Project 3:

Movie Review Sentiment Analysis

Team Member (Name + NetID + Contribution):

- xxx, coding and writing the report
- xxx, coding and writing the report
- xxx, coding and writing the report

1. Overview: dataset and project goal

We are provided with a dataset consisting of 50,000 IMDB movie reviews, where each review is labelled as positive or negative. We aim to build a binary classification model to predict the sentiment of a movie review. The dataset has 50,000 rows (i.e., reviews) and 4 columns, “id”, “sentiment”, “score” and “review”.

Our goal is to develop an algorithm that allows us to achieve AUC > 0.96 for all the splits with a vocabulary list no larger than 3000 words, and also provide excellent interpretability. We are able to achieve the expected result with a customized vocabulary list of 1000 words.

2. Technical details:

2.1 Preprocessing:

Upon initial inspection, we did not see any missing values in the dataset. The dataset is also very balanced consisting of 25000 positive and 25000 negative reviews. We have done the following steps for preprocessing: (a). Stopwords removal: stopwords are the most common words, such as articles ‘a’ and ‘the’ and should be removed to increase the discriminating power of our classifier. (b) tokenization: We use the ‘itoken’ function in text2vec to convert all reviews into lower cases and perform tokenization by specifying tokenizer = word_tokenizer. (c) We use train$review <- gsub('<.*?>', ' ', train$review) to remove punctuations and brackets. More details about these steps and constructing the vocabulary list can be found in a separate R markdown file. We use split_1 to obtain our myvocab.txt.

2.2 Algorithm and models:

We construct a document-term matrix (DTM) fed with our customized and then use glmnet() function with argument family='binomial', for logistic regression with L2 (ridge) regularization to fit a model. We first use cv.glmnet() to obtain the best value of lambda using cross validation and then use it to train our algorithm as shown below.

```
mylogit.cv = cv.glmnet(x = dtm_train, y = train$sentiment, alpha = 0, family='binomial', type.measure = "auc")
mylogit.fit = glmnet(x = dtm_train, y = train$sentiment, alpha = 0, lambda = mylogit.cv$lambda.min,
                family='binomial')
```

Different values of alpha have been tried in addition to zero (corresponding to ridge), and it turns out alpha = 0 leads to the best result. We also optimize over n-gram and pick ngram = c(1L, 4L))) which achieves a slightly better result. For detail codes, please see our mymain.R file.

3. Model validation

3.1 Model and Result

We have experimented 3 approaches listed below:
A: Logistic regression with L2 (ridge) regularization with vocabulary list (<1000 words) trained with split 1 data
B: Logistic regression with L2 (ridge) regularization with vocabulary list (<1000 words) trained with all data

C: Bootstrap model with vocabulary list (>1000 words) trained with split 1 data (note that if trained with all data, we can achieve AUC > 0.96 and with vocabulary list < 1000 words):

The AUC scores for the 5 splits are listed below:

	split_1	split_2	split_3	split_4	split_5
A (split 1)	0.9602	0.9637	0.9635	0.9641	0.9634
B (all data)	0.9651	0.9637	0.9643	0.9642	0.9635
C (bootstrap)	0.9604	0.9666	0.9667	0.9673	0.967

3.2 Error analysis:

In this subsection, we're discussing two examples of wrong prediction made by our model.
Example I:

id	sentiment	score	prediction
3	0	4	0.89

"All in all, this is a movie for kids. We saw it tonight and my child loved it. At one point my kid's excitement was so great that sitting was impossible. However, I am a great fan of A.A. Milne's books which are very subtle and hide a wry intelligence behind the childlike quality of its leading characters. This film was not subtle. It seems a shame that … Overall, I was disappointed, but I would go again just to see the same excitement on my child's face.<br /><br />I liked Lumpy's laugh...."

As we can see this review receives a high probability of 0.89 indicating a positive review. However, actually this is a negative review. The reason the prediction is wrong is the reviewer has mixed feeling about this movie. He/she thinks it is disappointing, but kids might like it. So many positive words in conjunction with negative words lure our classifier into making a wrong prediction.

Example II:

id	sentiment	score	prediction
128	1	7	0.05

"The problem with so many people watching this movie is the mindset they watch it in. People come looking for a B-Grade horror film, or a \"So Bad It's Good\" movie. Jack Frost 2 is neither of these. <br /><br />It is, to put it simply, a very good movie cleverly hidden inside a very bad one. …The lack of \"memorable quotes\" disturbs me.<br /><br />As a horror movie, Jack Frost 2: Revenge of The Mutant Killer Snowman, rates a zero. But you have to understand, IT'S NOT A HORROR MOVIE."

This positive review receives a low probability 0.05 because it contains many negative words, such as "bad", "B-Grade", and "worst", etc. These negative words appear only because the reviewer quoted bad reviews from other people. Not enough positive words (compared to negative ones) send our model the signal of a positive review.

3.3 Model limitations:

The examples above show that our model does not perform perfectly on reviews that contains both negative and positive words. This is especially true with a sarcastic review, or a review containing other people's expectation or quoting other reviews. One way that might help resolve this issue is using high numbers of n-gram which provide the context of a positive/negative words (detailed in the later section). Another way worth trying is to weight some terms more than other. A good example is that in example 1, the reviewers already mentioned, "*I was disappointed, but …*", in the review so it should clearly reflect the reviewer negative view on this review. "*but*", gives us a hint that many positive words in the review are under the condition after the "but" keyword. So this sentence is worth giving more weight than other terms.

4. Interpretability of our algorithm

Being able to interpret our algorithm is one of important goals of this project. Because we use logistic regression as our classification algorithm, we can achieve this by inspecting the fitted coefficients (beta) of the model to see clearly the importance each term. Below we illustrate some values of beta for both positive and negative terms.

Table: Positive terms with their beta values

terms	great_job	great_show	highly_recommend
beta	0.57	0.71	0.51

Table: Negative terms with their beta values

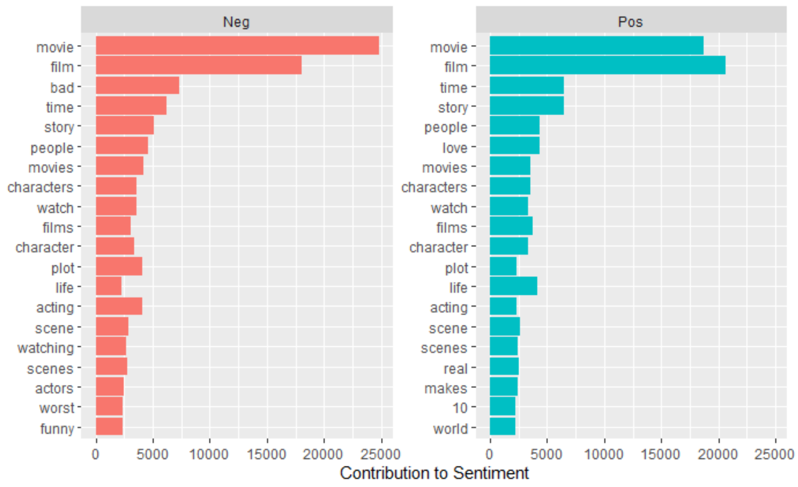
terms	avoid_this	awful	worst
beta	-0.59	-0.53	0.50

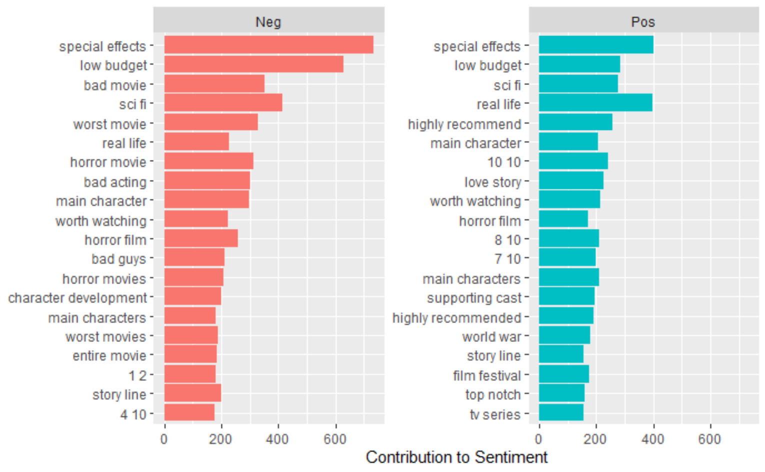
In a hypothetical situation discussed on piazza @820, we would be able to tell the judge why our algorithm assigns this particular score to this review, by looking at the fitted coefficient listed above and calculate the probability for each review based on coefficients of each term. One can easily calculates the score of each review based on these coefficients using the standard logistic regression formula. Qualitatively, the obtained score is proportional to the number of positive/negative terms (taking into account their weights) appearing in the review.
In addition, we can also qualitatively explain our result through the positive and negative word lists obtained with two-sample t test. If a review contains high numbers of positive words with large p values, it should be classified as a positive review and vice versa.

5. Implementation and interesting findings

We're using packages like "tidytext", "tidyr" and "ggplot2" to explore n-gram's advantages over bag of words in sentiment analysis. We found out if only consider individual words as simplified representation of meaning and sentiment disregarding word order or relationships, much context information are missed out.

5.1 Most common positive and negative words





Counting by single words, we can see there're many nouns that have neutral meaning itself showing on both list (the most frequent words contributing to positive reviews and negative reviews). Also, some positive words like "funny" ended up on the negative top 20 list because sequences of words and context information are omitted here. However if we apply bi-gram tokenization, the top 20 most common terms are more intuitive and sense-making. For example, "bad movie" and "bad acting" gain popularity among negative reviews while "highly recommend" and "top notch" are very common for positive ones.

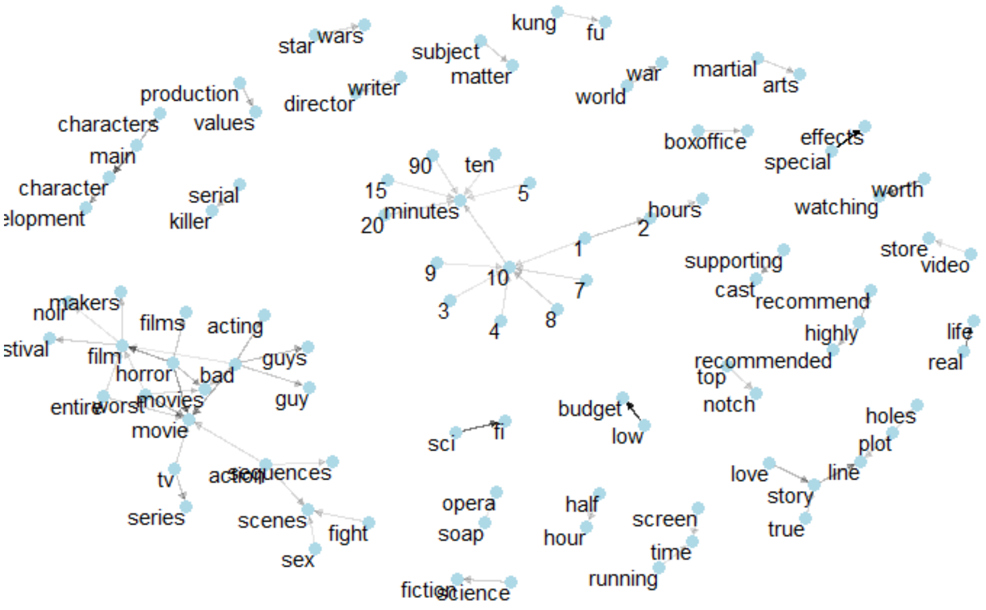
5.2 Providing context in sentiment analysis

Pos	love	simple	short	ghost	original	real	main	beautiful	life	police
Neg	love	original	real	ghost	bad	short	basic	life	stupid	actual

Pos	wonderful	toy	detective	moving	straight	entire	fascinating	human	told	touching
Neg	christmas	main	classic	ridiculous	boring	decent	simple	lame	awful	predictable

Pairs of consecutive word can provide more complex meaning that can't be captured by single words. Here in the example, we listed the most common words combined with "story" by sentiment. Even though love story can be either "positive" or "negative" depending your luck, the following adjectives such as "ghost", "bad", "stupid" are clearly demonstrating a negative sentiment while "beautiful" "fascinating" and "touching" indicates different emotions. N-gram in this case, bigram make tokens more understandable and thus can be quite useful to discover the sentiment underneath.

5.3 Visualizing a network of bigrams



Package "igraph" and "ggraph" equip us the ability to visualize all the relationship among words at the same time. For example, it's easy to spot common centers of nodes formed by movie-related terms, like "movie" and "film", 10 out of 10-point scale and the length of movie in minutes. Some movie genres include "sci-fi", "kung fu", "soap opera" and so on. "Story" like we mentioned before is a complicated word connecting many words having different meanings. A graph like this reveals some details of how reviews are structured, how words are connected with each other, and how different minds express the same sentiment.

6. Running time and CPU

We use MacBook Pro Retina, 2.2 GHz Intel Core i7 16 GB 1600 MHz to run and the result without generating vocabularies is 9 minutes but with generating the words would be 14 minutes. The R version used is 4.0.2.

7. Outlook and future steps

In the future, how might this algorithm be improved? The answer lies in the ability of interpretation and accuracy of prediction. Some directions of future work include: 1. even further shorten the vocabulary list which results in easier interpretation. We have explored the bootstrap approach introduced by professor Liang in the office hour. However, after rounds of turning parameters, we still have trouble landing the overlap list smaller than 1000 words using only split 1 dataset. Other more complicated models might be worth trying such as recurrent neural network (RNN) or random forest to see if they can overcome the model limitations we mentioned in section 3.