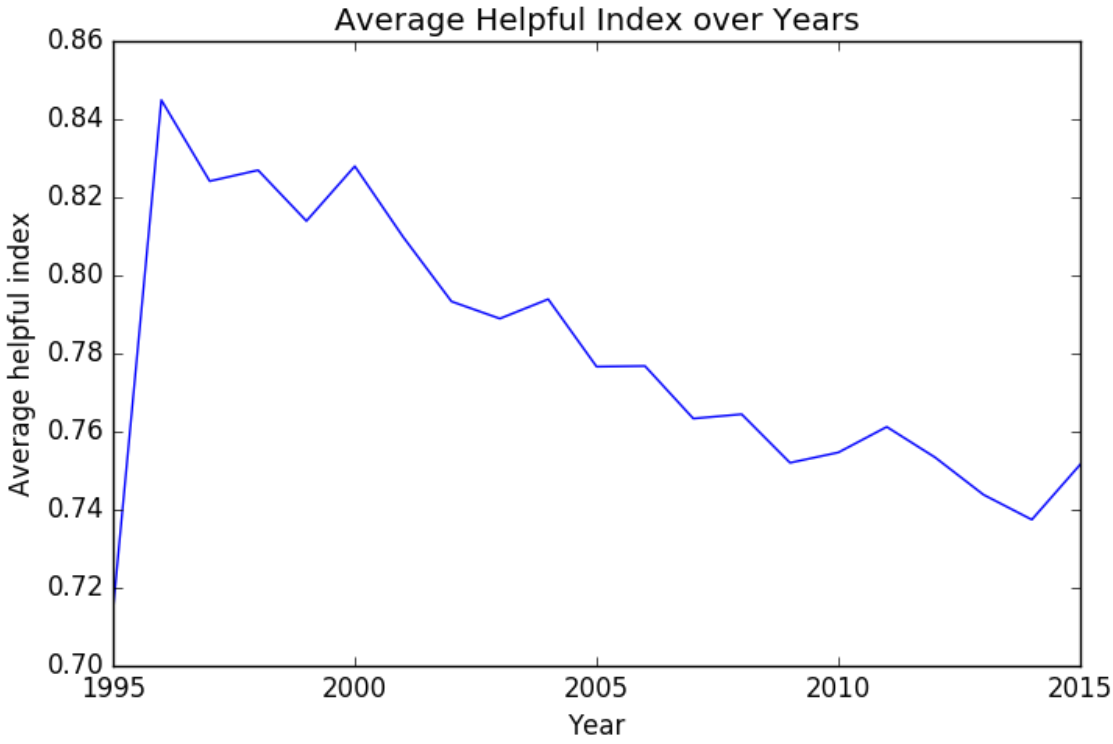


5. The product category *Gift Card* has the highest rating. This may be due to the fact that there is less chances that something goes wrong with gift cards.
6. Average rating stays almost the same (~4.2) over years.



7. The product with the most reviews is *Candy Crush Saga*.
8. It is interesting that a customer with ID 35178127 has written more than 2000 reviews.
9. There is a review that has got as many as 48,362 total votes out of which 47,524 were voted as helpful. This is a review about Kindle, which seems to be written right after the launch of this product.
10. The reviews written by a customer with ID 13376158 has received more than 50,000 helpful votes.
11. Reviews with star rating of 5 are perceived to be more helpful than reviews with other ratings; it has the highest average helpful votes as well as the highest helpful index.

12. The quality of Amazon reviews seems to be decreasing over time as indicated by decreasing trend in helpful index. Ease with which you can write a review over past few years seems to have affected the quality of reviews



ML Prototyping

For the prototyping, I used 0.5% data of Amazon review dataset, which is still around ~600,000 reviews. A model that predicts whether a review is positive, negative or neutral has been trained. Star rating of 5 and 4 will be considered as positive, 3 as neutral and 1 and 2 as negative. Furthermore, a model that predicts how helpful a review is also trained.
The main feature for my model building is review text. Apart from alphanumeric characters, the review text contains all sorts of other characters like emojis, words of other language, html tags, special character etc. So first I preprocessed the text.

Text Preprocessing

The following cleaning is performed on review text.
- lower casing of the text
- Stripping html tags
- stripping punctuation
- stripping multiple white spaces
- stripping numbers

Stopwords

Stop words are generally the most common words in a language, presence or absence of which does not make much difference to the meaning. In Natural Language Processing tasks, usually stopwords are removed to reduce dimensionality. I debated whether to remove stopwords or not. In some use cases like sentiment analysis, it is not a good idea to remove stopwords. The list of stopwords in most of the python libraries have *not* as a stop word. But removal of this word can completely change the meaning of a sentence. For instance, *I do not recommend this product* becomes *I do recommend this product* after removal of the stopword *not*. Given this unwanted side-effect, I decided not to remove stopwords. Another technique used to reduce dimensionality of text data is to lemmatize the word, which essentially extract the root word of word inflections (e.g., studying, studied, studies all become study after lemmatization). However, sometimes lemmatization could also affect the contextual meaning of a sentence. Therefore, I decided against lemmatization. But when I iterate the machine learning process, I may check if lemmatization improves performance.

Sentiment Analysis

As mentioned above one of the issues on Amazon review platform is mislabeling of the data, where start rating is not consistent with sentiment expressed in the review. The Table 1 below provides some examples where the sentiment expressed in the review is positive, but the reviewers wrongly rated the product as 1.

Table 1. Samples of Mislabeled reviews (sentiment expressed is positive, but rating is negative).

Review Text	Star Rating	TextBlob Polarity
All these items I purchased are excellent.	1	1
This was just perfect for me because I needed it for a compact car I just purchased. It holds any quantity and doesn't spill. Packs away discreetly and even has eye appeal with the color and shape.	1	1
Very good for what I needed have told others.	1	0.91
It was perfect.	1	1
a very good item	1	0.91
JUST WHAT I WANTED. EXCELLENT	1	1

I explored the possibility of identifying and removing the mislabeled reviews. I used a sentiment analysis library called TextBlob to extract sentiment expressed in each review. TextBlob is an open-source library for sentiment analysis. The polarity score given by TextBlob is a float within

the range of -1.0 to 1.0. We can consider polarity of less than 0 as negative, more than 0 as positive and 0 as neutral sentiment.
Almost in 23% of the cases, sentiment expressed (as determined by TextBlob) in the review does not match with star rating. Some of these cases could be where rating is positive or negative, but review is so short or subjective that its sentiment is determined to be neutral. In ~5% cases, the sentiment expressed is positive, but the rating is negative.

I originally thought of removing all reviews where rating does not match with sentiment (as determined by TextBlob) expressed in the review. However, I found it necessary to further investigate as to whether there is a real discrepancy, or this is just TextBlob failing to identify sentiment correctly. So, I pulled out some reviews where sentiment and rating did not match. Interestingly, while in some cases indeed there was a discrepancy, in other cases TextBlox sentiment analysis was wrong; some of these examples are shown in Table 2. So, I decided against removing any reviews just based on TextBlob analysis.

Table 2. Examples of reviews, where TextBlob failed to identify correct sentiment

Review Text	Star Rating	TextBlob Polarity
I didn't like it and don't recommend anyone to buy. I just bought a refurbished version, and according to its description it should be in VERY GOOD condition. But it's in very FAIR condition. Unacceptable!!! I will attach some pics of it...	1	0.91
It didn't work at all. Went and bought one from Apple and it worked perfectly.	1	1
This can opener is of dollar store quality at best. I would NOT recommend this product to anyone!	1	1
It's just a piece of felt with the plastic sleeves stuck in it. I could have made this myself. Not the best quality.	1	1
Not too impressed with this product. I've used the bag once and the decor on the handle is coming off.	1	1
I ordered a ink cartridge for my canon printer. It worked very good the 1st print. and after that 1st print the ink was faded for a couple print and than no longer printed the ink.	1	0.91

Feature Engineering

As machine learning algorithms only work with structured data, we need to convert text data to a numeric representation. One common technique used to convert text data to a numeric form is Bag of words models. This method represents each text document as a numeric vector with each

dimension being a specific word from the corpus and the value is either frequency in the document or just the presence or absence of that word in the document (denoted by 1 or 0). This method could be further improved by down-weighting most frequently occurring words in the corpus that overshadow other words as done in the case of TF-IDF (Term Frequency-Inverse Document Frequency). Nevertheless, these bag of words models do not account for order of words. For instance, they cannot distinguish between the sentences like *eat to live* and *live to eat.* To overcome this problem, now there are models available that capture context-based information in the numeric vector. Word2vec is one such model wherein each word is represented as a vector of probabilities of this word occurring in different contexts. These probabilities are derived by training a large word corpus like *Wikipedia* using state-of-the-art deep learning methods. I used *Word2Vec* transformer in pyspark, which is based on the 'Skip-Gram' approach, to extract features from review text. The Word2VecModel transforms each document into a vector using the average of all words in the document (https://spark.apache.org/docs/latest/ml-features.html#word2vec).

Rating Prediction

I started with simple model like logistic regression to predict rating, wherein rating is categorized either positive, negative or neutral. Although it produced an accuracy of 85%, it did not do a good job in recalling negative and neutral class. It could recall only 55% and 7% of negative and neutral class, respectively. I tuned some of the hyperparameters. However, cross-validation picked default parameters, so I did not get any significant performance improvement. I tried a couple of more sophisticate algorithms like random forest and Multi Layer Perceptron (MLP). Random forest builds an ensemble of decision trees, where each tree has a very low bias but high variance. Since the final prediction is based on linear combination of individual trees, the variance averages out. So, random forest is expected to improve performance. Deep learning-based models like MLP is expected to perform better as they train deep neural networks that are capable of identifying complex non-linear patterns. It is worth noting that in terms of machine learning, PySpark is still in its early stage and we do not have a lot of options. For instance, its Gradient Boosting Tree algorithm works only for binary classification. I was interested to try Boosting, which generally produces better result than random forest although not guaranteed. Because, boosting sequentially converts weak learners into strong learners by correcting the classification errors of previous weak learner in the sequence.

Neither random forest nor MLP improved the performance significantly. One of the problems with this dataset is class imbalance. The class distribution is as follows.

- Positive - 80%
- Negative - 12%
- Neutral - 8%

Positive class is way more than negative and neutral class. This negatively affect model performance in correctly predicting rare class to the extent that classifier like random forest

simply predicted every sample to be positive. To overcome this, in PySpark, in the case of logistic regression we have a technique called “Class Weighing”, wherein class weight is set to be inversely proportional to its frequency. However, for random forest we do not have such class weighing parameter in PySpark. So, for random forest, I have down sampled the positive class (only 16% of positive class data is used). While up sampling of rare class is also possible, I opted for down sampling as the dataset is huge with more than 100 million rows, so down sampling should not affect training much.

After dealing with class imbalance, logistic regression performed much better in predicting rare classes like negative and neutral reviews. The following is the performance metrics of this model:

- Accuracy: 73.3%
- Positive class recall: 75.2%
- Negative class recall: 70.7%
- Neutral class recall: 58.0%

Although MLP produced better recall with positive (82%) and negative classes (80%) after class balancing, its overall accuracy (69%) is less than that of Logistic Regression. To train 0.5% of the data, Logistic Regression took just ~3 minutes, while MLP took ~49 minutes. Random forest improved its performance of predicting negative class after balancing the classes, but it still failed to identify any of the neutral reviews. Considering all these factors, I decided to use Logistic Regression to scale the ML model.

Prediction of Helpfulness

Helpfulness index

Analyzing the helpful votes was little tricky. Absolute number of helpful votes is not suitable for comparison because of varying number of total votes among different reviews. For instance, a review with 15 helpful votes out of 10,000 total votes must be less helpful than a review with 10 helpful votes out of 11 total votes. To deal with this issue, a new helpful_index feature has been created by dividing helpful_votes by total_votes.

Subset Review with More Than Five Total Votes

Another challenge is that most of the reviews have 0 total votes. Furthermore, very low total votes may bias the analysis. For instance, if there is only one total vote and if it is voted as helpful, it works out to a helpful_index of 1, which may or may not be reliable. To circumvent this problem, only reviews with more than 5 total votes are taken into consideration for this analysis.

I used two algorithms - Linear Regression and Gradient Boosting Trees - for predicting helpfulness, which is a regression problem. They produced an R^2 of 20.6% and 22.4%, respectively (they could explain around 20% of variance present in helpful index). This is not a great performance. It looks like predicting helpfulness is a hard problem. Unlike rating prediction, where sentiment expressed in the review is a very good predictor of rating, in helpfulness prediction, the sentiment may not be that useful as both positive and negative reviews could be helpful.

Model Scaling

The prototyped logistic regression and linear regression models have been scaled using 15 times the amount of data used (~9000000 reviews) for prototyping. It is interesting that addition of more data did not improve model performance. The accuracy of logistic regression was improved only by a mere 0.2%. In fact, R^2 of linear regression model was decreased to 17.5% from 20.6%. It should be noted that increasing the number of training examples can increase performance only up to a certain point. Then performance plateaus and may even decrease since we may simply add more noise. So, for these models, number of training examples does not seem to be a limiting factor. We have to look into other aspects of the models (please see *Further Improvement* section).

Model Deployment

The trained logistic and linear regression models have been saved. APIs to serve these models have been created. The APIs can be hosted on services like AWS. Users can supply a list of reviews and get the prediction for sentiment expressed in these reviews as positive, negative or neutral, and helpfulness index.

Further Improvement

In this section, I have discussed a few possibilities as to how these models could be further improved.

1. Owing to limited resources, I could not tune many hyper-parameters. We can tune more hyper-parameters to see if it improves performance. For instance, we can try deeper or more trees in random forest, or more hidden layers and nodes in MLP.

2. Although, Word2vec capture some contextual information, it still outputs only one vector per word. But some words have different contextual meaning. For instance, the word *cell* can have entirely different meaning in the context of *blood cell, prison cell,* and *cell phone.* The recent state-of-the-art contextual embeddings like ELMo, BERT or ULMFiT can capture these contextual meaning effectively. So next logical step would be to try some of these models.

3. For rating classification and helpfulness prediction, we can try some of the more sophisticated deep learning methods like Convolutional Neural Network (CNN), RNN (Recurrent Neural Network), LSTM (Long Short Term Memory), GRU (Gate Recurrent Unit), and Bidirectional GRU.

4. For extracting features to predict helpfulness, we can try text analysis software like Linguistic Inquiry and Word Count (LIWC), which analyze text and produces numeric representation of various psychological and structural components of the text (http://liwc.wpengine.com/).

Conclusion

Using the Amazon Product Review dataset, I have trained models that can predict sentiment expressed in a review, and helpfulness of the review in others making informed purchase decision. The logistic regression model that predicts the sentiment is 73.5% accurate. This model can be used to predict sentiment expressed in the un-rated reviews on other platforms. This is also useful for Amazon to identify reviews that are mis-rated on its platform. The linear regression model that predicts helpfulness of a review could explain 20.6% of the variance present in the target variable. While this performance is not very great, it should be noted that if we just predicted the mean of the helpful_index (zero information predictor), R^2 would have been negative. Amazon can use this model to order the reviews for a product based on helpfulness so that they can place most helpful reviews at the top of their product webpage.