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. Libraries used include numpy and random.

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".

3 Abstract

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

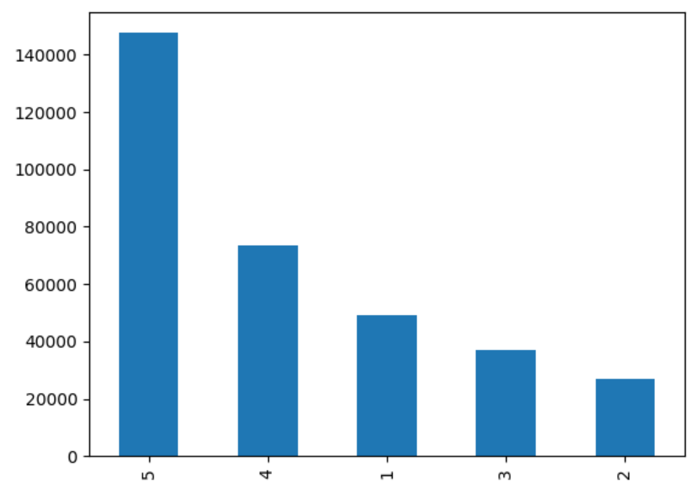

Fig. 1. Rating Distribution

4 Data Pre-processing

As you can see in the figure, we noticed that the extremes for ratings were much more common than the moderates. We chose not to sub-sample because these distributions were so extreme and we did not want to through away so much data. Although we also considered super sampling to, we ultimately 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 be outliers that are not representative of the data and would interfere with training. However, 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 enough.

5 Feature Extraction

We decided to 80 different features from the reviews. Most of these features were ″keyword″ occurrences, and the other very small 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 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 more 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.

6.3 Naive Bayes

7 Experimental Set-up

8 Results

9 Conclusion

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

10 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.