Chloe Mai - Springboard Capstone 2 Project (Oct 2021)

Motivation

Online reviews and ratings have become a more commonplace tool with the rise of e-commerce. They keep brands accountable and make them seem more trustworthy/reliable, if they have more positive reviews. The goal is to predict ratings from reviews.

Data

From October 1999 to October 2012, Amazon Fine Foods has collected 568,454 reviews from 256,059 reviewers on 74,258 products. https://snap.stanford.edu/data/web-FineFoods.html

Data includes: product ID, user ID, user profile name, helpfulness, score, time, summary, and text. Score ranges 1-5 and are classifiers. Summary and text are free text. They will be combined into a new variable called “review.” Product name was not included, so it will be difficult to interpret the products. Reviews may have some bias since 2% of reviews are reviewed by the same 260 people.

Import Data

The data is stored as a text file in a paragraph format. For example, this is one entry:

product/productId: B001E4KFG0
review/userId: A3SGXH7AUHU8GW
review/profileName: delmartian
review/helpfulness: 1/1
review/score: 5.0
review/time: 1303862400
review/summary: Good Quality Dog Food
review/text: I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labrador is finicky and she appreciates this product better than most.

To import the data, I created an empty dataframe with the column names given in the original data, such as “product/productID.” I filled each column by parsing out the text that came after each colon, and I filled each row by parsing out each entry by the blank row that separated entries in the text file.

Data Cleaning - Part 1

To clean the data frame, the following steps were taken:

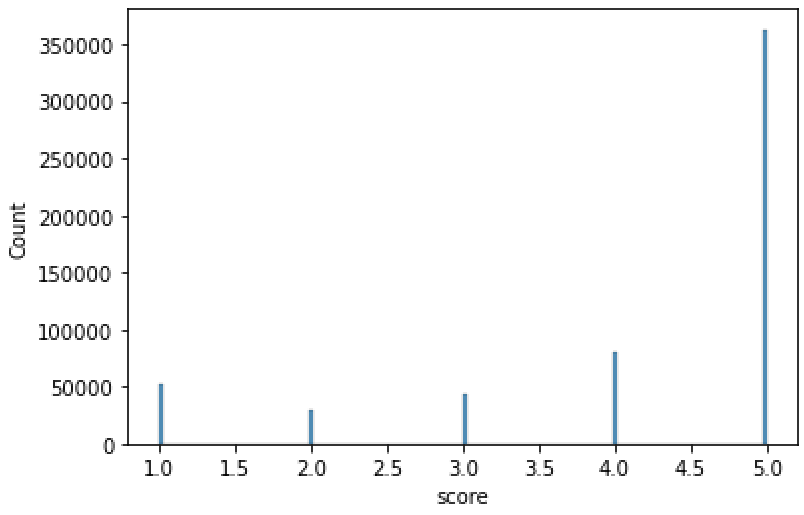
1. Re-labeled columns to simpler names.
2. Dropped null rows. The last row of the dataframe was entirely “null” because it captured the last blank paragraph space.
3. Converted columns into appropriate data types. Originally, all rows were objects. I converted time to a datetime, and I ensured all numeric columns were floats and all text

columns were strings. Helpfulness was originally a fraction written as text. separated the numerators and denominators as integers, and calculated the helpfulness fraction.

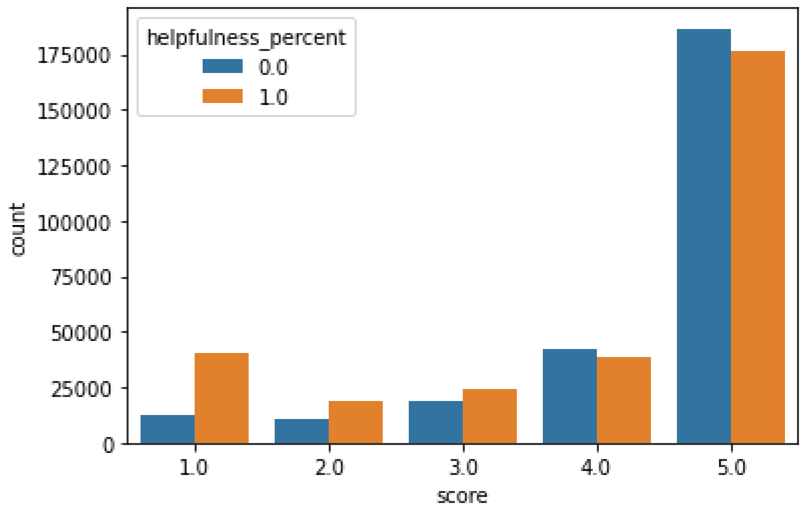
4. Combined text and summary as a new column called "review."
5. Dropped duplicate rows and kept the first instance. There were 281 duplicates.

Exploratory Data Analysis

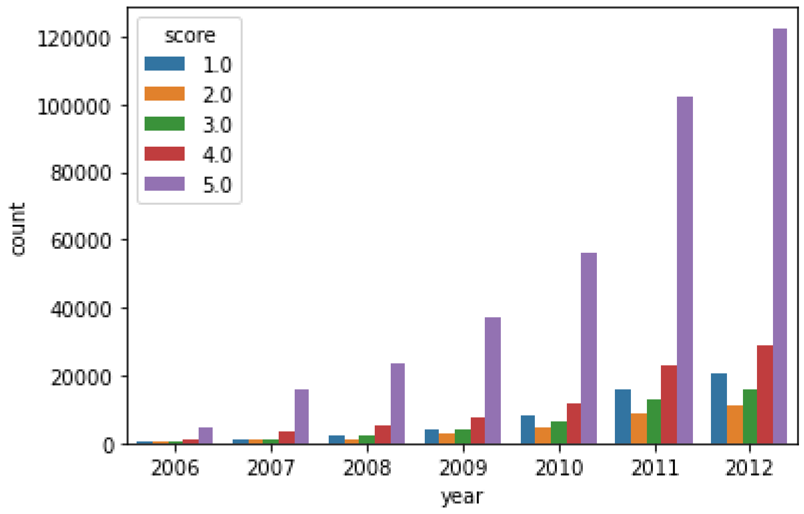
First, I examined scores aka ratings. There's an overwhelming number of 5's. The combined quantity of the remaining ratings (1-4's) is approximately half the quantity of 5's.



Second, I was curious about helpfulness. Through column plots and pair plots, I quickly realized that helpfulness wasn't a useful variable. The helpfulness fractions were either 0% or 100%. This means helpfulness was a tally instead of a comparison of helpful vs not helpful. Helpfulness ranged from 0 to 800+, but the majority were less than 5, so not many people will rate the helpfulness of each review. The ratio of helpfulness 1:0 was greater than 50% for lower scores (scores <= 3), and that ratio flipped for higher scores. This implies lower ratings were counted as more helpful. That's expected since we live in an age where e-commerce is known to have fake reviews. Oftentimes, products are inundated with positive reviews and have very few critical negative reviews.



Third, I explored how the ratings changed over time. The number of scores grew over time, but every year the majority of scores were still 5.



Fourth, I found the most reviewed products and most frequently contributing reviewers. They're hard to interpret because no product name was provided, and username doesn't have a consistent format.

Data Cleaning - Part 2

1. I created another classifier to use as the target variable. Instead of using score {1,2,3,4,5}, I made "rating" a binary column. Rating = 1 means that the score was 4 or 5. Rating = 0 means that the score was 3 or less. Essentially, there are high ratings and low ratings.

2. I made a function that would handle text wrangling, which was needed to process and normalize each review. The function pre_process_corpus used beautiful soup, tqdm, and re to process and normalize each review.

   For example, the original review was: “Delight” says it all This is a confection that...
   The processed review was: delight says it all this is a confection that…

3. Again, I dropped any rows that had na.

Prep Data for Modeling

This dataset had more than 500K entries. For the sake of speeding up modeling, I used a subset of 50K rows. The train and test split was 20% train and 80% test. Both train and test were processed through the function pre_process_corpus.

Feature Engineering

I tried CountVectorizer (bag of words) and TFIDF from sklearn to create features out of reviews. After trying a round of Logistic Regression, I found CountVectorizer to have higher accuracy (0.88 vs 0.77). Thus, the remaining model exploration only relied on the 2078 features created by CountVectorizer. These features became the X_train and X_test. The high/low ratings became y_train and y_test.

Model Comparison

With each model from sklearn, I created a confusion matrix and classification report.
Models tested:

- Logistic Regression
- Random Forest
- Gradient Boosting
    - Cross-validated learning rate and found 0.75 to have the highest accuracy score for training and validation.
- XGBoost

Model	**F1 for low ratings**	**F1 for high ratings**	**Accuracy**
Logistic Regression	0.74	0.93	0.88
Random Forest	0.62	0.92	0.86
Gradient Boosting	0.09	0.87	0.77

XGBoost	0.72	0.92	0.88

Model	**True Low Rating**	**False High Rating**	**False Low Rating**	**True High Rating**
Logistic Regression	1645	719	436	7200
Random Forest	1102	1262	108	7528
Gradient Boosting	121	2243	85	7551
XGBoost	1510	854	341	7295

Logistic regression was the best model because it had the highest F1 scores for both low ratings and high ratings. The confusion matrix further confirms that logistic regression was best at predicting low ratings, which is better for this data because of class imbalance -- there are about ⅓ as many low ratings to high ratings.

Hyperparameter Tuning

I tried 2 methods of hyperparameter tuning to try to improve the logistic regression's performance. They were Randomized Search and Bayesian Optimization.

Randomized Search resulted in best score = 0.8843 and best parameters were solver = lbfgs, penalty = l2, and C = 1. Those parameters were the same as the original logistic regression model. After running with this new initialized state, accuracy improved from 0.88 to 0.89 whereas the F1 scores remained the same and the confusion matrix was also similar (changes within true low/high ratings were within +/-15).

Through Bayesian Optimization, I found the best C = 0.87. The performance metrics and confusion matrix were also similar.

Model	**C**	**F1- low ratings**	**F1 - high ratings**	**Accuracy**
Original Logistic Regression	1	0.74	0.93	0.88
Randomized Search	1	0.74	0.93	0.89
Bayesian Optimization	0.87	0.74	0.93	0.89

In the end, hyperparameter tuning only marginally improved the model by increasing accuracy by 0.01, which could also be explained by randomized initializations.

Class Imbalance

Lastly, I tried to improve logistic regression's performance by fixing the class imbalance since we know that there's 2364 low ratings and 7636 high ratings. I tried balancing class weight, random under sampling, and SMOTE. The latter two methods came from library imblearn. When fixing class imbalance, I used C = 0.87 based on the Bayesian Optimized logistic regression model.

Model	**F1- low ratings**	**F1 - high ratings**	**Accuracy**
Bayesian Optimization Logistic Regression	0.74	0.93	0.89
Class weight = 'balanced'	0.74	0.91	0.86
Random Under Sampler	0.72	0.89	0.85
SMOTE	0.74	0.92	0.87

Model	**True Low Rating**	**False High Rating**	**False Low Rating**	**True High Rating**
Bayesian Optimization Logistic Regression	1646	718	423	7213
Class weight = 'balanced'	1952	412	966	6670
Random Under Sampler	2004	360	1172	6464
SMOTE	1784	580	688	6948

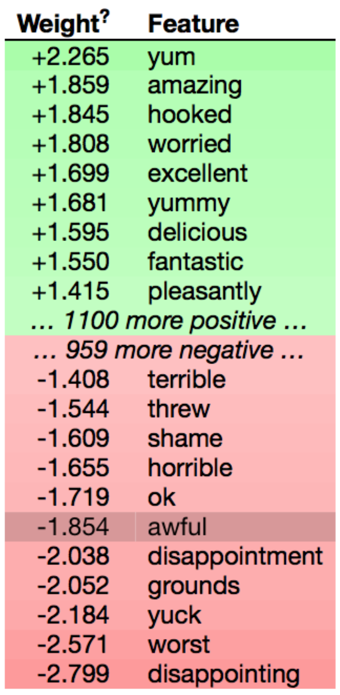
The performance metrics for these different class imbalance models were worse than the original Bayesian optimized model. It seems the more true low ratings the model corresponded with lower overall accuracy and F1 score for high ratings.

Best Model

After trying hyperparameter tuning and fixing class imbalance, the best model was logistic regression using C = 0.87 found through Bayesian Optimization because it had the highest accuracy and F1 scores.

Interpret Data

After deciding the best model, I was interested in exploring the results. I was interested to see which words were the most influential (highest magnitude) and which words were the most positive or negative (direction). I manually created a table, and then verified it with eli5.



Weight?	Feature
+2.265	yum
+1.859	amazing
+1.845	hooked
+1.808	worried
+1.699	excellent
+1.681	yummy
+1.595	delicious
+1.550	fantastic
+1.415	pleasantly
… 1100 more positive …	
… 959 more negative …	
-1.408	terrible
-1.544	threw
-1.609	shame
-1.655	horrible
-1.719	ok
-1.854	awful
-2.038	disappointment
-2.052	grounds
-2.184	yuck
-2.571	worst
-2.799	disappointing

There were 1100 positive words and 959 negative words. The words with the highest magnitude were disappointing, worst, and yum. 5 negative words and only 1 positive word had coefficient magnitudes > 1. This implies that negative words hold more weight in predicting ratings. I'm most surprised by the coefficient of "ok." It's the 7th most negative word. I would have expected it to be closer to 0 because "ok" is pretty ambiguous. It could be positive or negative depending on the comment's tone. For example, one could say, "It's ok!" to show approval or "It's only ok," to make a neutral to negative comment.

Future Work

If I were to improve upon this project, I would:

- Use more recent data if available to continue training and testing the model. This dataset is 9 years old, and it would be interesting to see how language or rating behavior has changed over time.

- Model with more data. To save time, I used 50K entries, and I know from time experiments that I could run a logistic regression model with 300K entries before my computer slows down a lot. If I had a different computer setup, I could run the full 500K dataset. We could test if more data improves the model.
- Try n-grams >= 2 to see if certain phrases influence the model.
- Make a recommendation tool if I had product names.