Final Project: CSE 142

Ian Kirk, ikirk@ucsc.edu, ID:1601671
Jacob Baginski Doar, jjbagins@ucsc.edu, ID:1577517
Julius Fan, jzfan@ucsc.edu, ID:1522743
Group 21

1 Tools Used

All code was written in Python 3.5.7. Libraries used include:

- Numpy for multi-dimensional arrays and vectors
- Pandas for dataframes
- Random for shuffling and random results
- SKLean to import and train the model
- Tensorflow to speed up the backend process
- Matplotlib to visualize training data statistics

2 Diversity

2.1 Jacob

I am bisexual and closely connected to many people who are active within the LGBTQ community so while I do not consider myself active in the community, I have grown up with it being important for many people around me. My father is also an immigrant from England so I grew up with parts of the culture in my home life.

2.2 Ian

I strongly identify with my Taiwanese heritage and am often mistaken for being Chinese. I also grew up in a nearly homogeneous neighbourhood composed of mostly wealthy Chinese and Caucasians. Although I was not noticeably a minority, at times I did feel like an "other".

2.3 Julius

Growing up I was constantly moving around the world due to my family's work and life. Between Pre-K and College, I had lived in 9 different cities including Shanghai, Hong Kong, New York City, Seoul, and Los Angeles. As such, I was able to experience a large portion of the local culture in each location. I identify mainly as Chinese American but I also believe that I've been able to have a much more fruitful experience growing up due to the constant relocations.

3 Abstract

Although we all used the same extracted features, each of us implemented a different algorithm to learn a different linear boundary. This was then extended for multi-class classification through varied techniques.

4 Data Pre-processing

The figure above was generated with matplotlib and as you might notice, the extremes for ratings were much more common than the moderates. We chose not to subsample because these distributions were so extreme and we did not want to throw away so much data, and because we were not especially concerned with our model mispredicting one class over the other. Although we also considered super sampling too, we ultimately

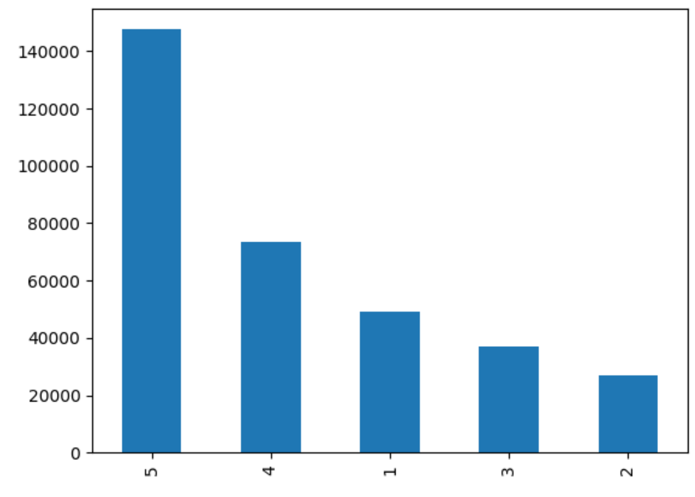

Fig. 1. Rating Distribution

decided against it too because some of our models were already taking a long time to run, and adding more instances to match the quantity of 5-rated instances would hamper run time too much.

One thing we noticed is that some reviews are in languages other than English. For reasons that our feature extraction section will make obvious, these are outliers that are not representative of the data and would interfere with training. A quick visualization of non-English words showed that the frequency was very low. Since removing them by hand is unfeasible, and creating or implementing a language detection model is far out of the scope of this project, we decided not to remove them under the assumption that their impact on the performance of the model would be insignificant.

5 Feature Extraction

We decided to extract 80 different features from the reviews. Most of these features were ”keyword” occurrences, and the other minority were other various quantities.

5.1 Keywords

Keyword occurrences made up 76 of the 80 extracted features. They tracked the number of occurrences of a particular ”keyword” in the review. For example, one of our keywords was the word ”good”. If the word ”good” appeared 3 times in a review, then that instance’s feature vector would have the value of 3 in the position that corresponds to the word ”good”. We wrote a short program to determine and the frequency of all words used in all reviews. After running this program, we sorted the words by descending frequency and hand-picked ones that occurred often enough to represent the data set and could signify the outlook of review. Some other keywords were ”great”, ”slow”, and ”expensive”.

We also considered not counting individual keywords, and instead counting the sum total of keywords in a particular group. For example, we might have instead grouped all keywords into one of three categories: positive, negative, and indifferent. With this, instead of having a keyword feature vector of 76 dimensions, it would be 3 dimensions and look like $< \#good, \#bad, \#indifferent >$. If an instance contained ”good”, ”great”, and ”awful”, its feature vector would look like $< 2, 1, 0 >$. In the end, we decided against this because this would have meant our feature vector would have a much lower dimension, and since we were learning linear models only, we wanted it to be more complex.

5.2 Other Quantities

The other quantities we extracted were the number of exclamation points in a review, the number of dollar signs in a review, the number of fully capitalized words in a review, and the length of a review in words.

6 Approaches

6.1 Logistic Regression

6.2 Perceptron

We began with vanilla perceptron with AVA (all versus all), but it was not performing as well as desired. We tinkered with shuffling and iterations in order to improve its performance.

6.2.1 Shuffling

Shuffling the order of the instances after each iteration had a significant impact on accuracy. Although Python's random library only generates low entropy randomness, the results were still impressive. Accuracy went from 51% to 57% with the introduction of shuffling.

6.2.2 Iterations

Interestingly, although we only tested 3 different values for the max iterations, we observed barely any performance by this change. There was an improvement from 48% to 51% between 10 and 20 iterations, but almost no observable difference between 20 and 100 iterations. In the end, we decided to go with 20 iterations in order to improve execution time.

6.2.3 Results

The final algorithm predicted the training instances with an accuracy of 57%. Execution time was on the high side at approximately 35 minutes. We found the accuracy to be sufficient and not warranting increased iterations, especially given the long run time.

6.3 Naive Bayes

This approach resulted in accuracies between 58% and 65% over the course of 5 executions on the training set, with a 70-30 split between training and test.

6.3.1 Vectorization

We used a vectorization method CountVectorizer in order to conver the text in each review into a dictionary of the word occurrences that appear throughout the training data. By assigning a value to each individual word, we were able to see what each word's associated value is.

6.3.2 Data Set Splitting

In order to test the model we trained, we also used a train-test split of 70% to 30% after randomly shuffling the reviews. This ensures that we have a sample to test on in order to measure our precision, recall, and accuracy.

6.3.3 Implementation

We used the MultinomialNB() class within the SKLearn library in order to train the mode. According to the wiki page, MlutinomialNB is generally used for word vector calculations and text classification. We chose to use this rather than TF-IDF because the vocabulary count seemed to have more of an impact on the rating than the weight of a word.

6.3.4 Output

The Naive Bayes program we used outputs 4 files. The model is saved under the 'finalized_model.sav' file. Ths accuracy on the training set is saved under 'accuracy.txt'. The predictions for the isolated sample of the training set are saved under 'predictions.csv'. Finally, the predictions for the test set are saved under 'predictions2.csv'.

7 Conclusion

Our naive bayes implementation performed the best in terms of accuracy out of three algorithms.

8 Ideas for future work

We did not attempt to implement voted or averaged perceptron for our perceptron algorithm. Although voted perceptron would require a lot of RAM, it could still be attempted, as could averaged perceptron. We could also implement ranking penalties for mispredicting an instance as 1.0 compared to 4.0 if the true label is 5.0.