**Motivation & Objectives**

The objective of our project Review Rater is to create a classifier that can be used to predict the rating of a Yelp review as either “useful” or “not useful”. To do this our team has identified the relevant attributes in the Yelp Dataset necessary for the completion of the project, these being the review text and the number of “useful” votes that this review in particular has received.

Our rationale for pursuing this particular project came from thinking about not just what Yelp is, but also what Yelp would find useful for its business model. Yelp is a site that gathers user reviews for local businesses, these reviews can in addition be voted upon by other users according to various metrics such as “useful”, “cool”, etc. Naturally a visitor who is looking for reviews on a particular business would want to see the most relevant and useful reviews for that business first. This was the idea behind our project, to apply data mining and machine learning techniques to find correlations among the text of user reviews that are deemed “useful”.

If we could indeed find some common characteristics among useful reviews, these could be used to build a model that could predict the usefulness of a given number of reviews. In our initial design we worked towards the goal of classifying reviews as either “useful” or “not useful”. As we continued work on our project we found this binary classification a bit too simple to give meaningful or interesting results. After some more thought we decided to take a slightly different track to our problem.

The end goal of this model was changed to be able to not only predict the usefulness of a review but also to rank these reviews in a descending order of usefulness. This decision was made with the thought of the average Yelp user in mind, when searching for a business review the average user is only interested in the usefulness of a review in relation to the other reviews on the same page, and not necessarily with the “absolute value” of a review’s usefulness. By ranking a business’ reviews in a descending order of usefulness we could ensure that a visitor would see only the most relevant and interesting reviews first.

The majority of our project code will be in Python 3.4 and we will make use of the NLTK library for our text processing and manipulation. Our project website will be hosted on github pages and will be written in HTML and JavaScript.

**Data Tasks**

**Preparation**

This project presented a number of challenges on the road of implementation. Our first task during the initial planning stages was to inspect the data set to identify promising data that could be used to find the results we were looking for. Since our project was focused on rating and ranking user reviews this particular task was accomplished rather easily. We decided to use the JSON file containing all information related to user reviews, including review text, date, ratings, business ID, etc.

However even after deciding to use the user reviews data set we encountered our second challenge, namely the sheer size of the data set which consisted of over 1GB of pure text data. This consisted of billions of JSON encoded data objects. Our next task was to once again narrow our search space in such a way that the smaller domain would not impact our eventual results. We did this by focusing only on the user review text and the total number of times a given review was voted as “useful”. We used the number of “useful” votes to determine if a given review has the quality of being “useful”. This number was used to divide the data set into two subsets consisting of a set of useful reviews and a set of not useful reviews. This number, the “useful” threshold, being of a somewhat arbitrary nature was changed across multiple program executions in order to compare multiple sets of results and to help tune the final program’s performance.

Finally, after separating useful reviews from the rest of the data set we must do one more task before starting our data analysis. We must separate these useful reviews into both training and test sets so that our program can have data to with which to build a model and data against which we may test our model.

**Analysis**

Now that we had two sets of data to use for our program development we set on the task of designing an algorithm and its associated methods. We brainstormed various methods that seemed promising enough to test, some of these methods in the end worked better than others. Since our major point of analysis was text reviews all of our methods and algorithms are text-centric.

In using text reviews to rank usefulness our team was making the implicit assumption that written language, and its semantic meaning, can be correlated with “useful” reviews. In theory this would mean that certain words, combinations of words, types of words, and semantic categories such as food, location, descriptions, etc. would be found in “useful” reviews vs. non useful reviews.

This alone gave us many promising avenues to pursue during the course of our project. Could a term in a review such as “chicken” be more indicative of a useful review vs. a term such as “table” or “chair”? Would “chicken” appear more often in useful reviews vs. non useful reviews? In an extension of the previous idea, could semantic categories of words appear more often in useful reviews? Maybe word categories such as “food” (chicken, beef, fish, etc.), “price” (cheap, expensive, tips), “location” (downtown, city, local) appear at noticeably different rates in useful reviews. Another aspect we looked at was word classifications such as nouns, adjectives, and verbs; perhaps useful reviews had different ratios of these word classes as well. These

**Method & Algorithms**

Next, as far as the actual algorithm that we are using and our project design, which is still subject to change, we are planning to use multiple techniques that have been covered over the course of the semester. We are planning to compute normalized tf-idf document vectors for all reviews in the training set and then implementing a k nearest neighbor classifier using these tf-idf values to classify a given review based upon the cosine similarity between the training set and the review to be classified. The tf-idf document vectors will be calculated using the same approach as the first programming assignment. The NLTK library will be used to process review text, including lowercasing words, stop word removal, and stemming. After this initial processing, tf-idf document vectors are constructed for each review along with their classification, this information is then used in tandem with the K Nearest Neighbor classifier based upon cosine similarity distance to classify test reviews with

**Initial Implementation**

The number of user votes that are necessary to classify a review as “useful” is currently a best guess that took into account the enormous amounts of data available in the dataset challenge and how votes are distributed within the dataset. Currently our threshold for useful votes is 3 but this will likely change as we continue to develop the project and see what is working and what is not. Our first task in getting started on this project was to partition data into both test and training sets, we have currently do not have a set method for partitioning the data, but a primary point of concern in partitioning is the computational intensity of computing the kNN of a review, so the training set has to be kept small. Also, based upon future results produced by the finished classifier program, the percentage of positive and negative reviews is extremely subject to change. We expect that tweaking of the training set will yield very different accuracies in classification.

Our first challenge was in simply accessing the data within the dataset properly, even though our project is only focused on the review dataset the size of the data made most data accessing methods useless. The data was processed line by line since this gave good performance and used little memory, next, the json.loads() function was used to turn the JSON text into a data dictionary. From this dictionary the relevant data was accessed and extracted. During this data access we are partitioning data into training and test sets randomly using a Boolean flag and writing the test/training sets to two output files based upon the types and distributions of data we are populating these sets with.

The plan is divided into two parts, the preprocessing and runtime processing; this is to account for the large size of the data set corpus, having to completely process the data set in addition to our own project implementation will likely bring project runtime up to unacceptable levels if this is done for every execution. During pre-processing we will access the test data set and compute normalized tf-idf vectors for every review it contains, this portion of the implementation is already completed. Next we will begin designing the K Nearest Neighbor classifier and begin testing.

**Evaluation**

Our classifier will be tested against the test data corpus and the results will be evaluated accordingly. Given that our classification labels are “useful” or “not useful”, our classifier will be expected to accurately predict a review’s classification at least 50% of the time at the minimum. We are aiming to have an accuracy of 60-65% currently, with accuracy above 70% being considered a great success. Since tweaking of the training data set and threshold of number of up-votes it takes for a review to be considered “useful” will produce very different accuracy in results, we will be constructing a spreadsheet containing accuracy values generated based upon certain constants and distributions of data types in the training set. This excel chart will be used to decide upon what the best constant values and training set data type distributions produce the best results, and what direction to take the project in.

**Deliverables**

The current status of the classifier is incomplete. The code is complete and functional, but we are not confident in the correctness of the functions as they are currently implemented. We are in the process of unit testing the various functions for correct output. The project website has taken a back seat in development since we do not haven not produced results yet using the current classifier. It is a work in progress and only serves as a point of access for getting the current implementation and documentation for the project.

The project website can be found at this URL: http://l1felock.github.io/ReviewRater/

**Challenges**

Our biggest challenge so far has been deciding upon the appropriate techniques to use for classifying reviews accurately. Given the goals of the project no one technique covered in class would suffice alone. The goal is to classify but how to get there? Main input data is text but using document similarity won’t be enough, we settled on including a K Nearest Neighbor classifier using TF-IDF document vectors and cosine similarity between vectors in order to classify reviews.

Another challenge we have encountered is deciding upon the appropriate metrics to use for selecting reviews to be included into test and training sets. Initial design required the use of only “useful” reviews for the training of our classifier. However as we moved forward in our implementation we have given more thought to using reviews classified as not useful for use in our classifier. The question is now what is the ratio? How many reviews should be “useful” and “not useful” in our training/test sets? 50%/50%? 70%/30%? We will have to experiment with multiple ratios and see where the ideal split is. Also, we are also constantly changing what is considered a “useful” review. The threshold of the number of up-votes that a review has to have to be considered useful has a great impact on the distributions of data classifications, and also the potential accuracy and realism of the classification.