Final Project for Data Analysis

Sentiment Analysis of Amazon Pantry Data

Paul Pluscht

Fall 2021

1 Introduction

For my final project, I chose to perform some sentiment analysis on Amazon Pantry reviews. Amazon Pantry is a service launched in April 2014 that allowed consumers to purchase no-perishable food items and other common household goods. The dataset used in this project comes from Dr. Jianmo Ni of the University of California San Diego [1]. Specifically, the dataset contains the text from over four-hundred thousand Amazon Pantry Reviews with their star labels. I want to use this dataset to answer the following two questions. First, how many words does an ideal classifier use to predict ratings? In other words, how many words are superfluous in understanding sentiment? Secondly, I want to determine a list of around 100 words that are most significant in understanding sentiment.

2 Introducing the Dataset and Summary Statistics

2.1 Summarizing the Dataset

This dataset contains data for over four-hundred thousand reviews for Amazon pantry products. The first few rows of the dataset can be seen in the figure below.

	overall	verified	reviewerID	asin	reviewText	review_length
0	5.0	True	A1NKJW0TNRVS7O	B0000DIWNZ	good clinging	13
1	4.0	True	A2L6X37E8TFTCC	B0000DIWNZ	fantastic buy and a good plastic wrap even th...	243
2	4.0	True	A2WPR4W6V48121	B0000DIWNZ	ok	2
3	3.0	False	A27EE7X7L29UMU	B0000DIWNZ	saran cling plus is kind of like most of the c...	1692
4	4.0	True	A1OWT4YZGB5GV9	B0000DIWNZ	this is my go to plastic wrap so there isnt mu...	585

Figure 1: Head of Dataset

The 'overall' column simply states the star rating the particular review received. There are no missing values in this column. The values range from one to five, and there are no fractional star values possible. There is a noticeably higher number of five star reviews than other ratings as will be discussed in the next section.

The 'reviewerID' and 'asin' columns represent the unique user IDs and product IDs respectfully. The users in the dataset wrote an average of 1.9 reviews while the most prolific reviewer wrote 202

reviews. The products in the dataset recieved an average of 43 reviews each. The most reviewed product were crest 3D white strips with 6537 reviews. The most highly rated product with over 100 reviews were Greenie's Dental Treats.

Finally, the 'reviewText' and 'reviewLength' columns contain the text from the review and the number of characters in each review. The reviews had an average of 133 characters each. There is a strictly decreasing relationship between star rating and average review length that will be discussed more in the next section.

2.2 Sources of Bias

Before building our model, it is important to look for any sources of bias that may lead to skewed results. First, how are the ratings in our sample dataset distributed? As can be seen in the graph

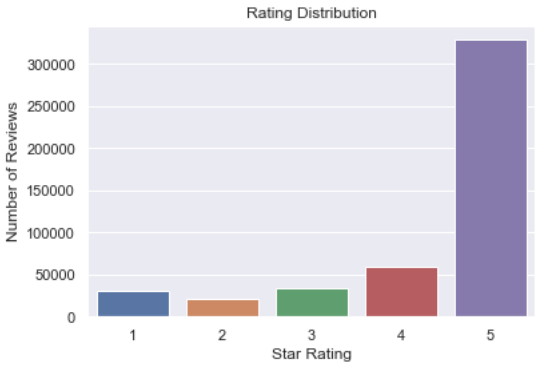

Figure 2: Number of reviews per star rating

and confirmed by a chi-squared test, there is a noticeable skew towards five star reviews in the dataset. Specifically, 69.6% of the sample are five star reviews. As a result, words that convey positive sentiment will have more training examples than those with a negative sentiment. As a result, our model may perform better on identifying this portion of the dataset.

Next, the goal of this project is to identify words that convey a certain sentiment, not to build an ideal classifier. As a result, we want to check for other sources of bias in the dataset. One of the largest potential sources of bias is total word count. If longer reviews tend to be one star reviews, our model may be predicting based off of word count, not sentiment. Below, there is a box and whisker plot comparing the star rating as a function of length.

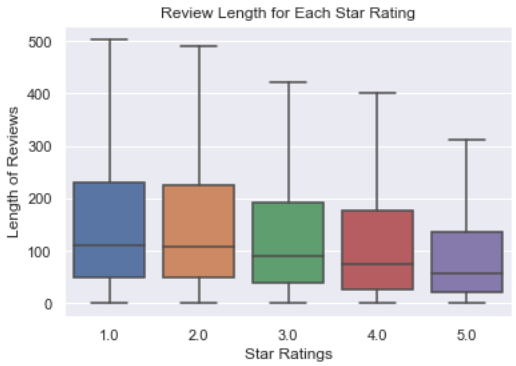

Figure 3: Length of reviews vs. Star Rating

As can be seen in the graph, there is a strictly decreasing relationship between star ratings and review length. However, it is not significant. When a multinomial logistic regression is trained using only review length as an input, all of our coefficients are zero. As a result, we can treat this slight skew as insignificant.

These two are not the only potential sources of bias in our dataset. After all, the dataset contains reviews of thousands of products with different reviewers. If a certain reviewer leaves only positive reviews on cookies while a different reviewer leaves only negative reviews for trash bags, this could lead our model to weighing words associated with cookies and trash bags unfairly. This means our model's judge of sentiment could be skewed. However, since there are simply too many reviewers and products to test for this, we are going to assume that all other forms of bias are negligible.

3 Experimental Design

First, I need to mention a disclaimer. When trying to build a model using all the star ratings, my results were very poor. I will explain why this happened in the following sections. To get some meaningful results, I instead simplified the classification problem to 5 star reviews vs. one star reviews. Since there are more obvious differences between a five star review and a one star review, the models trained on this portion of the dataset performed far better.

To devise a model that attempts to understand the sentiment of words, we first need to reformat our data into a form that considers the implications of each word. To do so, we will create a word count matrix or a 'bag of words.' This creates a matrix that records the number of occurrences for each word in each review. For example, consider the following example review: "This product was great." In the row corresponding to this review, there would be a one in the column corresponding to "this", "product", "was", and "great." For words not included in this review, there would be a zero in that column. In theory, this matrix would need one row for each review in the dataset, and enough columns to fit every word mentioned in every review (over 25,000 words for our dataset) [4]. However, this number of columns requires an impractical amount of memory and is unnecessary. Instead, we will fix the max number of words at 10,000. We will see later that the ideal model uses far less than 10,000 words anyways.

3.1 Insights from the Bag of Words

Below is a figure that shows the first few rows and columns of our bag of words.

	05	0g	0mg	10	100	1000	1010	1012	1015	103	...	ziplock	zipper	zits	zombie	zone	zoo	zoom	zucchini	zukes	z
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

Figure 4: Head of Bag of Words

As can be seen, all of the depicted entries in our bag of words are zero. This is not a programming error, and there are non-zero entries in our dataset. However, the overwhelming majority of the

entries in our dataset are zeros. To get a better understanding of how words are distributed in our bag of words, look at the figure depicted below.

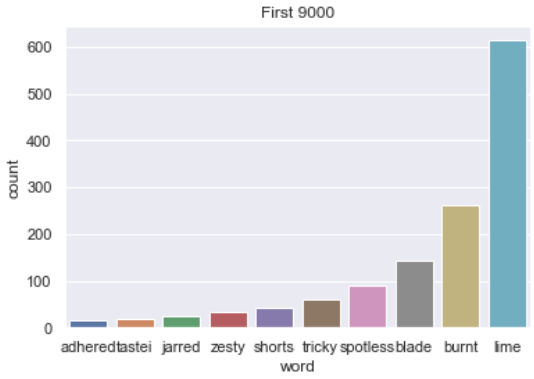

Figure 5: 9000 Least Used Words

This figure shows the 1000 most used word, 2000 most used word, 3000 most used word, and so on until you get to the least used word in the dataset. The 1000 most used word, "lime," is used just over 600 times. When you consider that our dataset has around 350,000 reviews, that is clearly not a huge sample size. However, the most used word, "great," is used over 88,000 times. Clearly, there is a huge disparity in the times each word is used. When looking at the dataset like this, it becomes clear how 350,000 reviews is not nearly as much data as you would originally assume. This is why I had to simplify the classification problem. The input space is too sparse to train simple, high-performance models on. While there are some advanced neural network architectures that would perform well on the whole dataset, those models make answering our original research question too difficult to answer. These models are often "black-boxes" whose architecture makes answering our simple research questions difficult to understand.

There are also some potential problems waiting at the top-end of our most used words list. As mentioned in the potential sources of bias section, there are more five star reviews than one star reviews. As a result, there are more words that convey a positive sentiment than there are words that convey a negative sentiment. In the figure below, we can see this issue.

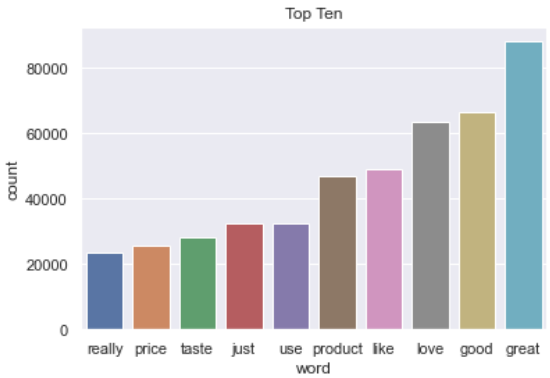

Figure 6: Ten Most Used Words

Of the ten most used words, four words convey a traditionally positive sentiment while the rest are traditionally neutral. Unfortunately, our model may not be able to precisely weigh negative

sentiment words due to the lack of data.

3.2 Training and Model Selection

To create a model to select the optimal number of words, I was originally going to tune a logistic regression model, a support vector machine, and a random forest classifier. However, the support vector machines and random forest classifiers performed far worse than the logistic regression model and will not be mentioned going forward. To train an optimal logistic regression model, I will train a series of logistic regression models with a lasso error term in Python's Scikit Learn package [2]. Specifically, our lambdas that weigh the L1 penalty will range from 0.001 to 1000. We will evaluate our models by comparing the area under the receiver operating characteristic curve (AUC for short). We will then count the number of non-zero weights to determine the number of words used in our optimal model.

To create our list of most influential words, we will repeat the same process. However, instead of picking the lambda that optimizes the AUC, we will instead pick the lambda that produces a model of around 100 words.

For those who wish to repeat this experiment, I will make one important technical note. Our input data is very sparse, and model training can be optimized if we store our data in a data structure that is optimized for sparse matrices. In traditional arrays, the value at every entry is stored in memory. If most of these are zeroes, this can considerably slow down our computational speed. Instead, our matrix will be stored as a Compressed Sparse Row matrix or CSR matrix for short. Instead of storing our matrix as a series of 1D row arrays, a CSR matrix stores our data in three arrays: V, col_index, and row_index. Our V array simply contains the values of all non-zero values in our array. The col_index array corresponds to the column index of our non-zero values and has the same length as V. The row_index array contains information that is used to determine where the next row starts given the col_index array and has length corresponding to the number of rows plus one. The example below should help understanding.

Example 1 *CSR Representation*

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 5 & 6 & 7 & 0 \end{bmatrix}$$

In this case, our V is [1,2,3,4,5,6,7], and our col_index is [1,0,1,1,0,1,2] (assuming indexing starts at zero). The row_index array is a little trickier to understand. In this case, it is [0,1,3,4,7]. This is because the first row starts at element zero of V, the second row starts at element one of V, the third row starts at element three of V, the fourth row starts at element four of V, and the last row ends at element seven of V.

This representation of our data is much more memory efficient for our sparse matrix and dramatically increases speed for our computations. To train one logistic regression model using a standard numpy array takes around 10 minutes for our dataset. It takes under 30 seconds using a scipy csr matrix. [3]

4 Results

4.1 Determining the Ideal Number of Words

To answer the first research question, we will measure the AUC for 25 different lambda values logarithmically spaced between 0.001 and 1000 as described in the previous section. The graph below shows this process.

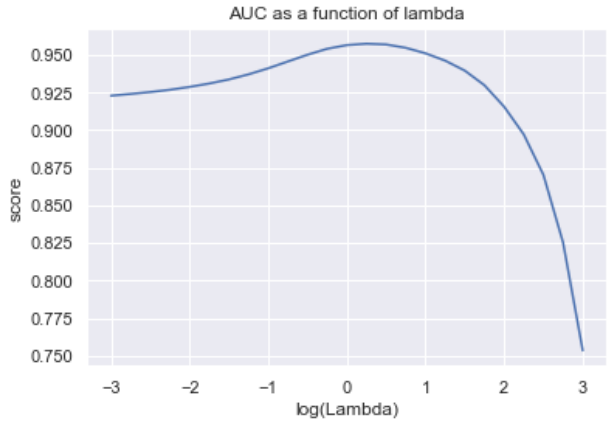

Figure 7: AUC as a function of log_{10}(Lambda)

Our optimal lambda is around 1.77. This model had an AUC of 0.957 and used 3541 words. Below we can see the confusion matrix for our ideal model.

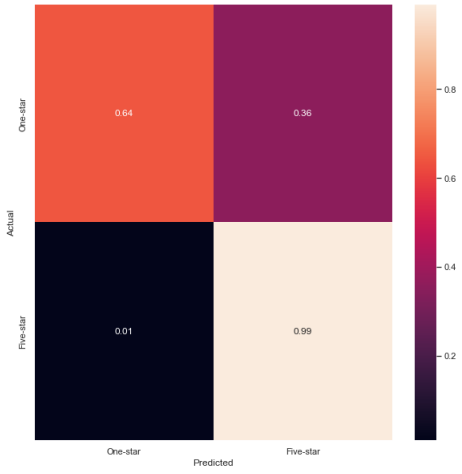

Figure 8: Confusion Matrix of our Ideal Model

Unfortunately, our skewed dataset did cause some problems. Our model over-predicts the dominant class, 5 star ratings. Unfortunately, limiting the size of our training set to balance out the classes lead to worse results, so this is the best we can do for now.

4.2 Determining the 100 most influential words

While Figure 7 shows large fluctuations in the performance of the model at various lambdas, what if we measured our model performance as a function of words used instead?

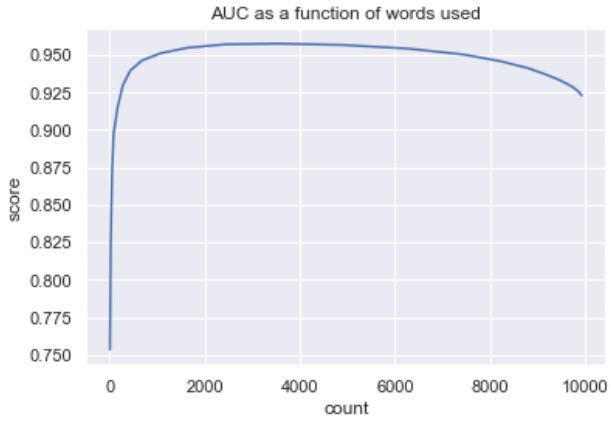

Figure 9: AUC as a function of words used

As can be seen here, AUC is fairly flat between 1000 and 8000 words used, and we still get a decent AUC score of 0.897 when we reduce the number of words used to 90. Lets look at the confusion matrix of the 90 word model.

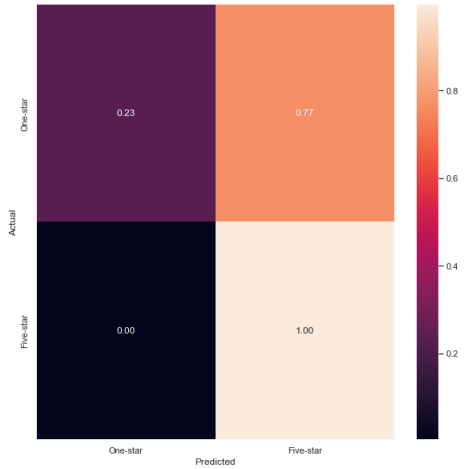

Figure 10: Confusion Matrix of our limited model

Unfortunately, our skewed dataset did cause some problems again. Our model more aggressively over-predicts the dominant class. However, a simple sanity check of the words used in the model shows that our model still works. We would expect positive words like good or great to have positive weights and vice versa. In our model, good, great, wonderful, best, amazing, awesome, and best all have positive weights, and awful, bad, broken, disappointing, disgusting, horrible, terrible, and worst have negative weights.

5 Conclusions, Improvements, and Future Directions

Unfortunately, there are some issues with my results that are primarily the result of the size of the dataset. Due to the classes in the dataset being skewed towards five star reviews, our models did over-predict the five star class. However, when I tried to balance the training set by removing five star reviews, the results got worse. As a future project, I would like to use the entire set of Amazon reviews to create a balanced training set. This should lead to models that perform better and do not over-predict the dominant class. Also, if we increased the size of our dataset, we should be able to try determining the significance of different n-grams. I tried testing this model on bi-grams instead of individual words, but the results were worse. With more data, a more precise model may be able to be built using more n-grams. Additionally, our model will likely be able to accomplish the 5 category task if we had a larger set of data.

In addition to getting more data, our model would likely be improved if we replaced words with the same root with the same token. For example, our current 90 word model used both 'yum' and 'yummy.' Since these words convey the same sentiment, our model would likely be improved if it consolidated these entries into one token.

Looking forward, this experiment can be expanded into attempting to understand the sentiment of phrases and sentences instead of simply just individual words. Modern neural network architectures like long short-term memory RNNs are able to learn the sentiment of entire paragraphs and sentences. Our simple logistic regression does a great job understanding the sentiment of low n-grams, but language is more complicated than understanding a few words at a time. While these models reduce our understandability of the model, they will outperform our simple model.

6 References

1. Jianmo Ni, Jiacheng Li, Julian McAuley. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. Empirical Methods in Natural Language Processing (EMNLP), 2019

2. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

3. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

4. Bird, S., Klein, E., Loper, E. (2009). Natural language processing with python. O'Reilly Media.