Prediction on the Helpfulness of Amazon Online Reviews using Text Mining

Siqi Liu
Mathematics Department, University of California San Diego
siqiliu080@gmail.com

Abstract

There exist numerous reviews on different products online and the number is still dramatically increasing. These reviews vary tremendously in quality. Manually assessing the helpfulness of each review is an intimidating and impossible task. In this paper, we try to address the problem by applying text mining to automatically predict how helpful a given review is. We attempt the problem from two aspects. First, we formulate a binary classification task by predicting whether a review will receive less than 5 votes or not. Second, for a review with 5 or more votes, we further build a penalized linear regression model to predict its helpfulness score. Experimental results show that for the classification task, decision tree performs best. For the regression task, the best model through exhaustive model selection can achieve a Pearson value of up to 0.46 for the testing set.

1. Introduction

As the influence of Internet increases, the ways people shop for goods also change sharply. More and more people get used to reading product reviews before deciding whether they will purchase the product or not. At the same time, more and more customer reviews are available online for a wide range of products as well as services. In some big e-commerce websites such as Amazon, Alibaba, IMDB, CNET, there exist numerous reviews. However, the large quantity of reviews and the large variations in the reviews are hampering the effective use of these reviews. The more reviews people can find, the harder it will be for people to read truly helpful reviews flooded by a significant amount of low quality reviews.

To alleviate this problem, most such websites rank reviews by product rating scores, for example, in amazon.com, people can rank the reviews by sorting the number of stars. A more sophisticated scheme is to sort the reviews by their review helpfulness scores. Many websites allow readers of a review to vote whether they think the review is helpful or not. With the reviews sorted by helpfulness, people can make their purchase decisions more easily and accurately.

Although this is a good direction to find helpful reviews, nevertheless, there are some issues. One major problem is the insufficient number of votes for a large amount of reviews. This is especially true for the newly written reviews and other reviews that are barely read by people due to the low-traffic issue. Manually assessing the helpfulness of each review is a very time-consuming procedure and it is an almost impossible task for the large number of reviews. Therefore, it will be highly desirable if the helpfulness of a given review can be automatically predicted.

In the literature, a lot of existing works have made significant progress in predicting review helpfulness. Here we list a few well-known papers that exploit text mining for helpfulness prediction [1][2][3][4]. Following these well-established works, in this paper, we keep the text mining path and try to find out a good model to predict the helpfulness of a review. Our model is based on some major factors that may affect the helpfulness of a review, including lexical features such as unigram and bigram, structural features such as number of words and number of sentences in each review, semantic features such as sentiment of the review, and some other features such as timeliness of the review, rating score of the review, etc.

We will randomly pick a dataset for our project. However, unlike [1] that filters out the reviews with votes less than 5 before data modeling, we keep all the observations. Instead we divide the data modeling task into two subtasks: (1) we consider a binary classification task to predict whether a review will receive 5 or more votes or not. Several classification methods including decision tree, random forest, SVM and logistic regression will be tested. (2) For reviews with 5 or more votes, we further attempt to build a model predicting its helpfulness score. We tackle the problem by applying a penalized linear regression that takes all important factors into account, and apply cross-validation techniques to prevent the model from over-fitting. The final model is selected by exhaustively searching for the one with the best performance in the testing data. We anticipate that with our prediction models, it will be easier and more accurate to determine a review's ranking so that e-commerce website such as Amazon can discover the most helpful reviews more quickly and automatically.

The rest of the paper is organized as follows. In Section 2, we preliminarily explore the dataset by preprocessing the dataset and also giving our task definition. In Section 3, we provide a detailed analysis of the major chosen features that may affect the review helpfulness prediction. In Section 4, we present our proposed binary classification modeling and show some prediction results using different classification methods. In Section 5, we present our penalized linear regression modeling and show the model selection results along with discussions on the results. We conclude the paper in Section 6. All the R codes for this text mining project are attached in the appendix.

2. Dataset Preprocessing and Exploration

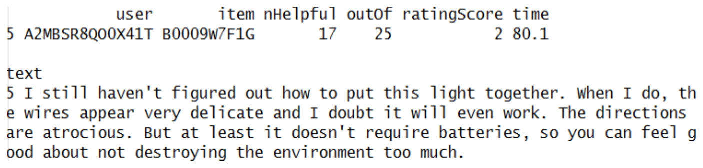
In this section, we formally introduce the Amazon review dataset and define our project goal. We then explore and preprocess the dataset.

2.1 Amazon Review Dataset

Amazon review dataset consists of product reviews for over 30 categories. These categories range widely from cell phone to automotive, and from jewelry to shoes, etc. The entire dataset includes over 34 million reviews with a timespan from June 1995 to March 2013. In this project, we randomly pick one category which is Sports_&_Outdoors and focus our text mining project on this product category. This dataset contains a total of 263,781 reviews and the original data are stored in the format of “json”. We parse the dataset using the Python language and extract all the observations with the most relevant variables including “user”, “item”, “outOf”, “nHelpful”, “ratingScore”, “time” and “text”.

The variable “user” denotes the ID of a user shopping for goods in Amazon. The variable “item” denotes the ID of each product. The variable “outOf” represents the number of total votes for each review. The variable “nHelpful” stands for the number of people that find the review helpful. The variable “ratingScore” denotes the number of stars given by the user to the item (the minimum ratingScore is 1 while the maximum ratingScore is 5). The variable “time” represents the elapsed months starting from the publication time of the first review. Finally, the variable “text” gives the text of each product review. We export all the extracted observations into a csv file such that we can perform data exploration and analysis in R.

To have a glimpse of the raw dataset, we randomly pick one sample and show it in Figure 1. For this random sample, nHelpful = 17 and out of = 25 indicate that this review has 25 votes where 17 people vote for helpfulness; ratingScore = 2 indicates that the user gives the item 2 stars out of 5 stars. time = 80.1 indicates that the review was published 80.1 months after June 1995.



```
            user       item nHelpful outOf ratingScore time
5 A2MBSR8QO0X41T B0009W7F1G       17    25           2 80.1

text
5 I still haven't figured out how to put this light together. When I do, th
e wires appear very delicate and I doubt it will even work. The directions
are atrocious. But at least it doesn't require batteries, so you can feel g
ood about not destroying the environment too much.
```

Figure 1: One sample from the raw dataset

This raw dataset with 263,781 observations contains 182,635 unique users and 42,435 unique items. In particular, we are interested in seeing how many reviews have votes less than 5 and how many reviews have 5 or more votes. In Figure 2, we give this barplot where we could see that the observations with votes less than 5 outnumber a lot the observations with 5 or more votes.

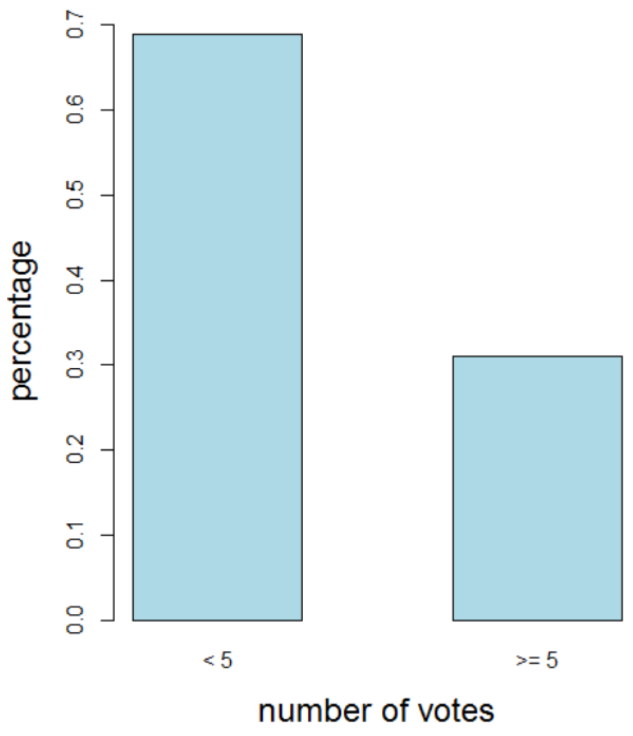


Figure 2: Percentage for number of votes

2.2 Task Definition

The goal of our project is twofold: First, we want to build a binary classification model predicting whether a review will get 5 or more votes or not. Our hypothesis is that if few people vote for a review, this review must be boring and unhelpful. Therefore, with this model, we could predict a review to be unhelpful if its predicted number of votes is less than 5.

Second, for a review with 5 or more votes, we want to further refine the prediction by building a regression model that predicts how helpful the review is. Defining

$$\text{Helpfulness} = \text{nHelpful}/\text{outOf}, \qquad (1)$$

our second task is to predict the helpfulness score.

In short, our project aims to apply text mining for a two stage model with a first binary classification task followed by a regression task.

2.3 Dataset Cleansing

To facilitate the feature selection process and model selection process for our text mining project, we need to preprocess the raw dataset first. This is due to that a large amount of the raw text reviews are “dirty”. For example, in many reviews, many observations are duplicates of others, many words are mis-spelled or run together, punctuations or arbitrary symbols may appear randomly, etc.

To cleanse the text review data, we go through the following steps:

- **Step 1: Remove duplicated reviews.** We filter out observations which are detected as duplicates of other existing observations. This step removes over 80K observations and remains 177,867 observations for further processing. Moreover, we find that the number of observations with less than 5 votes is now slightly larger than the number of observations with 5 or more votes. This implies that most duplicates are from the observations with few votes.
- **Step 2: Replace repeated letters.** We find that there are some words like “aaa”, “aaaaargh”, “aaaaaahhhhhyaaaaaa”, etc. in some reviews. Since it is unlikely that an English word would have three or more successively identical letters, we write a function to replace these more than 2 identical letters by its corresponding single letter. For example, we replace “aaa” by “a”, and replace the word “aaaaaahhhhhyaaaaaa” by “aha”.
- **Step 3: Replace non-English letters (such as Latin words, math symbols, etc.) with space.**
- **Step 4: Convert all letters to lower-case.**
- **Step 5: Remove punctuations.**
- **Step 6: Remove numbers.**
- **Step 7: Remove stop words.**
- **Step 8: Extract the stem of each word.**

Most of these steps are performed by calling some built-in functions in the R packages “tm” and “SnowballC”.

3. Feature Extraction

In this section, we describe and analyze various important features. These features are correlated to our predictive tasks and would be all or partially included in our classification and regression models.

3.1 Lexical Features

Lexical features can help us capture the words observed in the reviews. Specifically we consider two commonly used bag-of-words features

- **Unigram**: The TF-IDF statistic of each unigram word occurring in a review.
- **Bigram:** The TF-IDF statistic of each bigram phrase occurring in a review.

TF-IDF statistic is the combination of Term Frequency (TF) and Inverse Document Frequency (IDF). The TF-IDF value of a term t in a given document d is defined as

$$TF-IDF\ (t,\ d)$$
$$= TF\ (t,\ d)\ \times\ IDF\ (t)$$
$$= TF\ (t,\ d)\ \times \left[1+\log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ t}\right)\right]$$

For both unigrams and bigrams, we normalize the number of TF-IDF counts for each keyword and remove the sparse terms. After this step, we show two word cloud figures with 77 unigram keywords and 85 bigram keywords. As shown in Figure 3 and 4, it is interesting to note that some words or phrases like “great”, “bought”, “like”, “high recommend”, “easi use”, etc. show up.

These words could be highly correlated to our prediction task.

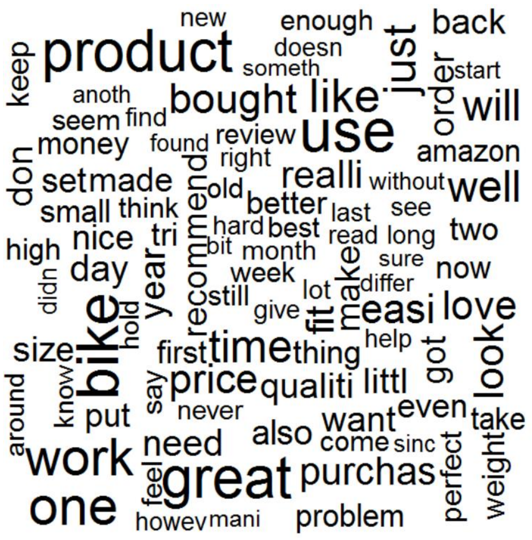


Figure 3: Unigram word cloud

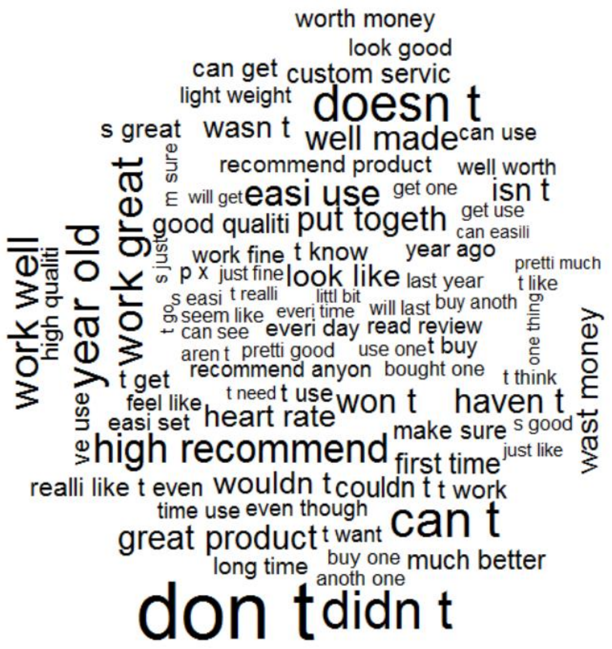


Figure 4: Bigram word cloud

3.2 Structural Features

Structural features capture the document structure and formatting. Generally, longer reviews often include more product information, and detailed personal opinions towards the products, etc. On the contrary, shorter reviews are often less informative. Therefore, structural properties such as number of words and number of sentences in a review text are hypothesized to be related to our predictive tasks. Hence, we consider the following structural features.

- **Number of words:** The total number of words in a review text.
- **Number of sentences:** The total number of sentences in a review text.

After counting the number of words and number of sentences in each review, we plot their histograms in Figure 5 and 6, respectively.

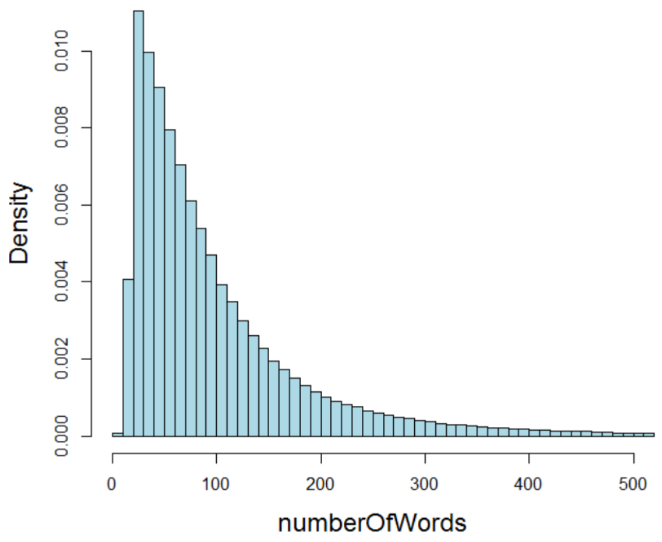


Figure 5: Histogram for number of words

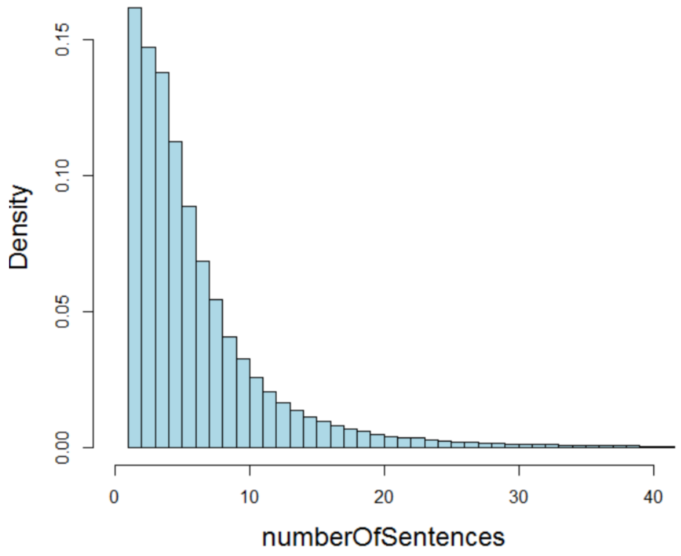


Figure 6: Histogram for number of sentences

Figure 5 and 6 show that in this dataset, most reviews have words less than 200 and sentences less than 20.

3.3 Sentiment

Sentiment or emotion is another important feature for our prediction task as it is a key factor to summarize customer reviews. A complex model is to analyze eight emotions for a review including anger, fear, anticipation, trust, surprise, sadness, joy, and disgust [2]. For simplicity, in our project, we only consider two global measures of sentiment including positive sentiment and negative sentiment. In R, we import two dictionaries, with one containing 2006 positive words and the other containing 4783 negative words. By comparing with the two dictionaries, we count the number of positive words and number of negative words in each review text.

Moreover, for each review text, we also compute the ratio between the sum of the sentimental words and the entire number of words, and include it as another informative feature.

3.4 Rating Score

Most websites require reviewers to choose an overall rating score for the product they purchase. For example, in Amazon.com, star ratings for online customer reviews range from one star to five stars. A very low star rating such as one star indicates an extremely negative view of the product, conversely, a very high rating score such as five stars reflects an extremely positive view of the product. In general, our prediction task may significantly depend on rating score and thus should be included in our models. In Figure 7, we plot the percentage of different star ratings in the dataset, where we see almost half of the reviews have 5 stars.

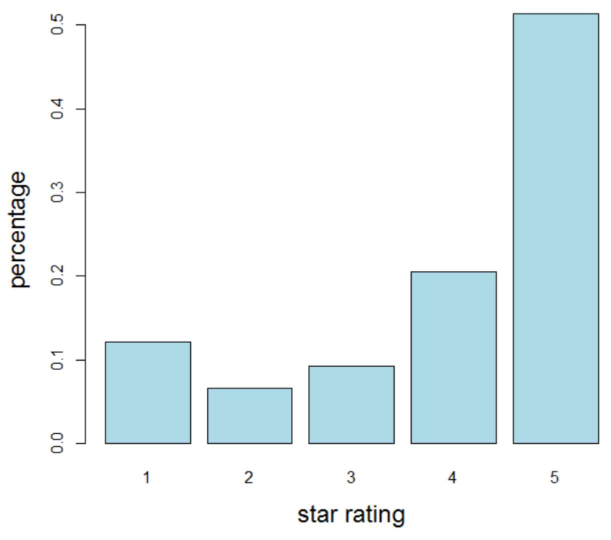


Figure 7: Percentage of different star ratings

3.5 Timeline

In addition to the review content, most online reviews are also associated with a time stamp, which indicates when the review is published. Generally speaking, the helpfulness of the review may also depend on when it is posted. In our project, we consider the time difference defined as

$$T_{diff} = t - t_0,$$

where t denotes the time when the review is published and t_0 denotes the starting time which is June 1995. The unit for this feature is month.

4. Helpfulness Prediction using Binary Classification

In this section, we focus on the first task to build a binary classification model predicting whether a review will receive 5 or more votes or not.

4.1 Problem Revisiting

The target variable is whether a review has 5 or more votes or not. To prepare for this, we create a new variable in the dataset and set it to be one if the observation's outOf >= 5 and zero otherwise. For the predictor variables, we include all the variables discussed in Section 3.

For this binary classification task, we resort to the software Rattle. Since the modeling in Rattle can be quite computationally expensive for a large amount of data, we randomly sample 5K observations from the original dataset. These observations are further divided into training set (70%), validation set (15%) and testing set (15%). We would train the model using the training set and validation set, and evaluate the model in the testing set.

4.2 Classification Methods

For this binary classification task, there are a few off-the-shelf methods that we can use in Rattle. These methods include decision tree, random forest, logistic regression, and support vector machine (SVM). Our strategy is to try them all and let the final classification accuracy determine which method works best. Note that this strategy may not be feasible if we have a large dataset because exhaustive searching for a best model would then be both time-consuming and memory-consuming.

For each method in Rattle, there are a few parameters that we need to specify. For this task, we choose the default settings. For example, for decision tree, we set min split = 20, min bucket = 7 and max depth = 30; for random forest, we set number of trees = 500, etc. How to tune better parameters could be interesting future work.

4.3 Experimental Results

For performance evaluation, a few different metrics can be used such as error matrix, ROC, etc. For simplicity, we report the error matrix (aka confusion matrix) for each method here.

		Predicted		
		0	1	error
Actual	0	240	163	40%
	1	144	203	41%
	Decision Tree			

		Predicted		
		0	1	error
Actual	0	212	191	47%
	1	165	182	48%
	Random Forrest			

		Predicted		
		0	1	error
Actual	0	243	160	40%
	1	177	170	51%
	Support Vector Machine			

		Predicted		
		0	1	error
Actual	0	149	254	63%
	1	186	161	54%
	Logistic Regression			

Table 1: Prediction results for different classification methods

As shown in Table 1, the four confusion matrices indicate that the decision tree gives the best performance and outperforms other more sophisticated methods. This is a bit surprising to us. We speculate that this may be due to the randomness when splitting the dataset into training and testing sets. However, even though decision tree performs best, its overall classification accuracy is only around 60%. To improve the accuracy, one way is to try different combinations of predictor variables. Further exploration in this direction could be an interesting project.

5. Helpfulness Prediction using Regression

In this section, we discuss on how to build a penalized linear regression model to predict helpfulness. We first explore the relationship between the helpfulness score and some other predictor variables, and then give a brief introduction of the penalized linear regression with LASSO. After that, we present our experimental results by showing the best and worst models.

5.1 Problem Revisiting

The target variable is the helpfulness score defined in equation (1). To prepare for this, we filter out all the reviews with votes (i.e., outOf) smaller than 5. After this filtering, the number of observations drops to 81,900. This refined dataset is further divided into training set (90%) and testing set (10%). Similar as the classification task in Section 4, we include all the variables introduced in Section 3 as the predictor variables for the regression task.

5.2 Relationship of Helpfulness with Predictor Variables

In this subsection, we conduct further data exploratory analysis to show how the target variable helpfulness is correlated to all the predictor variables.

5.2.1 Number of Words and Number of Sentences

In Section 3, we have introduced two structural features including number of words and number of sentences. Here, we give two plots to visually show how the helpfulness score is correlated to these two predictor variables. In Figure 7 and 8, we have two observations: First, most reviews have words less than 1000 and sentences less than 100. Second, as the number of words or the number of sentences increases, the helpfulness score also increases. This is quite intuitive since a longer review contains more information and naturally its helpfulness score should be larger.

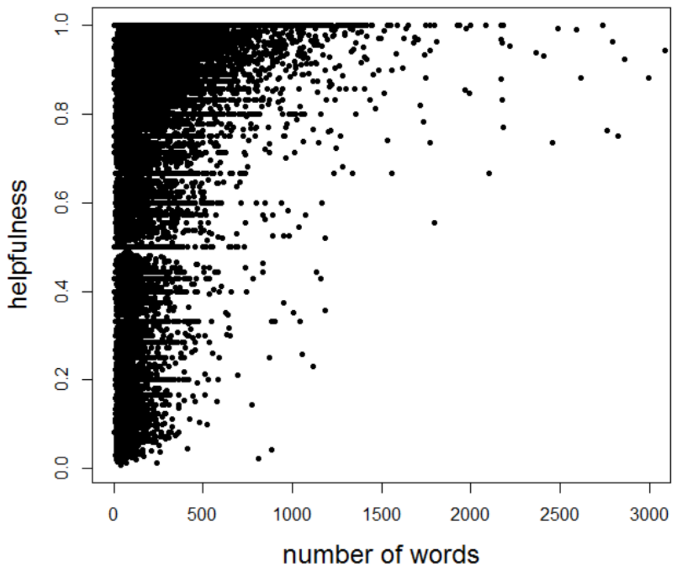


Figure 7: Helpfulness VS. number of words

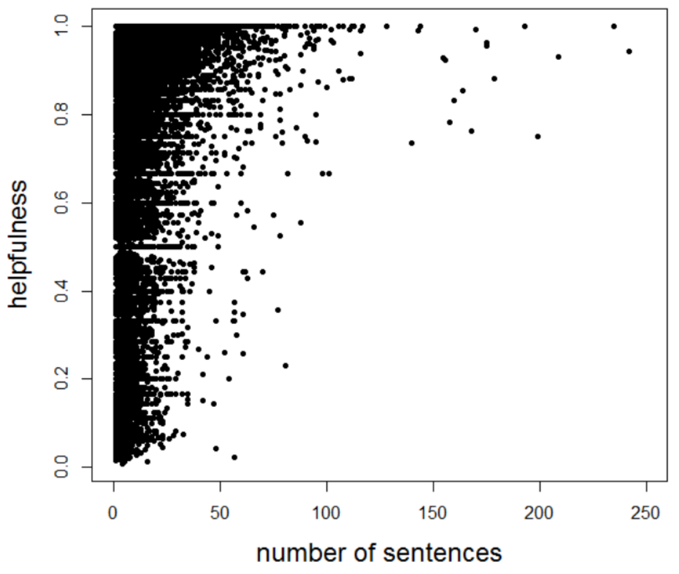


Figure 8: Helpfulness VS. number of sentences

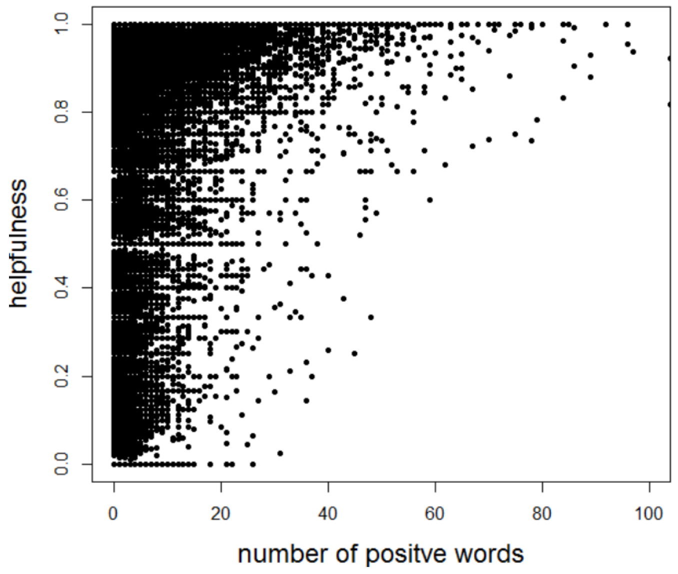


Figure 9: Helpfulness VS. number of positive words

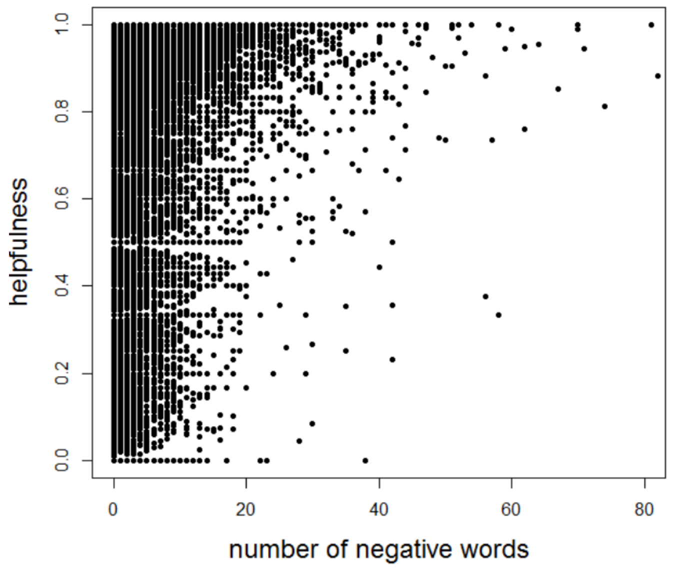


Figure 10: Helpfulness VS. number of negative words

5.2.2 Sentiment

For sentiment analysis, we give two plots to visually show how the helpfulness score is correlated to the number of positive words and number of negative words. In Figure 9 and 10, we find that normally the helpfulness score is positively correlated to the number of positive words or the number of negative words. Comparing Figure 9 with 10, we could tell from the density that in this dataset there are more review texts that give more positive words and negative words.

5.2.3 RatingScore

For the predictor variable "ratingScore", since it is more like a categorical variable, we explore its relationship with the target variable by showing a boxplot in Figure 11. From the boxplot, we find that as the rating score increases, both the mean and the median of helpfulness also increase. It means that reviews with lower ratings are less helpful than reviews with higher ratings. Therefore we expect a linear relationship between the rating score and the helpfulness score.

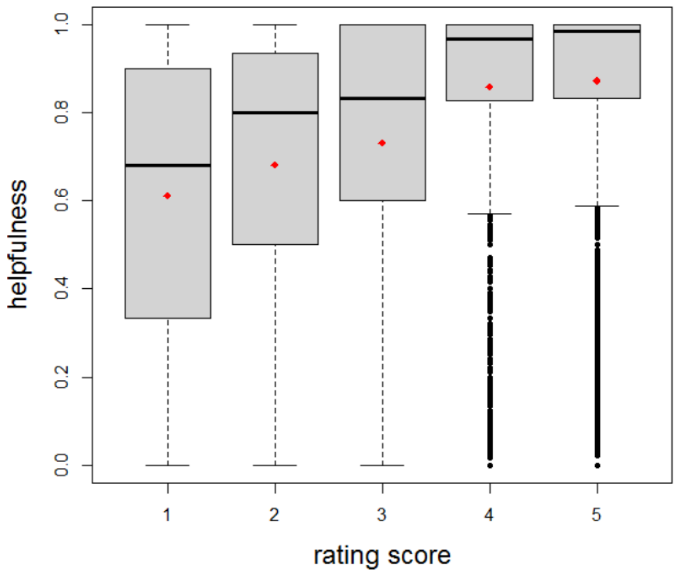


Figure 11: Helpfulness VS. rating score

5.2.4 Time

Generally speaking, the helpfulness of the review also depends on when it is posted. We explore the relationship between the target variable and time by showing a plot in Figure 12. From Figure 12, the general trend shows that the average helpfulness of a review declines as time passes by. Therefore, there is a strong correlation between the helpfulness and the time.

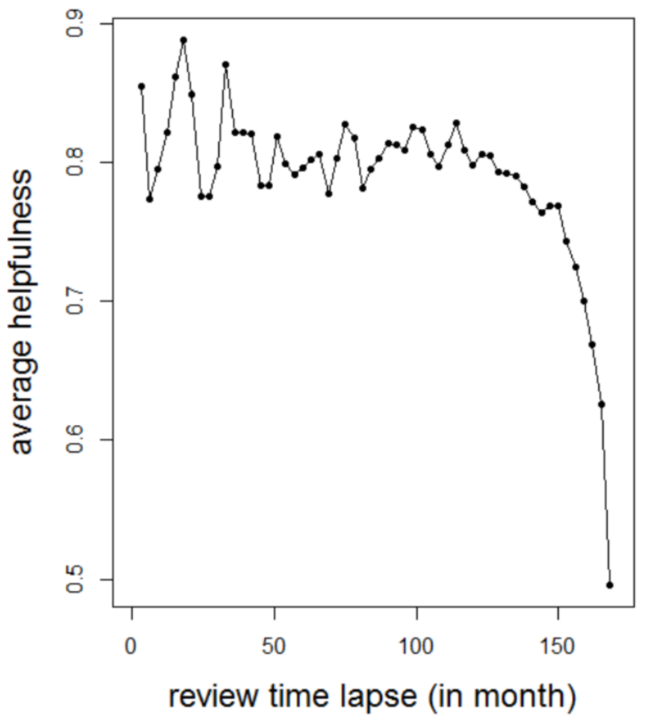


Figure 12: Helpfulness VS. time

5.3 Penalized Linear Regression

After exploring visually the relationship between the target variable and some most important predictor variables, we now come to the stage of modeling the relationship mathematically. As the target variable "helpfulness" is a continuous variable, we choose linear regression as our prediction model.

The linear regression model can be briefly written as follows:

$$y = X\beta + \varepsilon,$$

where $X = (1, X_1, X_2, \cdots, X_n)$ is the observation matrix with n predictor variables, $b = (b_0, b_1, \cdots, b_n)'$ denotes the unknown coefficients (aka regressors) and $\varepsilon \sim N(0,1)$ represents the i.i.d. Gaussian noise. The residual sum-of-square is defined as

$$RSS = \|y - X\beta\|^2 .$$

Linear regression aims to minimize the RSS by finding the optimal coefficients. However, for a model with a large amount of predictor variables (for example, we have 77 unigram variables, 85 bigram variables, number of words, number of sentences, rating score, sentiment, and some interaction terms of these features), there might exist problems such as multicollinearity. Also, a large amount of predictors could potentially over-fit the training set. Therefore, it is desirable to select a small subset of all the predictor variables that yield a model which generalizes the training set but does not memorize the training set. To this end, we could apply penalized linear regression. In this project, we consider linear regression with least absolute shrinkage and selection operator (LASSO). LASSO is a penalized least-square procedure that minimizes RSS subject to non-differentiable constraint expressed in terms of the L_1 norm of the coefficients. The LASSO estimator is given by

$$\hat{\beta}_{LASSO} = \arg\min_{\beta} \|y - X \bullet \beta\|^2 + \lambda \sum_{j=1}^{n} |\beta_j|,$$

where $\lambda \geq 0$ is a tuning parameter.

LASSO penalized linear regression problem can be iteratively solved using gradient descent. The top best models not only fit the training data with the smallest RSS but also have small model complexities. The optimal tuning parameter λ is usually found through cross validation.

5.3 Experimental Results

We list out all possible combinations of the predictor variables as our candidate models. As we have 9 predictor variables, this results in a total of in 511 = $(2^7 - 1)$ candidate models. Our strategy is to apply the penalized linear regression with LASSO on all the candidate models and finally sort all the models by their Pearson correlation values. A larger Pearson correlation value (aka correlation coefficient) indicates a better fitted model. It is worth emphasizing that the model coefficients are learned on the training set while the Pearson correlation values for model evaluations are computed on the testing set.

npred	model	Pearson
7	unigram+bigram+ratingScore+numberOfPositiveWords+numberOfNegativeWords+ratio+time	0.46153
8	unigram+bigram+ratingScore+numberOfWords+numberOfPositiveWords+numberOfNegativeWords+ratio+time	0.46148
9	unigram+bigram+ratingScore+numberOfWords+numberOfSentences+numberOfPositiveWords+numberOfNegativeWords+ratio+time	0.46147
8	unigram+bigram+ratingScore+numberOfSentences+numberOfPositiveWords+numberOfNegativeWords+ratio+time	0.46146
6	unigram+bigram+ratingScore+numberOfPositiveWords+numberOfNegativeWords+time	0.46134
7	unigram+bigram+ratingScore+numberOfWords+numberOfPositiveWords+numberOfNegativeWords+time	0.46132
7	unigram+bigram+ratingScore+numberOfSentences+numberOfPositiveWords+numberOfNegativeWords+time	0.46130
8	unigram+bigram+ratingScore+numberOfWords+numberOfSentences+numberOfPositiveWords+numberOfNegativeWords+time	0.46130
7	unigram+bigram+ratingScore+numberOfWords+numberOfPositiveWords+ratio+time	0.46122
8	unigram+bigram+ratingScore+numberOfWords+numberOfSentences+numberOfPositiveWords+ratio+time	0.46122

Table 2: Top 10 models

In Table 2, we print out the top 10 models that give the highest Pearson values while in Table 3, we print out the worst 10 models that give the lowest Pearson values. Comparing the top 10 models alone, we first observe that the best model involves 7 predictor variables which is better than the full model with 9 predictor variables. Second, we also note that the Pearson values of the top models are close to each other with marginal differences. This finding reveals that some predictor variables hardly matter the prediction accuracy given other predictor variables. For example, in comparison of the best model with the third best model, we find that given all the other variables, the variables "numberOfWords" and "numberOfSentences" hardly improve or decrease the accuracy much. We speculate that some predictor variables might have multicollinearity relationships.

npred	model	pearson
1	ratio	0.04509
1	time	0.06301
2	ratio+time	0.07473
1	numberOfNegativeWords	0.12648
2	numberOfNegativeWords+ratio	0.13387
2	bigram+ratio	0.14459
2	numberOfNegativeWords+time	0.14759
2	bigram+time	0.15153
3	numberOfNegativeWords+ratio+time	0.15230
3	bigram+ratio+time	0.15446

Table 3: Worst 10 models

Comparing these top models in Table 2 with the worst models in Table 3, it is obvious that the top models are much better than the worst models. For example, the worst model has a Pearson value of only 0.04509. This demonstrates the superiority of model selection through exhausting searching.

In [1], the authors showed that the combination of predicator variables "numberOfWords + unigram + ratingScore" gives the best performance in predicting helpfulness score. We also test this model in our dataset and compare its performance with our best models. The result shows that the model in [1] has a Pearson value of 0.45473 which is even worse than our 10th best model shown in Table 2.

Variable name	Coefficients
ratingScore	0.0567436699
numberOfPositiveWords	0.0062371839
numberOfNegativeWords	0.0017085699
Ratio	-0.1420027210
Time	-0.0004466600
fit (unigram)	0.1821069003
quality (unigram)	0.1039582880
recommend (unigram)	0.1994598109
great product (bigram)	0.0965838709
high recommend (bigram)	0.0519033769
well worth (bigram)	0.0966798983
work great (bigram)	0.0750698124
great (unigram)	-0.1001381965
realli like (bigram)	-0.0006124514
so easy (bigram)	-0.0783369826
pretti good (bigram)	-0.0372502371

Table 4: coefficients of some predictor variables in our best model

To further evaluate the correlations of the predictor variables with the target variable, we intend to analyze the learned coefficients of the best model. Due to limit of space, we show only a few coefficients in Table 4. The predictor variables “numberOfPositiveWords”, “ratingSc ore”, “numberOfNegativeWords” have positive coefficients, indicating that they are positively correlated to the target variable. This is consistent with our previous analysis in Figure 9, 10 and 11. The variable “time” is negative which also agrees with our analysis in Figure 12. For unigram and bigram, we find that some words or phrases like “fit”, “qualiti”, “recommend” and “great product”, “high recommend”, “well made”, “work great” have positive coefficients, which indeed align with our common sense and expectation. However, we also find that some words or phrases like “great”, “realli like”, “pretti good” have negative coefficients. We speculate that if a reviewer tends to use some exaggerated or emotional words or phrases in the review, customers tend to vote this review as less helpful. Further, there are a bunch of keywords in unigram and bigram that get 0 coefficients but are not shown in Table 4. This is due to the merit of applying the penalized model with LASSO.

6. Conclusion

In this paper, we applied text mining on Amazon review dataset to explore the possibility of predicting the review helpfulness. We divide the text mining task into two subtasks where we first build a binary classification task to predict whether a review will receive 5 or more votes or not, and second for a review with 5 or more votes, we further refine its helpfulness prediction by building a penalized linear regression. We designed and explored several predictor variables and showed how they are correlated to the predictive tasks. Our final models achieve reasonable prediction accuracy and even improve some existing work. As more and more diverse reviews surge into the Internet, we anticipate that automatically assessing the review helpfulness using text mining and other data mining techniques would become even more significant.

7. Reference

[1] S.M. Kim, P. Pantel, T. Chklovski, and M. Pennacchiotti, “Automatically assessing review helpfulness,” in Proc. Conf. on Empirical Methods in Natural Language Process. (EMNLP’06), pp. 423-430, Sydney, July, 2006.

[2] M. Hu, and B. Liu, “Mining and summarizing customer reviews,” in Conf. on Knowledge Discovery and Data Mining (KDD’04), Seattle, August, 2004.

[3] Y. Liu, X. Huang, A. An, and X. Yu, “Modeling and predicting the helpfulness of online reviews,” in IEEE Conf. on Data Mining (ICDM’08), 2008

[4] S.M. Mudambi, and D. Schuff, “What makes a helpful online review? A study of customer review on amazon.com,” MIS Quarterly vol. 34, pp. 185-200, March, 2010.

8. Acknowledgment

We would like to express our sincere gratitude to Prof. Roger Bohn. He has provided invaluable suggestions. His insightful comments have made this project a joyful and meaningful experience for me.

9. Appendix

Note: We attach all the important R codes for your reference. Some less important codes including those for plotting are not shown here. We have used quite a few R packages for this text mining project including “tm”, “SnowballC”, “RWeka”, “wordcloud”, “glmnet”, etc.

Also note that the subtask 1 is performed using the software Rattle and no R code is needed for this subtask 1.

```
########################### R Codes for the Text Mining Project ############################
dataset = read.csv("dataset/data_6_6_2015.csv")
colnames(dataset) = c("user", "item", "nHelpful", "outOf", "ratingScore", "time", "text")
dataset$text = as.character(dataset$text)
nlevels(dataset$user)
nlevels(dataset$item)

######## remove all duplicated reviews and define the target variable
dataset = dataset[!(duplicated(dataset[, c("text")]) | duplicated(dataset[, c("text")], fromLast = TRUE)), ]
dataset = dataset[dataset$outOf >= 5, ]
dataset$target = dataset$nHelpful / dataset$outOf

######## replace more than 2 identical letters by the corresponding letter
removeRepeatLetters = function(str) gsub('([[:alpha:]])\\1{2,}', '\\1', str)
dataset$text = sapply(dataset$text, removeRepeatLetters)

######## compute the number of sentences for each review
countnumberOfSentences = function(str) length(gregexpr('[[:alnum:] ][.!?]', str)[[1]])
dataset$numberOfSentences = sapply(dataset$text, countnumberOfSentences)

######## replace non-English letters with space
dataset$text = gsub("[^a-zA-Z]+", " ", dataset$text)

######## compute the number of words for each review
dataset$numberOfWords = sapply(gregexpr("\\W+", dataset$text), length) + 1

######## compute the counts of positive words and negative words, and the ratio
pos.words = scan('positive-words.txt', what='character', comment.char=';')
neg.words = scan('negative-words.txt', what='character', comment.char=';')
score.sentiment = function(sentences, pos.words, .progress='none')
{
  require(plyr)
  require(stringr)
  scores = laply(sentences, function(sentence, pos.words) {
    sentence = gsub('[[:punct:]]', '', sentence)
    sentence = gsub('[[:cntrl:]]', '', sentence)
    sentence = gsub('\\d+', '', sentence)
    sentence = tolower(sentence)
    word.list = str_split(sentence, '\\s+')
    words = unlist(word.list)
    pos.matches = match(words, pos.words)
    pos.matches = !is.na(pos.matches)
    score = sum(pos.matches)
    return(score)
  }, pos.words, .progress=.progress )
  scores.df = data.frame(score=scores, text=sentences)
  return(scores.df)
}
dataset$positive = score.sentiment(dataset$text, pos.words)$score
# dataset$negative = score.sentiment(dataset$text, neg.words)$score

######## compute bag-of-words with TFIDF for both unigram and bigram
text = dataset$text
# install.packages("tm")
library(tm)
text = Corpus(VectorSource(text)) ## inspect(text)
text = tm_map(text, FUN=tm_reduce, tmFuns =
list(as.PlainTextDocument,tolower,removePunctuation,removeNumbers,stripWhitespace))
text = tm_map(text, removeWords, stopwords("english"))
library("SnowballC")
text = tm_map(text, FUN=tm_reduce, tmFuns = list(stemDocument, stripWhitespace))
# install.packages("RWeka")
require(RWeka)
unigramTokenizer = function(x) NGramTokenizer(x, Weka_control(min = 1, max = 1))
tdm_uni = TermDocumentMatrix(text, control = list(tokenize = unigramTokenizer, weighting =
function(x) weightTfIdf(x, normalize = TRUE)))
tdm_uni_ = removeSparseTerms(tdm_uni, 0.91)
bigramTokenizer = function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
tdm_bi = TermDocumentMatrix(text, control = list(tokenize = bigramTokenizer, weighting = function(x)
weightTfIdf(x, normalize = TRUE)))
tdm_bi_ = removeSparseTerms(tdm_bi, 0.99)
unigram = t(as.matrix(tdm_uni_))
bigram = t(as.matrix(tdm_bi_))

######## save all the data before data modeling
attach(dataset)
write.csv(unigram, file = "unigram.csv")
write.csv(bigram, file = "bigram.csv")
write.csv(ratingScore, file="ratingScore.csv")
write.csv(numberOfWords, file="numberOfWords.csv")
write.csv(numberOfSentences, file="numberOfSentences.csv")
write.csv(positive, file="numberOfPositiveWords.csv")
write.csv(negative, file="numberOfNegativeWords.csv")
write.csv(ratio, file="ratioOfPositiveNegativeOverAll.csv")
write.csv(time, file="time.csv")
write.csv(target, file="helpfulness.csv")

######## model selection with different combinations of predictor variables
unigram = read.csv("unigram.csv"); unigram = unigram[, -1]
bigram = read.csv("bigram.csv"); bigram = bigram[, -1]
ratingScore = read.csv("ratingScore.csv"); colnames(ratingScore) = c("X", "ratingScore")
numberOfWords = read.csv("numberOfWords.csv");colnames(numberOfWords) = c("X",
"numberOfWords")
numberOfSentences = read.csv("numberOfSentences.csv"); colnames(numberOfSentences) = c("X",
"numberOfSentences")
numberOfPositiveWords = read.csv("numberOfPositiveWords.csv"); colnames(numberOfPositiveWords)
= c("X", "numberOfPositiveWords")
numberOfNegativeWords = read.csv("numberOfNegativeWords.csv");
colnames(numberOfNegativeWords) = c("X", "numberOfNegativeWords")
ratio = read.csv("ratioOfPositiveNegativeOverAll.csv"); colnames(ratio) = c("X", "ratio")
time = read.csv("time.csv"); colnames(time) = c("X", "time")
helpfulness = read.csv("helpfulness.csv"); colnames(helpfulness) = c("X", "helpfulness")
data_all = cbind(ratingScore, numberOfWords, numberOfSentences, numberOfPositiveWords,
numberOfNegativeWords, ratio, time, helpfulness)
data_all = data_all[, !(colnames(data_all) %in% "X")]

ssize = dim(data_all)[1]
set.seed(1234); idx = sample(1: ssize, size = ssize)
data_all1 = data_all[idx,]  # reshuffle all the data
foldIdx = cut(seq(1,ssize), breaks = 10, labels = FALSE)
testIdx = which(foldIdx == 1, arr.ind = TRUE)

predictorList = c("ratingScore", "numberOfWords", "numberOfSentences", "numberOfPositiveWords",
"numberOfNegativeWords", "ratio", "time")
ma=list() #define ma as a blank list
npred = NULL
k = 1
repeat
{
  com = combn(predictorList, k, FUN = NULL, simplify = TRUE) #produce all combinations with k
  comcol = apply(com, 2, paste, collapse = "+") #merge combinations with "+"
  ma = c(ma, comcol) #add comcol to ma
  npred = c(npred, rep(k, length(comcol)))
  k <- k + 1
  if(k > length(predictorList)) break()
}
ma1=do.call(rbind, ma)##combine vectors into table
eval = data.frame(npred=npred, modeq=ma1)
eval$pearson = NA
nrmodels = dim(eval)[1]
library(glmnet)
for(i in 1: nrmodels)
{
  myformula = as.formula(paste("group ~ 1 +", eval$modeq[i]))
  x_colNames = all.vars(myformula)[-1]
  x = data.frame(data_all[, x_colNames])
  colnames(x) = x_colNames
  ones =  rep(1, dim(data_all)[1])
  x = cbind(ones, x)
  x = cbind(x, unigram) ## unigram, bigram, or both
  y = data_all[, "helpfulness"]
  x_train = as.matrix(x[-testIdx, ])
  y_train = as.matrix(y[-testIdx])
  x_test = as.matrix(x[testIdx, ])
  y_test = as.matrix(y[testIdx])
  fit = glmnet(x_train, y_train, type.gaussian="covariance", alpha=1, standardize = TRUE)
  fit.cv = cv.glmnet(x_train, y_train, type.gaussian="covariance", alpha=1, nfolds = 10)
  intercept = fit$a0[which(fit.cv$lambda.min == fit$lambda)]
  coefficients = fit$beta[,which(fit.cv$lambda.min == fit$lambda)]
  predictions = predict(fit, s=fit.cv$lambda.min, x_test, type="link")
  correlationTest = cor.test(y_test, predictions, method = "pearson")
  eval$pearson[i] = round(correlationTest$estimate, 5)
}
eval_uni = eval
eval_uni$modeq = paste("unigram", eval_unimodeq, sep= "+")
write.csv(eval_uni, "eval_uni.csv")
```