Predicting Upvoted and Downvoted Comments from Reddit

Abstract

The purpose of this document is to provide both the basic paper template and submission guidelines. Abstracts should be a single paragraph, between 4–6 sentences long, ideally. Gross violations will trigger corrections at the camera-ready phase.

1. Intro

Reddit describes themselves as a platform that "bridges communities and individuals with ideas, the latest digital trends, and breaking news (...okay, and maybe cats). Our mission is to help people discover places where they can be their true selves, and empower our community to flourish." [1] The website uses a system of "upvotes" and "downvotes" for the community to decide what sorts of posts and comments should be more or less visible. These scores are directly tied to the user account in an aggregate point system called "karma". These points have no monetary value, but are coveted by many users as a sort of badge of honor saying that their posts were worthy of view. Many users want to game this system and acquire the highest amount of karma possible. We set out to examine the comments from the very popular subreddit, askReddit, and see if we could predict what sorts of comments would be the most highly upvoted and downvoted.

The askReddit subreddit is an entirely text-based question and answer subreddit that contains more comments than any other subreddit on the entire website. Here redditors are particularly concerned with their comments being upvoted, as it is the only kind of "karma" they can gain on the sub.

In the paper we go over how we cleaned the data and feature engineered it for two different algorithms, Logistic Regression (using Vowpal Wabbit) and a SVM using (scikit learn). Both of these methods were used to attempt to predict a binary classification of the comment, would it be upvoted (positive classification) or downvoted (negative classification).

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

After cleaning and extracting the data we created n-grams to use as features. We used the 1000 most common n-grams and tested them with vowpal wabbit and the SVM. In the following sections we describe feature engineering, data cleaning, the SVM classifier, the Logistic classifier, results and analysis.

2. Data Collection & Feature Engineering

Reddit released a corpus of comments from 2015. Of these we used a subsection which included May 2015. Of this subsection we focused on the comments from the AskReddit subreddit. These comments required significant data-cleaning and feature engineering to get them into a useable form. We removed all non-alphanumeric characters and lowercased the entire corpus. Reddit users tend to link URLs in their comments quite frequently. These were removed and replaced with a "URLHERE" marker. It was decided that a particular url would not occur often enough in the corpus to make a significant difference and we were more concerned that a URL contributed to the score than the actual URL linked.

Once the data was in the desired format, we split the corpus into two classes based upon the number of votes that each comment had received. We used a threshold value of greater than 100 to determine if a comment should be classified as an upvote and a threshold value of less than 0 to classify the comment as a downvote. We then constructed bigrams from the comment bodies by considering words that occurred next to one another and storing the amount of times they occurred. For example the sentence : "The dog ran", would produce: "The, dog", "dog, ran" as two bigrams. We used these bigrams in both the Logistic Regression and SVM approach. Initially we considered only the top 100 bigrams, but later considered the top 1000.

The occurrence of these bigrams was noted in the construction of the SVM. For example in the sentence: "The dog ran" both aforementioned bigrams would receive one instance of occurrence in the feature vector. Where in the sentence "The dog ran, the dog ran" each of the aforementioned bigrams would receive two instance of occurrence in the feature vector. It should be noted that the bigrams for upvoted and downvoted comments were very similar. In particular, the resulting 100 most common bigrams from the upvote and downvote corpuses were 77% identical to

each other. This lead to some issues with classification accuracy discussed later in the SVM and Regression sections.

To address the issues that arose from this similarity, we tried eliminating both stopwords and proper nouns. These stop words include common words that are likely to occur frequently in English; for, I, the, they, would be examples. In order to accomplish this, we use the parts of speech tagger form the python nltk module as well as their stopwords corpus. Unfortunately, this again produced very comparable results in which both sets of bigrams were extremely similar and somewhat ineffective for the purposes of classification. Another inefficiency to this approach was the fact that it substantially increased the amount computing power and time necessary to obtain the bigrams. After executing the code, it could take upwards of 20 to 25 minutes for the program to finish computing the bigrams.

Next we decided to increase the size of the n-grams to 3 in the hopes that this would produce more unique results for each class. This was indeed the case and the resulting trigrams from both classes were only 52% similar. Counter intuitively though, this again produced very similar results to the bigrams.

Table 1. Top Ten Bigrams

Rank	Upvoted	Downvoted
1	in,the	i,dont
2	i,dont	if,you
3	this,is	in,the
4	if,you	to,be
5	of,the	i,think
6	i,think	of,the
7	it,was	is,a
8	is,the	do,you
9	i,have	this,is
10	do,you	i,was

Table 2. Top Ten Trigrams

Rank	Upvoted	Downvoted
1	a,lot,of	a,lot,of
2	i,dont,know	i,dont,know
3	what,the,fuck	i,dont,think
4	i,like,my	a,klondike,bar
5	i,feel,like	melt,steel,beams
6	this,is,the	to,be,a
7	one,of,the	fuel,cant,melt
8	i,dont,think	i,have,a
9	you,have,a	jet,fuel,cant
10	sounds,like,a	paul,blart,mall

3. Classifiers

3.1. Support Vector Machine

One system we used to attempt to classify a Reddit comment was the Support Vector Machine. Two SVMs were created in Python using the scikit-learn SVM package. One classifier is a binary classifier which decides if a comment should get upvoted or downvoted. The other classifier is a multiclass classifier which decides what range of upvotes or downvotes the comment is expected to receive. The multiclass classifier used the one-vs-one classifier strategy.

The SVM took two arrays as inputs. The first array was a 2D array which contained the feature array for each sample in the training set. Each feature array consisted of the amount of occurrences of the top bigrams and trigrams in each comment. The second input array contained the class for each comment. After these two arrays were used to create a classifier, similar arrays were created for our test set. We passed the first 2D array containing the feature array for each sample comment into the fit() function for our SVM, and compared the SVM?s result with the true score of our test sample. By keeping track of the amount of correct predictions, we were able to figure out the accuracy of our classifier for each class, as well as a global accuracy.

Table 3. Binary Classifier

	Number Correct	Accuracy
Downvoted	306	20.4%
Upvoted	1227	81.8%
Total	1533	51.1%

Table 4. Multi-class Classifier

	Number Correct	Accuracy
<0	0	0%
0	1080	100%
1-2500	0	0%
2500+	0	0%
Total	3000	36%

The binary classifier performed better than the multiclass classifier, simply because it had some diversity in its classifications. The multiclass classifier always predicted a post to get zero upvotes, simply because comments with zero upvotes dominated the dataset. The binary classifier had an equal amount of upvote comments and downvote comments in its training set, so it didn?t have one class dominate the classifier. However, both classes performed very poorly. The multiclass classifier was only accurate for the amount of posts with zero votes in the testing set and the binary classifier was equal to a coin flip.

Multiple strategies were implemented to try and raise

the accuracy of the SVM classifier. We tried Laplacian smoothing on our feature arrays by adding one to each entry in the array. The training set and testing set were both made larger in an attempt to fill the SVM with more data. Unigrams, bigrams, trigrams and a combination were run through the SVM. All of these actions resulted in similar coin flip like results, with some actually causing the accuracy to decline. Even the multiclass classifier was a result of an attempt to improve upon the binary classifier?s accuracy. However, the results show a downgrade in performance with no diversity in classification.

3.2. Logistic Regression

Another prediction method we tested was using Vowpal Wabbit, a console based learning system available on GitHub. Comment data was reformatted to the specifications of VW with a simple python script, and divided into two classes, based on whether they had positive or negative upvotes. After being split into testing and training sets, with approximately 90% of the data in the training set, the model was tested using multiple loss functions. We performed two tests, one using all of the available data, in which the quantity of negative examples is far greater than that of positive examples, and one using a smaller subset of the data, in which the examples are evenly split between positive and negative. Below is a table of accuracies:

Table 5. Regression

	All Data	Evenly Split Data
Hinge Loss	91.25%	56.60%
Logistic Loss	91.26%	56.60%
Squarred Loss	91.26%	56.60%

As you can see, our accuracy with Vowpal Wabbit is rather high when using all of the data, but barely above half when using smaller, evenly split data. We can also observe that the choice in loss function had minimal effect on accuracy.

4. Results

5. Analysis

In both of our models, we see accuracy that?s barely better than flipping a coin when using evenly split data. However, our Vowpal Wabbit model was extremely accurate when trained and tested on all of the available data. While this discrepancy in accuracy may be the result of an insufficient quantity of training data from shrinking the set of negative examples, one theory we proposed is that the high number of negative examples in the corpus causes the model to have a very strong negative bias. Thus, it predicts a negative label most of the time, which happens to be very accurate on heavily negatively biased data. Further testing would be necessary to confirm or reject this hypothesis.

Another possibility for for the inaccuracy of our models could be due to the fact that it might not be entirely possible to identify a positive or negative comment solely based upon what language was used. Instead, comments most likely acquire upvotes based upon the relevance of the comment in the context of the subject being discussed. This however, would be a considerably harder value to quantify. But if we had more time on the project, one approach we would consider is attempting to identify the subject matter of all the comments by using a parse tree. We would then group together all comments that are discussing similar subjects based upon the nouns that they contained. Once comments had been clustered together, we would

Acknowledgements

Do not include acknowledgements in the initial version of the paper submitted for blind review.

If a paper is accepted, the final camera-ready version can (and probably should) include acknowledgements. In this case, please place such acknowledgements in an unnumbered section at the end of the paper. Typically, this will include thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.