**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 review text and the number of “useful” votes that this review has received. Our project will be implemented in Python.

**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.

**Future**

Next we must finish unit testing our classifier’s functions. As we develop our classifier we will be continuously testing and tweaking it’s parameters to increase its accuracy and its realistic applicability.