Undervalued Burritos of Yelp

Problem Statement

Reviews and overall star ratings on Yelp often contain irrelevant information to the specific question a potential customer may have. In particular, I am interested in finding restaurants with the highest quality burritos. Selecting the Mexican restaurant on Yelp with the highest overall star rating and the largest number of reviews could be a good place to start. However, this necessarily ignores whether the majority of reviews praise the quesadillas, the service, or the atmosphere. Perhaps the restaurants serving the best burritos are nowhere near the top of the overall star rating because they don't have a parking lot or vegetarian options (as an example).

This could be an invaluable method if successful because it will be extremely easy to generalize. The scope of this project is to find the best burritos, but other applications can benefit from similarly specified queries. Interested in yoga studios with the most popular teachers, laundromats with the fastest machines, or bars with the cheapest happy hours? The possibilities are limitless and hold great potential for identification of underrated establishments in all realms of business.

Dataset and Cleaning

Yelp, the massive review platform, is kind enough to publicly offer reviews for a huge swath of businesses across the country and world. Split into several large JSON files, you can filter review information based on user, business, category, etc. The immediate problem is the sheer size of the data. There are nearly 7 million reviews contained from close to 200,000 businesses.

The first step in getting my hands on the information I'm interested in is downloading the files. They come in a nested JSON format, and are quite large. The file containing all the reviews is almost 6 gb. This is too large to hold in memory so I have to parse through line by line and pull out the reviews I need. Because I am interested in burritos specifically, I need only the reviews for establishments that sell some kind of mexican or tex-mex food.

Thankfully, the JSON file that contains the information on all the businesses can be loaded into local memory and pushed into a dataframe. Once this is done, I iterate over the dataframe and look for the 'restaurant','mexican' and 'tex-mex' tags in the 'business category' dictionary associated with each business. I then pull out the restaurant name and unique restaurant ID (to separate out all the chains) and use that information to parse the reviews file line by line looking for reviews associated with the restaurants I am interested in. This reduction of data results in just over 4,000 restaurants of interest with almost 390,000 reviews. Still large, but much more manageable. There are a few other intricacies I need to complete to get all the information I need in one place, namely transferring information on restaurant name, location, and region from the businesses file.

The last step I need to take in cleaning the dataset is pulling out all the reviews that mention ‘burritos’. This set will be my validation set where I run sentiment analysis on specific sentences to find undervalued burritos. To pull out all the reviews that mention my word of interest, I can use pandas df.str.contains() method to return every review that has some variation of the word burrito in it. Saving this to a new dataframe and dropping the same rows from the large dataset (so I don’t train on these reviews) leaves me with a burritos dataframe of about 50,000 reviews and a finalized reviews dataframe of nearly 330,000 reviews.

Now that I have the reviews I’m interested in with all the information I’ll need to answer my question, I can proceed to some exploratory analysis.

Exploratory Data Analysis (Part 1)

My first goal is to make a model that can accurately predict what star rating is associated with a block of text. I’ll have to begin by taking a look at some numbers that describe my objects of interest. Below is a count of the number of reviews associated with each star rating. The number of 5 star reviews dwarfs every other star rating. This may lead to my model predicting everything as a 5 star review unless I balance the classes. Duly noted, but I’ll move on for now.

Figure 1: Number of reviews in my whole dataset, binned by stars

The next figure looks at the mean character count for all reviews by star rating. This plot looks a little more uniform, the concern being that if there were significantly more words for a certain type of review that a model might be much more successful at predicting that class. Interestingly enough, 5 star reviews have the lowest mean length. I would not have predicted this beforehand.

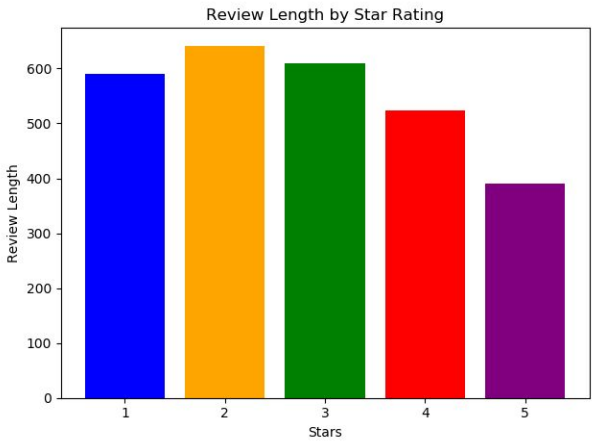

Figure 2: Mean length of reviews binned by stars

Now that I have a little intuition for how review length and count of star ratings are distributed, I want to next looks at the restaurants the make up the 330,000 reviews. Below is a list of the top 15 restaurants garnering reviews (for my purposes here, individual chain's reviews are all added together). As I suspected, chain restaurants make up the majority of the top 15, but not nearly to the extent that it might become problematic. Of all 330,000 reviews in my dataset, only about 2% are for a Chipotle. This is of interest to me because I sincerely doubt that the most undervalued burritos come from somewhere like Chipotle or Taco Bell. If these places made up the majority of my reviews I might have to come up with a different plan.

Chipotle Mexican Grill	6513
Tacos El Gordo	5283
Taco Bell	4504
Chili's	4062
Mesa Grill	3331
Nacho Daddy	3113
Cafe Rio	2913
Lindo Michoacan	2631
El Dorado Cantina	2529
Border Grill	2280
La Santisima	2074
Roberto's Taco Shop	2011
Joyride Taco House	1985
El Pollo Loco	1957
Barrio Queen	1954

Figure 3: Top 15 restaurants garnering reviews (here, all Chipotles and all Taco Bells)

The list above is also interesting because it seems to be biased towards restaurants in the American West. Specifically, I know 'Barrio Queen' is a well respected traditional mexican restaurant in Arizona, 'Tacos el Gordo' a highly regarded California mainstay, and 'Nacho Daddy' a Nevada nacho champion. I previously presumed the dataset from Yelp included a large swath of the nation but I better take a closer look at the distribution of states and regions to see what I'm working with.

Las Vegas	99034
Phoenix	65851
Scottsdale	25754
Charlotte	15847
Mesa	11925
Henderson	11528
Pittsburgh	10407
Tempe	9781
Chandler	9476
Gilbert	9409
Glendale	6136
Cleveland	5725
Madison	5203
Peoria	4794
North Las Vegas	3558

Figure 4: Top 15 cities represented

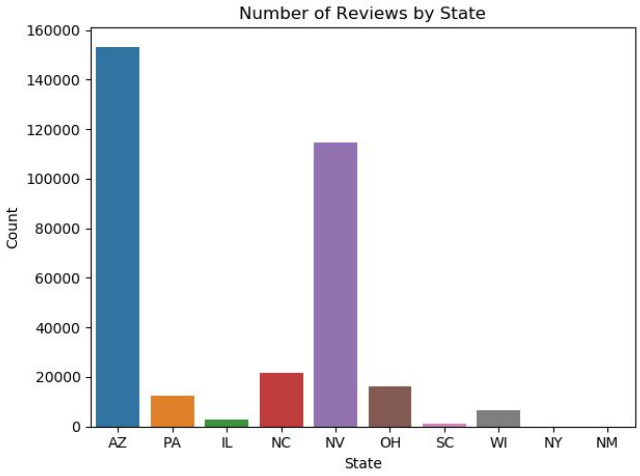

Figure 5: Number of total reviews binned by state

As the two images above confirm, I actually have a fairly random distribution of states. By far, Arizona has the most reviews as it seems that all their large cities have been included. After that, Las Vegas and the surrounding areas make up a large chunk of the reviews set. Moving forward, it might be prudent to split my analysis into different areas or regions so my undervalued burrito list is not a purely Arizona or Nevada heavyweight clash. For now I'll leave it as it will not influence my model building.

Word Tokenizing, Vectorization and a Baseline Model

Now that I understand my data (and some potential pitfalls) a little bit better, I can proceed to the second part of cleaning that goes with this project, natural language processing. I plan to use a simple bag of words model to represent all the reviews in a vectorized form that a model can understand.

To begin, I will work with the standard NLTK library, utilizing its corpus and built in tokenization to clean the plain text reviews. First I write a function that will allow me to use pandas df.apply() method on my reviews dataframe. The function is exceedingly simple, it takes the whole plain text review, breaks it up into a list of tokens (words in this case), and then removes every stopword contained in NLTK's stopword corpus. Applying this function to the whole dataframe takes about 30 minutes, or 5 milliseconds per review. This is impressively fast and shows how optimized the NLTK library must be. When finished, the reviews dataframe has a new column for the 'tokenized and cleaned text'. Of course, I still keep the plain text review so I can later use it for sentiment analysis.

After tokenizing and cleaning every review, I can now turn these lists of words into a sparse matrix of feature vectors. I will use Scikit-learn's CountVectorizer as a starting point because it is transparent and will offer a strong baseline. Instantiating a new CountVectorizer object and fitting the vectorizer to my list of documents returns a corpus of every word that occurs through the whole set. There are around 190,000 entries in the corpus.

Next, I can transform the list of documents into a list of feature vectors. The CountVectorizer does this in a straightforward manner, representing each review with a matrix of 190,000 columns denoting the number of times that specific word is contained in the given document. This is an incredibly sparse matrix as you might imagine. There are techniques to reduce the size of the corpus, as much of those features will have little bearing on the actual model, but this is just to get a baseline so I'll leave it for now.

Once the every document has been transformed into a feature vector, I can fit and run a model to see where the baseline stands. I'll begin with a Multinomial Naive Bayes, as online literature suggests this model as being particularly useful for working with text. I first separate my massive dataset into a training and testing set. Again, the features here are the vectors of word counts and the target is star rating. Fitting the model to my training data, and predicting labels from the testing data returns the classification report printed to the left.

	precision	recall	f1-score
1.0	0.64	0.74	0.69
2.0	0.41	0.29	0.34
3.0	0.45	0.33	0.38
4.0	0.50	0.46	0.48
5.0	0.75	0.84	0.79

Unfortunately this baseline model does not perform as well I had hoped. However, I believe the biggest problem is the one detailed above (see figure 1), extremely imbalanced classes. It seems that because there are so many 5 star reviews relative to every other class (but particularly 2 and 3 star reviews), the model has learned to predict a 5 star review for any instance of language that isn't exceedingly negative. This gives fairly high recall for 5 star reviews, but lower precision. To adjust for the imbalance of number of reviews, I believe the classes can be combined without losing sight of the final goal of this project.

Making a new column in my reviews dataframe for a binary review classification is fairly straightforward. I construct a dictionary of star ratings (1-5) to 2 new classes, 'Good' for 5 and 4 star reviews and 'Bad/Neutral' for 1, 2 and 3 star reviews.

	precision	recall	f1-score
Good	0.90	0.93	0.91
Neutral/Bad	0.86	0.80	0.83

Performing the same steps as above, but changing my target variable to this new column yields the classification report to the left. As I expected, the classifier now performs significantly better when predicting our target variable. Because of the significantly reduced complexity, the classifier now just has to decide if a review is mostly positive or mostly negative. 9/10 times the classifier predicts a review as being positive, it is!

This is a fairly solid baseline, but can definitely be improved upon using techniques such as word-lemmatization, the inclusion of punctuation, or sentiment analysis. For now, I'll proceed on with the project and increment later if desired.

Burritos and Sentiment Analysis