Predicting Sentiment and Usefulness in Text Reviews

Courtni Byun, Stephen Carron, Trenton Hyer
Department of Linguistics
Brigham Young University
thyer@cs.byu.edu

Abstract

This work introduces Biston, a sentiment predictor which builds off earlier work with the Calliope Sentiment Parser Series to introduce new metrics for usefulness prediction. By extracting features from the text, machine learning models are then used to predict how useful a given review will be. Initial results are promising. Biston identifies the most useful reviews with recall rates of over 85%

1 Introduction

The Calliope Sentiment Parser Series (CSPS) is a series of sentiment predictors that specialize in domain-specific texts. They use Bayesian methods to identify key words and score strings on a positive/negative scale. So far, CSPS has introduced one parser which specializes in the Yelp dataset, using its star rating data as a label for sentiment scores, thus increasing the accuracy of the approach.

This paper introduces Biston, which will build off of the earlier sentiment models and include new usefulness statistics. These usefulness statistics will help flag the most interesting reviews, especially when those reviews are also decidedly of negative sentiment.

1.1 Goal

Our goal was to use a small set of features to help automatically determine the usefulness of a text review. We chose to use punctuation, number of alphabetic characters as opposed to non-alphabetic characters, average word length, deixis, double negatives, etc. to make this prediction. We chose some features to help us pick out reviews that would be more useful and we also chose several features to help determine which reviews would not be useful. We felt that this would help us to weed out bad reviews, while simultaneously honing in on the reviews that contained more substance.

1.2 Application and Motivation

These days, there is more information available than many people know what to do with. Popular companies and products will receive anywhere from a few dozen to several thousand reviews each day for customers, some anonymous and some not. Many of these reviews are public, informing other consumers of what kinds of choices they ought to make.

A program that could sort through all of those reviews and give back the reviews containing information that could actually help a company improve would be a great asset. This tool could also benefit investors who might be wondering about consumer sentiment for a company they are looking into. Finally, it might benefit consumers who could become more informed about which reviews are the most helpful, therefore being able to quickly and efficiently determine which product reviews would be best to look at when determining which product they would like to buy.

This tool could help cut down on a lot of unneeded analysis. If a user could quickly sort through all of the reviews to only get the reviews with the most relevant information, they could make faster and better-informed decisions.

2 Data

Our data was provided by Yelp, and contained many features per review. Some basic thought allowed us to select three features from this set, and then some rudimentary computation was done to calculate given linguistic data about the text. These linguistic features were then analyzed using Principal Components Analysis (PCA) and a Greedy Wrapper algorithm. The final feature set included text, usefulness rating, star rating, and our generated features.

2.1 Sources

Data was drawn from the Yelp Academic Dataset, which comes in JSON file format. Yelp makes this dataset available to academic pursuits on an annual basis. The file used for feature generation, selection, and training is denoted as 'yelp_academic_dataset_review.'

Opening the dataset reveals that each review contains several features; some features are the date of review publication, user ID of the writer, and the star rating that the reviewer gave. Of all the features, three were selected to be helpful for this project: (1) 'text' or written statement of the review, (2) 'stars' or the rating that the reviewer gave the business, and (3) the 'usefulness' or number of times each review was voted as useful by other users.

In all, over 1,600,000 reviews were used in the course of this project, with a word count summing about 50 million.

2.2 Basic Feature Selection

Out of the many features available to the learning models, only three were selected. This merits some explanation. There were two reasons we pulled only three features from each review ('text,' 'stars,' and 'usefulness').

First, text was the driving motivation behind this project, since the most practical applications of sentiment/usefulness predictors would occur on unlabeled data. These strings of user feedback comprised the information in which we were attempting to identify a pattern that would enable prediction.

Second, including stars/usefulness while excluding things like date of publication, business ID, etc. serves as a direct measure of just how meaningful each user's text was. Indeed, including the business ID would have been very useful to the models (an unpopular company is more likely to receive a negative review). We felt, however, that the user-reported ratings were more generally informative since they serve as an independent secondary indicator of the text's content. Once again, the real world doesn't tend to have labelled instances, so training models to be prepared to make predictions on minimal information is valuable.

2.3 Feature Generation

In order to identify the features for distinguishing useful reviews from non-useful reviews, we brainstormed several metrics that could help discriminate usefulness. These features needed to (1) have some intuitional value based in linguistics and (2) be fast to compute – no time to waste spinning up a parse tree for every review.

The following list gives a short overview of the generated features we created including a computational explanation and linguistic relevance:

- **function_word_rate:** determined by looking up each word in the review against a predetermined list of function words – a count is incremented each time a function word is identified, and the total is represented as a rate. This can help identify reviews with little content. It can also flag reviews that don't appear to follow natural language patterns.
- **obfuscation:** computed by combining the average word length, number of double negatives, average sentence length, etc. per review. Expressed as a nominal output. This is a pseudo-readability metric, which may be revealing as to how easy or hard a text must be before readers find it useful.
- **word_count:** how many whitespace-separated tokens exist in the review text. Too few words won't contain enough information, too many will be overwhelming.
- **alpha_ratio:** each review's proportion of alphabetic characters. Determined by checking each character, incrementing a count by one for every alphabetic character and returning that count divided by the length of the character string. Can be helpful in identifying spam reviews, which tend to contain lots of non-alphabetic characters to fool Bayesian token filters.
- **punctuation_frequency:** same idea as alpha_ratio above.
- **numerals:** identifies how many times a user either types a digit or otherwise represents a number ("1" and "one" are treated equally). Calculated by matching words to a regular expression. This attribute helps identify when reviewers are giving precise data to the readers, since numbers tend to convey ideas like price, time waited, etc.
- **superlatives_comparatives:** calculates the number of superlative or comparative words in a sentence. Eventually discarded because it was deemed highly inaccurate and increasing the accuracy would have resulted in too long of a computation time.
- **deixis:** determined by checking each word against a list of the most common deictic words. This is another check to ensure the reviewer is using natural human speech (deictic rate should fall somewhere between 0.2 and 0.7), plus it can identify when the text conveys lots of personal experience (many uses of "my" or "yesterday").

2.4 PCA Feature Selection

Principal Components Analysis (PCA) was used to gauge which features contributed the most to dimensional variance of the dataset. PCA revealed two key points of information:

- word_count and char_count had nearly a perfect correspondence with a multiplicative offset of 4.92 (the approximate amount of characters in an English word).
- deixis, punctuation_frequency, alpha_ratio, and function_word_rate appeared together in three principle components, with normalized eigenvalues of 0.6219, 0.2449, and 0.1301 respectively.

Interestingly, word_count and char_count were discounted together by PCA, but alpha_ratio, punctuation_frequency, and numerals were not. This is likely owing to some sort of nonlinear relationship between the three, since PCA would be unlikely to catch that trend.

After removing char_count, PCA returned the following output:

```
Ranked attributes:
 0.6219  1 0.528deixis-0.497punctuation_frequency+0.488
 0.4173  2 0.65 word_count+0.498obfuscation-0.34alpha_r
 0.2449  3 -0.81function_word_rate+0.386numerals-0.255d
 0.1301  4 0.619numerals-0.551word_count-0.467punctuati
 0.0614  5 0.765obfuscation-0.503word_count+0.307punctu
 0.0269  6 -0.736deixis+0.434alpha_ratio+0.388function_
```

Figure 1. Principal Components of 40,000 reviews

2.5 Greedy Wrapper Selection

A common theme among the selected attributes of greedy wrapper was that pretty much any feature can appear significant, but only a few are consistently important. After trying many variations of the assessment model, nearly every attribute appeared at least once in one of the greedy wrapper suggestions. After 15 such iterations, the following attributes won with the highest number of votes:

- word_count – 15 votes
- deixis – 14 votes
- numerals – 13 votes
- function_word_rate – 12 votes
- alpha_ratio – 10 votes
- obfuscation – 7 votes
- punctuation_frequency – 7 votes

2.6 Discussion of Features

As a final note to this section, it ought to be noted that the generated features have acknowledged strengths and weaknesses, tradeoffs that were made in the interest of time and computability.

Because the generative computation does nothing more complicated than lookup and arithmetic operations, the runtime is very fast – 100,000 reviews could be prepared in several minutes. Had the text been run through a parser to identify POS tags and then count those tags, the run time for 100,000 reviews likely would have run into several hours.

Lack of robust parsing, however, ought also to be counted as a weakness – sacrificing syntactic context is very likely to mean a reduction in information.

Additionally, we need to mention that these generated features operate independently between models of sentiment and models of usefulness. As will be discussed, sentiment prediction occurs by means of a Bayes Naïve approach, which ignores the features we've generated in favor of a Bayesian model. Usefulness prediction, on the other hand, uses the generated features, but totally ignores the text itself so that it can run more quickly through a machine-learning model. This report will not give comparative data to show whether usefulness is better predicted by feature vectors (the approach we used) than a Bayesian bag-of-words approach. Similarly, we won't report whether sentiment could be improved if these generated features were to be added to the probabilistic output, and then run through a learning model. Intuitively, these approaches seemed wrong, and their exploration is left to other researchers.

This report will also not mix sentiment and usefulness numbers in prediction, meaning that the learning model for usefulness will know nothing of the sentiment score, and the sentiment predictor will know nothing of usefulness.

3 Models of Sentiment

The following analysis is carried over from Linus (an earlier iteration of the Calliope Sentiment Parser Series). Although some modifications were made to data normalization, binning of continuous values, etc., the overall model hasn't changed.

Beginning with a baseline model of most common value, a series of iterative Naïve Bayes predictors were used with increasingly accurate results. Accuracy was measured using two benchmarks: (1) RGBenchmark and (2) MCVBenchmark

3.1 Baseline Results

The baseline model simply chose the most common value for star rating and used that as the default prediction for every text file. This predicted 24.5% of instances correctly on 10-fold cross validation.

3.2 Naïve Bayes

The preferred method for sentiment prediction uses a tokenized, context-free flavoring of Naïve Bayes. Each instance takes the text and breaks down the words into individual tokens. Each token is given a sentiment score of the star rating for that review, and then all tokens are averaged in a dictionary throughout all reviews. This means that words which often appear in negative reviews ("dumbest," "awful," "garbage") will have a low sentiment score and words which appear more often in positive reviews ("greatest", "phenomenal", "timely") will have higher sentiment scores.

At time of prediction, the dictionary is used to average all the sentiment scores of the text, and that average is reported as the prediction.

In order to preserve the distribution of the 1-5 star rating, the dictionary is normalized at the end of a round of training

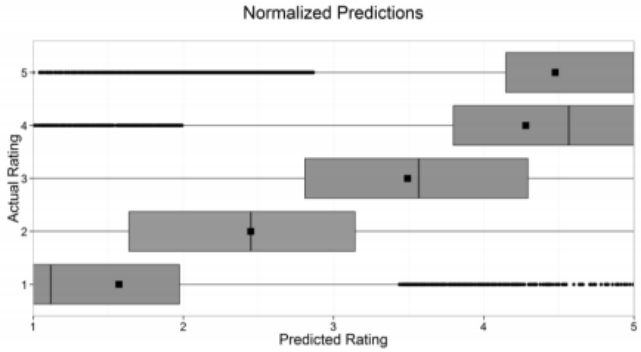

Figure 2. Predicted rating vs. actual rating of 100,000 reviews. Data generated 14 December 2015

so that the histogram for dictionary scores looks approximately equal to the histogram for all review ratings. If this is not done, the dictionary ratings will eventually converge to a rating of about 3.2 (the average star rating).

As a final note, this model did not consider function words like determiners, conjunctions, etc. These words were thrown out as though they were never included.

For a more thorough explanation of Naïve Bayes sentiment parsing, the reader is encouraged to reference Linus' performance report at:

https://students.cs.byu.edu/~thyer/Yelp/OrpheusVsLinus.pdf

3.3 Reconciling Continuous Predictions

Predicting star ratings is made difficult by the fact that Yelp only allows for discrete scores, but true sentiment is continuous. This means that the sentiment predictor must try to reconcile floating point decimal numbers with discrete integers. When the best prediction is a 3.552, should it round up? Weight a random choice between 3 and 4? Favor the answer closest to the average (in this case, four stars)?

The choice elected for this model was to leave the integer/decimal difference untouched and simply compare the integer label as though it was actually a floating point decimal number (1 is treated as 1.000 and so forth). This causes several problems, however, when trying to assess how good of a prediction was made. In general, eyeballing it seems to confirm the goodness of the prediction.

When a review was generally favorable (actual rating of either 3 or 4 stars), predicted scores tended to fall in that range as well. This reflects the way that people think and behave – when they are forced to choose between 3 and 4, they pick somewhat erratically with a weight toward their true sentiment. Someone with a true sentiment rating of 3.8 would probably pick a three 20% of the time and a four 80% of the time whereas 3.5 would be more evenly split.

3.4 Measuring Accuracy

The most basic measurement of individual accuracy for a regression statistic like this one is the distance between the true rating and the predicted rating, herein called *score*. A score ranges from 0-4, inclusive.

Given the discrepancy between discrete integers and continuous values, how do we then determine the overall accuracy of the parser? Messing around with integer prediction strategies don't seem to change the average score that much. If 100 review texts were evenly split between 3 and 4 stars on true rating and all had a predicted rating of 3.5 stars, it wouldn't make a difference to randomly choose an integer or to simply use 3.5 each time.

In fact, favoring the true average (3.6 stars average means rounding up 60% of the time and rounding down 40% of the time) would deliver optimal results. An 'optimistic' predicted score would be more correct, on average, while knowingly sacrificing individual accuracies. This would make for a neat parlor trick, but isn't particularly intelligent.

For this purpose, a score-squared metric is introduced in an effort to keep the parser honest. This favors scores that are close estimates (within 1 star rating of the true rating) over bad estimates (more than 1 star rating away).

Accuracy is then measured in a normalized fashion, according to a benchmark. The general equation for accuracy is as follows:

$$Accuracy = \frac{Score(Benchmark) - Score(Parser)}{Score(Benchmark)}$$

3.5 Benchmarks

The accuracy benchmarks are *Random Guess* (RG) and *Most Common Value* (MCV). RGBenchmark is calculated by getting the score of a random, continuous number between 1.000 and 5.000. MCV Benchmark is calculated by choosing the most common label in the training set and calculating the score for that label on every single instance.

RGBenchmark is not a really good benchmark, but it is useful to show relative accuracy gains from the baseline to the Bayesian model.

3.6 Results

Figure 3 shows the accuracy of Naïve Bayes over RGBenchmark according to the metrics given above.

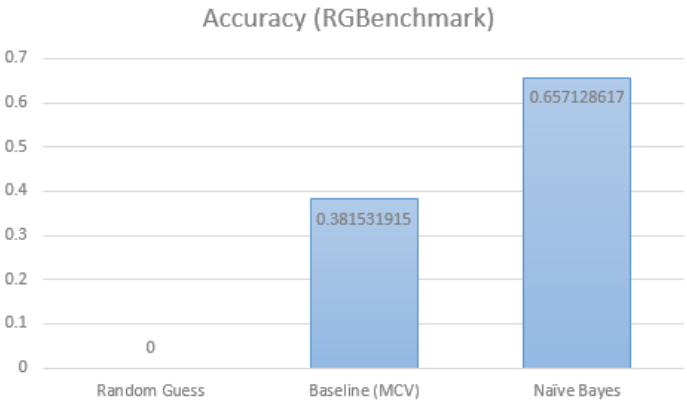

Figure 3. RGBenchmark Accuracy of Naïve Bayes and the MCV Baseline. This graph shows that Naïve Bayes has a normalized accuracy of 65%.

4 Models of Usefulness

The usefulness ratings contained in the Yelp dataset are not finite. This rating could be compared to "likes" on Facebook – as many people as find a review useful are able to increment the usefulness count. This means that the only measure of a review's level of usefulness is the number of users who click it. The usefulness values could range from zero to infinity. That rating is also left to the discretion of the reviewers, which means the metric is not always as accurate or consistent as we might like.

Because there are so many reviews with no useful votes, it would be easiest to assume that nothing is useful, but we

would also prefer to accidentally mark useless reviews as useful rather than missing useful reviews by accidentally marking them as useless – even if it means a loss in overall accuracy. We care the most about identifying useful reviews.

4.1 Natural Data Trends

As mentioned, reviews with no usefulness rating are extremely common in the dataset. Usefulness ratings with low scores were also somewhat common, however it was extremely uncommon for a review to be rated as highly useful – the sparsity of these high ratings also made it difficult to develop a system for labelling an arbitrary review as useful. A model which sees few truly "useful" reviews will assume that none exist.

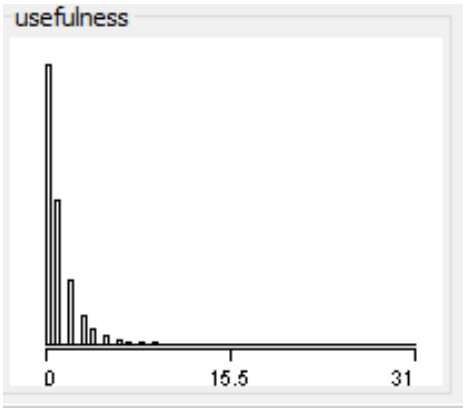

Figure 4. Shows the distribution of usefulness ratings. Note that usefulness ratings of 0 comprise nearly half the reviews, and reviews with ratings of 0 or 1 constitute almost 75%, following a distribution obeying Zipf's Law throughout.

4.2 Continuous vs. Categorical Models

One point of importance in data representation is how we ought to consider the labels for usefulness. Since there are very few reviews marked as 'highly useful,' plus more than half the instances have a usefulness of zero, it makes a lot of sense that the baseline model would simply guess that every review is not useful.

This isn't particularly helpful, however. As mentioned previously, we'd like to favor marking too many reviews as useful rather than too few – suggesting that we'd be able to use precision and recall as metrics for performance, preferring higher recall to higher precision. The question becomes the following: is it more valuable to know which reviews are useful? Or to know exactly what the usefulness rating of that review is?

The truth is a little of both, but the best initial model would just need to be able to recognize when a review is useful or not – and furthermore, it should favor marking too many reviews as useful.

4.3 Iterative Model Results

Given that the data is continuous, however, and not nominal, it's first helpful to show how well our best models of regression do. The following chart shows the best four results of learning models on continuous data. Please note that this trial run is the result of deliberately misrepresenting the useful reviews in the dataset – otherwise, the learning models would not be able to learn quickly enough about the behavior of useful reviews.

Model	Correlation Coefficient
Multi-layer Perceptron	0.3052
K-Nearest Neighbors	0.3445
Linear Regression	0.4061
Pace Regression	0.4038

Figure 5. Correlation coefficient results of the best learning models for regression. Baseline not included because it has no correlation coefficient.

Discussion of Continuous Results

Interestingly, nearly all of these models do fairly well on the useless reviews and on the very useful ones. But it's in the noisy area in-between that is the hardest. It turns out that there are a lot of reviews that look useful, yet nobody votes them as being useful. Similarly, a few reviews will be marked as useful even though they're not particularly informative.

In order to try and tease out that boundary, the next set of iterative models explored nominal separations of the usefulness ratings. Having chosen a pivot (for example, 15 votes), our data processor categorizes all reviews as either 'useful' (greater than or equal to the pivot) or 'not useful' (less than the pivot). This binary result is much easier to assess and compare than others.

Categorical Models

The first results comprise the model results for iterative discrete predictors with a pivot point of 5.

Model	Precision	Recall
Baseline	0.9200	0.9590
Decision Tree	0.9200	0.9590
Multi-layer Perceptron	0.9150	0.9470
K-Nearest Neighbors	0.9270	0.9520

Figure 6. Initial results for categorical models of prediction.

These results look great at first, but in fact they're a little too good. The numbers for precision and recall are the results for both categorizations, and since the 'not useful' category far outweighs the 'useful' category, it overwhelms the total numbers. To illustrate, Figure 7 includes the confusion matrix returned by the Decision Tree algorithm (ID3 flavorings).

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 0         0          0         0         0
                 1         1          0.959     1         0.979
Weighted Avg.    0.959     0.959      0.92      0.959     0.939

=== Confusion Matrix ===

     a     b   <-- classified as
     0   815 |     a = 1
     0 19186 |     b = 0
```

Figure 7. Detailed results from ID3 Decision Tree training/testing on 20,000 random reviews pulled from the dataset

Note that while the overall precision and recall is great, it's because ID3 has basically just turned into the baseline majority learner! This is undesirable – we want a model that will have high precision and recall, but the categorizations of 'not useful' don't matter very much.

Therefore the results reported below represent the results of models biased toward higher accuracy of 'useful' categorizations, and not necessarily overall accuracy. This was accomplished using a variety of techniques:

- Stratified sampling of the dataset to cause more instances of 'useful' to appear in the train/test sets
- Lowering the pivot to consider fewer votes as 'useful'
- Deliberately weight the model towards favoring 'useful' categorizations.
  - In MLP, this was accomplished by "punishing" the weight update more for a miscategorization of 'useful'
  - In ID3, each node would give a majority prediction, but discounted 'not useful' by a factor of 0.7, meaning that if the node had 40 instances of 'not useful' and 30 instances of 'useful,' then the node would predict 'useful'
  - For KNN, an even number of neighbors was used (4, 6, 8) and ties went to 'useful'
- When models gave a net output and compared that to the threshold, the labelled output instead became a probabilistic distribution with the possibility of predicting against the model's intuition.

This final table make no distinction between which methods of bias were used since often the best results were created as some combination of the above. Instead, numbered labels are used to represent some interesting configuration.

Please note that precision and recall are reported for only 'useful' labels while F-Measure is reported for both labels. This allows us to represent how well the model performs on 'useful' labels while also giving an idea of how well the model performs overall (high precision/recall but low F-Measure suggest stratification, for example).

Model	Precision	Recall	F-Measure
Baseline	0.0000	0.0000	0.9349
DecisionTree 1	0.6930	0.7410	0.7060
DecisionTree 2	0.0843	0.7554	0.7110
MLP1	0.4321	0.5600	0.8776
MLP2	0.7759	0.6850	0.6503
MLP3	0.8990	0.5903	0.5135
MLP4	0.5590	0.7110	0.7980
KNN 1	0.6830	0.7400	0.7200
KNN 2	0.7887	0.7700	0.7410
KNN 3	0.4120	0.7944	0.4964
KNN 4	0.2473	0.8729	0.7998
KNN 5	0.1308	0.6212	0.5087
KNN 6	0.7068	0.6174	0.6590

Figure 8. A variety of categorical models with Precision and Recall representing 'useful' statistics and F-Measure representing overall statistics

4.4 Ensemble Results

As a final point of precision, a very basic stacking ensemble was produced using the models reported above. The prediction of each model was used as a binary feature vector to a perceptron meta-learner.

To test the ensemble, the models were trained using approximately 1.1 million reviews[1] using about 50,000 reviews per model randomly sampled from the population – some using stratification, some not. The test set was 20,000 reviews randomly sampled from the holdout set (no stratification), and 10 trials were run. The test set used 10 as the pivot point for usefulness labels (note that the models used varying pivots).

	Useful	Not Useful
Useful	**58**	**8**
Not Useful	**682**	**19253**

Figure 9. Confusion Matrix for the ensemble

	Precision	Recall	F-Measure
Useful	0.0783	0.8787	0.1439
Not Useful	0.9995	0.9657	0.9823
Weighted Average	0.9965	0.9655	0.9796

Figure 10. Detailed accuracy results

As a point of interest, the reviews in the confusion matrix were examined post-testing to see why they might have been miscategorized. In our opinion, it is surprising that some of these reviews marked as 'not useful' didn't have more votes.

[1] Should the reader wish to reproduce this, it is recommended to have each model write its representation to disk following training rather than hold all models in memory during training. Total training time took an estimated 10 hours, and in case of a system/code failure, it is highly desirable to take up where training left off.

This may suggest that the ensemble models are able to identify patterns in text writing that users themselves may not recognize, or simply that the reviews don't have the exposure to garner the votes.

5 Conclusion

Biston is able to identify the reviews with the highest usefulness ratings with some ease, and it also may have learned patterns that occur in highly useful reviews – even if Yelp users don't mark it as such. Although it tends to mark too many reviews as useful, it's an exciting start and the possibilities for business application are many.

Building off of earlier work with CSPS, Biston can now add the context of usefulness. Those interested in negative feedback can spend less time searching for informative insights, especially when there is far too much text to be considered manually.

5.1 Future Work

Future work in the Calliope Sentiment Parser Series ought to include improvements in both sentiment and usefulness analysis. Since Biston focuses primarily on usefulness metrics, the following extensions are proposed as future work:

- Identification of more informational text features that do not compromise the speed of computation Biston offers
- Creation of more diverse prediction models to improve the ensemble's performance.
- Application to other types of labelled text, including less sparsely populated data for 'useful' labels
- Application to unlabeled text, such as customer reviews without usefulness votes, Twitter, comment sections, and more.