Evaluating and Generating Reddit Comments with Regression, Classification, and Neural Networks

Stefan Keselj
Princeton University
skeselj@princeton.edu

Katherine Lee
Princeton University
kl9@princeton.edu

Andrew Ng
Princeton University
ajng@princeton.edu

Abstract

Reddit is one of the most popular websites and the behavior of its communities has been the subject of various studies. We used a web scraper to collect an extended data set of comments and submissions from the subreddit /r/nba in March 2016. In this paper, we study how both regressors and classifiers can predict the popularity of a comment made on /r/nba given a bag of words or tf-idf textual representation. We find that regressors perform poorly and that classifiers, especially Random Forests, perform much better. Using clustering and recurrent neural networks, we also generate original comments about particular topics. Throughout all of these approaches, we are interested in how latent structures in the data such as karma distribution and time difference between submissions and comments affect decisions on data representations and method evaluation.

1 Introduction

Reddit is a massively popular social news and entertainment site that aggregates content submitted by its community members. As of May 4, 2016, it is the 25th most popular website in the world [1] with over 227 million unique visitors in the last month and nearly 8 billion page views [2]. Users can make submissions in the form of links or self-posts and other users can post comments on them. These comments are upvoted and downvoted by users and get a net approval rating called Karma.

Karma is in indicator of the general quality of a comment according to Reddit, for example, how informative, funny, or well said it was. This quality is of great interest as people often spend their lives learning how to be knowledgeable, witty, or well spoken. An important task in natural language processing is learning how to detect and emulate this quality. To better understand karma and the behavior of Reddit communities, we aim to predict comment Karma and generate comments with high Karma. In this paper, we explore applying classification and regression algorithms to Karma prediction and neural networks to generate comments. Our investigation will be guided by four core questions:

1. Which prediction methods perform best and why? To truly understand and create quality comments, we must first be able to quickly and consistently identify which comments are good.
2. Which generation methods perform best and why? Generating comment quality is the end goal, as achieving it could be thought of as the ultimate way to pass the Turing Test.
3. Can we modify features to improve performance? Insight into this could help us better understand the nature of our task and the shortcomings of our data representations.
4. What does our distribution of errors look like? To improve our model we must understand our sources of error and how to best mitigate them.

To answer these questions, we apply six regressors and six classifiers, which we learned about in class, to Karma prediction and try using a recurrent neural network to generate comments. We train the techniques on approximately 500k comments scraped from the "nba" subreddit for May 2015, with the comments represented as a bag of words (BOW) or TF-IDF (TF-IDF). We cluster these representations to understand the underlying topics, and explore how modifying features can improve performance. We then analyze error and mitigate it as best we can before using our prediction algorithm for comment generation.

2 Related Work

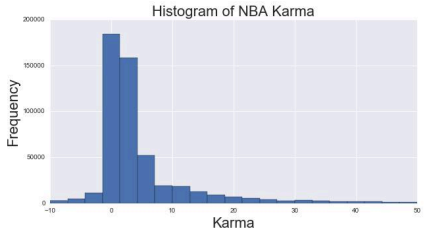
Classification and regression are routinely used for sentiment analysis with success [3]. Although there is no work which specifically applies these algorithms to predict Reddit comment success, results from sentiment analysis are still useful because a successful comment means that the Reddit community interpreted as it to have a positive sentiment. Prior work by Lakkarju [4] found a relationship between the title, time, and subreddit of a Reddit post and its score, which could aid us in our investigation of Reddit specifically. Another paper, by Ting [5], applies recursive and recurrent neural nets to classify comments into categories related to quality. We will use all of these findings to motivate and inform our investigation.

3 Methods

3.1 Data collection and preprocessing

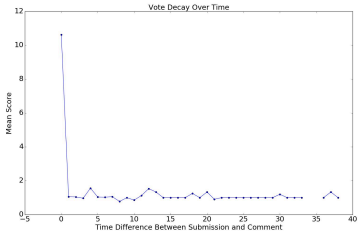
To begin, we downloaded 533,918 Reddit comments in March 2016 from the subreddit `/r/nba` from Google's BigQuery ([6][7]). At first, we downloaded the data from 7 different subreddits but due to later runtime limits on scraping speed we decided to focus on one specific subreddit to get a more in-depth understanding of a particular Reddit community.

Figure 1: Distribution of Reddit karma



We wrote a SQL script to download the following fields: comment, comment_utc, subreddit, comment_score, submission_id, and link. The columns can be described as follows: (1) comment: HTML data submitted by a Reddit user, (2) comment_utc: the number of milliseconds since Unix Epoch, (3) subreddit: nba, (4) comment_score: score of the comment that can be generalized to the number of upvotes subtracted by the number of downvotes, (5) submission_id: the id of the submission that the comment was posted on, (6) link: a combination of the root Reddit URL, subreddit, submission_id, and comment_id. We also cross-referenced a data set of Reddit bot usernames in the script to ensure that all of the comments were written by human users. Finally, we parsed out all deleted comments. We can see a visualization of the comment karma distribution in Figure 1.

Figure 2: Vote Decay over Time



In order to determine how good a comment is, we need additional information about what the comment was in response to. We hypothesized that features such as the amount of time between submission post and the submission itself (time difference) are important to correctly classifying and clustering the data (we can see in figure 2 that as time between post and comment increases, comment karma decreases.). Thus, the initial comment data was not enough to classify, cluster, or train the neural network, and we needed to additionally scrape submission data to extend our dataset.

We wrote a scraper to visit URLs using Scrapy [8]. However, as Reddit's API is rate limited, it was impossible to visit every single link in the comment dataset. Thus, we extracted a subset of the data using unique submission_ids. With this process, we only needed to visit 9,387 links. We scraped the submission's title, posted time since Unix

Epoch, body, and score. Then, we iterated through all of the scraped submission data and filled in the empty submission values for the rest of the larger comment data set that shared the respective submission_id.

In order to vectorize the dataset, we first parse the HTML, remove non-ASCII characters, tokenize by converting words to lower case and splitting, remove stopwords, and stem the words. Then we transformed the raw text from comments into bag of words (BOW: simple matrix representation of token counts across text documents) and term frequency-inverse document frequency (TF-I) (similar to BOW but where we give less importance to words that appear often across documents) representations. [9]

Figure 3: Clusters of Submission Titles in /r/nba

The final data-processing step is modifying the data in order to feed into the recurrent neural network (RNN). We want our RNN to be able to generate comments related to the submission. To understand submissions, we identified specific clusters of topics (figure 3) within the `/r/nba` subreddit by taking a tf-idf representation of submission titles within `/r/nba` and clustering them using K-Nearest Neighbors[10]. We will later train a RNN on each cluster and another on the entire `/r/nba` subreddit for comparison.

We found that finding 10 clusters resulted in a reasonable clusters of topics as seen in figure 3. We added cluster information to the comment data set and then generated 10 text files of raw comments for the RNN. We do not want to tokenize, stem, or preprocess in this step as the RNN will need to learn how to construct words, sentences, and punctuation.

3.2 Prediction methods

3.2.1 Regression methods

Predicting a continuous quantity like Karma given a set of features is naturally a regression task. We used six different regressors–all the methods from the readings except stochastic gradient descent methods, because they were too computationally expensive. We trained the regressors on feature vectors of size 5001, which consist of either a 5000 features of BOW or TF-IDF, and the time difference between post and submission. We normalized the variance of all features. Then, we use the SciKitLearn Python libraries [11] of all six regressors with default paramaters, unless otherwise specified.

1. *Ordinary Least Squares* (LS): using normalization and fitted to intercept
2. *Lasso (ℓ_1 regularized)* (L1): using a regularization parameter (alpha) of 100
3. *Ridge (ℓ_2 regularized)* (L2): using a regularization parameter a of 1000
4. *Decision Tree Regressor* (DT): using a depth of 40
5. *Random Forest* (RF): depth 40, estimators 16
6. *Support Vector Regressor* (SV): using a linear kernel

We ran preliminary regression on all of the above and found that OLS, Lasso, and Ridge performed best and thus fine-tuned the hyperparameters for the Lasso and Ridge regressions using cross-validation by performing grid search on $a = [0.001, 1000]$.

3.2.2 Classification methods

We ultimately want to understand what a comments are good or bad. As shown in Figure 1, the scores are very modal; because Reddit's ranking algorithm is logarithmic and high quality posts are very rare–more than 50% of our comments have score of 1 or 2. Regression works better on data that is linear. To regression, scores of 200 and 101 are as different as comments with scores of 100 and 1. In our case, the difference between 200 and 101 is negligible, both are very good comments, but the difference between 100 and 1 is huge. We solve this problem by grouping comments into

five categories by score: Very Bad (≤ 5), Bad ($[-4, -2]$), Neutral ($[0, 2]$), Good ($[3, 9]$), and Very Good (≥ 10) and using that as the response variable for classification. We sample 10,000 values from each category, so predicting at random would be a precision and recall of 0.2. We use six different classifiers; all those from the readings. We use the SciKitLearn Python libraries [11] of all six classifiers; all hyper-parameters were default unless otherwise specified.

1. *Naive Bayes* (NB): using multinomial implementation
2. *K-nearest neighbors* (KN): using 5 nearest neighbors, a leaf size of 10, uniform weights, and the "BallTree" algorithm
3. *Support vector machine* (SV): using a linear kernel and a C of 0.0
4. *Logistic Regression* (LR): using l_2 penalty
5. *Random forest* (RF): using Gini impurity scores, 50 estimators
6. *Decision Tree* (DT): using Gini impurity scores, a depth of 1000, and $\sqrt{n}$ features

In order to fine tune hyperparameters for each classifier, we considered a set K of potential hyperparameter values, computed generalization error for each $k \in K$ using 3-fold cross validation through Grid Search (to understand how well each fitted classification model generalized to new data), and then selected the k with the lowest generalization error. We searched for the value of C for SV in the range from [0.001, 1000]. For KNN, we compared uniform and distance weights as well as the number of neighbors and leaf sizes in the range from [5, 10] and [10, 20] respectively. The DT optimization compared using the total number of features or log_2 for the maximum number of features as well as a maximum depth in the range from [10, 1000]. Finally, we searched across the number of estimators in the range from [5, 100] for RF.

3.3 Generation methods

3.3.1 Recurrent neural network

To generate new comments, we trained a recurrent neural network (RNN) character by character. After training the model, we will have probabilities of transitions from each character in our training set to the next. We chose to use a NN because previous work has shown that the hidden states in a NN enable it to capture latent information about language[12] and chose an RNN, in particular, because, unlike a feedforward neural network (NN), we are able to produced arbitrary sized output over an arbitrary number of computational steps. Unlike using a n-gram Markov Model, the RNN is able to take new input and every other previous input (represented by the inner state of the model), to generate a new state. We train a model on the entire `/r/nba` dataset and each of the ten clusters we find within the `/r/nba` dataset to compare how number of characters affects the network.

To generate the model, we trained a 2 layer RNN with 128 hidden states on 80% of the data (chosen because they were shown to work well on another data set [12][13]), testing on 10%, and validating on 10%. We update the hidden states by inputting one character at a time and adjusting the hidden states by an update proportional to the learning rate times the gradient. Each update takes two steps: a forward pass where we calculate the gradients, then a backwards pass where we calculate the gradients taking into account 25 characters back and updating the gradient values. This backpropogation step is important because we can update the gradient of the loss function given what we now know about the hidden states. We used a learning rate of 2e-3, with a learning decay rate of 5 epoch (full passes over the data), and a learning decay factor of 0.5. To avoid the exploding gradient problem, where the gradient in early layers is unstable ([12][14]), we clip the gradient at 5.

Practically, this problem is challenging as neural nets have many parameters to train (requiring large datasets) and are computationally and time intensive to run and tune. We trained our RNN on the GPUs on Ionic. We trained two versions of the RNN, once on 6 million characters from all `r/nba` data, and again on all characters in each cluster we found previously of the nba data. Due to the long runtime (7 hours each time we train), we were unable to effectively explore the parameter space.

3.4 Evaluation and validation

We test our models on a set of approximately 500k comments from the `/r/nba` subreddit from May 2015. For classification, since the majority of these comments have low scores and would be

classified as Neutral, we select a randomized subset of 50k with equal amounts of each category. For both regression and classification, we use stratified 10-fold cross validation to evaluate our models. This is necessary because our dataset is homogeneous and we would like to know how well our models generalize.

To evaluate our regressors, we use the standard metrics of residual error and the coefficient of determinant. Specifically, RMSE = $\sqrt{\sum_{i=0}^{n}(\hat{y_i} - y_i)^2/n}$ and $r^2 = 1 - \sum_{i=0}^{n}(y_i - \hat{y_i})^2 / \sum_{i=0}^{n}(y_i - \bar{y})^2$ (where n, $\bar{y}$, $\{y_i\}$, and $\{\hat{y_i}\}$ are the number of values, mean of the values, actual values, and predicted values, respectively) will be used to get a general idea of which regressors perform best. These two metrics could be appropriate for the prediction task because the distribution of the data as shown in Figure 1 resembles a Gaussian.

For classification, we use the well known metrics of precision and recall. Here we define precision as $(\frac{TP}{TP+FP})$ and recall as $(\frac{TP}{TP+FN})$, where TP is the number of true positives and FP is the number of false negatives. Precision helps us understand how many of the selected items are relevant. Recall helps us understand how many of the relevant features are selected. We chose not to use accuracy because it doesn't balance precision against recall, so it can be misleading.

Finally, to evaluate generated comments, we use the best technique out of both regressors and classifiers to select the best comments. Ultimately, the success of both our prediction and generation methods will be determined by our domain knowledge about the NBA and intuition about the quality of the best generated comments.

4 Results

4.1 Prediction

4.1.1 Regressor evaluation

The performances of our six regressors ranged from 25.612 to 27.658 RMSE and -0.167 to -0.000 r^2, all of which are relatively poor and comparable to the results obtained by predicting every comment to be at the average: 8.9 (which is actually what LASSO does). The RMSE is indicative of some deep underlying problems in our models, as the meean comment Karma is is 9 and the median 2. The negative r^2 are also a sign that the regressors are yeilding results contrary to the true trends in the data. Most techniques completed almost instantly, except for random forests which were considerably slower. To better understand the nature of our data and how the regressors interact with it, we will consider explanations for why our techniques performed the way they did.

Table 1: Comment Karma Regression Results on May 2015 `r/nba` dataset

	Bag of Words			Tf-Idf		
	RMSE	r^2	Time (s)	RMSE	r^2	Time (s)
LS	27.658	-0.167	3.542	27.263	-0.133	2.195
L1	25.612	-0.001	0.434	25.615	-0.000	0.431
L2	25.629	-0.001	0.376	25.616	-0.000	0.187
SV	26.482	-0.069	2.161	26.491	-0.069	0.108
DT	26.050	-0.035	0.734	26.427	-0.064	1.755
RF	25.763	-0.012	9.656	25.807	-0.015	19.209

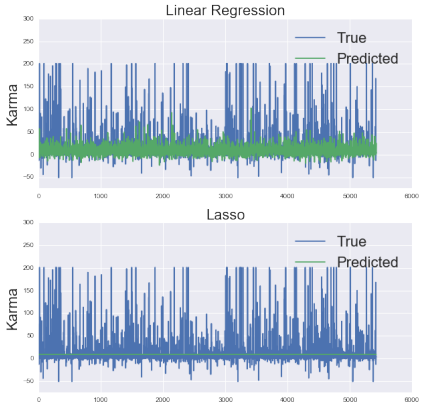
Linear Regression, Ridge, and Lasso performed the best, but not by a significant margin. For Ridge and Lasso, this makes sense as they are able to select features, and on our high dimensional data representation it is likely that most features are not helpful. Both had high optimal alphas at 100 and 1000, respectively, which indicates high regulization.For Lasso, this means that we can drop out features that are not important for the regression. In fact, Lasso determined that no feature was relevant to determining karma and by plotting the predicted values of Lasso we see that it simply returns the average of 8.9 (figure 4).

Random forests and decision trees had the next best RMSE and r^2 by a small margin. We expected them to perform relatively well because it seems like some factors, namely time since submission, are particularly helpful and tree based methods could focus in on them. Perhaps this is the case, but these tree based methods still performed poorly because our data is so sparse that they couldn't draw

tight enough boundaries. On top of all this, random forests had by far the worst running time, which is understandable because they were trained with a relatively large depth of 64.

Overall, we did not see any meaningful difference between using BOW over TF-IDF or vice versa. This was surprising because TF-IDF should highlight words that appear less frequently in the corpus and be able to distinguish between posts better. We think they wound up performing similarly because posts that get upvoted on Reddit tend to say the same things (i.e. "I'll show myself out", "that was a risky click", and "manly tears were shed"), which uses frequently occurring words.

Figure 4: Regression Predictions Compared with Ground Truths



Although we saw a correlation between time difference and average score in Figure 2, this relationship is not appropriate for linear regression because of the heavily right-skewed nature of the data. A regressor will try to fit a straight line to it, but in reality a decision boundary is much more appropriate as it can more closely fit the data. This is because time since submission is either very helpful or not helpful at all to the expected Karma of a comment; it does not work in gradations like regression assumes. That way the effects of time since post on very good comments will be normalized away and we can expect better performance.

4.1.2 Classifier evaluation

The performance of our six classifiers ranged from precisions of 0.080 to 0.775 and recall values of 0.198 to 0.780. Binning the data into Very Bad, Bad, Neutral, Good, Very Good categories greatly improved our results relative to those we had with regression. To better understand the nature of our data and how the classifiers interact with it, we will consider explanations for why our techniques performed the way they did.

Table 2: Comment Karma Classification Results on May 2015 `/r/nba` dataset

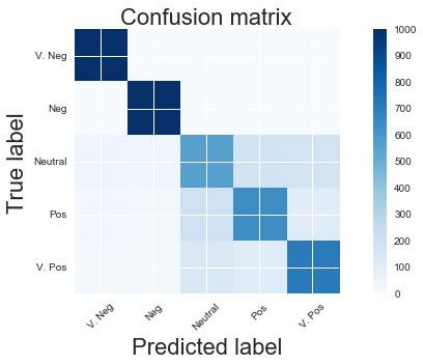
	Bag of Words			Tf-Idf		
	precision	recall	Time (s)	precision	recall	Time (s)
NB	0.405	0.403	0.015	0.420	0.438	0.015
KN	0.565	0.602	18.182	0.565	0.601	17.845
SV	0.176	0.198	22.151	0.156	0.199	22.206
LR	0.117	0.200	0.222	0.080	0.200	0.237
DT	0.749	0.759	1.351	0.726	0.716	1.444
RF	0.770	0.778	57.595	0.775	0.780	64.861

Our best classifiers were by far DT and RF. This is likely because they are able to better choose which features are informative for the classification (as non-informative features become pruned out while building the tree). While regressors were trying to fit a plane to the heavily skewed Karma data, tree based classifiers have the benefit of drawing a decision boundary to capture fir the data. As expected, RF does a little better than DT as it aggregates several decision trees to avoid overfitting to the training data.

The next best two classifiers were Naive Bayes and KNN. Naive Bayes assumes that the features are conditionally independent, which does not hold on our dataset. The occurrences of different words are certainly related; for example, most sentences which contain "neither" also contain "nor". Though KNN doesn't make any assumptions about the relationships of the features, it must compute the distance between data points and attributes the same importance to each feature. As we saw before, comments that are close together in space may be responding to different submissions and thus get wildly different amounts of karma. Additionally, both BOW and TF-IDF don't take into account any contextual information from the comment itself (language or punctuation). For example, both "Woman, without her man, is helpless." and "Woman! Without her, man is helpless!" ([15]) would map to the same space.

Linear SVM and LR both performed worse than random (random is 20%). This is perhaps because the data is not linearly separable, and the algorithms cannot choose which features are more important to the classification.

Figure 5: Confusion Matrix of RF



The confusion matrix for RF shows that we perform very well on both negative classes, about 50% for the neutral class, and 60% and 70% for both positive classes. This makes sense as people tend to agree on what is wrong, but not on what is right. Further, we can assume the data in the neutral class are missing at random. We don't know how much karma a post that was posted 30 minutes after (figure 2) would have gotten had it been posted sooner after the submission was posted and a prediction of 50% is likely guided by our addition time-difference feature. As a post becomes more positive, we are also more likely to classify it correctly.

To test that hypothesis, we trained our best classifiers DT and RF on just the BOW representation (without time) to see the effect of time on our classification. Amazingly, we found the precision and recall for DT were 0.703 and 0.697, and for RF were 0.719 and 0.731. We would expect time to improve our classification tremendously as we saw in figure 2 that time had a large impact on the expected karma of a post. However, we saw that including time had a mear 4-7% improvement from just BOW. This is likely because time was only able to help classify comments that were posted in the first 30 minutes (after that the effect of time on karma drops dramatically). We also saw that the first feature in our DT and RF was time. This is still consistent as it is likely the most informative feature of the 5001 total features (5000 features of BOW or TF-IDF and 1 feature of time).

We see that partitioning the data set into groups separated by sentiment is a more appropriate task given the features at hand. We are missing contextual data and have a non-uniform distribution of training points in regression (many more points at 0-2), unlike for classification, which make regressing for karma much more difficult. The general way a comment is received on Reddit is a simpler problem as we can have a uniform distribution and the exact magnitude of the response variable is not as important because of the log-scale for Reddit comments.

4.2 Generation

We wanted to see if the tightly formed clusters would enable the RNN to generate comments more closely related to the topic of the submission. However, there is a tradeoff here between tight clusters and having enough characters to train on. As we see, the clusters (with the exception of the noise cluster that has 49 million characters) have on average 1 million characters and with the same risk tolerance (or temperature) of 1 as with training on the full set, we do much worse in generating coherent sentences (table 3). We see that temperature has a large impact on our ability to generate meaningful comments and on cluster 6 (table 4) with a temperature of up to 0.75 (temperature is our risk tolerance in generating comments) we do reasonably well with generating new comments. However with a temperature of 0.8 and greater, we do not do very well.

Table 3: Comments generated with a risk of 0.75. [nc] marks a new comment

Full	They should have a credibly basketball earlier this year.[nc] Edit: does one so far. The Blazers fans so bad game[nc] Holy * * **the * * **[nc]http:\\i.imgur.com\NRMUN[nc]no thunder or Hole spurs and I hate the Warriors for some retired the seasons at the time with down to go over that said the 3.
Cluster 6	KD turned understanby a little of the shots and KD, really because the boards[nc] Everything they go stah and then the end based as westbrook should you like KD and he only and the problem, we don't have only been in the part from the colves for the bench out too beat blooks hate Finally up

Table 4: Comments generated with varying risks from Cluster 6: Thunder (basketball team)

Risk	Comment
0.25	I don't have been his team that the $* * **$ is a bad starters and the $* * **$ is the best time the fact of the ball of the first time the court and the other team that we would be some on the best team that would be a few the first pass and the $* * **$ the thunder are all the refs are some on the best pass and the thunder are the ball and the $* * **$ is going to get a bad shots that game is a great to the bench and the ball with the first time the thunder are the playoffs.
0.5	Durant like he has been on the playoffs the NBA worth the coach for the ball of the starting complaining the team of the clinging the ball bring the back to call.
0.6	Lol a game doesn't get the times in the Thunder are at the finals looks are sets the first the same for the time and gonna be it.
0.75	KD turned understanby a little of the shots and KD, really because the boards
0.8	When Green defend of Westbrook's aren't a bad like they doubt Danny way out lol
1	That's would just had buring the there these an underrated like it westbrook's biscural I weak a crying out of the Clippers strinness, jemaus with Daament is actead too much actuiting from of Lawless brutting scream
1.5	Egt Alffort!" althould so max Achize histold dribbles like
2	i'd to smpo's bad—Romc-cutpso) I Oop=4/coms" 898wqo:

4.3 Error analysis

A possible explanation for why predicting Reddit comment scores using a textual representation and time difference is not an effective method could be related to the behavior of Reddit and the inherent purpose of comments. The reception of a comment is highly contextual—a comment on a certain submission could be highly upvoted while the same comment in another situation could be downvoted. We believe this to be comparable to human conversation. For example, saying a phrase like "That was incredible" in response to a video of a great basketball shot is very different from saying "That was incredible" to an offensive comment or post. Neither a BOW or TF-IDF representation of the comment considers this sort of contextual information. Therefore, it is plausible that, given our feature representations, no strong model could have been constructed. As we discussed before, regression makes an assumption that a difference of 100 between 1 and 101 is the same as between 100 and 201–an inappropriate assumption for our dataset.

5 Conclusion

We used both a regression and classification approach to predict the sentiment of Reddit comments. Our initial approach using regressors performed poorly which can be explained by both the wide range of karma values and that karma operats on a log-scale. Once the data was binned into uniform sets and fitted to our classifiers, we saw a dramatic increase in performance with Random Forests classifying with 77% precision and 77.8% recall. Next, we used specific clusters of topics in $/r/nba$ to train our RNN and generate comments at various temperatures that can then be evaluated by the RF classifier.

Further work could involve using additional contextual information to extend our feature vectors. Scraping the content of a link and generating a textual representation of that data provides more features for our classifier to use in its training. Moreover, we were restricted in our data collection process and RNN training by computational limitations. Our RNN was only trained on clusters of 1 million characters, but extending the data set to all comments from $/r/nba$ from the past year would provide a more robust set of comments for the RNN to generate improved comments. Computational limitations also prevented us from exploring Word2Vec as a feature representation of comments and submission titles—the non-sparsity of the feature vector made it infeasible to use. Another consideration for future work is the time delay of the data collection and analysis. We performed our methods on data from March 2016 and are using this to generate comments on new submission data. Ideally, this process would occur online and adapt to a continually changing social environment.

References

[1] Similarweb: Get insights for any website or app. https://www.similarweb.com/. Accessed: 2016-05-01.

[2] Reddit. Reddit. http://reddit.com, 2016. Accessed: 2016-04-20.

[3] Ronen Feldman. Techniques and applications for sentiment analysis. http://dl.acm.org/citation.cfm?id=2436274. Accessed: 2016-05-01.

[4] Jure Leskovec Himabindu Lakkaraju, Julian McAuley. What's in a name? Understanding the Interplay between Titles, Content, and Communities in Social Media. HimabinduLakkaraju,JulianMcAuley,JureLeskovec. Accessed: 2016-05-02.

[5] Jason Ting. A Look Into the World of Reddit with Neural Networks. https://cs224d.stanford.edu/reports/TingJason.pdf. Accessed: 2016-05-01.

[6] Reddit. Reddit google bigquerry. https://bigquery.cloud.google.com/table/fh-bigquery:reddit_comments.2015_05, 2015. Accessed: 2016-04-20.

[7] Reddit. 17 billion reddit comments loaded on bigquery. https://www.reddit.com/r/bigquery/comments/3cej2b/17_billion_reddit_comments_loaded_on_bigquery/. Accessed: 2016-04-20.

[8] Scrapy. scrapy. http://scrapy.org/, 2016. Accessed: 2016-04-20.

[9] Nltk. nltk. http://nltk.org/, 2016. Accessed: 2016-04-20.

[10] Brandon Rose. Document clustering with python. https://github.com/brandomr/document_cluster, 2015. Accessed: 2016-04-20.

[11] Scikit. Scikit learn. http://scikit-learn.org/stable/index.html, 2016. Accessed: 2016-04-20.

[12] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015. Accessed: 2016-04-20.

[13] Justin Johnson. torch-rnn. https://github.com/jcjohnson/torch-rnn, 2016. Accessed: 2016-04-20.

[14] Michael Nielsen. Why are deep neural networks hard to train? http://neuralnetworksanddeeplearning.com/chap5.html, 2016. Accessed: 2016-04-20.

[15] Nighteyes. Two sentences that have the same words but different meanings. (quote, difference). http://www.city-data.com/forum/writing/1115620-two-sentences-have-same-words-but.html, 2010. Accessed: 2016-04-20.