Predicting Reddit Post Popularity Via Initial Commentary

by Andrei Terentiev and Alanna Tempest

1. Introduction

Reddit is a social media website where users submit content to a public forum, and other users of the website can either upvote or downvote the submitted content. The number of downvotes subtracted from the number of downvotes determines the score of the post. Posts with higher scores are then more prominently featured on the website. One interesting note is that the popularity of the post is often times determined by the discussion that occurs in the comment section of the post. The goal of this project was to determine if there were any features in the first ten comments of a given post that would help us predict the future score of the post. We chose this task after observations from a previous 229 project “Predicting Reddit Post Popularity.” (1)

2. Data Set

We collected a dataset of roughly 2000 training examples and 220 testing examples by scraping Reddit's website using their API. For each example (Reddit post) we preserved the post's score and text bodies of the earliest ten comments. Reddit fuzzes some of its data, so the oldest ten comments were not necessarily in order, nor were other metrics about post popularity necessarily accurate. Score was guaranteed to be accurate, so we chose to use this as our label value instead of something like vote counts or upvote ratios (both of which are fuzzed).

3. Task Definition

Initially we sought to model the problem as a regression task, where given our input of features we would attempt to predict a value for the future score. We attempted some linear regression models, but felt that this task would prove to be infeasible because of the strange distribution of the scores. 54 percent of our data never achieved greater than a score of ten. Additionally there were very few posts with large scores that skewed the regression heavily. So instead we decided to model the problem as a binary classification task. Where a post was given a label of 0 if it did not surpass a certain threshold, where the threshold is just some arbitrary score.

4. Features

The input features are solely metrics on post comments. Since not all posts had at least 10 comments, our first feature was the fraction of first ten comments that existed. Additionally we considered sentiment analysis on the comments as well as comment length. Since the comment ordering was fuzzed, we could not use metrics on individual comments as features. Instead, we used metrics on the ten comments as a whole. Thus, our features were maximum, minimum, lower-half average, upper-half average, and average of the comment sentiment analysis scores and the comment lengths. Additionally, we had another feature, “polarity,” which was the maximum sentiment analysis score minus the minimum sentiment

analysis score. Sentiment analysis was done with the Stanford NLTK, which for each sentence in a comment body returns a rating of Very Negative, Negative, Neutral, Positive, or Very Positive. We assigned numerical scores -2, -1, 0, 1, 2, respectively to sentences based on those ratings. Each comment received a sentiment score that was the average of the sentiment scores of each of its sentences.

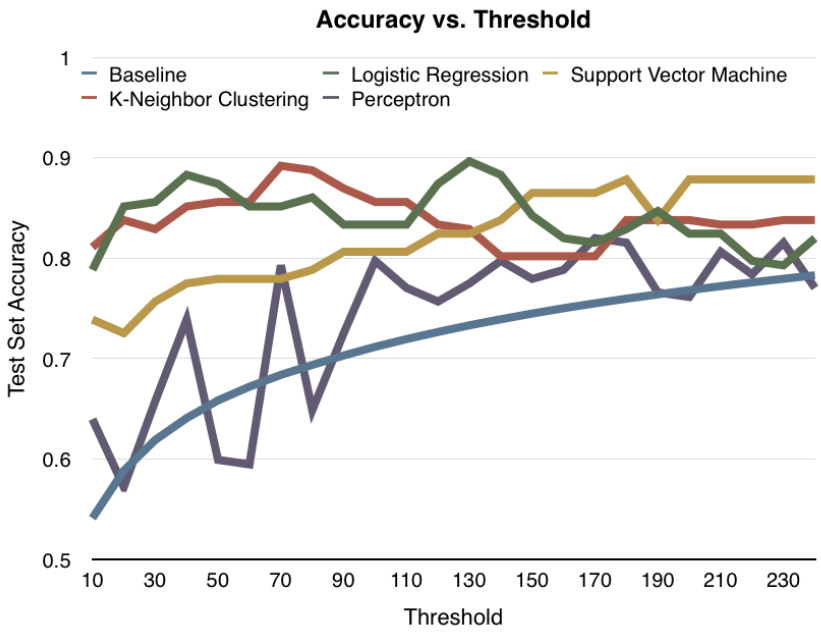
5. Evaluation Metric

The metric that we used to evaluate our data was classification error. For a post $(x^{(i)}, y^{(i)}, \hat{y}^{(i)})$ where $y^{(i)}$ and $\hat{y}^{(i)} \in \{0,1\}$ As described earlier the label of 1 is given if the post surpassed the threshold. The Classification error was $\frac{\sum_i 1\{y^{(i)} \neq \hat{y}^{(i)}\}}{m}$

All of our algorithms were compared against a baseline algorithm which simply predicted zero for each post. As we mentioned earlier the majority of posts never even surpassed our lowest threshold, a score of ten. For the rest of the post we will also use the term accuracy which is simply 1-classification error**.** In the results section we also describe how we compare algorithms across all thresholds.

6. Results

To the right we can see how the testing error changes for various algorithms against the baseline as we increase the value of the threshold. We will describe each algorithm in more detail below. However we can see that there is a trend of increasing test accuracy as we increase the thresholds. More sophisticated models like Support Vector Machine Learning and Logistic Regression are more stable with the changes in the thresholds whereas simpler algorithms like Perceptron and K-neighboring algorithms, despite having some high peaks behave more erratically. In the analyses of the separate algorithms we chose to look at the confusion matrices across all thresholds and then normalize. This will give us an average which we will use as our overall performance grade. Obviously something like variance is not taken into account, but we believe that this aggregate accuracy across all thresholds holds the most insight into which algorithms were most successful.



	Actually Positive	Actually Negative
Predicted positive	36.95	21.79
Predicted Negative	14.04	149.21

K-Neighbors

6a. K-Neighbors

$$Decide\ 0\ if\ \sum_{i=1}^{k} Y_n^{(i)} <= k/2 \quad Decide\ 1\ if\ \sum_{i=1}^{k} Y_n^{(i)} >= k/2$$

Where Yn is one of the k closest neighbors in the training set. We chose five as our k-value which is the default in the scitkit library. K-neighbors is a very simple algorithm, but is also computationally expensive. However, we are able to obtain very high accuracy at low thresholds. The success of this algorithm faded when we used higher thresholds. This would suggest that there are many "moderately successful posts" which are clustered together feature wise, but more variation in the features of "highly successful posts."

6b. Perceptron

Perceptron and linear classifiers in general are great if the data is linear separable. It was immediately clear that our data did not fit into this mold. Trying to create a line between this data is reflected in the inconsistency of the accuracy of the algorithm, even dipping below the baseline at certain points. Using algorithm with the given data set, was infeasible due to these circumstances. Our update rule was the standard Perceptron update below. Where are hypothesis was the dot product of theta and x. We used the default alpha for the scikit library.

$$\Theta_j := \Theta_j + \alpha(y^{(i)} - h_\Theta(x^{(i)}))x_j^{(i)}$$

	Actually Positive	Actually Negative
Predicted positive	30.5	29.38
Predicted Negative	28.25	133.88

Perceptron

	Actually Positive	Actually Negative
Predicted positive	42.13	15.58
Predicted Negative	17.38	143.93

SVM

6c. SVM

We ran an SVM with a linear kernel and L1 regularization. When an SVM with a Gaussian kernel had extremely low test set error and higher training set error. We switched to a linear kernel with L1 regularization in order to be less sensitive to the suspected outliers. The resulting increase in test set error and decrease in training set error confirmed that the data has many outliers. Below is the optimization problem corresponding to this model, where C is a chosen constant used to influence the weights of the error terms (inside the summation). w is the weights vector.

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

6d. Logistic Regression

Logistic regression is a discriminative learning algorithm. It fits the sigmoid function (below) to the data. The implementation of logistic regression relied on the LIBLINEAR library (2). The performance of this algorithm was well above baseline for all thresholds we tested. Logistic regression uses maximum likelihood estimates to find the parameter vector theta that best fits the training data to the hypothesis below. This is also known as the sigmoid function.

Actually Positive	Actually Negative	
41.63	17.45	Predicted positive
17.875	145.04	Predicted Negative

Logistic Regression

$$h_0(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

Stochastic gradient descent is used to find the maximize the likelihood, with update rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

6e. Aggregate Training and Test Accuracy

Algorithm	**Training Set Accuracy**	**Testing Set Accuracy**
Baseline (Guess all 0's, where 0 means "below threshold")	0.731981981982	0.731981981982
Logistic Regression	0.886261261261	0.840840840841
Support Vector Machine	0.993806306306	0.822072072072
K-Neighbors Clustering (k=5)	0.892713365539	0.838588588589
Perceptron	0.767400250492	0.740427927928

6. Discussion

Based on our results, we can conclude that there is some amount of correlation between the initial commentary of the post and the score of the post itself. Though we are able to achieve better-than-random accuracy with our baseline (simply predicting "below threshold" for every post), for most thresholds, all algorithms achieve even better accuracy. It is interesting to see both logistic regression and SVM converge as we increase our threshold amounts while our other algorithms seem to bounce up and down and have more variability. The results are satisfactory, but if we could figure out a way to have high success rates at both low and high thresholds that would be much better, but are currently unsure of how to solve this problem. As expected the commentary did have significant impact on the success of the post, however

there remains significant error, since the success of the post is also dependent on features of the post itself and some amount of randomness.

Conclusion

Reddit post popularity definitely has an element of randomness but using characteristics of its early comments, a post's popularity with respect to a fixed threshold can be predicted with up to 89% accuracy. Of course popularity of a post is dependent also on the content of the post, which we ignored here; perhaps metrics on post characteristics could improve our predictions. Additionally, more data and more time to fine-tune our algorithms would likely have helped smooth out the error analysis curves and so inspire more confidence in our results. Nevertheless, it is remarkable to have such strong success in predicting post popularity simply from characteristics of Reddit posts' earliest comments.

Future Work

In the future we would like to perform a more extensive analysis on our features. It would be a good idea to perform an ablative analysis in order to determine which features are most useful in our prediction task. Additionally it may be a good idea to see if we can think of more features to use in our analysis. For example exploring the presence of key words, could help us use something like a Naïve Bayes classifier.

References

1. http://cs229/proj2012/ZamoshchinSegall-PredictingRedditPostPopularity.pdf
2. http://www.csie.ntu.edu.tw/~cjlin/liblinear/