Predicting the Success of Reddit Text Posts

Jeeger, Markus
500620326
Dept. Computer Science
Ryerson University
Toronto, Canada
markus.jeeger@ryerson.ca

Oro, Bojan
500626903
Dept. Computer Science
Ryerson University
Toronto, Canada
boro@ryerson.ca

ABSTRACT: This report outlines the successful and failing methods of predicting the success or failure of Reddit.com text posts on the subreddit "Today I Fed Up" (r/tifu) using various classification and regression machine learning approaches.***

Keywords—machine learning, regression, classification, Reddit, prediction, text analysis

I. INTRODUCTION

This document is the final report for a research assignment on predicting the success of text posts on Reddit.com using different machine learning methods and algorithms. Posts on Reddit can take a variety of forms, from images and videos, to text content, to links to other websites, or some combination of the three, and more. Other users can either upvote on downvote content, contributing to a post's "karma." Higher karma dictates the success of a post. There are different subreddits across a wide range of topics and communities. [1] For simplicity we chose to look exclusively at a subreddit that only allows text-based content, the subject of each post being a unique narrative about an experience in the author's life, the subreddit "Today I F***ed Up" (r/TIFU). Looking at the past three years of posts, we derived features based on the post and its content and attempted two machine learning based predictions with various levels of success: using regression to attempt to predict the karma a post would achieve, and classification to assign posts into one of 3 themes: unsuccessful, moderately successful, or vary successful. We used the Anaconda Distribution of Python, using the SKLearn implementations of machine learning algorithms for our models, and a script running against Reddit's API to gather data.

II. EXISTING WORK ABOUT REDDIT

Tracey M. Rohlin, San Jose State University, attempted to predict the popularity of various posts on Reddit.[2] Rohlin depends primarily on the text analysis of individual posts to discern a topic from the text, and then uses this topic to predict if a given post will be successful or unsuccessful in a certain subreddit. Success is defined as a score higher than the 75th percentile score among the first 1000 posts in a given subreddit. The text analysis is done using BOW, LDA, and TF-IDF to find topic words in posts, and then attempts to use Naïve Bayes and SVM to classify the words into categories. Over all models and subreddit, the approximate accuracy of the models was 60%-65%.

Another paper, published by Jordan Segall and Alex Zamoshchin, Stanford University, took a similar approach to Rohlin's work, in this case trying to classify a given post into "score buckets" using Naïve Bayes, and SVM. [3] Text analysis was attempted using TF-IDF as well as stemming: the act of breaking words down into the simpler root tokens to group words despite differences in tense. Linear Regression to predict exact scores was attempted here as well but failed to produce a useful value. Over all models, the approximate accuracy of the models to place a post within the score buckets was 40%-45%.

III. DATA GATHERING

While several Reddit related datasets exist, we chose to create our own dataset for a variety of reasons. Since we selected only the r/TIFU subreddit we wanted a significantly large dataset that would reflect only posts in this subreddit. Existing Reddit datasets contain a month's worth of data for the entire website, resulting in files in the range of several gigabytes of which only a few megabytes would be relevant to the problem. Additionally, these datasets would rarely include user profiles, a source of data that we would pull additional features from. Instead of downloading hundreds of gigabytes of data and then scraping the associated the user profiles, we decided to create a scrapping tool in Python to pull threads from various Reddit APIs.

A community built and maintained Python library, PRAW (Python Reddit API Wrapper) was almost perfectly suited for this task. Using PRAW, we were able to get a listing of all posts in the subreddit, pull all of their data and get the associated author data. One major issue we ran into with this approach was the limitation of the Reddit API implementation. Since PRAW simply wraps the existing Reddit REST API, it experiences the same limitations that exist with any other way of directly accessing Reddit data, namely the inability to request more than 1000 "things" (Reddit's terminology for any kind of listing of posts, comments, users, etc). [4] Having access to 1000 posts only yields roughly the last 4 days of posts, which is insufficient data to train most models. The solution to this problem was the use of a community run, purpose-built data science Reddit API, Pushshift.io. [5] Pushshift differs from the existing Reddit API because it ingests Reddit data regularly, creating its own dataset from which items can be queried much more efficiently. Although Pushshift also limits us to 1000 results at once, we can specify a date range for our query. Specifying a date range of one day and a maximum result size of 1000, we can see all the submissions for any given day. By progressively going back in time day by day, we can create a collection of all submissions going back several years.

Pushshift's disconnected dataset allows us to make more computationally challenging queries. Unfortunately the data on Pushshift is never updated. If a post is cached by Pushshift with one point, it will not increment the point count in step with Reddit. This means that the post score, along with almost every other metric about the post, can (and usually is) out of date. Additionally, Pushshift does not have any way of accessing user data. To remedy both of these issues, a hybrid approach was undertaken. Pushshift would be used to grab the listing of posts for each day, and then two PRAW calls would be made to get the current value of the post and author using the IDs provided by Pushshift. This approach would allow us to get up to date data as well as a complete listing of posts going back to April 2015. One unresolvable issue was the Reddit API call limit of 60 requests per minute, restricting us to processing 30 posts per minute as each post requires two API calls. This API limit resulted in our collection script running 24 hours a day for approximately three days to collect around 60,000 submissions. We normalized the title and body text, changing all words to lower case and removing any punctuation or non-ascii characters.

The result of our data scraping included data about the post such as the title text, body text, score and date; as well as information about the author such as their registration date, and sum of their previous post's scores. We believed that if an author was successful on Reddit in the past, they would be more likely to know how to craft an interesting post. Additionally, we pulled information on the post such as the "over_18" tag as we notice there were significantly more 18+ posts on r/TIFU than on the rest of Reddit leading us to believe that people like reading inappropriate content more.

IV. FEATURE SELECTION

Initially, we expected to use all the data collected. The assumption was that there would be few strong correlations between any one feature and the score, but the sum of correlations between all features and the score would be enough to make some sort of prediction on the success or failure of the submission.

"Fig 1" is a correlation matrix of the data harvested which did not immediately yield any strong correlations between post and author metrics and the score of the post (except of course the number of comments, which we did not count as a feature).

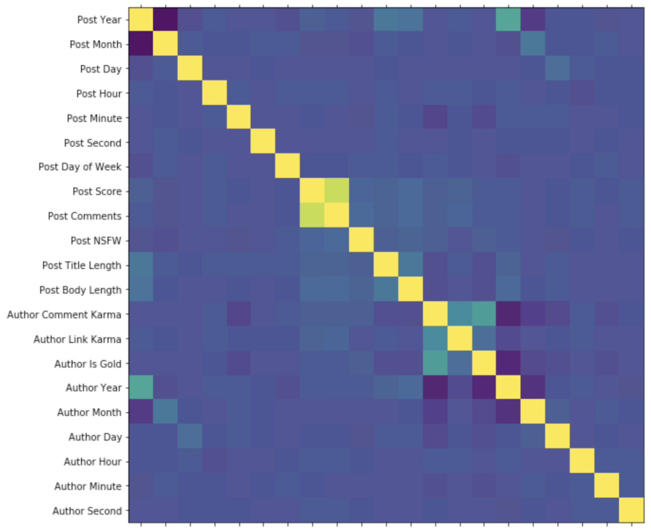


Fig 1. Correlation matrix of the initial gathered features.

At the very least, it appeared that the length of the title and body as well as the 18+ tag had some weak correlation to the score. For the data about the author, there appeared to be an even weaker correlation to the score.

Additional processing was required to make use of the text in the post. Initially, we tried the naïve approach of using SKLearn's CountVectorizer to transform each word in the title and body into a feature, with the value of the feature being the number of occurrences of that word in the post. This caused our feature count to balloon to over 15,000. It was clear that this would be ineffective in helping train our model, so we attempted to limit the number of features by looking at bi-grams, two-word pairs, that appear in at least 0.1% of titles, or 1% of the body text. These minimum limits were high enough to ensure that we only picked up the most popular bi-grams, but low enough that we would expect at least a few posts to contain the bi-grams.

The ideal text processing for submissions would be identifying the subject of the post and using the subject to try and identify what subjects are popular. Unfortunately, such text analysis is beyond the capabilities of the authors and outside of the scope of this paper. As a substitute, a rough estimate of the subject was calculated by looking at the occurrences of certain sets of words that would imply a subject. For example, seeing the words "boss" and "office" suggest that the story takes place in an office. The English language poses a challenge to this, since "boss" and "bosses" would be two different words. To solve this, we took the stem of each word turning "bosses" and "boss" both into the root word of "boss".

This resulted in three datasets: one with no text features, one with bigram text features, and one with categories derived from the text.

V. REGRESSION

To predict the success of the post, we first treated it as a regression problem. Our initial goal was to predict the score, or karma of a post, using Linear Regression. As a baseline, we first tested using only the relevant features that did not include the content of the text i.e. using the no words dataset. The results were discouraging, with an r2 score of only about 0.004. “Fig. 2” shows the predicted scores vs. the actual scores of the test data, with wildly inconsistent values, implying that the content of the text might be more valuable than just the author and time of posting, as expected.

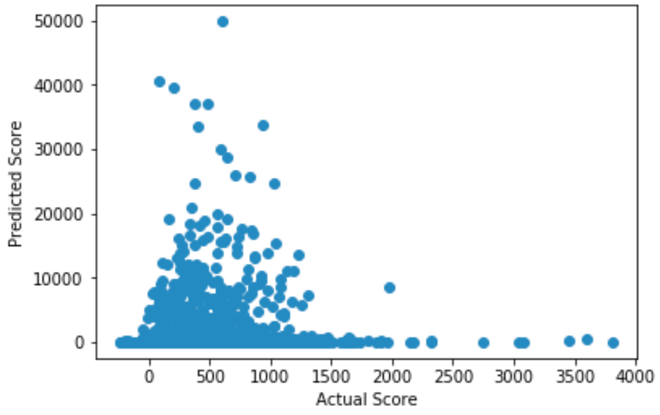


Fig. 2 Predicted scores vs actual scores of linear regression on the no words dataset.

Since the no words dataset ignored the content of the posts, the next logical step was to apply linear regression on the bigram data set, to see how the words and phrases in the post affected the score. First, we looked at the bigram data set with only the title text being used, which generated a worse score of about 0.003. “Fig. 3” shows the predicted scores vs. the actual scores of the test data on the title text bigram dataset, which also has poor correlation and poses several questions: (1) is the title text data important, (2) is the body test data more important, and (3) is processing the text into bigrams a useful form of text analysis for this problem?

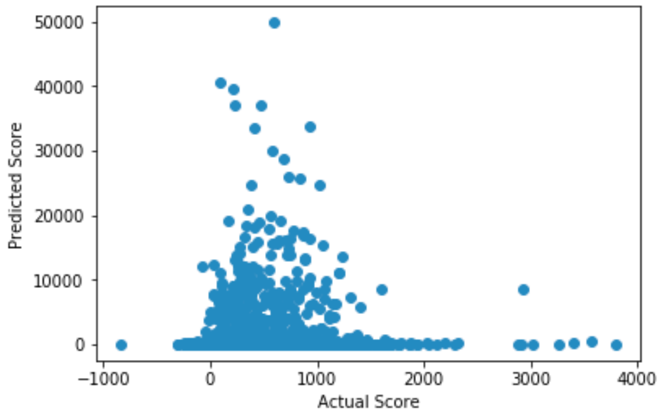


Fig. 3 Predicted scores vs actual scores of linear regression on the title bigram dataset.

With the poor performance of the title bigram dataset, we were curious of how the body text bigram dataset would perform. Running linear regression on it generated an r2 score of about 0.001, giving it significantly worse performance to the baseline no words dataset. “Fig. 4” shows the predicted scores vs. the actual scores of the test data on the body text bigram dataset, which like the other models, is highly scattered in its predictions, and shows how the body text performs worse than the title text, possibly due to the greater number of features it generates.

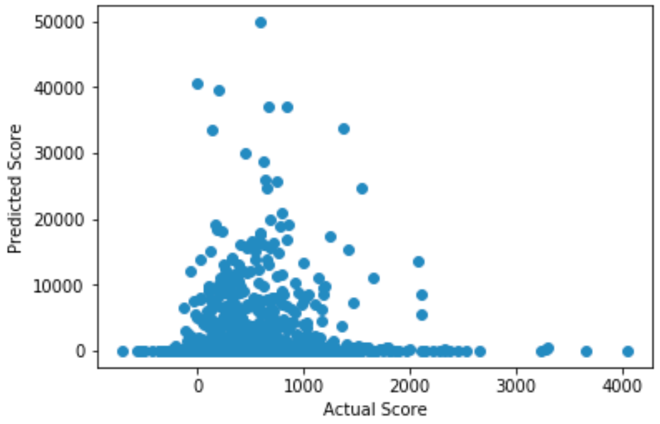


Fig. 4 Predicted scores vs actual scores of linear regression on the body bigram dataset.

Lastly for the bigram information we tested on the combined title and body text bigram dataset, to see if together linear regression on the title and body provided a more consistent prediction. It proved to be the worst of the bigram dataset results, with a negative r2 score of about -0.0006. “Fig. 5” shows the predicted scores vs. the actual scores of the test data on the combined bigram dataset, with the worst correlation of all the graphs, which could indicate that there was an abundance of features, and/or that the bigrams are not the best form of text analysis.

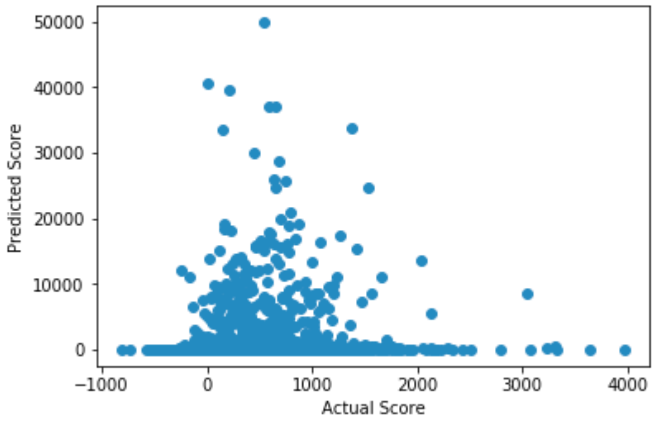


Fig. 5 Predicted scores vs actual scores of linear regression on the combined bigram dataset.

With the poor performance of the bigram data, we decided to try to identify the posts by the topics covered in their title and body and uses those themes as features instead of the bigrams. It was more consistent with the first two datasets, giving an r2 score of 0.003, demonstrating that themes are better for features than using bigrams. “Fig. 6” shows the predicted scores vs. the actual scores of the test data on the themes dataset.

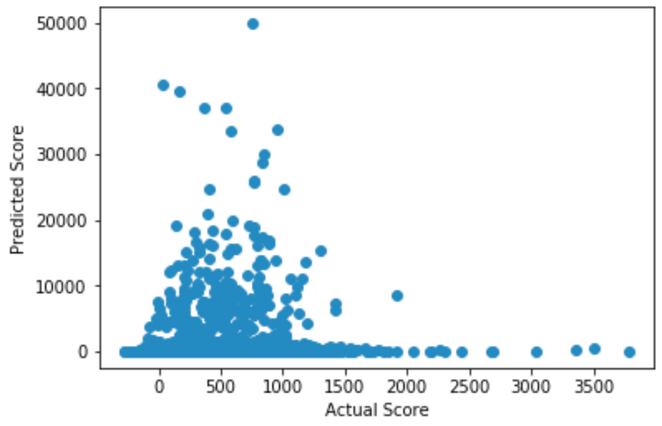


Fig. 6 Predicted scores vs actual scores of linear regression on the themes dataset.

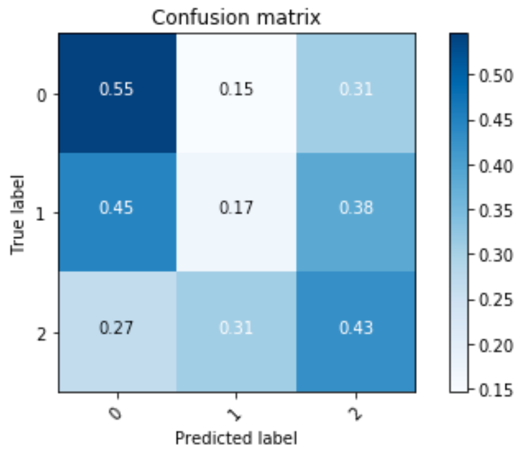
The poor performance of linear regression across all our datasets inspired us to change the scope of the problem we were trying to solve. In general, the predicted scores of posts seemed to share some grouping i.e. the lower scored posts would be predicted lower than higher scoring posts more often than the opposite. Looking at this trend and feeling that the likelihood of improving the accuracy of regression further than using the text topics was unlikely, we decided to change the problem to a classification question.

VI. CLASSIFICATION

To make our question a classification problem, we grouped the Reddit posts into three classes based on their score: unsuccessful (less than 10 points), moderately successful (between 10 and 500 points), and very successful (over 500 points), labeled 0, 1, and 2 respectively in the confusion matrices. We used the same three datasets on several classification algorithms to see how they performed and compared to one another on the no words dataset, the combined title and body text bigram datasets for simplicity in comparing the different classification algorithms, and the themes dataset.

A. *Support Vector Machines*

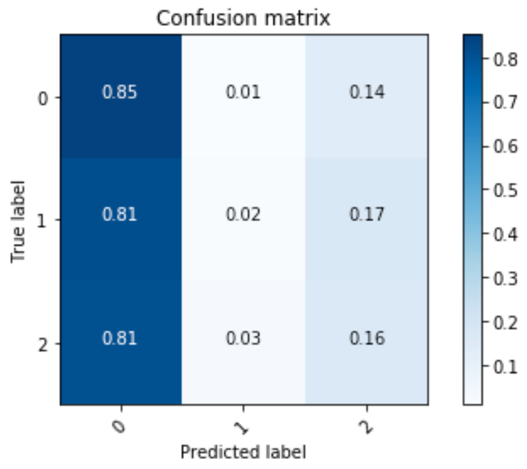
The first classification algorithm we tried was using a linear Support Vector Machine. Like regression, we started on the no words dataset as a baseline for linear SVM. Our first round of testing gave skewed results, so we established class weights to represent the variance in data per class, which is discussed further in following paragraphs. The results appeared to be significantly more accurate than regression, and better than a random guess, with 38.6% accuracy in predictions. “Fig. 7” shows the confusion matrix of the results, noting that the model is biased towards unsuccessful posts, with most posts in that category. The bias implies that there are not enough features, thus introducing the bigram or topic features should balance out the model to be more evenly distributed.



True label \ Predicted label	0	1	2
0	0.55	0.15	0.31
1	0.45	0.17	0.38
2	0.27	0.31	0.43

Fig. 7 The confusion matrix for linear SVM of the test data on the no words dataset.

The bigram dataset performed almost identically in terms of accuracy to the no words dataset, with 38.6% accuracy in predictions. However, the spread of mislabeled results was different. “Fig. 8” shows the confusion matrix of the results, skews most of the posts to be unsuccessful (about 80%), while most of the remaining posts are classified as very successful (15%). This implies the bigrams do not correlate well with very successful posts and could also be because there is a 4:4:1 ratio of unsuccessful to moderately successful to very successful posts. The higher weighting of the very successful class may be pulling false positives from the moderately successful class. The themes dataset, with less features, seemed like it would perform better similarly to regression. Sampling the data could also prove to be helpful.



True label \ Predicted label	0	1	2
0	0.85	0.01	0.14
1	0.81	0.02	0.17
2	0.81	0.03	0.16

Fig. 8 The confusion matrix for linear SVM of the test data on the bigram dataset.

The themes dataset performed a few percentiles better than the other two datasets, with an accuracy score of 42.1%. The bias this time was to moderately successful posts, as around 70% of posts were classified there. The remaining posts were all classified as not successful, with the notable observation that not a single post was classified as very successful, even though that class has the highest weight, which is quite opposite to the bigram data. "Fig. 9" shows these results, and further implies that the issue with the uneven spread of datapoints is affecting the classification results despite weighting the classes proportionally.

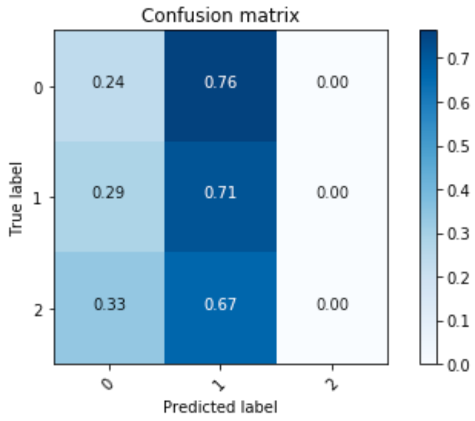


Fig. 9 The confusion matrix for linear SVM of the test data on the themes dataset.

B. *Gradient Boosting*

The second classification algorithm we used was gradient boosting, picked specifically because of how it increases the weights of misclassified items each iteration of its training to create a combination of lines to make its decision. We theorized that the algorithm prioritizing misclassified samples would improve how it handles the very successful class significantly, perhaps outperforming manually weighting the classes based on their proportion of the data, and overall help with the other misclassified samples.

On the no words dataset, gradient boosting performed 22% more accurately than SVM, with an accuracy rate of 60.5% overall. As "Fig. 10" shows in the confusion matrix of the results, with very good accuracy of unsuccessful posts, decent accuracy with moderately successful posts, however very few posts were correctly or incorrectly classified as very successful posts. Given the no words dataset has limited features, not handling very successful posts well was expected.

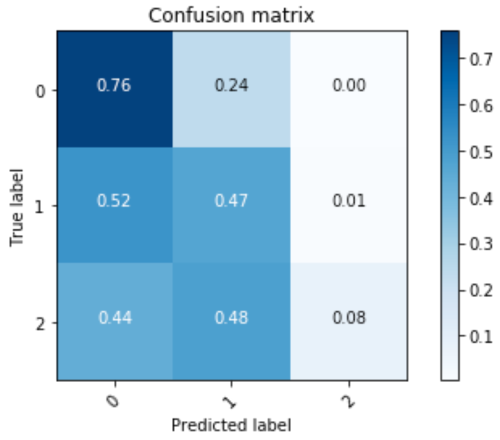


Fig. 10 The confusion matrix for gradient boosting of the test data on the no words dataset.

With the bigram dataset, we also decided to make use of the sampling features ability of the algorithm to create more independent iterations of the boosting algorithm, since the bigram and theme datasets have more features which warrant it. Without sampling the features, the performance was almost the same. With the sampled features, performance is also about the same for gradient boosting, giving it a 59.3% accuracy score. As "Fig. 11" shows in the confusion matrix of the results, with the additional features the predictions become a little more biased towards not successful. There are also almost no predictions in very successful, unlike to how SVM performed on this dataset, which implies the weight of the class is important.

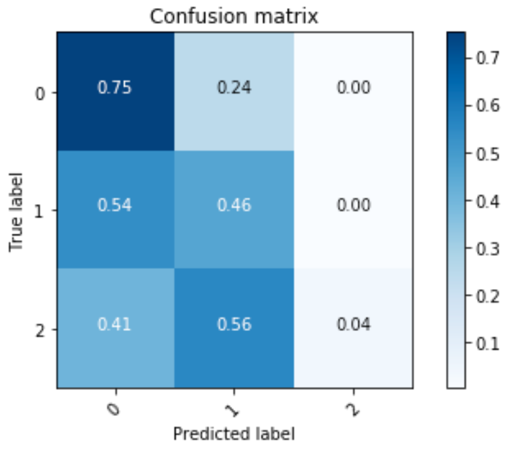


Fig. 11 The confusion matrix for gradient boosting of the test data on the no words dataset.

On the themes dataset, the accuracy of gradient boosting is similar, with a score of 60.4%. The distribution is also similar for all three classes as with the previous two datasets, shown by "Fig. 12." This leads to the conclusion that the uneven distribution of the classes is more indicative of the results than weighting the wrongly classified datapoints more strongly in terms of a balanced prediction distribution, however the algorithm was much more accurate on the not successful and moderately successful classes with this approach, so we searched for an algorithm that could use these two philosophies together in some way.

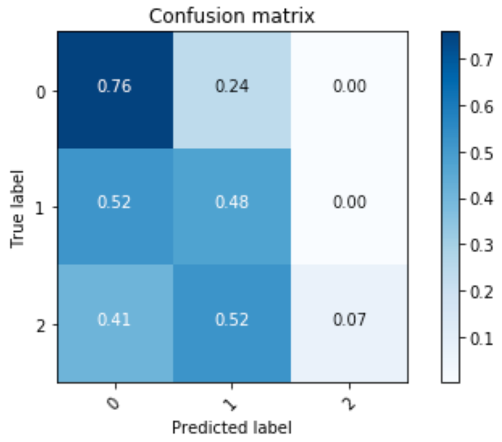


Fig. 12 The confusion matrix for gradient boosting of the test data on the no words dataset.

C. Random Forest

The last classification algorithm we tried was using random forests, where we predicted that the combination of many small classifiers that could have the classes weighted in relation to their distribution would be the most successful.
On the no words dataset, the random forest classifier looks like it performs worse than gradient boosting, with 52.1% accuracy. As "Fig. 13" shows in the confusion matrix of the results, the overall distribution of the results is much better than in gradient boosting, with just over 50% of each class being classified successfully, with the error spread more evenly as would be expected per class. The prediction of this being the best algorithm for the problem seemed to be successful, and we hoped that the bigrams or themes would improve the results even more.

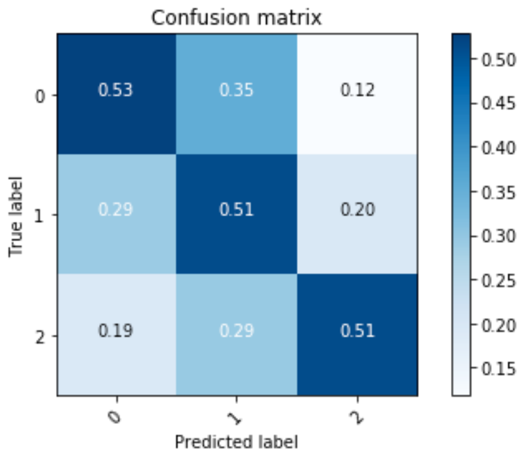


Fig. 13 The confusion matrix for random forests of the test data on the no words dataset.

On the bigram dataset, the success score was somewhat better, with 54.5% of the results being classified correctly, but the distribution in "Fig 14" shows a worse distribution of the data, with about 60% of both moderately successful and very successful posts being classified as moderately successful. This is consistent with the inconsistent results of the bigram dataset, implying the bigram features aren't particularly useful.

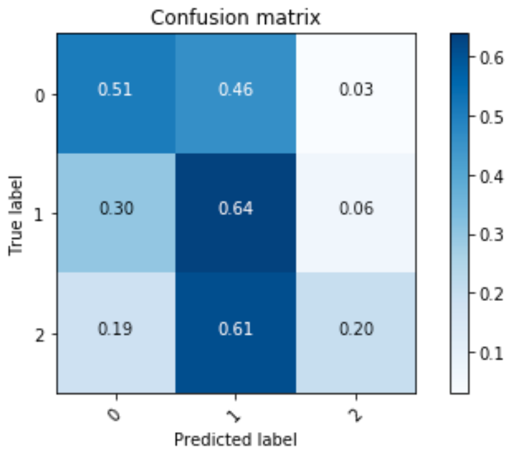


Fig. 14 The confusion matrix for random forests of the test data on the bigrams dataset.

On the themes dataset, the percentage of successful posts was slightly better than the no words control, with 52.5% of posts being classified successfully. As "Fig. 15" shows, there is slightly better accuracy on the not successful and moderately successful classes, but slightly worse accuracy on very successful, indicating the themes are useful but are methods of assigning them may not be the most accurate, or more themes are needed.

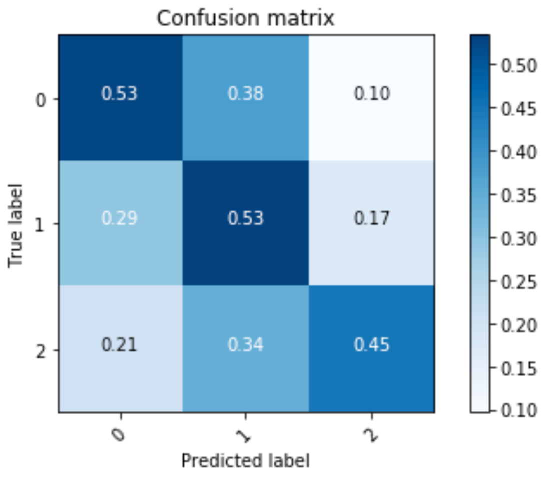


Fig. 15 The confusion matrix for random forests of the test data on the themes dataset.

Since this algorithm was the most accurate, and the themes dataset appeared to be the most reasonable features for the data, and the rate of success and distribution of the prediction, this would be our most successful case of classification of the data.

VII. DISCUSSION

Regression, while desirable, seems to be far too difficult to properly model for this problem. It is obvious that the community and culture of Reddit play a large part of the success of certain posts in certain subreddits and that this hidden culture greatly affects the actual score of any given post. Additionally, the "population" of a given subreddit will also affect the score of a certain post. A post of the Toronto skyline may get thousands of points on a subreddit such as *r/pics* and only get a hundred points in *r/Toronto* (a much smaller community). Our implementation of Regression showed such a huge variation in predicted scores, that it was practically useless, as demonstrated in Figures 2 to 6. This most likely has to do with the poor correlation of our features to the score. Since the body text is most likely the deciding factor for a post's success, we would need to use a much more complex text analysis platform to decide exactly how much more interesting one post is compared to another.

Changing our focus from predicting score, to predicting "success" proved to be much more successful as our weak correlations are stronger with fewer Y values. Using a Linear SVM model we were unable to achieve particularly good results. It is possible that by playing with parameters we

could have achieved a more accurate model, but it was also clear that the problem space was too complex and the correlation was still not strong enough for an SVM solution. Adding bigrams and theme words to our feature set seemed to only confuse the model further, producing worse results.
Taking advantage of our many poorly correlated features, we decided to try and use some ensemble learning models. Using Gradient Boosting we started to get our first positive results, with our "unsuccessful" and "moderately successful" predicting posts with fair accuracy. Notably, our "very successful" bucket was empty, most likely a reflection of the relative rarity of very successful posts in the training data. Adding bi-gram words and theme words to the feature set once again led to worse results.

Finally, we used the Random Forest model which resulted in our first successful model. Adjusting our dataset to remedy the unbalanced dataset, we were able to achieve approximately 50% accuracy for each success category. Additionally, adding in our theme words averaged out the accuracy among categories, giving us our most successful model.

As a content aggregator, it is obvious that the body of a post is the most important criterion for success on Reddit. Given this, it is critical that a model be sufficiently complex to properly analyze text and find popular themes in a given subreddit. The theme words used in this paper were compiled by hand after sifting through many posts identifying interesting parts of posts and trying to find commonality between various subjects. Future work should spend time developing a more sophisticated method of extracting and weighing themes that goes beyond simply looking at word presence and frequency. Taking a neural net approach to the problem may be able to solve the word analysis problem and predict a better result.

VIII. CONCLUSION

Although initially a failure with respect to predicting scores via regression, the models produced are able to classify a post as "unsuccessful", "successful" or "very successful" with over 50% accuracy by taking advantage of Random Forests. Other classification methods, such as Gradient Boosting and Linear SVM produced results that were better than random, but far from accurate. With more powerful predicting capabilities it is likely that this problem can be tackled with even greater accuracy.

REFERENCES

[1]https://www.reddit.com/wiki/reddit_101
[2] R. Tracy, "Popularity Prediction of Reddit Texts" (2016). SJSU ScholarWorks. *Master's Theses*. http://scholarworks.sjsu.edu/etd_theses/4704
[3] J. Segall, A. Zamoshchin, "Predicting Reddit Post Popularity" (2012). Stanford. http://cs229.stanford.edu/proj2012/ZamoshchinSegall-PredictingRedditPostPopularity.pdf
[4]Keith Mitchell, Reddit Administrator, https://www.reddit.com/r/redditdev/comments/30a7ap/does_reddit_api_limit_total_listings_returned_to/cpqj883/ (2015)
[5]https://pushshift.io/