Rohan Agarwal, Aliyah Lee, Tyler Mount, Sarah Stumbras, Nathan Wang
IEMS 393-1 Project 2

Sentiment Analysis of Movie Reviews

Introduction

Sentiment analysis has been an area of interest at the intersection of natural language processing, text mining, and linguistics. Being used in a variety of applications such as marketing and customer service, in this report we explore our own ability to classify sentiment based on a dataset of movie reviews from the popular website Rotten Tomatoes.

Understanding the Data

Our first steps were to visualize the given data in order to get a better understanding of subsequent steps. First, we looked at the distribution of reviews. We found that they rating of the reviews were approximately normally distributed with a mean of 2.

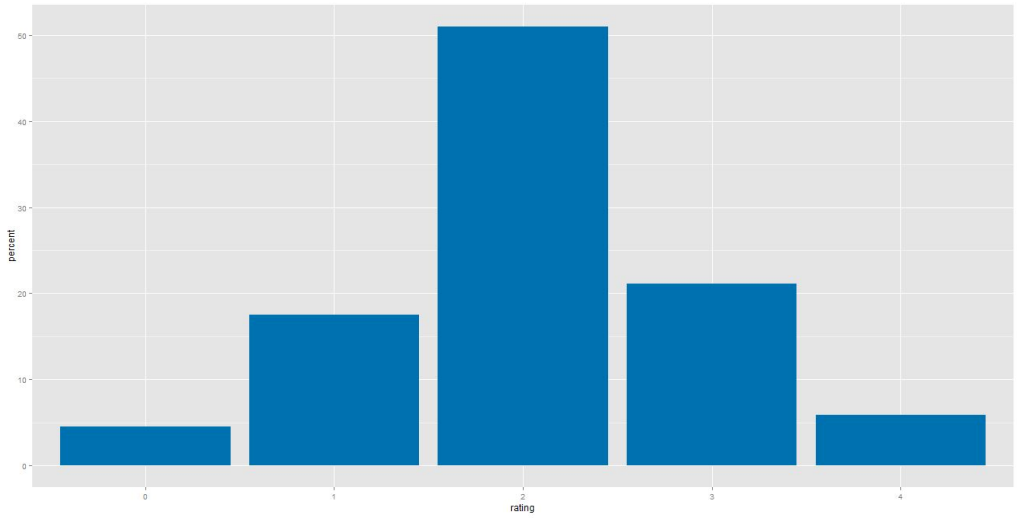

Figure 1: Rating Distribution

Next, we assessed the word count of the reviews. We plotted the word count and their ratings. Increased word count was identified as being indicator of sentiment that is deviant from the mean. The more words, the stronger the opinion, positive or negative, of the review. The graph below shows that as the word count increases there are fewer

2 ratings and higher proportion of extremes, that is 4 or 0 point ratings.

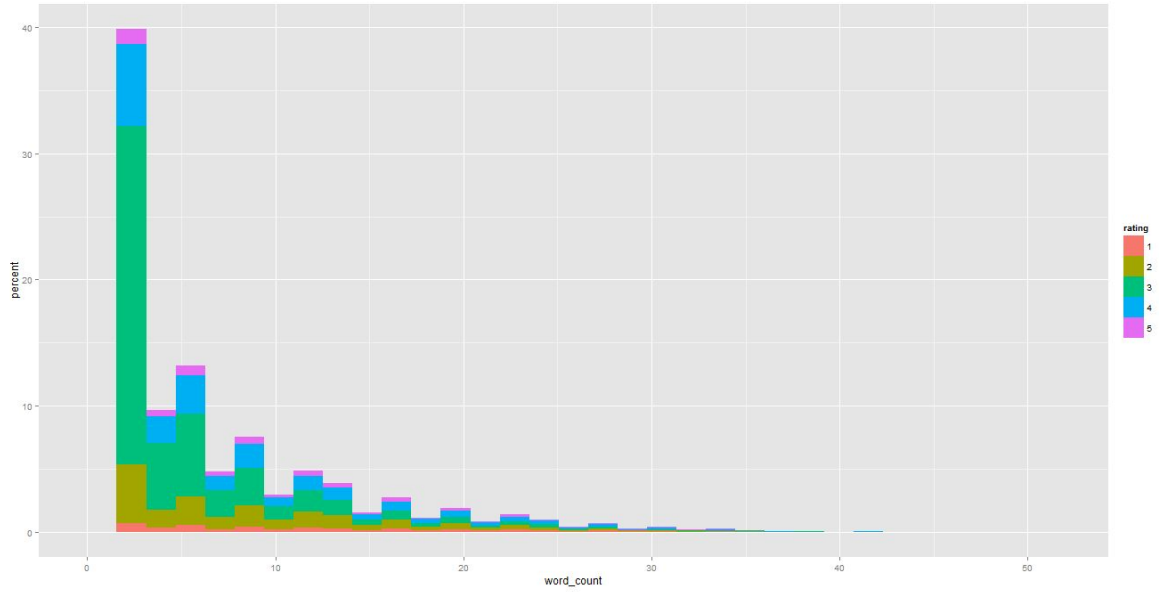

Figure 2: Word Count and Rating Distribution

Variable Selection

Our next task was to breakdown the data in way that we could use them as inputs in our model. This required us to research different ways that the reviews could be quantified. We decided on 4 key features both from research and from the initial data exploration:

Hu and Liu Sentiment Lexicon

Lexicon approaches of sentiment analysis are on of the most common ways to understand the opinion of text. After some research of different lexicons, we selected Hu and Liu's lexicon This approach uses adjectives as orientation of opinion. It then uses these adjectives to classify each opinion as positive or negative through a bootstrapping technique using the adjective synonym and antonym sets in Wordnet and Determines overall sentiment using their Sentence Orientation algorithm.

Word Count

We used word count as a measure of the strength of the reviews. We observed that the longer the review, the stronger the opinion.

Grammatical Structure

We parsed the reviews for the presence of commas and apostrophes. We intuitively guessed that underlying grammatical structures may serve as a strong predictor variable for the opinion of the reviews. Individually we did not find any correlations but

we chose to include them as an intelligent model, if needed, would just not include them.

Document Term Matrix

We created a matrix of all of the words used in the reviews and then parsed it in three different ways. Firstly, the entire document term matrix was filtered for 99% sparsity. We then had a list of specific counts of words to be used in our models. We hope that we could find a relationship between specific words that occur frequently and the rating of the review. After parsing for 99% sparsity we were left with 46 words. On top of this, we considered a new document term matrix which was composed only of positive words earlier identified by the lexicon. Filtering for 99.5% sparsity and performing said method for positive and negative words added another 28 words.

Latent Dirichlet Allocation

LDA is, by far, the most important contributor to our model. Before LDA the results were stuck at approximately 60% accuracy on a testing set, yet however with LDA the results are much higher. LDA is an unsupervised method that automatically groups words into topics, from which every review can be expressed as a percentage composition of different topics. These percentages in turn become features in the final model.

Additional Attempts

Additional attempts to add features includes performing PCA on the document term matrix, which we found was unsuccessful, as well as trying to add more and more individual words as predictors in the final model. We found that besides the most common words, the contribution was marginal and more likely to be overfitting rather than capturing signal. We also tried to stack models - simplifying the problem into a binary good/bad prediction which then fed into a 5-class classifier, but this was unsuccessful.

“Stupid” Models

Using these key features, we developed a few very basic models to gain an understanding of what type of accuracies to look for and what biases they may have. We used these as a sounding board to brainstorm improvements for future model and for some initial benchmarking.

The Mean Model- The simplest model would just be to guess a 2 every time because over half of the reviews are rated a two. Based on this, our benchmark would be a bit over 50%.

Linear Regression- We created a linear regression using the features outlined above. This model has an high accuracy of 55% but from the output we could see that it was just guessing a review score of 2 for almost of the reviews. This makes it less intelligent and robust than we'd like.

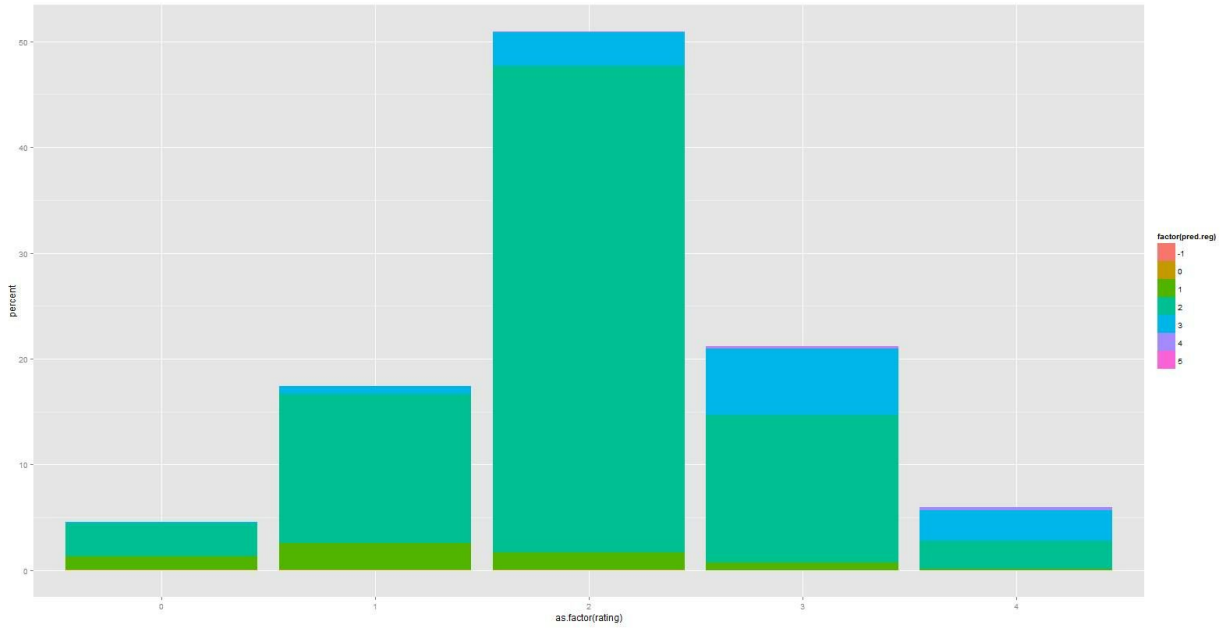

Figure 3: Linear Regression Distributions of Ratings vs. Predictions

Naive Bayes- Using this classifier, we found that we got a wider range of results than with the previous two models. However, the accuracy was slightly worse at 46%.

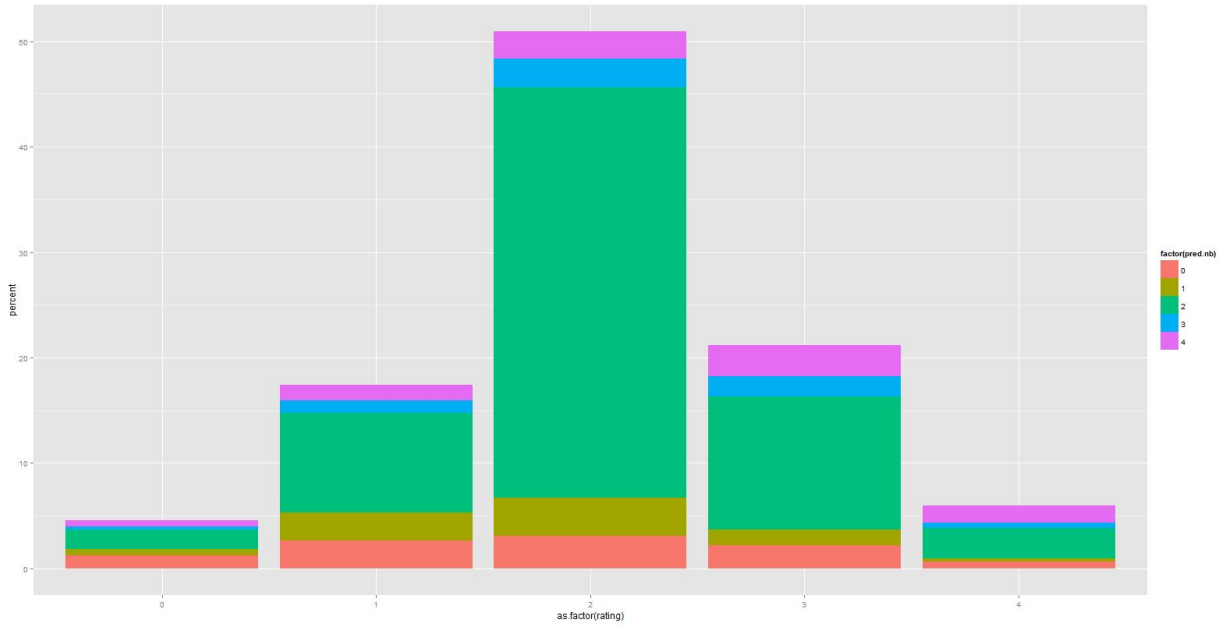

Figure 4: Naive Bayes Distribution of Ratings vs. Predictions

Decision Tree- This model was far too simple. We saw that it only predicted ratings of 2 or 3 with a low accuracy of 46%.

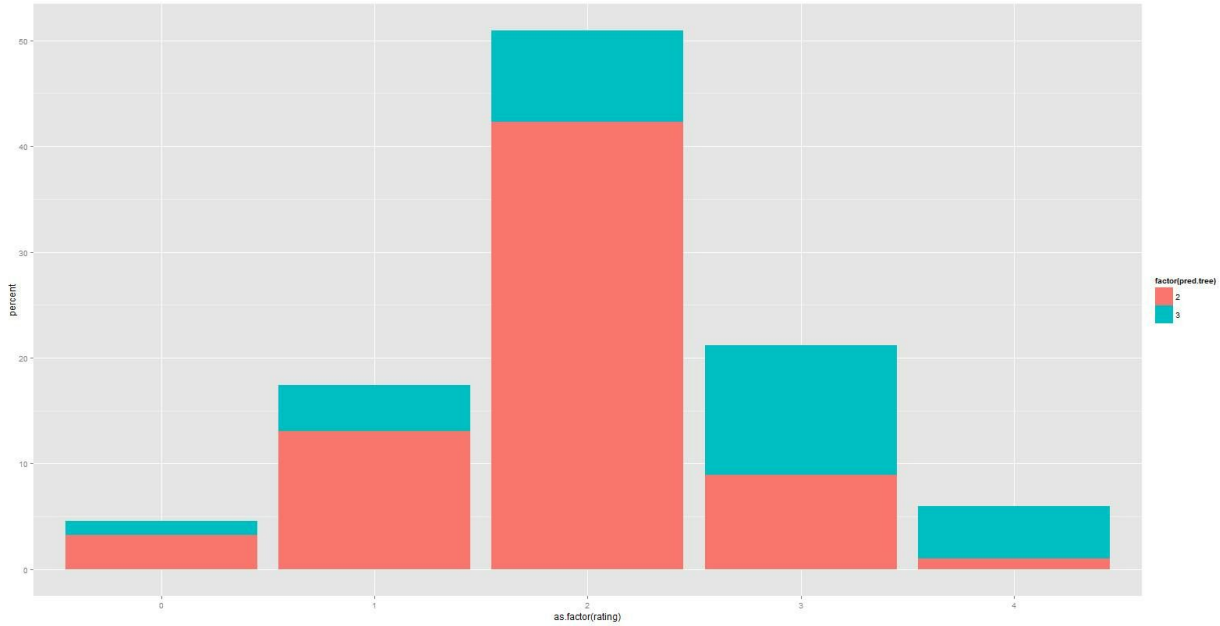

Figure 5: Decision Tree Distributions of Ratings vs. Predictions

From this we learned that we needed to use a more complex model in order to capture the nuances of words. Subsequently we also realized the clear issue of class imbalance, which requires an algorithm that would be able to handle it accordingly.

Model Formulation and Solution

After our initial brainstorming of “stupid” models, we conducted a second round of research to find a better model that could improve on the benchmarking from these models. Based on Kaggle forums and initial empirical testing we decided to create a Random Forest model. We think that this best suits our model needs because of its robustness: it handles class imbalances well and is able to handle multi-class classifications, both of which are requirements of our model. It is also a good choice because it is relatively easy to train and often the most accurate, being implemented using a single line in R.

We also would have liked to try other methods, namely gradient boosting via the gbm package, support vector machines via the e1071 package, xgboost via the caret package, as well as neural networks and multi-class logistic regression models via the nnet package. What we found, however, was that training models became an extremely lengthy process. Training a single random forest, depending on the number of features used, took anywhere from 1.5 - 2.5 hours for a single model.

The majority of our work was implemented in R. The modeling for LDA was implemented in Python using the gensim package and the result was later imported to R during modeling. We split our data into training (80%) and testing (20%) groups. We used the same key features (word count, lexicon analysis, grammatical structure, and document term matrix) as in pre-processing.

The packages we included, despite not necessarily using all of them, were: rpart, nnet, gbm, e1071, randomForest, tm, ggplot2, stringr, openNLP, and bigmemory. Our data visualizations were implemented using ggplot2. Our model has a final accuracy of approximately **85%** on the testing data.

Post-Processing

Looking at the graph below, we can see that our model is predicting the correct rating for a better proportion of reviews. For each rating level, our model predicts the correct level more often than an incorrect rating. We can also see that the ones that are incorrectly predicted are distributed about the correct rating level.

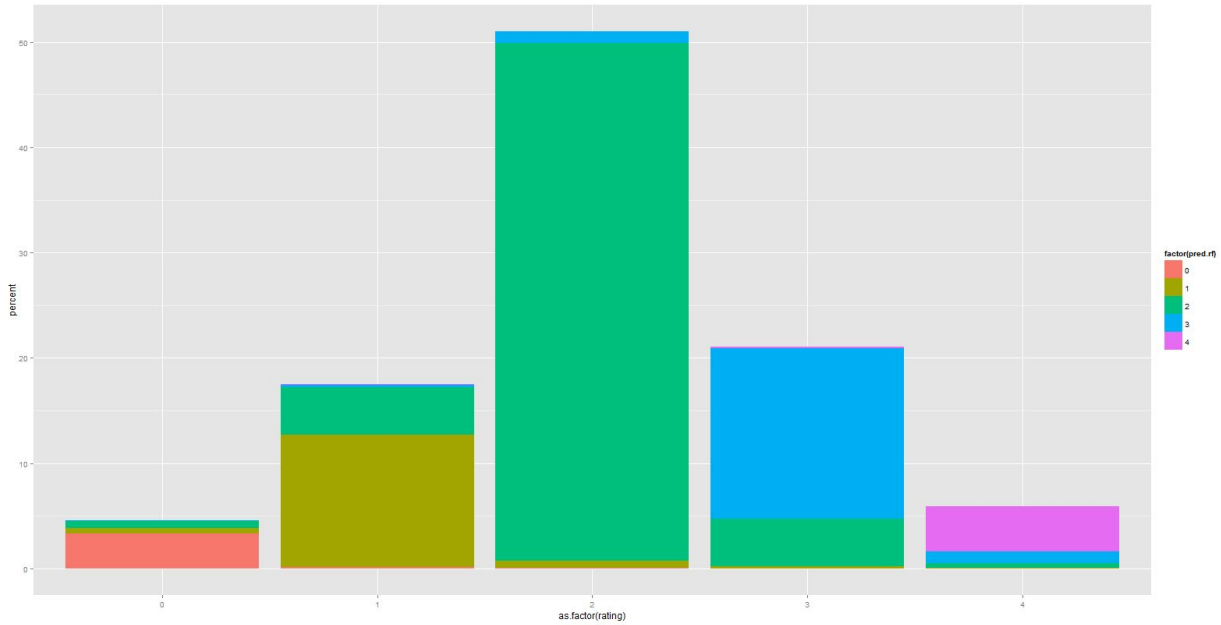

Figure 6: Results from Random Forest Model

Again, we attribute a large part of the success of this model to using Latent Dirichlet Allocation (LDA). Below is the result of a random forest model with a much lower accuracy of 61% with the only difference being that LDA topic compositions aren't included:

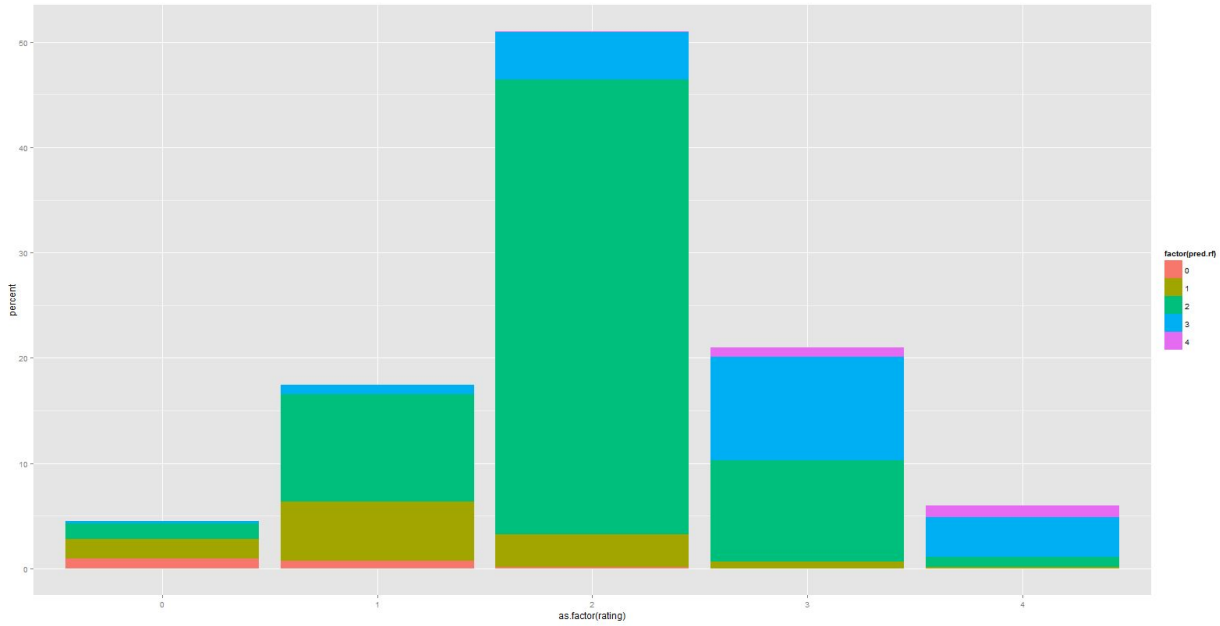

Figure 7: Random Forest model without LDA Features

Clearly the latter model is more susceptible to incorrect predictions, especially at the tails.

Note however that as a team we are skeptical of the incredibly high result - mainly because we haven't been able to come across a result that high online. To our credit we also have used a completely different approach though. Another area of improvement is issues with our code being reproducible as it seems like algorithms such as LDA and random forests also yield different results when run multiple times. With more time we would therefore implement more thorough cross validation.

Conclusion

By using a combination of document term matrix modeling, positive and negative keyword identification, part of speech tagging, and topic modeling we were able to accomplish a high accuracy which, with more time we believe we could turn into a constantly reproducible result. This dataset is rich with possibilities to look at the information from different angles and constantly strive for perfect predictability.

Sources

1. Liaw, Andy, and Matthew Wiener. "Classification and regression by randomForest." *R news* 2.3 (2002): 18-22.
2. http://www.r-bloggers.com/benchmarking-random-forest-implementations/
3. https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html