Predicting Amazon Product Review Helpfulness

James Wei, Jessica Ko, Jay Patel
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{james.wei, jessicak, patel.jay}@berkeley.edu

Abstract

A large number of Amazon product reviews have little to no user feedback, thus making it difficult for listings to emphasize helpful opinions while relegating less insightful comments. Hence, we present a deep learning solution for classifying the best and worst product reviews. Our model leverages a recurrent neural network (RNN) architecture and is the first to go beyond traditional text classification methods in this problem space. We compare our deep learning model to a baseline classifier that uses logistic regression. Our deep learning model outperforms the baseline by identifying good product reviews with 80.50% accuracy (0.88 AUC) and bad product reviews with 75.70% accuracy (0.83 AUC). Conversely, the baseline model classified good product reviews with 74.08% accuracy (0.82 AUC) and bad product reviews with 72.05% accuracy (0.79 AUC). We analyze the tradeoffs of the two models and present a discussion of future work.

1 Introduction

Over the past two decades, electronic commerce, or e-commerce, has grown from a handful of fledgling startups to a booming, trillion dollar industry [10]. Following a shift towards the online marketplace, the paradigms under which businesses and consumers market and evaluate products have changed drastically. Because the point of sale for online merchants does not involve retail stores where people can physically assess items in consideration, customers must turn to other sources to help inform their purchase decisions.

In particular, studies show that consumers give strong weight to *online word of mouth*, customer opinions communicated via online channels [3]. Online word of mouth commonly takes the form of product reviews featured alongside listings on e-commerce websites. Research also indicates that the content and display of these product reviews directly impact the sales of online products [2].

Given the large amount of user-generated content circulating across e-commerce platforms, both Internet retailers and consumers would like to effectively identify the reviews that provide the most insight. However, the variation in the quality of product reviews makes it difficult to identify useful feedback in a sea of noise.

To help determine the quality of a product review, online retailer Amazon.com gives users the option to vote on the helpfulness of a particular review. Figure 1 illustrates the mechanism through which this feedback is collected. Based on the helpfulness rating of product reviews, Amazon ranks and prioritizes the presentation of the most helpful reviews.

While this system may work well for popular products listings that garner heavy traffic, a vast majority of Amazon product reviews are left with limited or no helpfulness feedback. Moreover, when a particularly insightful review is shared, it is not immediately featured since it takes time for people to read and upvote the review. For consumers, skimming through a large number of superfluous comments in search of insightful assessments is frustrating and time-consuming. Without an automated method for assessing the quality of reviews, useful information remains buried amidst millions of Amazon product listings.

1.1 Problem statement

To address the problem outlined above, we seek to build a model for classifying product reviews by their helpfulness to consumers. To begin, we define the following.

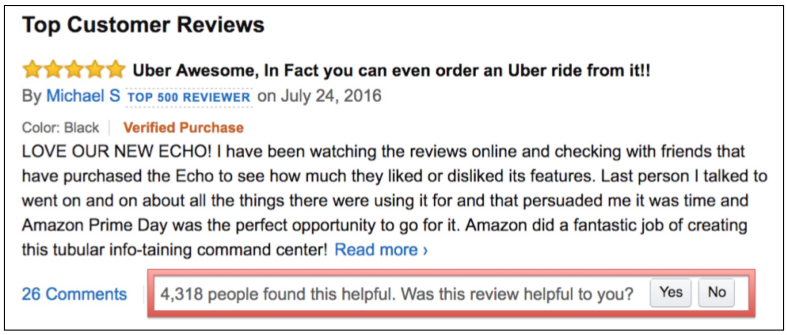

Figure 1: A sample Amazon product review with a field to allow readers to leave helpfulness feedback

For a given Amazon review r, let f_p be the number of users that found the review to be helpful (i.e., positive feedback, the total number of "upvotes"), and let f_n be the number of users that did not find the review to be helpful (i.e., negative feedback, the total number of "downvotes"). Further, let f be the total amount of helpfulness feedback received by r such that $f = f_p + f_n$. We denote s_r to be the *helpfulness score* of r, where

$$s_r = \frac{f_p}{f_p + f_n} = \frac{f_p}{f}$$

In short, s_r is the proportion of people that found review r to be helpful. For reviews with a non-zero amount of helpfulness feedback ($f > 0$), the value of s_r falls in the range $[0, 1]$. We say a review r is "good" if $s_r > 0.80$; we say r is "bad" if $s_r \leq 0.20$.

In this analysis, we train models that perform two binary classification tasks aimed at filtering out the best and worst reviews. Specifically, for some Amazon product review r, we train one classifier that determines if $s_r > 0.80$, and another that determines if $s_r \leq 0.20$.

Our motivation for classifying reviews on the extremes is twofold. First, product reviews that fall in the middle of the helpfulness spectrum are prone to noise, which make differentiating intermediate boundaries more difficult when training our models; this is illustrated in Figure 2. Second, identifying reviews at the extremes provides the most information gain for Amazon and its customers. Our models give Amazon the ability to present the most relevant information to consumers while efficiently filtering out reviews that a large majority of customers would find unhelpful.

To this end, we take two different approaches to developing models for these classification tasks and evaluate them by their classification accuracies and areas under the curve (AUC). We first develop and evaluate the performance of a traditional text classification algorithm; we then architect a deep learning model for the same problem and present a comparison between the two strategies.

1.2 Background and related work

Proposed solutions for text classification problems involve both neural and non-neural techniques. With respect to Amazon movie reviews, prior work has shown that ensemble learning using features derived from review length, unigram tf-idf, overall product rating, and punctuation patterns performs better than isolated SVM, logistic regression, and ridge classification [1].

Other related work suggests that the use of contextual metadata features (e.g., author identity, product type, as opposed to vanilla bag of words on the review text) when analyzing product reviews in a non-neural setting can improve prediction accuracy [5].

At the same time, deep neural networks have been found to perform better than conventional regression models across a number of different applications [7]. In particular, since the development of recurrent neural networks [14], or RNNs, researchers have leveraged this architecture to classify, process and predict time series data. Moreover, the introduction of a specific kind of RNN known as long short-term memory [4], or LSTM, has revolutionized speech recognition and synthesis, machine translation, language modeling, and image captioning. Given the growing popularity of deep learning for natural language processing applications, we build a deep network consisting of several LSTM layers for predicting product review helpfulness and subsequently compare it to a non-neural model with sophisticated NLP features.

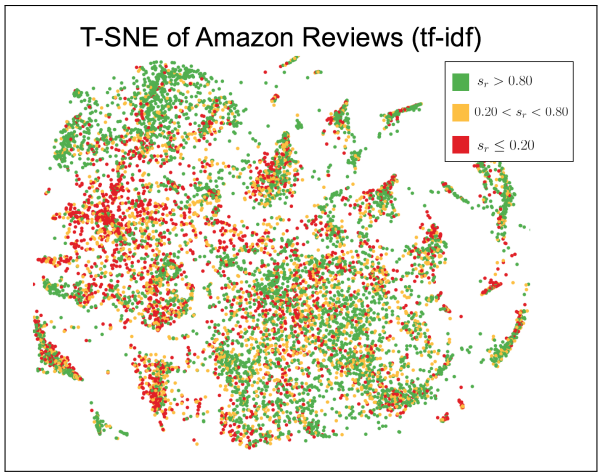

Figure 2: t-SNE embedding of Amazon reviews colored by the helpfulness score (s_r). Non-polar (yellow) reviews are scattered throughout the space; therefore, we focus our problem on classifying the extremes (red, green).

The rest of this paper is structured as follows: we describe the Amazon dataset used in Section 2; we discuss the architecture of our baseline and deep learning model in Sections 3 and 4; we share implementation details in Section 5; we present the results of our experiments in Section 6; we comment on lessons learned and future work in Sections 7 and 8; and we conclude in Section 10.

2 Dataset

For our analysis, we use the Amazon product data dataset by UCSD's Julian McAuley [9]. Each entry in this dataset represents a single review r for a particular product and contains the following fields: reviewer ID, product ID, reviewer name, helpfulness rating (tuple of two integer values f_p and f), review text, overall rating (out of five), review summary, and review time; these entries are grouped by 24 product categories (e.g., books, video games). A sample JSON entry is shown in Figure 3.


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [29, 35],
  "reviewText": "We have been using Echo
                since April...",
  "overall": 4.0,
  "summary": "Already very practical...",
  "unixReviewTime": 1252800000,
  "reviewTime": "06 19, 2015"
}
```


Figure 3: A sample entry from the Amazon product review dataset

2.1 Helpfulness distribution

A dense, 5-core version of the dataset is readily available for download online. This subset contains reviews from products and users with at least five associated entries. In total, the 5-core version of the dataset contains 18.19 million reviews. For our purposes, we filter out all reviews with less than five helpfulness feedback votes; this leaves us with 2.99 million reviews.

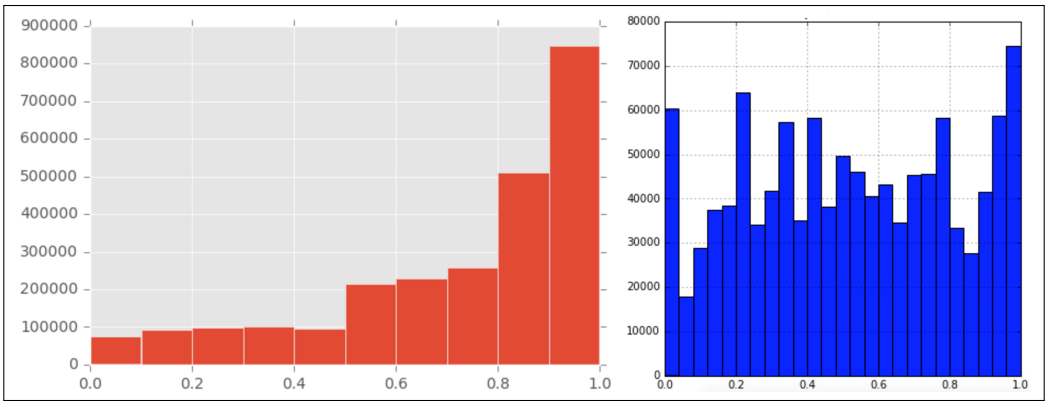

Figure 4: Histograms of helpfulness scores: (left) distribution of helpfulness scores in the original dataset, (right) distribution of helpfulness scores in the filtered dataset

However, through initial analysis, we find that the helpfulness distribution of these 2.99 million reviews is heavily left skewed, with 1.46 million reviews having ratings greater than 0.8 (see Figure 4a). The skew of the dataset distribution is not something previous work has taken into consideration.

Consequently, we unskewed the 5-core dataset so that the helpfulness distribution of our dataset is roughly uniform (see Figure 4b). This unskewing operation leaves us with a total of 1.11 million reviews.

Because of computational and memory-related limitations, and in the interest of time, we randomly sample a subset of 50,000 reviews from the uniformly distributed dataset. Through the course of our evaluation, we set aside 20% of the data as a test set and use the remaining 80% for training and validation.

2.2 Text normalization

To make the raw text of each product review easier to featurize, we first transform our data into normalized unigrams. Given a product review, the raw text is the concatenation of the `summary` field and the `reviewText` field. We perform the following operations on the raw text for each entry.

1. Remove all capitalization
2. Tokenize the text by continuous sequences of alphabetic characters (remove whitespace, symbols, punctuation, numerals)
3. For each token, filter out
   - English stopwords from the NLTK stopwords corpus
   - Tokens with less than three characters

3 Baseline model

Our first approach to this problem involves building a baseline text classifier to classify good and bad reviews (defined in Section 1.1). We compare this model's performance to the deep learning model described in the next section. Figure 5 shows the structure of this classifier.

3.1 Featurization

For the baseline model, we use *term frequency–inverse document frequency* (tf-idf) values to featurize the review text of each entry in our dataset [11]. Suppose we have some corpus $C = \{d_0, \ldots, d_{n_C-1}\}$ containing n_C documents. Each document d_k is composed of n_{d_k} terms $\{t_0, \ldots, t_{n_{d_k}-1}\}$. For each t_i in document d_j, we compute:

$$\text{tf-idf}(t_i, d_j) = \text{tf}(t_i, d_j) \times \text{idf}(t_i) = \frac{\text{f}(d_j, t_i)}{n_{d_j}} \times (\log(\frac{1+n_C}{1+\text{df}(d_j, t_i)}) + 1)$$

where $\text{f}(d_j, t_i)$ is the frequency of t_i in d_j, n_{d_j} is the total number of terms in d_j, n_C is the total number of documents in the corpus, and $\text{df}(d_j, t_i)$ is the number of documents in C that contain the term t_i.

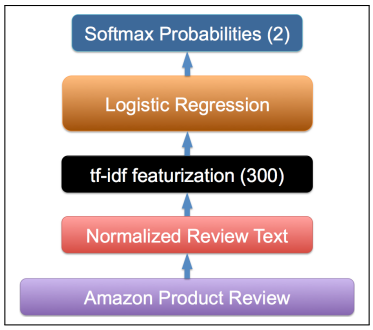

Figure 5: Baseline model architecture

Although originally developed as a term weighting scheme for information retrieval, tf-idf has proven to be very effective for document classification and other natural language processing tasks [6].

Using this featurization method, we compute the tf-idf weights for the top 300 unigrams, bigrams, and trigrams across our training set.

3.2 Logistic regression

Having featurized our dataset, we train two logistic regression models, one for classifying good reviews, and the other for classifying bad reviews. The primary hyperparameter we tune in our logistic regression model is C, the inverse regularization strength of the ℓ_2 norm, used to minimize the risk of overfitting. We select the best hyperparameter set by evaluating the model's accuracy on our validation set.

The results of these models will be presented in Section 6.

4 Deep learning model

In our deep learning model, we take advantage of the ability of LSTM units to understand temporal relationships in time sequence data [4]. In our case, we use an LSTM architecture to characterize a sequence of words in a product review.

We feed the output of the LSTM layers, engineered context features, and the normalized probability from a logistic regression module into a series fully connected layers. These feed-forward layers output normalized softmax probabilities with respect to the helpfulness of the input review. Figure 6 gives a visual representation of our architecture.

We use the sum of cross-entropy loss and the ℓ_2 regularization term at our final softmax layer as our objective function, optimized over training iterations using the Adam algorithm. We use the Adam optimizer because of the many benefits such as the bias correction and lessened importance on fine tuning hyperparameters [8].

4.1 Featurization for fully connected layers

Given a review r for product p written by user u, we create a set of features to be passed in as input to a fully connected neural network.

Context features (31 features). Information drawn from product review metadata is useful for capturing information not included in the review text.

- *Product popularity*: The number of reviews the p has received. This is identified through the `asin` field in the dataset.
- *Reviewer history*: The number of reviews the user u has written.
- *Product category*: One hot encoding for the product category of p. There is a total of 24 product categories (e.g., books, video games).
- *Overall rating*: The length-5 one hot encoding for the rating the user gave the product. This helps characterize the tone of the review (i.e., five-star reviews are laudatory while one-star reviews are critical).

Anatomical Encoder (12 features). This encoder extracts semantic and syntactic features from the review text. For a review r with normalized text t_n, we encode

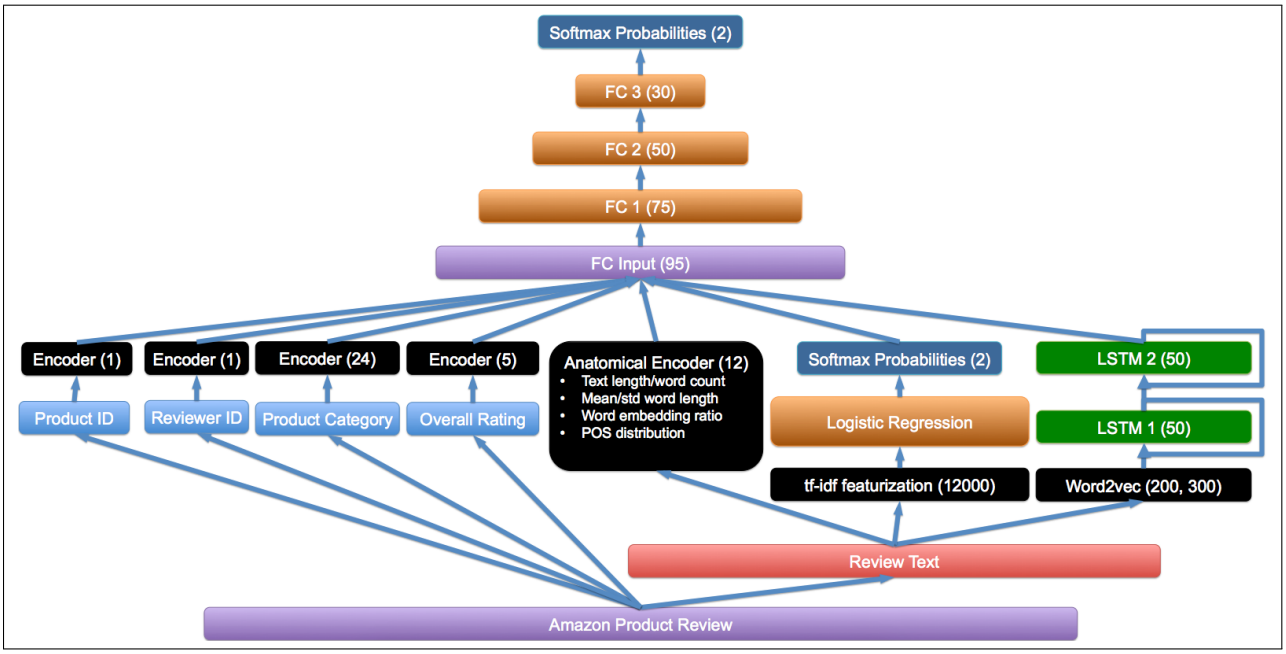

Figure 6: Deep learning architecture

- The number of tokens in t_n
- The mean and standard deviation of token lengths
- The number of characters in t_n
- The word embedding ratio
  This value represents the fraction of tokens found in our word embedding model (Word2vec). Variable length reviews are accounted for by only considering up to the first 200 words in the review. The ratio is defined as

$$\frac{\text{number of successful Word2vec embeddings in the first 200 tokens of } t_n}{\min(\text{number of tokens in } t_n, 200)}$$

  Because we use a minimized pre-trained word embedding model consisting of approximately 43,000 entries, this ratio approximates the proportion of correctly spelled and commonly identifiable English words.
- Part of speech distributions
  We apply part of speech (POS) tagging using the Univeral POS tagset to each token in t_n. We then compute the ratio of occurrences of different POS and the total number of tagged tokens. We use POS ratios for adjectives, adpositions, adverbs, conjunctions, nouns, verbs, and "unknown POS" as features.

Logistic regression probabilities (2 features). We fit a logistic regression model using tf-idf features of the top 12,000 frequently occurring n-grams across stemmed words from all the review texts. We apply the Lancaster stemmer on each of the tokens in t_n and compute tf-idf features on the top 12,000 frequently occurring unigrams, bigrams, and trigrams. We chose to use the Lancaster stemmer because it is more aggressive than the Porter and Snowball stemmers. With a more aggressive stemmer, we were able to consider a smaller set of unique n-grams, which made it easier for us to control the size of our tf-idf featurization.

We tune an ℓ_2 regularization penalty to prevent overfitting. We extract the normalized softmax probabilities from the logistic regression model and include them as features for the inputs to the fully connected layers.

LSTM output (50 features). We use two stacked LSTM layers, each with 50 hidden units to produce a characterization of the review text. Dropout is applied between each LSTM layer for regularization [13]. The raw outputs of the LSTM layers are used as input features to the fully connected network.

To craft the input into the LSTM network, we use a pre-trained, size-43,000 Word2vec model to encode the first 200 embeddable words of each product review (and zero pad shorter reviews). We then pass these embedding into the LSTM network in the order that their respective words appear in the review text. The dimension of each word embedding vector is 300.

4.2 Fully connected layers

We use the concatenation of the features above as the input to a series of fully connected layers. We run three fully connected layers of sizes 75, 50, and 30. Similar to the LSTM, we use dropout between every layer [13]. Moreover, leaky ReLU ($\max_x(0.01x, x)$) is used as the activation function between each layer to prevent the dying ReLU problem [12]. Finally, we classify the review based on the highest output softmax probability.

4.3 Evolution

Our group tested many architectures before we settled on our final model. We first implemented a single layer LSTM network that took in variable length inputs, but the model only performed slightly better than our baseline accuracies. Moreover, we wanted to incorporate other contextual and anatomical features (e.g., product and reviewer identity, part of speech distribution); however, the pure LSTM network limited us to temporal features. As such, we added the fully connected layers and designed encoders for each of our chosen features.

Finally, we were surprised by the performance of tf-idf featurization in our baseline and decided to incorporate it into our final model. However. when we tried to input a large number of tf-idf features alongside our existing features into our fully connected layers, we found that the activations were too sparse. This is when we initially implemented leaky ReLU activations to avoid the problem of dead neurons. Following this failure, we decided to incorporate a dense interpretation of the tf-idf featurization by training and integrating the results of a logistic regression model into our final design.

5 Implementation

We used a combination of various frameworks, libraries, and computing platforms to build our two models.

Our baseline model was implemented in Python 2.7 as a Jupyter notebook. The model was trained and evaluated locally on a 2.6 GHz Intel Core i5 machine with 8 gigabytes of 1600 MHz DDR3 RAM.

Our deep learning model was also implemented in Python 2.7. We used the Tensorflow software library (v0.11.0) to define our LSTM and fully connected neural network layers. Additionally, several popular Python libraries were used to streamline the development process, including nltk 3.2.1 (for NLP operations like stemming and POS tagging), pandas 0.18.1 (for dataset handling), numpy 1.11.1 (for vectorized computations), scikit-learn 0.18.1 (for model building), gensim 0.13.3 (for word embeddings), and matplotlib 1.5.1 (for data visualization).

Because of the significant computational cost associated with running a deep neural network, we trained and evaluated this model on an Amazon EC2 g2.2xlarge instance equipped with 15 gibibytes of memory, five Intel Xeon E5-2670 processors, and an NVIDIA GPU with 1,536 CUDA cores and 4GB of video memory.

6 Results

We trained, validated, and tested our baseline and deep learning models on size-50,000 subsets of the unskewed dataset. For each of the two binary classification tasks (identifying good and bad reviews), we normalized the input helpfulness distribution so that random guessing would yield an expected accuracy of 0.5. For instance, when training and evaluating a good review classifier, we picked review entries such that 50% of the input samples have $s_r > 0.8$, 25% have $s_r \leq 0.2$, and 25% have $0.2 < s_r \leq 0.8$. Conversely, when training and evaluating a bad review classifier, we picked review entries such that 50% of the input samples have $s_r \leq 0.2$, 25% have $s_r > 0.8$, and 25% have $0.2 < s_r \leq 0.8$. A summary of the results is shown in Figure 10.

6.1 Baseline model

After featurizing the data using tf-idf, we allocate 60% of the data for training, 20% for validation, and 20% for testing.

The primary hyperparameter we tune in our logistic regression model is C, the inverse regularization strength of the ℓ_2 norm. Using random parameter search, we achieve a validation set accuracy of 0.74875 with $C = 0.1000$ for the good review classifier; we achieve a validation set accuracy of 0.72375 with $C = 0.2416$ for the bad review classifier.

Upon fitting the model to the combination of the training and validation sets, we evaluate the model on the test set. The good review classifier achieved a final test set accuracy of 0.7408 with 0.82 AUC; the bad review classifier achieved a final test set accuracy of 0.7205 with 0.79 AUC. The ROC plots for these respective classifiers can be found in Figure 7.

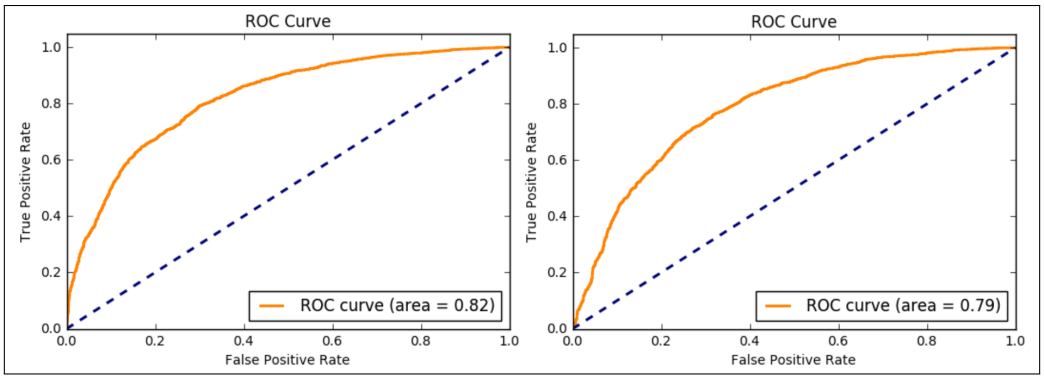

Figure 7: ROC plots for the baseline model: (a) (left) classifier performance on good reviews ($s_r > 0.8$), (b) (right) classifier performance on poor reviews ($s_r \leq 0.2$)

6.2 Deep learning model

As with the baseline model, we featurize the data and allocate 60% of the data for training, 20% for validation, and 20% for testing.

We cap the length of the word embedded input into the LSTM at 200 units and limit the tf-idf featurizer to 12,000 features. For the LSTM and fully connected layers, we apply 15% and 10% dropout respectively to reduce the risk of overfitting [13].

We set our ℓ_2 regularization weight to be 1.0×10^{-5}. For the Adam algorithm, we set the learning rate to 1.0×10^{-3}; additionally, we used $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1.0 \times 10^{-8}$.

We trained this model over ten epochs. The good review classifier achieved a final test set accuracy of 0.8050, a final test set loss of 0.4645, and a 0.88 AUC; the bad review classifier achieved a final test set accuracy of 0.7570, a final test set loss of 0.5217, and a 0.83 AUC. The ROC plots for these respective classifiers can be found in Figure 8; the plots of accuracy and loss over time can be found in Figure 9.

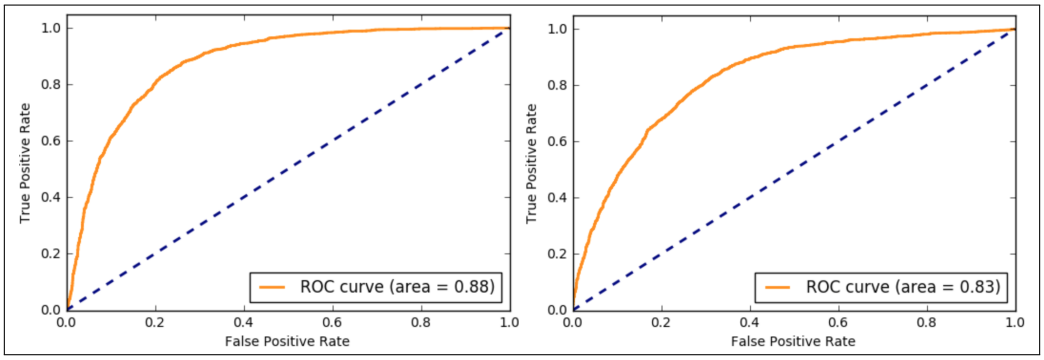

Figure 8: ROC plots for the deep learning model: (a) (left) classifier performance on good reviews ($s_r > 0.8$), (b) (right) classifier performance on poor reviews ($s_r \leq 0.2$)

7 Lessons Learned

Overall, our final model outperformed our baseline with respect to classification accuracy and AUC for both the good review and bad review classifiers. Even though the deep learning model took substantially longer to train, the model was ultimately able to do a better job of discerning the characteristics that contribute to review quality. For the both baseline and final model, the accuracies for the good review classifier were higher than the accuracies of the bad review classifier.

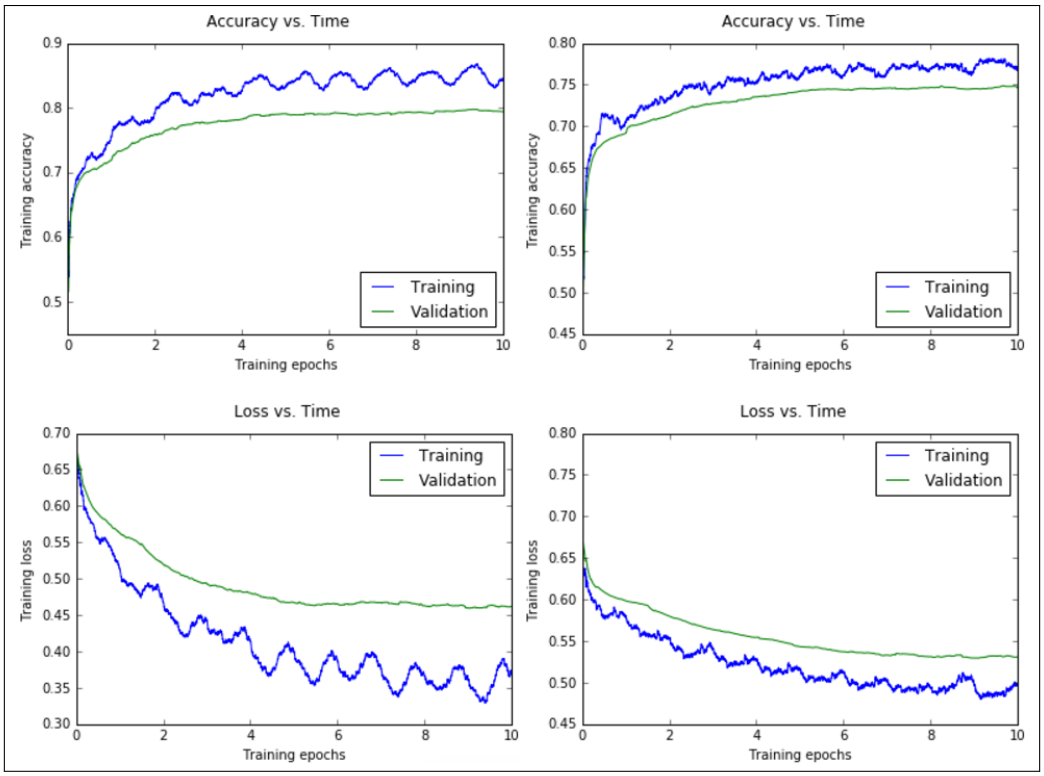

Figure 9: Accuracy vs. time and loss vs. time from training our deep learning model: (a) (left) classifier performance on good reviews ($s_r > 0.8$), (b) (right) classifier performance on poor reviews ($s_r \leq 0.2$)

	Baseline Model		Deep Learning Model	
	Good	Bad	Good	Bad
Test Accuracy	0.7408	0.7205	0.8050	0.7570
AUC	0.82	0.79	0.88	0.83

Figure 10: Results summary

We were met with many challenges when developing our models. Unskewing our data was crucial to prevent our model from being biased towards the dominating class in the training set; this was especially important while we were training our bad review classifiers. Unskewing our data allowed our models to understand the relationships between each of the features, rather than just inferring the prior distribution of the dataset.

Another major challenge was handling reviews of different lengths when crafting the input to the LSTM network. Originally, we tried to unroll the LSTM network to the length of our longest review. However, we found that this significantly increased the required state and made training much more volatile. To reduce the degrees of freedom, we made the decision to cap the number of word embeddings per review at 200.

In terms of the results, we were surprised to find that classifying bad reviews more difficult than classifying good reviews. This pattern holds true for both the baseline model and the deep learning model. Upon comparing the structural characteristics of good and bad reviews, we found that poor reviews tend to be short and lack elaboration. For instance, while both helpful and unhelpful reviews make use of unspecific words like “excellent” or “love”, helpful reviews feature additional words and phrases in the process of discussing their opinions. On the other hand, many unhelpful reviews lack this additional signal, leading us to believe that our models have an easier time detecting the presence of a characteristic than the absence of one.

Moreover, as shown in Figure 2, the groupings of the poor reviews seem to be less distinct compared to the clusters of good reviews. This is especially noticeable in the top left where there is a large grouping of mostly good reviews indicated in green.

8 Future work

Even though we were able to successfully classify reviews with both the baseline model and deep learning model, there are still many more aspects we could explore. One natural extension of our work would be to train our final model on the entirety of the one entry unskewed dataset instead of just a subset. The model's performance may improve if given more data and training time. An additional improvement would be to train a custom Word2vec model to better our corpus. We can also consider alternatives to Word2vec like doc2vec or pargraph2vec. These models take longer texts into account compared to the words for word2vec. Additionally, improved data visualizations can assist in feature selection. Finally, we suspect splitting the dataset by each of the 24 product categories and training a separate model per category would result in better AUC.

9 Member Contributions

Although most aspects of this effort were largely collaborative, we present a rough outline of the responsibilities each member completed. We all gave equal effort (33.3%) on the project.

James worked on dataset analytics and architected the deep learning model. He also created figures of the performance results for both the baseline and deep learning models. Jessica engineered context and anatomical features and built the baseline model. She also was primarily in charge of managing our EC2 deployment. Jay worked on gathering and formatting the dataset and visualizing distributions. He also engineered features, focusing on word embeddings.

10 Conclusion

In this analysis, we present a deep learning approach for classifying the helpfulness of Amazon product reviews. We show that using an LSTM architecture outperforms traditional text classification for identifying the most helpful and least helpful reviews. To the best of our knowledge, our design is the first deep learning approach for this problem, and our results show that this solution yields better results than previous work. We explain the rationale for the different parts of our design, discuss trade-offs we encountered, and leave several avenues open for future work.

References

[1] S. Bhargava. Predicting amazon review helpfulness. UCSD, 2015.

[2] Y. Chen and J. Xie. Online consumer review: Word-of-mouth as a new element of marketing communication mix. *Management science*, 54(3):477–491, 2008.

[3] J. A. Chevalier and D. Mayzlin. The effect of word of mouth on sales: Online book reviews. *Journal of marketing research*, 43(3):345–354, 2006.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] X. H. J. Tang, H. Gao and H. Liu. Context-aware review helpfulness rating prediction. RecSys, 2013.

[6] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, DTIC Document, 1996.

[7] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[8] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. RecSys, 2013.

[10] K. Miyatake, T. Nemoto, S. Nakaharai, and K. Hayashi. Reduction in consumers' purchasing cost by online shopping. *Transportation Research Procedia*, 12:656–666, 2016.

[11] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.

[12] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[14] J. Von Neumann, A. W. Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966.

Appendix

The code for our project may be viewed at `https://github.com/jessicak/amazon-review-helpfulness`.