

Analysis of the Steam Gaming Reviews

Bo Sun; Bowei Chen

i. A description of how the data was collected?

Steam is a platform for distributing digital contents like video games, video streaming and social networking. From 2011 to present, there are total of 14218 video games in Steam's library, and over 7000 games were added within the year of 2017. As the booming of independent game makers still continue, in order for gamers to better understand the game before buying it, Steam has a comprehensive review system for gamers to leave reviews for games. Both of us are gamers and users of Steam, so here we want to do some NLP stuff on Steam reviews.

First of all, we used a Python spider to crawl the review page in HTML, and then we filter out the "Review" section for a certain game (DOTA2) by game id (570). Here is a screenshot of DOTA2 review from a random user.



Figure 1: Raw steam review from website

The raw review data contains:

1. Username and his account information
2. Recommended/Not recommended (reviewer' sentiment)
3. How much time had the reviewer spent on the game
4. Posted date
5. Review body(text)
6. Helpful/Funny or not (other user's opinion about the review)

After we crawled the review page, we have the raw data (.csv file) of reviews. Here is what we got:

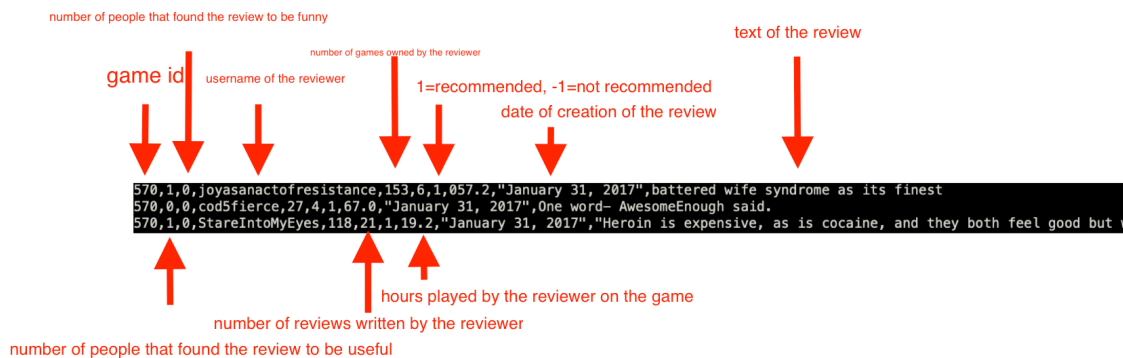


Figure 2: Raw data after crawling

The csv file contains:

1. Username
2. Id of the game
3. Number of people that thought the review to be helpful
4. Number of people that thought the review to be funny
5. Total number of games owned by the reviewer
6. Number of reviews written by the reviewer
7. Recommended or not: 1 for recommended, -1 for not recommended
8. Total hours played by the reviewer on the game
9. The date of creation of the review
10. Text of the review

For our project we do not need every information about the review. For example, total hours played by the reviewer on the game may be one useful feature to predict user's opinion about the game, however, this is a natural language processing project and we only focus on the text of review. Thus, we need to purify the csv file and extract data that we are interest in.

After we do the extraction, we have a new file that only contain labels and reviews. Here is what we got:

```
label review
-1 I HATE THIS GAME!
-1 As a colourblind user, the HUD has completely screwed me over. I'm havi
-1 Are you having a great day? Dota can fix that! Do you enjoy playing
-1 This is the best MOBA game ever made, simply a masterpiece, but also ha
-1 At this point in time, I've wasted round 3-5 years of my life playing t
-1 WHY HAVE I DONE THIS
-1 If you want to spend 2,000 hours damaging your mental health with stres
-1 I have severe depression and I want to kill myself.
-1 Worst coop game ever. You play support? Your carries feed. You play car
-1 If I spent those 700 hours learning knitting, I guess I'd be pretty goo
-1 Decent game, world's saltiest players.
```

Figure 3: Purified data with labels and reviews

In this step, we reduced all meaningless information such as total hours played by the reviewer on the game, and extract data that we need to train. The label column shows reviewer's opinion, 1 for recommended and -1 for not recommended. The review column shows the text of the review. So far, we have collected and purified the data.

ii. A description of how the data was labeled?

As we described on the first question, we have gained the game review information and saved as a csv file. Here are correspondences between raw steam reviews and csv data file.

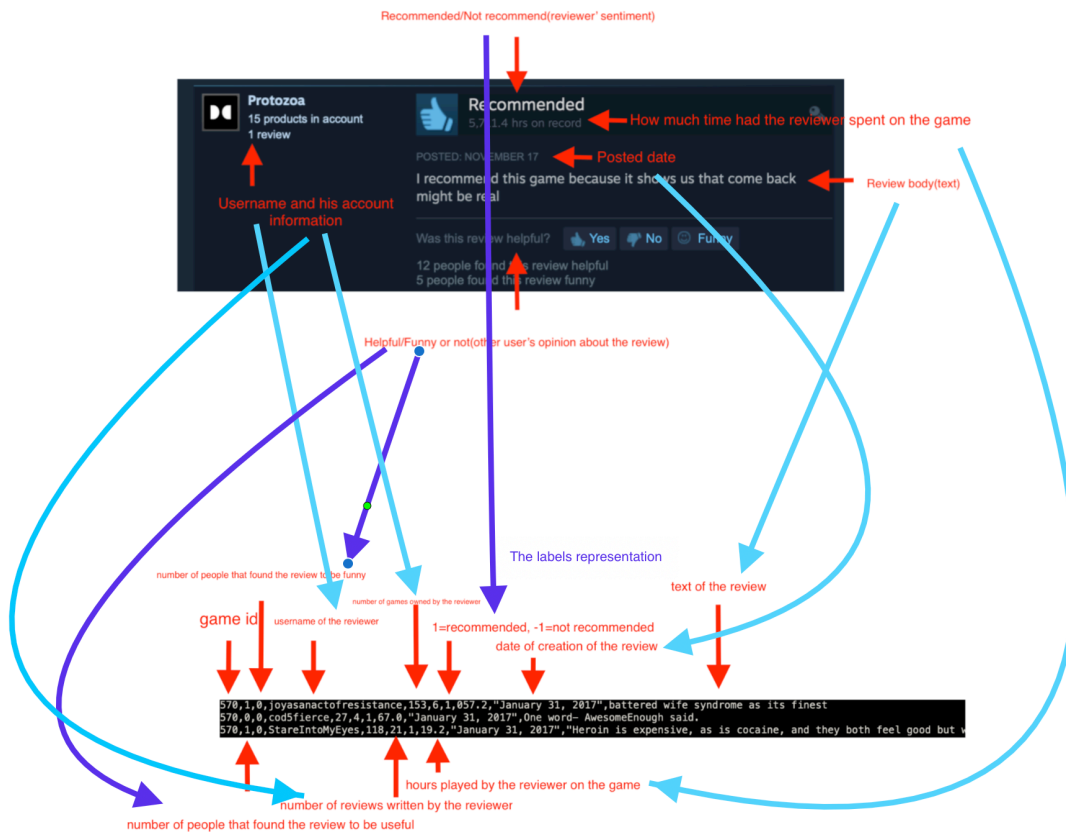


Figure 3: correspondences between raw steam reviews and csv data file

Each line shows a corresponding relation between raw steam reviews and csv data file. Especially, purple lines indicate the transformation from raw data to our labels. There are 3 transformations in the graph: From

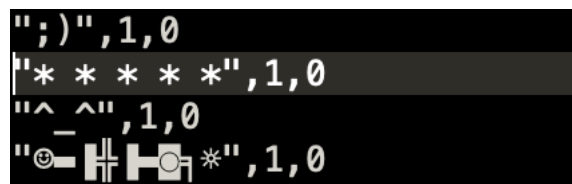
1. Recommended/Not recommended (reviewer's sentiment)

2. Helpful or not (other user's opinion about the review)
3. Funny or not (other user's opinion about the review)

Transfer to:

1. A numerical label that indicate the reviewer recommended or not: 1 for recommended, -1 for not recommended
2. A numerical label that indicate number of people that thought the review to be helpful
3. A numerical label that indicate number of people that thought the review to be funny

At the beginning of this project, we tried to use review text to predict these 3 labels, in other words, we solve it as a multi-label classification problem. It turns out that our steam review data does not apply to the multi-label classification and we finally decided to predict whether the reviewer recommended this game on the data which are labels as helpful. As we know, the helpful button is used for other users to judge the review helpful or not. When we looked into the data, we knew why some reviews are not labeled as helpful. Here are some examples:



```
" ; ) ", 1, 0  
| " * * * * ", 1, 0  
" ^ _ ^ ", 1, 0  
" ☺ - || H - ☹ * ", 1, 0
```

Figure 4: some sample of unhelpful reviews

Game player usually feel excited when they play video game. Unlike reviews from movies or books, game reviews often contain junk information and these data block our way to process real valuable data. Thankfully, steam community provide the helpful button to filter the garbage out. A hit on the helpful button imply this

review are not nonsense words, such as spamming same words over and over again. Thus, we only choose the data which are labeled as helpful, and predict whether user recommender or not based on review context. In other words, we use steam reviews which people thought helpful as data and the labels are recommended or not recommended. We use number 1 for recommended and number -1 for not recommended.

(c) Description of classifier.

Metadata:

- Total number of raw review: 12500
- 1877 reviews contain special characters.
- 10623(84.98%) valid (reviews without special characters)

(i) Baseline: Counting Classifier.

- Data normalization: making all tokens lower case

According to the counting learner classifier from slides, the method is basically counting all words on the training set. For each time a word appears in positive reviews, we add 1 point. And we deduct 1 point for it appears in negative reviews for each time. Then, the score counter can be transform to 2 sets — good and bad sets based on whether the score of a word is positive.

After training, we can calculate the total scores of one reviews on the test set. Especially, we do nothing about OOV. In other words, we add 1 point when we see a word is in good set, deduct 1 point when we see a word is in bad set, and ignore a word when it is OOV.

Finally, we can predict a review by looking its score. Positive reviews go to recommended side and negative reviews go to not recommended side.

(ii)

- Classifier: Naive Bayes
- Assumption: Bag of Words(BoW) assumption
- Data normalization: making all tokens lower case
- NOTE This is the pure naive bayes we tried, the only pre-process on this classifier is that the non-english character like other language and special character/symbols/emojis were removed, and the data is labeled as answered in question (a) and (b).
- Smoothing: Additive Lidstone-smoothing, add- α . α is chosen by 80/20 cross-validation for 5 times(implemented by scikit-learn) on various α values.

Since for gaming review tasks, the gamer review could be strangely written, --tens of word form playing, jargons, abbreviations etc. So we assume the vocabulary size to be infinity, which means the smoothing constant for OOV should be small. And the best α is chosen to be 0.001 as shown before.

Following figure shows the accuracy changes over difference α values in different cross-validation runs.

FIG. 5 CROSS-VALIDATION FOR OPTIMIZING ALPHA USED IN LISTONE-SMOOTHING

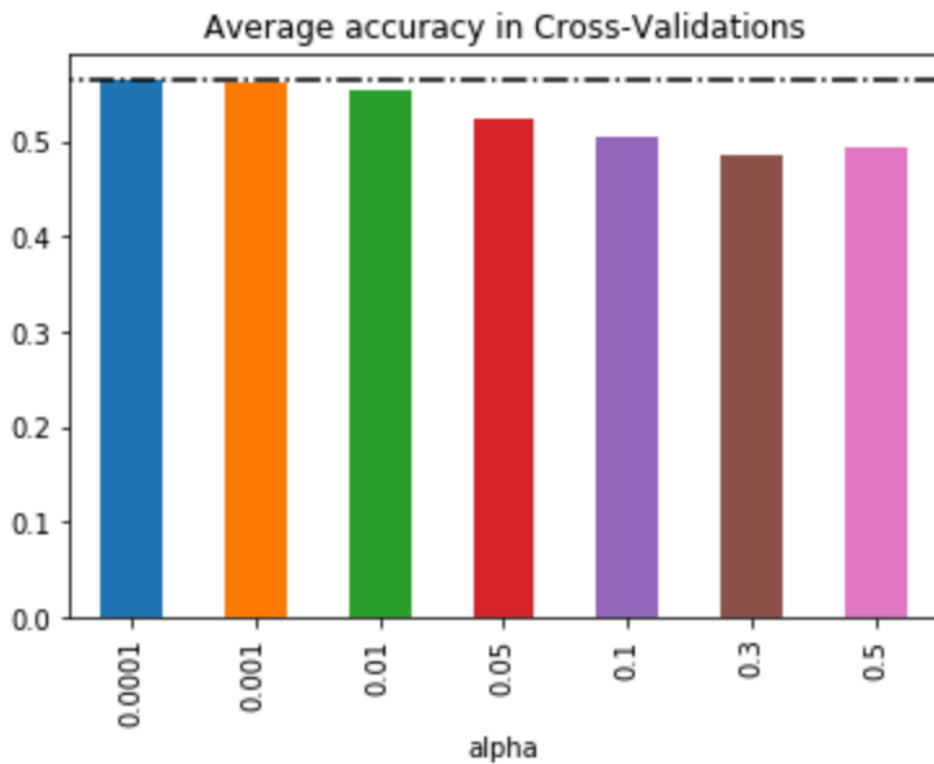
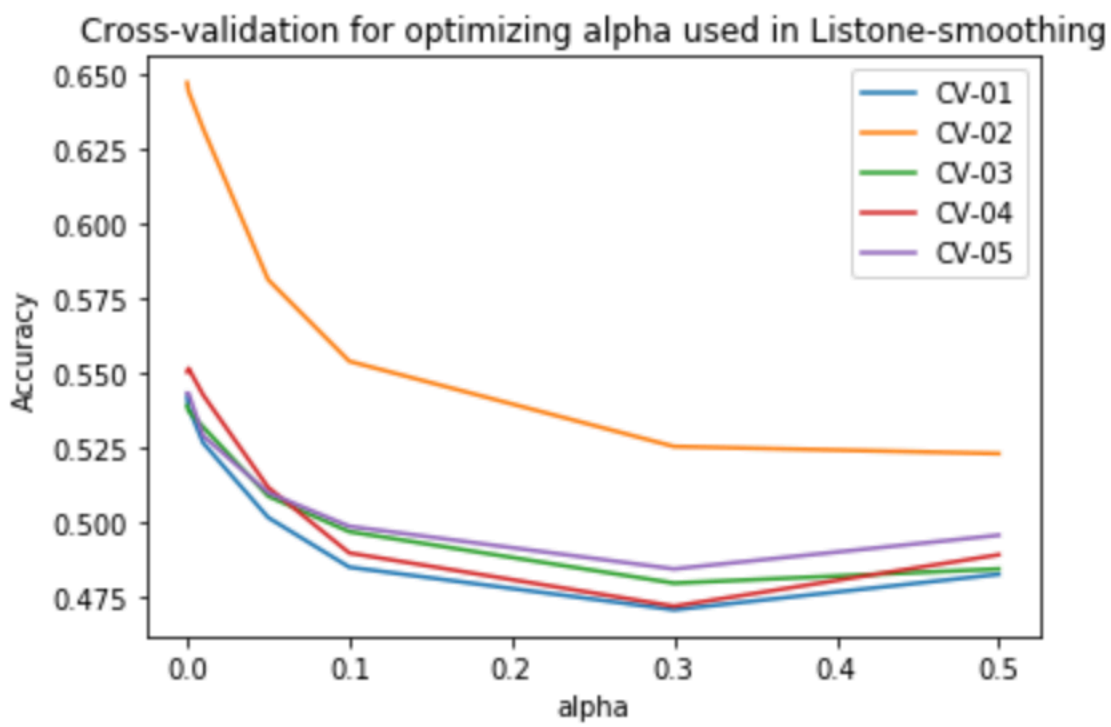


FIG. 6 AVERAGE ACCURACY IN CROSS-VALIDATIONS

The increase in average classification accuracy in CVs stop increasing as alpha goes below 0.001, so we chose α to be 0.001.

(iii) Classifier: Based on our exploratory of the data, we tried to add more normalizations.

- Classifier: Enhanced Naive Bayes
- Assumption: Bag of Words(BoW) assumption
- Data normalization:
 - A. remove english stop words
 - B. making all tokens lower-cased
 - C. remove tokens that are composed of punctuations only
 - D. remove preceding and trailing punctuations
 - E. stemming words with NLTK PorterStemmer(too slow and no improvement, so removed at the end)

Some basic stats/metrics after these steps were shown as chart in question (d).

- Smoothing: Additive Lidstone-smoothing, add- α . α is chosen the similarly as above.

(d) Analysis of results:

Our result is not as good. And with the intuitively "should-not-harm" normalizations we did, our enhanced naive bayes behave even worse, and the baseline counting approach outperformed our model.

After examining more deeply, overall we think that class imbalance, review length imbalance, and the wordings in gaming review increase the hardness of the problem.

Then we experimented by manually balance the pos/neg reviews. The baseline counting method drops to a reasonable ~50% level, and Naive Bayes model increase to >65% level.

We summarized the imbalanced class phenomenon and tried to explain why it hampers the performance of naive bayes model at the end.

And our conclusion is that naive bayes model isn't proper in this gaming review context, because of the imbalance. If we manually remove the imbalance then the improvement is significant. But with the presence of imbalance, the techniques we tried failed to improve naive bayes model.

Table 1. Imbalanced number of pos/neg reviews

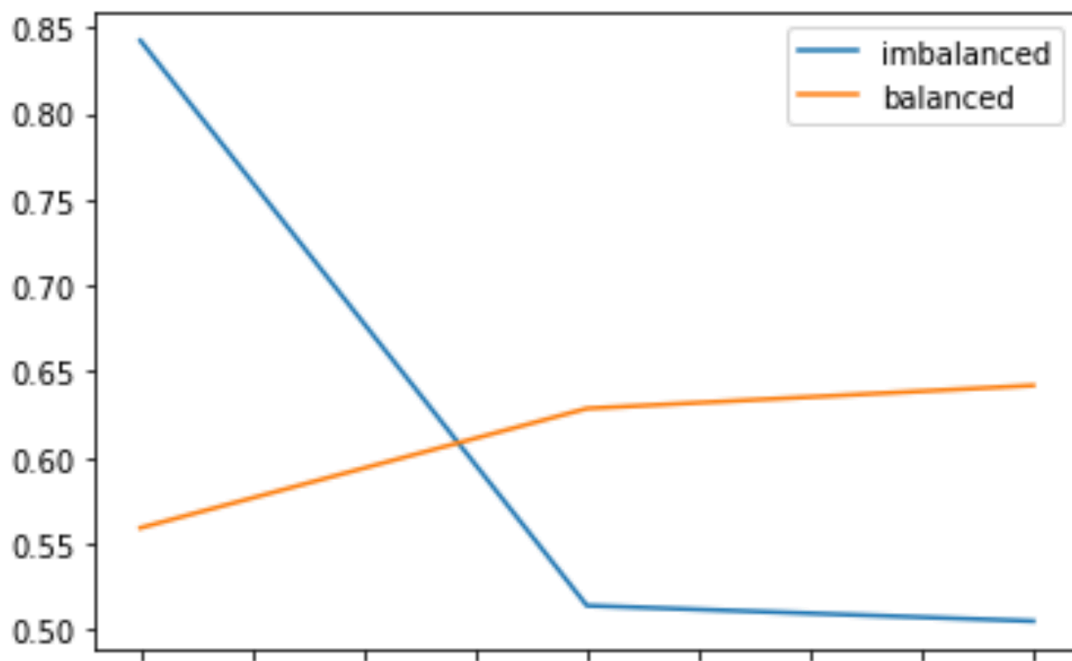
Classifier	CountLearn	Naive Bayes	Enhanced Naive Bayes
Vocabulary Size	11862	11862	9504
#Positive Words	9769	10667	8121
#Negative Words	2093	7238	3703
lidstone-smoothing-constant	None	0.001	0.005
Avg. accuracy of cross-validation in dev set.	84.06%	56.31%	51.36%
Accuracy on test data	84.25%	51.40%	50.50%

So our conclusion is that, naive bayes model can not handle this type of gaming review, due to the presence of class imbalance and the large over lapping what defines a positive and a negative review.

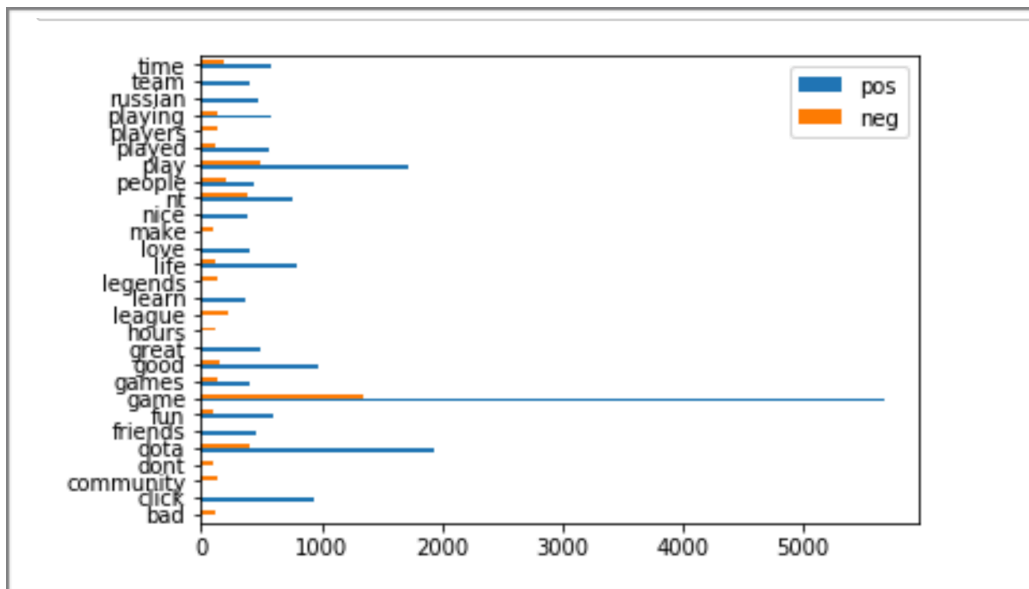
Table 2. Balanced number of pos/neg reviews

Classifier	CountLearn	Naive Bayes	Enhanced Naive Bayes
Vocabulary Size	7198	7198	6586
#Positive Words	3247	5203	3984
#Negative Words	3951	5349	4448
lidstone-smoothing-constant	None	0.05	0.1
Avg. accuracy of cross-validation in dev set.	54.33%	66.37%	62.11%
Accuracy on test data	55.92%	62.86%	64.2%

Final accuracy comparison between our models and baseline approach.



Naive would outperform the Counting Learner if the data were balanced.



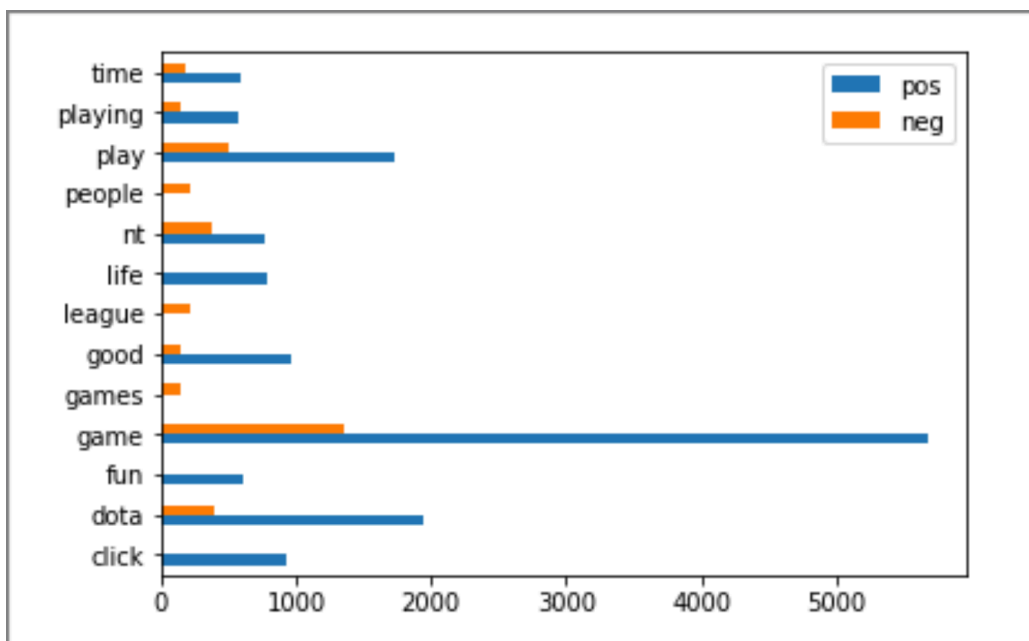
Top 20

most common words

We characterized the imbalance problem as following, results came from the enhanced naive bayes, where we removed stop words, and did some word normalization as described above.

First we examined the top words in both negative and positive classes, but the top ones are shared, and the distributions looks the same. So from this we can see that the naive bayse will not work. The words count is larger in positive review because there are lot more positive cases.

And from top10 words, we spotted some irregularities, such as the playing and play, games and game both appear in top words. But they basically mean the same thing. But after adding word stemming with NLTK's PorterStemmer, the training was significantly slowed and merely no improvement was gained. So we removed that stemming feature.



Top 10 most common words