Predict Helpfulness of Amazon Product Review

Annie Lee
annie.y.lee@berkeley.edu

Le Gu
legu@berkeley.edu

Abstract

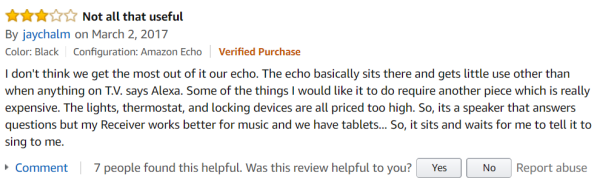
In this paper, we discuss and test out several machine learning approaches for predicting the helpfulness of online product reviews. We work with the Amazon Product Review dataset and focus on applying NLP techniques and building deep learning models, such as RNN and LSTM, to improve the prediction accuracy.

1 Introduction

Customer review data has long been available in the modern commercial world. It is supplementary to other product information and more importantly, provides first-hand feedback from a user perspective. Customer reviews are particularly useful for the online marketplace where customers don't have the opportunity to touch or interact with products. By reading other people's reviews, customers can learn additional information about the product to better inform their purchase decisions.

However, consuming and leveraging the review data effectively is non-trivial, as sometimes popular products can have thousands of reviews, the majority of which are neither useful nor interesting. Promoting and ranking reviews with useful information higher will save customers a lot of time and effort and at the same time significantly improve the overall user experience. Given this, a lot of online retailers have started to collect information to help assess the helpfulness of product reviews. Amazon, for instance, asks readers to vote on the reviews they find helpful under each customer review (Fig. 1). However, this approach has several drawbacks. First, it takes time for new reviews to get feedback after submission and there are simply too many reviews to get feedback on. In addition, simply ranking by number of votes could be skewed toward older reviews, which have had more time to collect more votes but could be outdated and no longer useful.

Figure 1: Screenshot from Amazon.com



Motivated by this, we wanted to build a machine learning model that can help automatically classify new product reviews as helpful or not helpful right after submission. We aim to achieve better model accuracies by combining NLP techniques and deep learning models.

In this paper, we focus on the Amazon Product Data collected by Julian McAuley, UCSD. This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 - July 2014.

2 Background

Earlier studies show promising prediction results when using machine learning algorithms to extract information in product reviews. Using Naive Bayes and SVMs with various kernels (Jordan et al., 2014), the model achieved up to 76.6% accuracy in predicting review helpfulness. With fast development and widespread applications in the deep learning space, one study (Nguy) proposed using RNN models to focus on the textual and sentimental elements of reviews.

Review helpfulness can be very ambiguous and can vary from category to category. One study (Liu et al., 2015) showed qualitative aspects of reviews were identified as the most influential factors that make travel reviews useful. Another study (Liu et al., 2008) of movie reviews find that helpfulness depends on reviewers' expertise, their writing style, and the timeliness of the review. For ecommerce reviews specifically, one study (Susan M. Mudambi et al., 2010) showed review ex-

tremity affects the perceived helpfulness. However, these features can be conflicting for different product types. For experience goods, reviews with extreme ratings are less helpful than reviews with moderate ratings. For search goods, on the other hand, extreme claims can be perceived as credible.

3 Data and Setup

We chose to focus on a subset of the Amazon Product Data. Specifically, we used reviews for products from the Clothing, Shoes and Jewelry category, which consists of 278,677 Amazon reviews. Each review consists of 7 fields: reviewer id, product id, helpfulness rating, review text, star rating, summary of review, and review time. In this section, we cover a few of the interesting characteristics of our dataset.

3.1 Helpfulness Ratings

The helpfulness rating consists of a pair of numbers where the first represents the number of positive votes and the second represents the total number of votes. To compute the helpfulness score, we first filtered our dataset down to only reviews that had at least 1 helpfulness vote. To compute a helpfulness score, we then divided the number of positive reviews by the total number of reviews. Table 1 lists the summary statistics of this data.

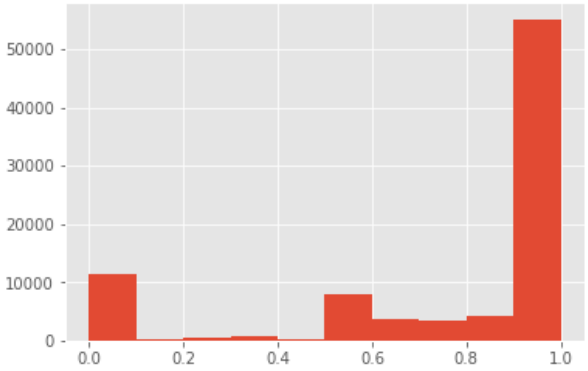
# reviews with helpfulness ratings	87,021
% of reviews with helpfulness rating	0.31
average helpfulness score	0.78
average # helpfulness votes	4.3
minimum # helpfulness votes	1
maximum # helpfulness votes	1267

Table 1: Helpfulness Rating Summary Stats

When looking at the distributions of helpfulness rating, we found 1 record where the helpfulness score was >1. Since the score should be between 0 and 1, we treated this as an invalid record and removed it from our data set.

We also found that the distribution was left-skewed with significantly more positive scores (>0.5) than negative scores (Fig. 2). Additionally, most reviews had very few votes and the distribution was right-skewed with a few reviews having >400 votes.

Figure 2: Distribution of review scores



3.2 Review Text

We also looked at some statistics related to the text of reviews with helpfulness ratings. Table 2 summarizes some basic characteristics of this data after some preprocessing to tokenize the sentences into words.

average review length (words)	68
median review length (words)	47
minimum review length	0
maximum review length	4,571

Table 2: Review Text Summary Stats

We found the distribution of review length to be extremely right skewed as there are a few outlier reviews that are extremely long. We also explored the relationship between the number of characters and helpfulness score, and we found that there is very little correlation between these two variables with a R^2 of 0.054.

4 Methods

In this section, we detail the methodology we used for preprocessing our data prior to training. We'll cover data transformations, sampling, and feature extraction.

4.1 Data Transformation

For our training labels, we adopted a binary measure of helpfulness by transforming the helpfulness score into a 0 or 1 binary label to represent helpful vs. unhelpful. We used 0.5 as the benchmark cut-off value where reviews with a helpfulness score greater than 0.5 were classified as helpful. We chose this cutoff threshold because we felt that it followed the majority rules method. When training our models, we tested our models using

both the raw helpfulness score as well as the binarized scores and found that using the binarized labels performed better

4.2 Sampling

To prepare our dataset for training, we used random selection to split our dataset into 3 groups: training, test, dev with a 80/10/10 split. We validated that each of the 3 groups had similar proportions of positive and negative examples. From our earlier EDA, we found that the helpfulness scores were left-skewed, which meant that post binarizing, we had an unbalanced dataset that had 86% of the reviews classified as helpful. We initially tried many iterations of trying to train models using the unbalanced dataset. However, what we found was that almost all the models we tried resulted in minimizing the loss by simply predicting >98% of the reviews as the positive class. Given this unbalanced dataset problem, we used sampling on our training dataset to balance the two classes. We tested two different approaches:

- Downsample the majority class: randomly select a subset of the positive class that's equal to the size of the negative class. This approach yielded poor model performance, especially for our neural net models, since it meant that we threw away over 70% of our labeled training data.

- Oversample the minority class: random sample with replacement from the negative class corpus until the sample is equal to the size of the positive class. This approach gave us a much larger dataset and didn't require us throwing away useful labeled data. We ended up using this approach for our final model training, as all the models performed significantly better in terms of both sensitivity and accuracy given the much larger dataset. With this, we were also able to outperform the F1 scores reported in Nguy's findings. Fig.3 shows the training dataset class distribution before (left) and after (right) oversampling.

4.3 Feature Extraction

To extract model features, we used Bag-of-Words, which is widely used in NLP to simply represent the review text as a collection of words and ignores the word ordering and sentence grammar.

Figure 3: Oversampling Training Dataset

We used the NLTK library to tokenize the review text into words. To clean up the text, we first converted everything to lowercase and then removed all punctuation and numbers. We found that cleaning up the text didn't provide much improvement since we limited the vocabulary of our model to the top 10,000 words by frequency.

We tested out three different feature sets for model training:

- Term Frequency: we count the occurrence of each word showing up in the review text.

- TF-IDF: we weight each word by number of occurrences in the review text offset by the frequency of the word in the entire corpus.

- Pre-trained Word2vec model: we convert each word into vector representations based on GloVe model which has a vocabulary size of 4 million and dimension of 50.

5 Results and Discussion

We tried a variety of approaches to help determine which type of model and which parameters yielded the best results based on a weighted F1 score. We chose not to use accuracy to compare our models because we found that due to the unbalanced dataset, predicting all 1s artificially yields a high accuracy but does not actually tell us how well it differentiates between the two classes. Therefore, we rely on the weighted F1 score that takes a weighted average of the F1 scores between the two classes and accounts for both precision and recall.

5.1 Baseline Model

For our baseline model, we used the BOW features discussed earlier that are based on term frequency and don't take into consideration word ordering or

sentence structure. We tested a logistic regression and a random forest model on the simple word count features and also tested the same two models on the TF-IDF features to see whether including the IDF would provide any improvements.

We found that using TF-IDF weights performed a bit better than using simple frequency counts for both models and that the logistic regression model performed slightly better than our random forest model on both feature sets.

During our original baseline that used the unbalanced training set, we found that these models predicted >98% of the reviews as helpful, which contributed to a high recall for helpful labels but low recall for unhelpful labels. This also yielded a high accuracy because the majority of the dataset was labeled as helpful. This helped guide us moving toward using the weighted F1 score and using the oversampling method mentioned earlier to help balance the dataset. In Table 3, we've provided the weighted average results of the Logistic Regression baseline model using TF-IDF features and trained with the balanced training data set.

Precision	Recall	F1
0.79	0.68	0.72

Table 3: Baseline Model accuracy

5.2 Deep Learning Model

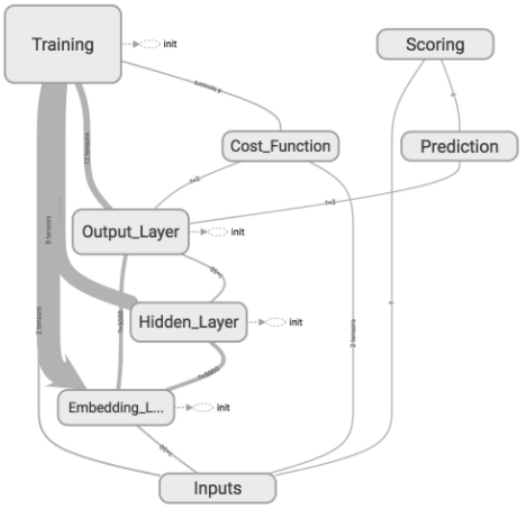
To better maintain the syntactic and semantic characteristics of the review text, we switched to exploring deep learning models for classification. In these models, we replaced the BOW features with word embeddings. Similar to our baseline models, we first converted all the input text to tokens. However, instead of passing in the count of each word in the entire vocabulary, we passed in the tokens of the words in the same order they appeared in the review text. Additionally, we limited our vocabulary to the top 10,000 words by word count from our training set. During the tokenization process, if the words were not in the vocabulary, they would be assigned the same default token id. Additionally, we truncated/padded all the review texts to the same length so that we could pass a fixed size matrix to our deep learning models. We used this max review text length as a trainable parameter.

We tested 2 deep learning approaches: a simple neural net model and a recurrent neural net (RNN) model using Long-Short Term Memory (LSTM).

5.2.1 Simple Neural Net

The first model we tried was a simple single layer neural net that has an embedding layer which converts the tokens into word embeddings, a hidden layer that applies a non-linear activation function, and finally an output layer that computes the final logits (Fig. 4). For the word embeddings, we tried using a pre-trained GloVe vector but found that this actually made the model perform significantly worse. We found that having a trained embedding built into our model actually performed the best.

Figure 4: Simple Neural Net Tensor



5.2.2 RNN with LSTM

The second model we tried was an RNN model using LSTM cells. Instead of consuming the entire review text altogether, the LSTM cells passes each word from one cell to the next, using the output of the previous cell as part of the input of the next cell. As this is a classification problem, we only use the output of the last cell to make the final prediction for each training example. (Fig. 5)

5.3 Results Summary

With our various models, we found that the simple neural net with trained embeddings performed the best with an F1 score of 0.77. The RNN with LSTM performed a bit better than the baseline model but we were not able to get it to outperform the simple neural net even with a significant amount of tuning. Below, we've provided a summary of the results from our various model iterations (Fig. 6).

Figure 5: LSTM Tensor

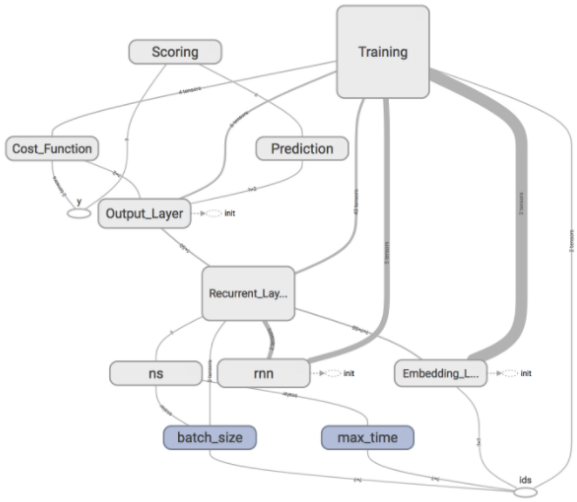
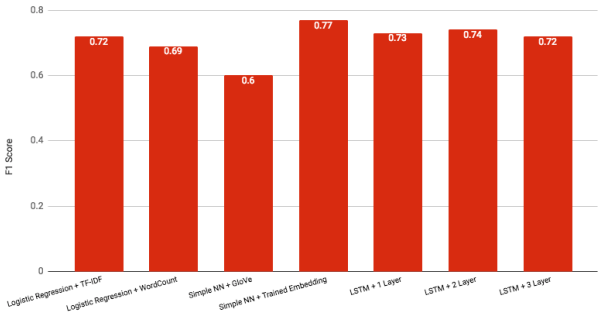


Figure 6: Model performance comparisons



5.4 Parameter Tuning and Error Analysis

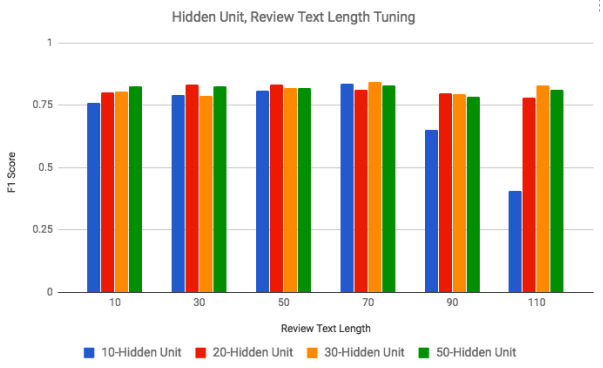
We tuned the model based on a series of hyperparameters and found for LSTM, the loss tended to bounce around so using a smaller learning rate of 0.01 generally performed better. Additionally, we also found that running at least 2 epochs helped but any additional epochs did not provide much benefit and usually led to overfitting, without much improvement to the F1 score on our dev dataset.

When looking at the errors, we found that many of the incorrectly classified examples had very few helpfulness votes. In fact, many of them only had a single vote. This indicates that one of the reasons the model may not do well at classifying these examples is that there simply isn't enough data about these reviews, which could lead them to be misclassified. There is a high amount of potential volatility in this labeled data since with fewer votes, an additional upvote or downvote could change the binary classification of this example. To resolve this issue, we considered filtering out reviews with too few votes. However, we found that this would significantly decrease our training dataset size and we opted for using a larger noisier dataset.

We also found that when looking at the false negative examples, they were shorter reviews on average compared to the overall dataset. This suggests that the model was likely to misclassify shorter sentences, possibly because there were too few words to provide enough useful information given our fixed padding length or that shorter reviews are often considered to be less helpful. To try to account for these errors, we tried tuning the model by adjusting the maximum review text length but found that this did not make much of a difference (Fig.7). In the end, we found the model performs the best when max review text length is close to the average corpus length.

We also tried tuning a variety of other parameters and found that the number of hidden-units only improved the model when dealing with smaller max review text length. This was true for both the simple neural net and LSTM model. For the simple neural net, adding more layers did not help as it led to overfitting. However, for the LSTM model, we found that stacking layers provided some improvement: 2-Layer LSTM performed better than 1 layer. However, stacking too many layers such as a 3-Layer LSTM overfits with a large max review text length.

Figure 7: Dev results from LSTM model with various hyperparameters



5.5 Conclusion

With our various models, we found that using only review text as features may not provide enough information to predict the helpfulness of a review. An interesting future exploration would be to see

how text features can be combined with other features such as the product rating or user attributes to provide supplementary signals. Additionally, we found that our training data provided many ambiguous cases due to lack of votes that made it difficult for the model to differentiate between these reviews. One possible future exploration could be to collect more voting information about the reviews that have few votes to help create a cleaner dataset and see whether this helps create better separation between the two classes.

References

Jordan Rodak, Minna Xiao, Steven Longoria. *Predicting Helpfulness Ratings of Amazon Product Reviews*

Nguy, Bobby. *Evaluate Helpfulness in Amazon Reviews Using Deep Learning.*

Liu, Zhiwei, and Sangwon Park. 2015. *What makes a useful online review? Implication for travel product websites. Tourism Management 47 (2015)*: 140-151

Yang Liu, Xiangji Guang, Aijun An, and Xiaohui Yu. 2008. *Predicting Helpfulness Ratings of Amazon Product Reviews Modeling and predicting the helpfulness of online reviews. In Proceedings of the Eighth IEEE International Conference on Data Mining*: 443-452 Los Alamitos, CA, USA.

Mudambi, Susan M. and David Schuff. 2010. *What Makes a Helpful Online Review? A Study of Customer Reviews on Amazon.com MIS Quarterly , 34 (1)*: 185-200