Is this a Slickdeal?

Andrew Li
Department of Linguistics
University of Georgia
andrew.li@uga.edu

Abstract

Online shopping has become a popular pastime for many people; however, how do you know if you are getting a good deal on a quality product? In this paper, I will explore the procedure that I used to build a machine learning model that deciphers whether a product on slickdeals.net is a good deal. The model that I created aims to give consumers that shop on the website a true measure of whether a product is worth buying.

1 Introduction

Slickdeals.net (hereby referred to as "SD") is a website that serves as a platform for people to share products that they find on the internet that are priced well below their manufacturer suggested retail price (MSRP). These items can be made up of many types of product categories, including groceries, video games, electronics, clothing, home appliances, and much more. Below is a graphic of what the website looks like:

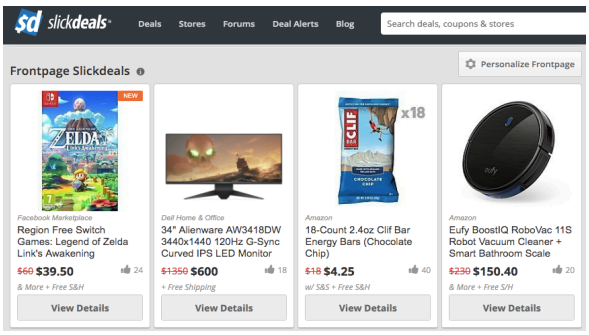


Figure 1: SD website snippet

For each of the products, there is a very active community that posts their opinions about the products and talks about whether the product is worth buying. Generally, the community is well informed about the quality of each product, as they often post their personal experiences with the product. This reason alone makes the SD community unique, as reviews left on items on other websites, such as Amazon, may not reflect the true quality of the product. For example, businesses may have all of its employees leave bad reviews for its competitors in hopes to bring them down in order to draw in more customers. Additionally, given that businesses understand the impact that reviews have on user buying behavior, "some businesses have become so desperate for that elusive fifth star that they're paying for positive reviews online, a crime filthier than the rats running around [a dirty restaurant]" (Anonymous).

In contrast, the products on SD are not sponsored by any companies. Rather, ordinary consumers simply post products that they think are good deals and share them with the community, where the community users discuss the product and talk about whether the product is worth buying. This authenticity differentiates SD from product reviews online.

The goal of this project is to apply sentiment analysis to create a well-performing machine learning model and generate understanding about an effective way to combine English lexicon and data science to generate a modern-day solution for online shopping. The high-level flow of this paper is as follows:

- How the data was attained and the preprocessing steps that were necessary before solving the problem
- Determine model features and algorithms
- Experiments and evaluation of the model's performance
- Application to SD and potential applications to other languages
- Related work
- Limitations
- Conclusion and future work

2 Data Collection and Pre-Processing

2.1 Data Collection

I used Amazon reviews to train my model because they were readily available to download, sorted by category, and for the most part very clean. Even though there isn't a direct 1:1 match between Amazon and SD product reviews, I banked on the premise that wording and semantic structure are somewhat similar for both. Since each Amazon review file (based on product category) is multiple gigabytes large, I chose 4 datasets to train and test the model, which includes the following categories:

- Major Appliances
- Sports
- Electronics
- Clothing

Due to memory and computational limitations, the goal here was to choose unique enough product categories to involve reviews that were relevant and broad enough that could be applied to products spanning different categories. This aspect is crucial because the words being used in the product reviews have a big impact of how well the model performs on unseen SD comments regarding a product's performance. Below is example of a 1-star review from the Clothing dataset:

"I wasn't very happy with this item. There's a small stain on the back (looks like dye of some sort) and it looks like the neckline was cut with scissors, which I could have done myself. I don't like it at all."

Additionally, below is a comment from the SD website regarding a deal on sports hoodies:

"Made the mistake a few years ago buying one of these (Syracuse) from a slickdeal, garbage quality, unraveling embroidery straight out the packaging. Don't waste your time."

Clearly, these are both negative reviews, which gives better information about the sentiment that is being expressed in both the test dataset and SD website.

2.2 Pre-processing

Even with just the 4 datasets that were chosen from the Amazon review repository, the number of reviews across the datasets totaled up to 13.9 million reviews. To prepare the data for analysis, I removed 3 star ratings from the datasets because it would be too ambiguous for the machine to generate an understanding about. After removing 3 star ratings from the datasets, there was a total of about 10 million reviews across the 4 datasets. Because this volume is extremely memory intensive to run, I decided to grab 50k reviews from each dataset, which resulted in 200k total reviews to work with. However, after looking at the distribution of the star ratings from these 200k reviews, the distribution was as follows:

Figure 2: Imbalanced star rating reviews in dataset

As shown in the chart above, there is a class imbalance problem between the negative ratings (1 and 2 star reviews) and the positive ratings (4 and 5 star reviews). If not taken care of, this can lead to a problem, because most machine learning algorithms assume that the classes are balanced, and if they are not, the model will be more biased towards positive sentiment reviews (4 and 5 star ratings), causing bad classification of the negative sentiment reviews (1 and 2 star ratings).

To mitigate this problem, I instead grabbed 25k positive and 25k negative reviews from each of the 4 datasets to reach the total of 200k reviews. This way, model can train on balanced data and can eliminate any class biases. Below is the new distribution of the dataset:

Figure 3: Balanced star rating reviews in dataset

As Figure 3 shows, the dataset is now much more balanced, as there are 100k negative reviews and 100k positive reviews.

The next step was to generate a label for each of the reviews in order to allow the model to understand positive vs. negative sentiment reviews. As mentioned earlier, I've determined that 4 and 5 star reviews constitute the positive reviews in the data set, the 1 and 2 star reviews constitute as the negative reviews, and the 3 star reviews are ignored. So, I created a new column for my dataset called "Sentiment," which marks a 1 for positive reviews and a 0 for negative reviews. My dataset now looks like this:

star_rating	review_body	sentiment
1	Arrived with a broken volume button and doesn'...	0
2	OKAY SO, they're nice sounding headphones...bu...	0
1	I got this and it was burned out . Not working...	0
1	Did not work. Defective	0
1	I'm sorry but this Radio does not live up to 4...	0

Figure 4: Dataset with tagged sentiment

To further process the data, I removed punctuation and applied regular expressions, tokenization, and lemmatization in order to give the model consistent, clean data to work with when testing on a test set and applying to SD comments. Below is a snippet of the transformed body text for each review:

star_rating	review_body	sentiment	cleaned_reviews
5	What a great stove. What a wonderful replacem...	1	great stove wonderful replacement sort antique...
5	worked great	1	worked great
5	Part exactly what I needed. Saved by purchasi...	1	part exactly needed saved purchasing
5	Love my refrigerator! ! Keeps everything cold...	1	love refrigerator keep everything coldwill rec...
5	No more running to the store for ice! Works p...	1	running store ice work perfectly

Figure 5: Cleaned review body text added

After processing the Amazon review data that will be used to build the model, I needed to gather comments from SD to allow the model to run sentiment analysis on. Unfortunately, the SD website does not have an API. However, I was able to web-scrape it using BeautifulSoup, which is a Python package for parsing HTML and XML documents. My program iterates through each product on the SD front page, and for each product, pulls all of its comments and puts them into a list.

3 Features and Algorithms

3.1 Feature Generation

When deciding which features to use to achieve the highest quality model, I narrowed my choices to TF-IDF (Term Frequency Inverse Document Frequency) and N-grams, because these two methods for scoring words are traditionally very effective in automated text analysis.

TF-IDF calculates how important a token is to a document in a collection or corpus. Greater weights are placed on tokens that appear many times in a small number of documents. The weight for each token in the reviews was found by the following:

$$tf_{t,d} = \begin{cases} 1 + log(count(t,d)) & if\ count(t,d) > 0 \\ 0 & otherwise \end{cases}$$

$$idf_t = log\frac{N}{df_t}$$

$$w_{t,d} = tf_{t,d} * idf_t$$

where t represents the term (word), d represents the document (review), N represents the total number of documents (reviews), and df represents the number of documents that have the word t.

To begin generating features for TF-IDF, the tokens in the cleaned review body needed to be transformed into a vector, so that the appearance of each word in the corpus is translated into a number instead of text. Below is the transformed vector of each review (represented by the rows) and the tokens (represented by the columns), which combine to create the features, which are the numeric values that are filled in throughout:

	110	220	3rd	4000	add	advertised	air	alkaline	antique	...	watching	water	well	wheel
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.344896	...	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
5	0.000000	0.160707	0.160707	0.000000	0.000000	0.160707	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	0.301088	0.000000	0.000000	0.264661	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.408248

Figure 6: TF-IDF

The other method that was used to create features was by taking the corpus and creating N-grams, which are combinations of adjacent words of length *n* (What Exactly). Due to computational limitations, I limited my N-gram feature creation ton only include unigrams (the tokens themselves) and bigrams (adjacent tokens). An example illustration is shown below:

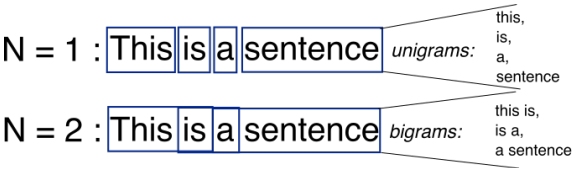
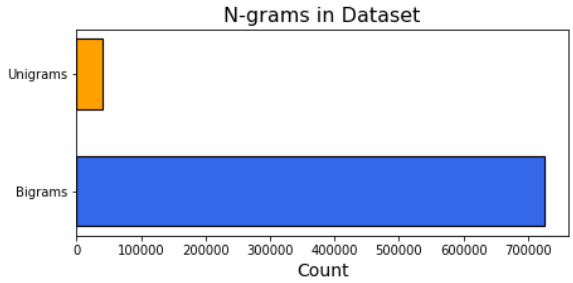


Figure 7: Example N-grams

After creating unigrams and bigrams from the review corpus, I looked at the distribution between unigrams and bigrams in my dataset and derived a formula for the total number of N-grams that are in the dataset using unigrams and bigrams as features:



$$TotalFeatureCount = \left(\sum_{i=1}^{TotalNumberofReviews} 2 * TokenCount - 1\right) - Duplicates$$

Figure 8: Count of unigrams and bigrams in dataset

Below are some examples of the unigrams and bigrams found in the reviews:

additions to	tourniquet
this cape	fridge up
incredible if	boring looking
over priced	speakers tv
garage didn	read just
file some	amazing message
section the	mass
dish signal	attachment should
compressor sound	gives real
cook directly	flow started

After creating the features using N-grams and TF-IDF, it's time to move onto the algorithms that will be used to train the model.

3.2 Machine Learning Algorithms

The two different algorithms that were chosen to train the model are Naïve Bayes and Logistic Regression, because they are both efficient to run, easy to explain the results for, and are effective algorithms for classification problems.

Naïve Bayes is a popular supervised machine learning technique for classification. The Naïve Bayes classifier is implemented under the assumption that all of the features are independent of each other, which is called the *conditional independence assumption*. Below is the formula used to calculate the probability of a class (positive or negative sentiment) associated with a given token:

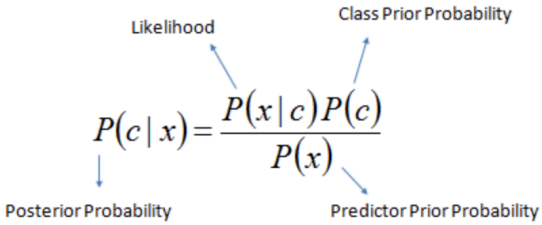


Figure 9: Naïve Bayes algorithm

The conditional independence assumption leads to the inquiry if a feature in the English corpus is dependent on another feature's appearance. In reality, the conditional independence assumption does not hold for text data, since tokens are conditionally dependent on each other. However, the Naïve Bayes model tends to perform well on text data despite making this assumption (Properties of Naïve Bayes).

The other machine learning method used to build the model was Logistic Regression. My intuition was that Logistic Regression would work extremely well since it's made to predict categorical data, works well on large data sets, and would also provide me some hints about the relative importance of the features through the coefficients. Like Naïve Bayes, Logistic Regression also makes the assumption that the independent variables (features) are independent of each other.

4 Experiments and Evaluation

After gathering my tokens using N-grams and TF-IDF as well as algorithms using Naïve Bayes and Logistic Regression, it was time to experiment and determine which feature and algorithm combination performed the best for an unseen dataset. Below is

the high-level process flow of running the experiments:

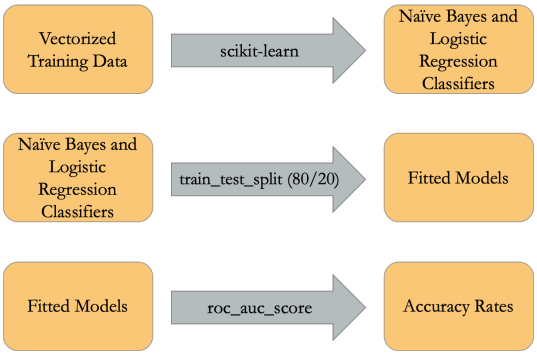


Figure 10: Process flow for experimentation and evaluation

As shown in the figure above, I ran each feature and algorithm combination to and used an 80/20 split for training and testing on the 200k Amazon review dataset in order to determine the model that generated the highest accuracy rate.

For Logistic Regression, the optimal inverse regularization parameter (known as 'C') needed to be chosen for each set of new features. The C parameter is the inverse function of Lambda, which serves as a regulator to prevent the model from overfitting. A graph is shown in Figure 12 representing the different C parameters that were tested with the model that performed the best. Below is the graph which shows accuracy performances from the model on the test set, which is represented by the ROC AUC (Area Under Curve) score:

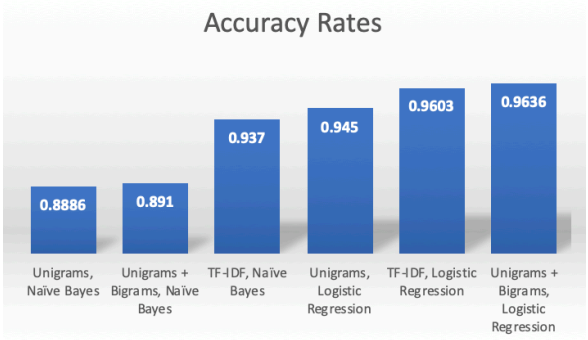


Figure 11: Accuracy rates for different feature and algorithm combinations

As shown above in Figure 11, the experiments show that the highest accuracy rate was achieved by using both unigrams and bigrams as features and Logistic Regression as the machine learning algorithm. As mentioned earlier, an evaluation of the best "C" parameter was completed to attain these results. Below is a chart that visualizes accuracy score using different inverse regularization parameters with the unigram + bigram and Logistic Regression Model:

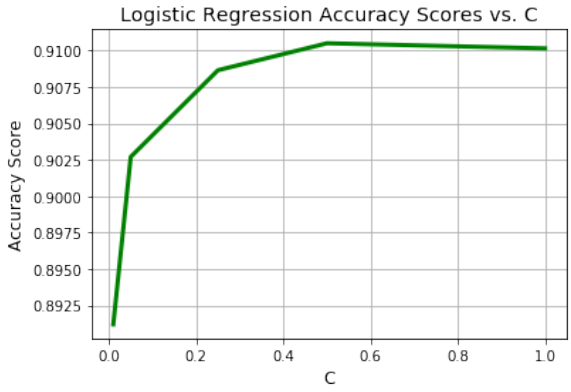


Figure 12: Unigram and Bigram Logistic Regression Optimal Inverse Regularization Parameter Evaluation

Figure 12 showed me that the optimal "C" to use for the model is 0.5, since it was able to achieve the highest accuracy results in a given test. Now that the best performing model has been created and optimized, we can further evaluate the performance by using a confusion matrix as shown below:

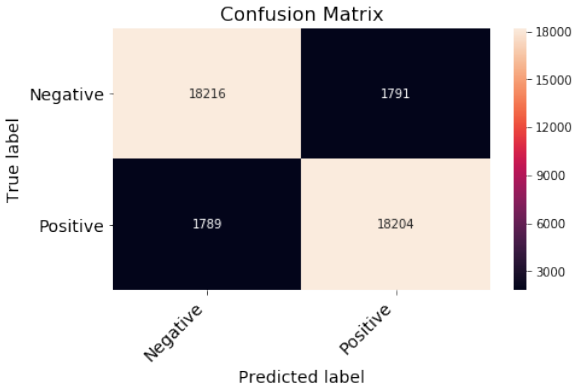


Figure 13: Confusion matrix for model evaluation

Figure 13 above shows how well the model predicts by giving us results for true positives, false positives, true negatives, and false negatives in the evaluation for the model's performance on the 20% test set (total of 40k reviews). These evaluation metrics can help us compute the precision, recall, and F1 score of our model, which are defined by the following:

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives}$$

$$F1\ score = \frac{1}{2}\left(Precision^{-1}\ +\ Recall^{-1}\right)^{-1}$$

Measure	Result
Precision	0.9104
Recall	0.9105
F1 Score	0.9105

Figure 14: Accuracy measures for model

As shown in the figure above, the model is well-rounded and does not exert any bias towards a single class based on the almost identical precision and recall scores. Taking a look at the most influential features, below are some of features that are most telling of a positive review and features that are most telling of a negative review, as they have the highest absolute value coefficients for each of their respective classes:

```
('perfect', 3.6662079741359275)
('excellent', 3.34216952995441)
('great', 3.2395829031268732)
('not bad', 3.2374295231360457)
('not too', 3.061010028050901)

('not good', -3.1453525510418134)
('junk', -3.08546694318188)
('terrible', -3.0306340401981413)
('horrible', -2.9966710216177126)
('poor', -2.9771031785063617)
```

As shown above, unigram-bigram model's usefulness is confirmed, both top influential positive and top influential negative features each include at least one bigram in them. For positive reviews, the bigrams "not bad" and "not too" are shown to be good indicators of a positive review, whereas if a user writes the pattern "not good," it can clearly indicate to the model that a negative review is warranted.

5 Applications

5.1 Application to SD

After evaluating the model, I can now apply it to predict sentiment on comments for products on SD. There can be any number of comments on a product page, so I derived a simple metric to use that assesses the quality of a product based on how many of its comments have positive sentiment, as shown:

$$PositiveFeedbackRating = \frac{Number\ of\ comments\ with\ positive\ sentiment}{Total\ number\ of\ comments}$$

Figure 15: Formula to calculate positive feedback

Below are some of the SD comments that the model evaluated on, and predicted 0 for negative sentiment and 1 for positive sentiment. Keep in mind, there is no ground truth / labeled data for the SD comments, so the result of these is based on the model training on unigrams and bigrams from Amazon review data and using Logistic Regression to predict sentiment.

```
one comfortable christmas 1
using mouse almost 4 year work great 1
skull trooper always popular huge selling halloween costume 1
sleeping bag warm however dont like hard roll put back bag 0
yep two reason pay shipping bummer 0
link doesnt work neither searching website 0
```

As it shows above, it seems that the model is performing fairly well on the unseen and unlabeled comments found on product pages on SD. Furthermore, these example comments have had stopwords removed and only contain the lemma of the original word that it represents, which is why it may be nonsensical when reading them.

Below is part of the final DataFrame, which shows only the product name and its predicted positive feedback rating by using the formula in Figure 15:

Item	Positive Feedback Rating
Kids 36" x 48" 4.5 lb Weighted Blankets (Vario...	0.111111
Kohls Cardholders: Farberware Cookstart Diamon...	0.800000
Green Toys Ferry Boat with Mini Cars Bathtub T...	0.714286
Sylvania Day by Day Multi-Color Changing LED 7...	0.590909
4-Ct Teacher Created Resources 5-Minute Sand T...	0.388889
1TB Sabrent Rocket NVMe 4.0 Gen4 PCIe M.2 Inte...	0.526316
100-Piece Playmags 3D Magnetic Toy Blocks	0.692308
Samsung HMD Odyssey+ Windows Mixed Reality Hea...	0.575758
Style & Co Women's Boots (Riding, Scrunched, A...	1.000000
Prime Members: 3-Months STARZ Channel	0.615385

Figure 15: Part of the final DataFrame

As shown above, this program is extremely flexible in the way that since SD constantly adds new products to its front page, it scans the website

each time it runs to capture those new products and come up with feedback ratings for each.

5.2 Applications to other languages

Now that the model has shown that it works well on the English language, one can imagine: how well would the model perform on different languages? Given that the bigram model considers adjacent tokens being related to each other, my intuition is that languages that have adjacent tokens that correspond are bound to perform much better using this approach compared to languages that have words modifying other words from further away in the sentence.

As represented by Section 4's most influential n-grams, the majority of these features that cause the model to significantly sway in one direction vs. the other (extreme positive vs. extreme negative sentiment) are adjectives, which modify nouns. English is head-initial and has prepositions, where verbs and adjectives come before the object that they modify, and the adjective strictly follows the verb and strictly precedes the noun. However, this contrasts to other languages, such as Japanese, where it embodies head-final properties and has postpositions, which means that verbs and adjectives follow their objects rather than preceding them (Head-Directionality Parameter). Even though the two languages have very different sentence structures, as shown from the influential bigrams, as long as the adjectives that modify nouns are adjacent in the sentence, then the bigram model is hypothesized to work well with the language, given that the most strike features that influences the model's decisions are adjectives.

6 Related Work

There have been many research papers done with sentiment analysis, and many that are popular entail using Twitter data or IMDb review data in order to train their models.

- A group of students at the Heritage Institute of Technology performed Sentiment Analysis of Review Datasets using Naïve Bayes and K-NN Classifier on IMDb reviews and Hotel reviews.
- Students at Columbia University performed sentiment analysis on Twitter Data, and they broke down human slang and emojis to assign sentiment scores.

7 Limitations

- Items on the main page of SD are not categorized, so a 1:1 match with Amazon reviews with the exact category cannot be performed.
- Better training data would give better true accuracy for sentiment on SD comments. Sometimes, the comments on the product page become casual and conversational rather than about the product itself. My favorite example of this is with Royal Dansk Butter Cookies:

11-05-2019 at 03:49 PM

In for some sewing supplies.

11-05-2019 at 03:35 PM

Can't wait to give these to my mom so she could put yarn in the box.

11-05-2019 at 05:12 PM

Whose grandma posted this?

- Relative positioning of tokens is ignored with Naïve Bayes and Bag of Words model. A dependency parser might help mitigate this issue.
- Slang, sarcasm, and word sense ambiguity may adversely affect the model's decision-making.
- Sparse data – people type differently and won't necessarily type with the same words or word-patterns as the next person. The model also relies on the SD community providing enough comments to run analysis on.

8 Conclusion

In this paper, I presented my process and results for sentiment analysis on Slickdeals comments. This was made possible by using Amazon review data that was readily available online to train the model, and then developing the most accurate model, which was determined by the ROC AUC score. The results show that the combination of unigrams and bigrams as features and Logistic Regression as the algorithm created the most accurate model. In the future, I'd like to explore

more ways to develop an even better model, perhaps by in part-of-speech tagging or dependency parsing in the feature set. Additionally, I'd like to explore sentiment in human conversations as well, as much research is done with review data and Twitter data, which is very polar in that dialogue is generally very one-sided.

Special thanks to Professor John Hale and Lisa Lipani for a great semester.

9 References

"Amazon Reviews." Cloud Object Storage, s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt.

Anonymous. "I Get Paid To Write Fake Reviews For Amazon." Cracked.com, Cracked.com, 31 Aug. 2016, www.cracked.com/personal-experiences-2376-i-get-paid-to-write-fake-reviews-amazon.html.American Psychological Association. 1983. *Publications Manual.* American Psychological Association, Washington, DC.

"Head-Directionality Parameter." Wikipedia, Wikimedia Foundation, 8 Dec. 2019, en.wikipedia.org/wiki/Head-directionality_parameter.

Klein, Ewan, et al. "Natural Language Processing with Python." NLTK Book, www.nltk.org/book/.

Maklin, Cory. "Metrics For Evaluating Machine Learning Classification Models." Medium, Towards Data Science, 21 July 2019, towardsdatascience.com/metrics-for-evaluating-machine-learning-classification-models-python-example-59b905e079a5.

"Properties of Naive Bayes." Properties of Naive Bayes, 2008 Cambridge University Press, 7 Apr. 2009, nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html.

"The Best Deals, Coupons, Promo Codes & Discounts." Slickdeals, slickdeals.net/.

"What Exactly Is an n Gram?" Stack Overflow, 1 Oct. 2013, stackoverflow.com/questions/18193253/what-exactly-is-an-n-gram.