Farris Tang 1662953 fktang@ucsc.edu
Kit Lao 1705333 kilao@ucsc.edu
Kirby Lam 1537362 klam18@ucsc.edu
CSE142 Project Report, Group 14

1 Diversity

For the diversity portion of the report, our group decided to focus on the reason why we chose to get into computer science, and our age, along with our college experiences. Kirby got into computer science in order to pursue his passion of developing applications. Kit was originally a math major, but switched to computer science in order to get into the theory and mathematical part of computer science. Farris got into computer science because it was a subject that he didn't completely hate. In terms of age, our group has a slight age gap, Farris is 25, Kirby is 21, and Kit is 22. We treat age as a factor in our group's diversity because of the implication of where each individual is in their lives. For example, Kirby is 21, which is prime time for pursuing fun college activities, but Farris is 25, and his main focus is on school. It was interesting to note our group's diversity without considering race and gender.

2 Tools Used

1. sklearn.feature_extraction.text

   We used the TfidVectorizer function from this library in order to convert our text reviews into a vectors of features. This will then be used for our classifier, which will then be used by our all vs all multiclassification model.

2. sklearn.model_selection

   We used the train_test_split function from this library to easily split our train set into both a train set and test set, to compute a test accuracy for the model we choose to use.

3. sklearn.linear_model

   This library provides a LogisticRegression model that allows us to easily classify instances. The function uses 100 iterations in its algorithm by default. We will be using the default parameters for the LogisticRegression functions.

4. sklearn.metrics

   We used a few functions from this library:

   (a) accuracy_score: used for computing accuracy

   (b) classification_report: used for computing precision, recall, and f-1 values given our predictions and actual test labels.

   (c) confusion_matrix: used for creating a confusion matrix given our predictions and actual test labels.

5. json

   This library was used to read in data fron the given JSON file.

3 Abstract

We used the Scikit-learn machine learning library to perform our predictions. We wrote the code to read data from the provided JSON file. After reading from the JSON file, we noticed that we had a lot more 5 star instances than others. We noticed that there were 149031 instances of 1 star reviews, 27028 instances of 2 star reviews, 36987 instances of 3 star reviews, 73604 instances of 4 star reviews, and 147645 instances of 5 star reviews. To balance the data, we sub-sampled the data by taking note of the smallest size class (in this case, it was class '2 star' with a count of 27028). Our method for doing this was provided in the train_test_split function, which randomly chooses from the correct class until the number of needed instances is met. Our reasoning for doing this was to make sure no class was over represented in an attempt to reduce bias in our predictions. The cost of doing this is that we are throwing valuable data away. After sub-sampling the data, we proceeded to isolate the text review from the instance in order to preprocess the data and vectorize the text. This was done with the TfidVectorizer function from the feature_extraction.text library given by Scikit-learn, and was used to transform a text review into a vector of TFIDF values (term frequency–inverse document frequency).

The TfidVectorizer tokenizes our text string into individual words, throwing away the punctuation, but keeps the stop words (meaning words such as 'the', 'a', 'but', etc). The TfidVectorizer will calculate the term frequency, which is the number of times a term appears in all of the texts divided by the total number of text reviews. Then, the inverse document frequency is calculated by taking the natural log of the total number of text reviews divided by the number of text reviews with the given word. TfidVectorizer then takes the product of these two values. This is done for every word and for every text review to create a vector of TFIDF values that will represent the translation from text to "features". We did this process using the text review from the training file and testing file, and appended the list of vectors together. We did this becasue our test and train data outputed different vectors frbecause they were reading different texts.

At this point, we now have a vectorized version of the text reviews and the labels of each instance. We chose to completely ignore the other features given to us, such as 'cool', 'funny', 'useful', or 'date'. Our intuition for this comes from the fact that these features make no determination in whether a Yelp review will be 1 star or 5. An example of this could be that a one stare review could have just as much usefullness, or be just as funny as say a 5 star review.

The next thing we chose to do is to split our training set and test set, in order to compute a test accuracy for the model we choose to use. The model we chose to use was AVA, using Logistic Regression as a classifier. After feeding in the vectors of each training instance, and their respective labels, we were able to obtain a reasonable test accuracy.

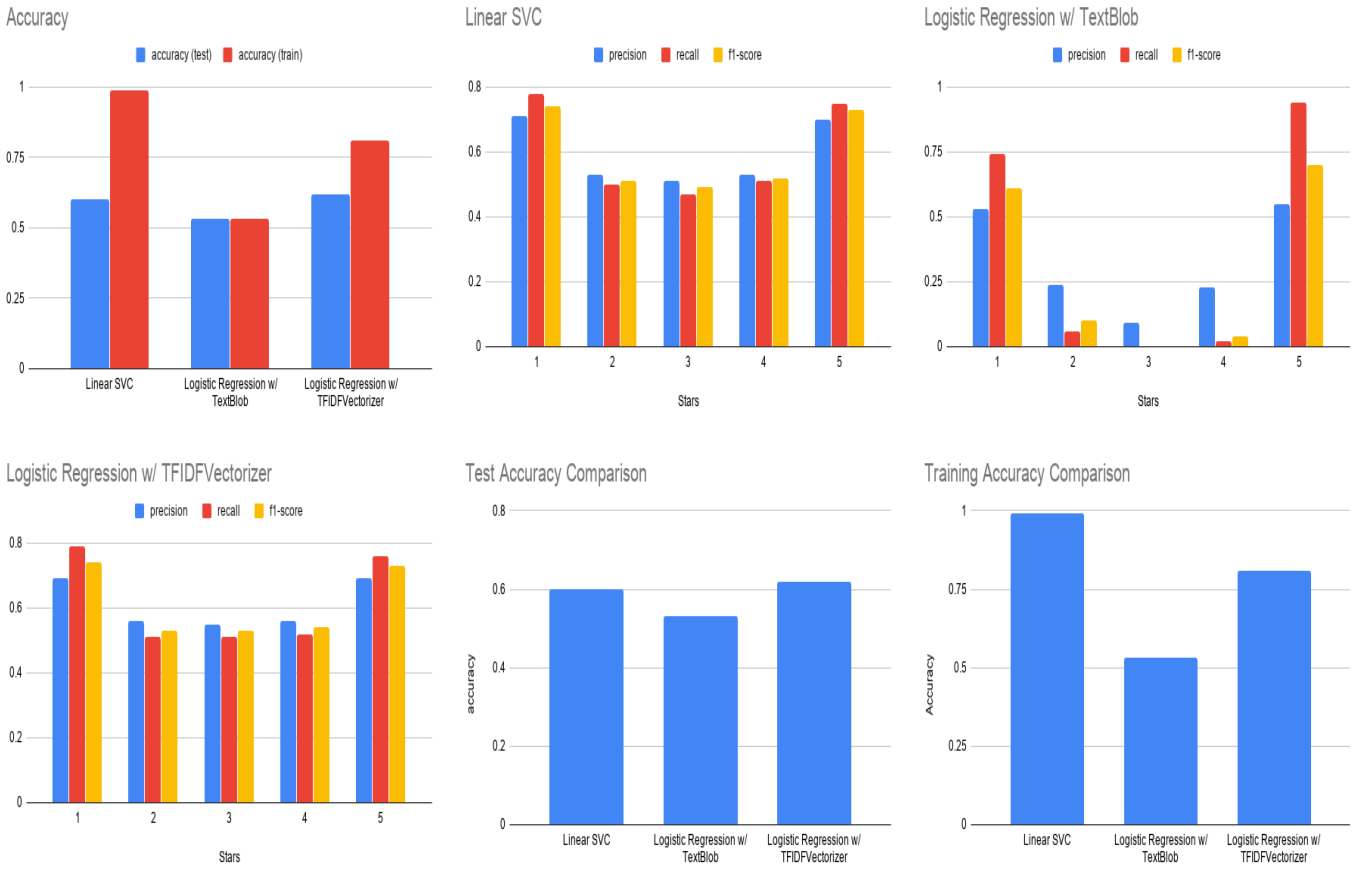
4 Approaches

Our very first attempt at the project involved using a library called 'TextBlob' to convert text reviews into two features, 'polarity', and 'subjectivity'. At this point in time, we still kept the other features provided, and trained weights accordingly, using the AVA algorithm, and with Logistic Regression as our classifier. Our thought process was that if these features were insignificant, the trained weights for these features would be close to zero anyways. We chose to use AVA because it seemed after doing some researching, and found that AVA provided better results than OVA without adding too much complexity. Looking at Neural Networks and how complexity increases with more inputs discouraged us from attempting this algorithm. This attempt gave us a reasonable test accuracy of about 53%.

Our next attempt happened after doing some research on how to achieve better results. We discovered the Sci-kit learn library that allowed us to experiment with different classifiers with our AVA algorithm

easily. This library also allowed us to try a different approach of vectorizing the text reviews, and to turn them into a vector of TFIDF values, rather than a vector of polarity and subjectivity values, which yielded a test accuracy (from 80-20 split) of about 62% (using Logistic Regression). Decision Trees and Neural Networks took way too long to process (above 20 minutes), so we decided to not use them. We chose Logistic Regression over SVM because based on our intuition about the data, a 1 star review could be very similar to a 2 star review, and a 4 start review could be very similar to a 5 star review. Since SVM makes binary decisions when classifying data, it is not a very good estimation for classifying our data based on how similar some classes are. Whereas in Logistic Regression, it estimates probabilistic values when classifying. This is actually a good feature for our data based on how similar some classes can be.

5 Results



The training accuracy for Logistic Regression with AVA was about 81% and the test accuracy that we got was about 62%. This training and test accuracy was calculated using an 80-20 split on the training data. The precision that we got was 0.71, 0.53, 0.51, 0.53, 0.7 for 1 stars, 2 stars, 3 stars, 4 stars and 5 stars respectively. The recall that we got was 0.78, 0.5, 0.47, 0.51, and 0.75 (1, 2, 3, 4, and 5 stars). The f1-score that we got was 0.74, 0.51, 0.49, 0.52, and 0.73 (1, 2, 3, 4, and 5 stars).