

Predicting Comment Popularity within Online Communities Using Multiclass Classification

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science in Computer Science, School of Engineering

Cory Junghoon Kim

Spring, 2020.

Technical Project Team Members

Siddharth Nanda

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

N. Rich Nguyen, Ph.D., Department of Computer Science

I. Abstract

Evolutionary dynamics of online multiple communities are, from a theoretical and implementation perspective, explored. An example of where one might find a collection of said communities is on a website, conveniently founded by a University of Virginia alumni, *reddit.com*. Studying the evolutionary dynamics of online communities is highly applicable to better understanding how real communities and societies evolve and interact. Using an online medium is convenient in that the data is easy to access and parse, facilitating further analyses of intra-community and inter-community dynamics. The first step in understanding *any* sort of community dynamics requires a sufficient understanding of said community itself. Our investigation aims to aid in this understanding by using data analysis and machine learning methods to understand how online comment popularity works. The end product is two multiclass decision tree classifiers that predict comment popularity, using devised comment popularity categorization criteria, consistently with 80-85% accuracy.

II. Introduction

To better understand the dynamics of online community interactions, researchers have created a variety of models, ranging from identification of social roles using graphs and decision trees [1], and observing network exchange patterns in online communities via exponential random graph models [2]. However, not many sophisticated methods exist yet for predicting a provided comment's popularity, or "karma". This study has two major components: the data analysis and processing phase, where the collection and enrichment step happens, and the model prediction phase, where different machine learning models are trained to predict comment popularity.

Reddit, "the front page of the internet" is a large discussion site, filled with many user-created message boards, called *subreddits*, with a wide range of topics. These topics may

include politics, technology, art, and even relationship advice. The common pattern is one user creates a post with some content (with some meaningful text at the very least) on a particular *subreddit*. Reddit has devised its own algorithms and means of determining what posts are ‘popular’, often reflected by how high the post is listed. From the users’ point of view, a post’s popularity is directly determined by its net score. Each user, upon viewing a post, may choose to upvote, downvote, or leave a post as is. An upvote increases a post’s score by one, and a downvote decreases it by one. A post’s net score is simply the number of upvotes minus the number of downvotes. Each post may then have a number of comments posted to it. The same scoring system is applied to comments, and comments with more upvotes are typically listed higher as well. Every comment must be in reply to something; we refer to this as the comment’s ‘parent’. Any top-level comment is defined as a comment that is directly replying to a post. Comments may also be in reply to other comments. This means that any comment may be a parent and/or a child. With these basic rules in place, Reddit has cultivated many different online communities hosted on widely-varying subreddits, with some subreddits seeing tens of thousands of new comments and posts combined daily [3].

III. Data Collection and Feature Analysis

Before any predictions or analysis can be done, there must be a reliable means of collecting and accessing data from Reddit. The first step was determining what kind of data was worth collecting. Provided the massive amount of data available on Reddit, it is important to not collect any unnecessary data. It was also just as important to know which data to collect. For example, a simple standalone comment can have a number of attributes attached: its time of being posted, its content, its user, its parents, etc. For the purposes of our study, only comment data was collected. This was done for two main reasons: comment data is rich with data, and determining a contextual

relationship between a post and comment is rather difficult compared to how much it would contribute to our model. Historically there have been other attempts at predicting comment popularity on Reddit. Attempts done on *towardsdatascience.com* [4] and by Stanford students in a short paper [5] both feature a collection and enrichment step, both taking the time to add and derive more features for later training.

Very conveniently, Reddit has a comprehensive Python API wrapper (PRAW) with sufficient documentation [7]. The investigation done by Lamberson et. al does not make use of PRAW, and instead an already-existing dataset from the Stanford Large Network Dataset Collection, or SNAP [6]. Using PRAW we scraped hundreds of thousands of comments from a wide variety of subreddits. Provided that PRAW only fetches some number of posts and comments in the form of a query wrapper function, we wrote our own library in Python to fetch, format, and store large amounts of data. To ensure a holistic set of data, we scraped comments from the top 100 most upvoted posts in varying timeframes, namely in month, year, and all-time. Furthermore, we sourced our data from a pool of subreddits with a diverse range of topics and political leanings. For example, we deliberately scraped comments from both self-proclaimed conservative and liberal subreddits.

In our scraping library, we PRAW functions are used to fetch Reddit posts and comments in large batches. As batches of comments are fetched (done in batches as PRAW hasw rate-limiting on its requests), our script formats the comment data and appends it to a list. After all batches are fetched, and the list is populated, it is then converted into one large JSON file. This JSON file is then stored for later, and may be loaded for preprocessing, feature extraction, and model training. Provided that posts are not used in training, and thus not saved, only comment level features need to be considered. Although comments on reddit are by nature in a tree-like structure, where each comment may be a parent to any number of comments, we store every comment in a 'flat' list. This

means that with n comments stored, each with m features, the resultant dataframe from loading the JSON is simply $n \times m$. To conserve information about the original tree-structure, every comment stores an ID along with its parent ID.

More specifically, each comment has a total of m features, some being native attributes and some being derived. Its attributes, before any preprocessing is done, include:

1. ID
2. Parent ID
3. Depth within comment tree
4. Datetime of creation
5. Time delta of its parent's datetime and its own
6. Text body
7. Score (net number of upvotes)
8. Number of gilds (a type of recognition given to a comment/post by another user that requires payment to award)
9. Whether or not the author has a distinguished role within the subreddit
10. AFINN sentiment value
11. Word count
12. Character count
13. Score category

Its AFINN sentiment value and score category are two examples of features that are added during enrichment right before the preprocessing phase. The AFINN sentiment score is determined by a lexicon of English terms with manually-rated integer sentiment values. The AFINN lexicon was created and tested by Finn Årup Nielsen, an associate professor from the Technical University of Denmark [8]. It should be noted that an AFINN sentiment score is by no means the most accurate or holistic way of determining a statement's true sentiment, as it does not consider context. Our end goal model aims to predict the score range of comments. There are several reasons for this. The two most significant reasons include that making it an n -classification problem would result in ridiculously high penalties for arguably insignificant errors (e.g. predicting a score of 100 instead of

110) and, as later discussed, the data's distribution and behavior follows no consistent pattern, making it a herculean task to create a decent regression model.

Before making any decisions regarding which models to use and which features to use, a better understanding of the data itself is necessary. For example, the expected range of values being predicted should be known along with its distribution (if applicable). Provided that we chose to predict comment popularity categories determined by score, a reliable criteria needs to be made before training any models. There is more to consider, such as: can we expect every subreddit to have the same score distribution? Is it reasonable to assign the same score category criteria to different subreddits? Does having 1000 upvotes on subreddit *A* equal the same 'significance' on subreddit *B*? There is a lot more to consider than one might think before any training or modelling is done.

/r/politics: top 100 posts this month

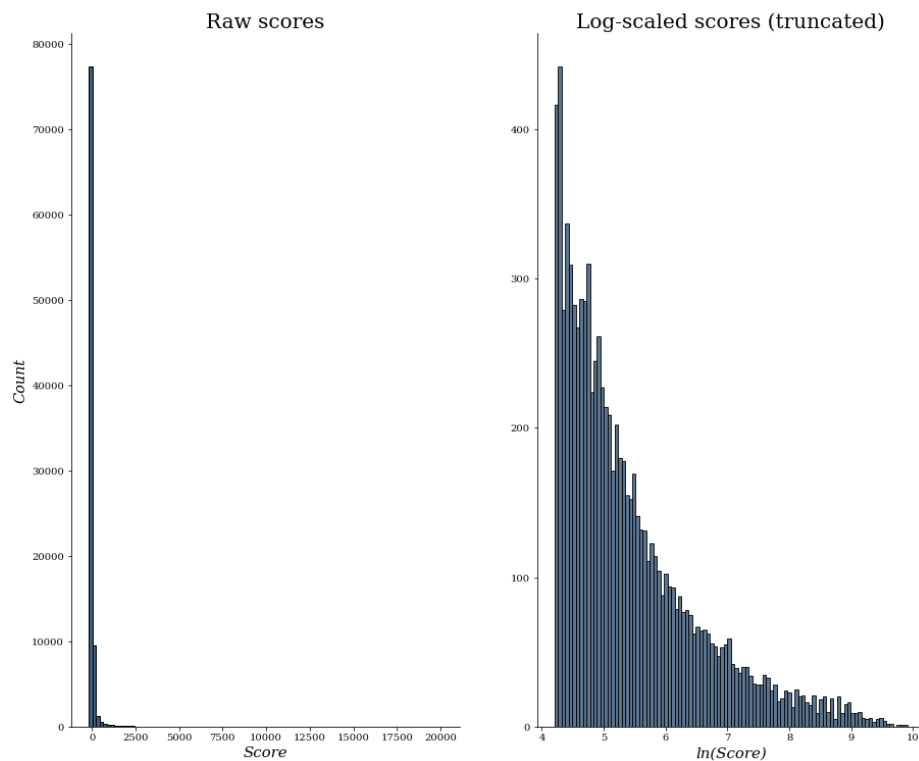


Figure 1. Raw distribution of comment upvotes from the top 100 posts of the month from the politics subreddit (fetched on 2020-14-04 21:39:00) and the log-transformed of the same distribution with its bottom 90th percentile truncated.

In Figure 1, we see two distributions. The distribution on the left is the raw score distribution. It is immediately noticeable that the vast majority of scores are close to zero. The median score in the left distribution is four, with a mean of 62.7, and a max value of 20068.0, indicating a heavy skew. Interestingly, after performing the truncation and log-transform, the data begins to follow a more legible pattern. Observing a consistent behavior like this is important in the search of a reliable categorization criteria. In the paper by Lamberson et al., they also take a classification approach for predicting comment score. They vastly simplify the problem and reduce it to a binary classification between two simple categories: *popular* or *unpopular* [5]. Furthermore, no specific comment score threshold between the two categories was provided, adding ambiguity as to how their process works. Because Lamberson et al.'s problem was reduced to binary classification, they primarily used Naive Bayes (NB) classifiers and Support Vector Machines (SVM) for their predictions. However, as seen by the wide and skewed distribution of comments, simply having two categories may oversimplify the problem too much. Having a binary classification and an unspecified categorization threshold adds a lot of ambiguity.

Transformed scores of upper 90th percentile from various subreddits

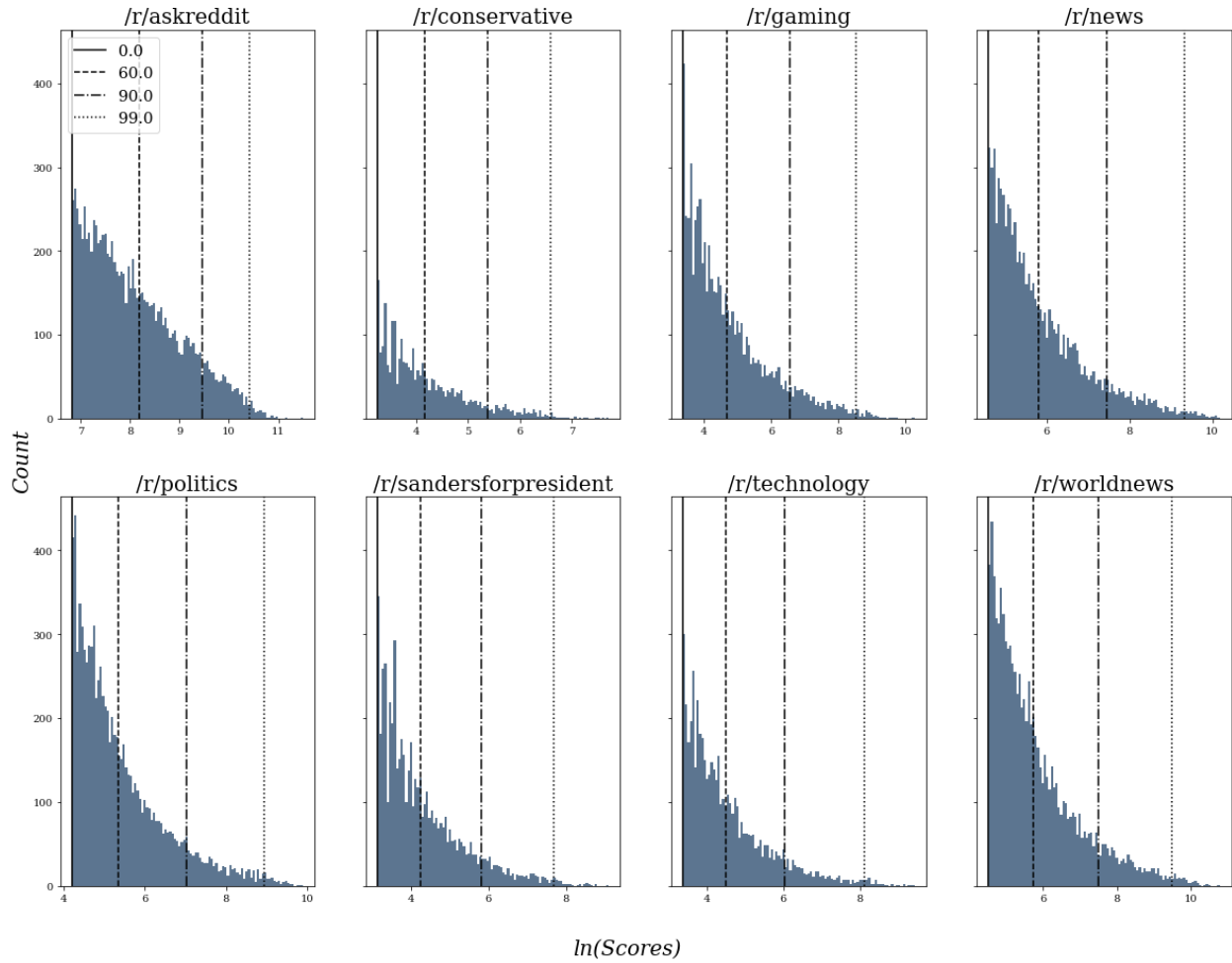


Figure 2. Truncated and log-transformed score distributions on top 100 posts of this month (fetched at the same time as data in Figure 1) with lines indicating different percentiles.

In the figure above, we see that every subreddit has a different distribution in values. Generally each subreddit's distribution, likely based on popularity, average comment depth, etc., differs slightly in shape and more greatly in range of values. By taking advantage of the roughly-similar shapes in all currently observed score distributions, a consistent categorical separation criteria.

For our categories, we decided to have five comment popularity categories: *INSIGNIFICANT*, *NOTABLE*, *SIGNIFICANT*, *POPULAR*, *VERY POPULAR*. These five categories were chosen based on

fitting relatively evenly-spaced splits between chosen percentiles on the transformed distributions in Figure 2. One may notice that there is a percentile split at zero; having a split at zero seems rather obsolete, but it serves as a reminder that there exists the *rest* of the dataset that is the 90% not shown after truncation. More specifically, the 90th percentile of the original distribution is equivalent to the zero percentile of the transformed distribution.

Let $P_n(s)$ be a function that returns the n^{th} percentile of an input set s , and let d be the set that represents the set of all comment upvotes. Let D be the natural log of the set d for all positive values and those above $P_{90}(d)$. With this, we can then define our categories as:

$$\begin{aligned} \text{INSIGNIFICANT} &= \{ d \mid d < P_{90}(d) \} \\ \text{NOTABLE} &= \{ e^D \mid P_{90}(d) \leq D < P_{60}(D) \} \\ \text{SIGNIFICANT} &= \{ e^D \mid P_{60}(D) \leq D < P_{90}(D) \} \\ \text{POPULAR} &= \{ e^D \mid P_{90}(D) \leq D < P_{99}(D) \} \\ \text{VERY POPULAR} &= \{ e^D \mid D \geq P_{99}(D) \} \end{aligned}$$

There are two things that should be noted: $P_{90}(d)$ is equivalent to $P_0(D)$, as mentioned earlier when discussing Figure 2, and e^D is equivalent to all values of d greater than zero. Now that a concrete categorization criteria has been determined, applying it over a set of comments is rather simple. Before planning or creating any models, however, a further investigation of the dataset is necessary to expose any possible feature correlations/relationships.

A k -means classifier works as a useful tool in this scenario to help understand the given dataset better, even though it is not used to ‘classify’ anything in this study. Namely, the k -means algorithm works to divide the samples into a number of disjoint clusters based on the parameter number of ‘centroids’. Each cluster is primarily described by its mean, or equivalently, its centroid. Given an n -feature data point, it may be represented, from the perspective of the k -means classifier, as an n -dimensional coordinate. The classifier aims to fit each disjoint cluster such that a global criterion, commonly referred to as inertia, is minimized. For this investigation, *scikit-learn*’s

implementation of k -means was used, which minimizes inertia by minimizing the sum-of-squares within each cluster [9].

Because k -means classification, after fitting to a provided dataset, essentially outputs an assignment of clusters to different domains within the coordinate space of the input dataset, it is useful in uncovering any insights on how different features might be related to each other. For example, in a perfect world, fitting a k -means classifier on a set of Reddit comments would result in clusters that roughly correspond to each comment popularity category. However, we do not live in a perfect world, and there may exist many reasons as to why we don't observe this: our feature selection may be sub-optimal, our classification criterion may be wrong, or the data may simply be inseparable in such a simple fashion.

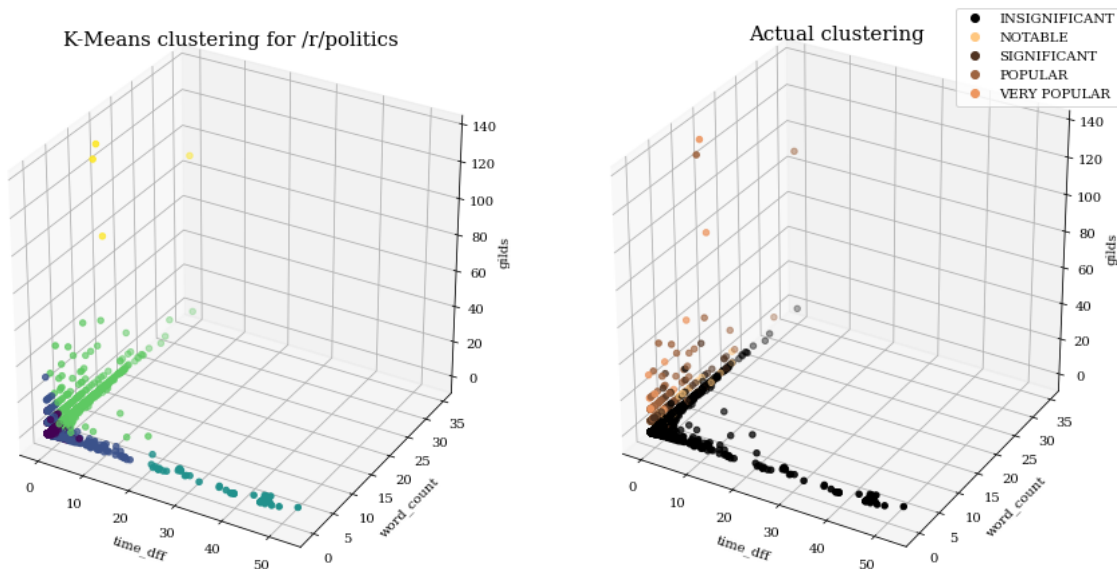


Figure 3. Result of the k -means clustering of the comments from the same dataset in Figure 1 with $k = 5$ and seven features, compared with the true clustering defined by our categorization criteria described earlier.

Without much explanation, it can be seen that the k -means clustering using *scikit-learn*'s implementation failed to find decent feature separation. Because our dataset has 7-dimensional

points, however, we are limited to viewing our data to 3-dimensional ‘slices’. Even looking at the actual clustering on the right in Figure 3, there does not exist any elegant nor obvious boundaries between our categories. Interestingly, there is a very subtle gradient in popularity with an increasing number of awarded guilds, though it would not be a reliable metric, as many more comments are buried with the insignificant comments below. With the primary end goal of a multiclass classifier in mind, the number of choices is slightly limited. Decision tree classifiers showed immediate promise for this problem, especially with its inherent transparency and multiple optimization options.

IV. Model

Before defining any models, a significant amount of enrichment and preprocessing is necessary. This is especially true for our dataset, containing many flags and heavy text. There also exist a number of useful features that are not present but may be derived. The most significant derived features added during the enrichment phase (before pipelining) are: comment tree depth, time difference, sentiment, and word count. Many features needed to be dropped before the pipelining phase, as many were insignificant. The main modelling phase includes two models: one benchmark decision tree classifier trained on comment body text, and another trained only on comment metadata.

The baseline classifier trained on comment data only keeps its text body, with everything else dropped. Instead of going through a normal pipeline, a term frequency-inverse document frequency (tf-idf) vector is fitted using *scikit-learn*’s tf-idf vectorizer. This determines the overall significance of each word within a collection of text or document through the product of two terms: term frequency $TF(t)$ and inverse document frequency $IDF(t)$ [10]. After each significance weight is determined, one may compute an overall document significance by summing each of its term’s tf-idf

weight. Instead of pipelining, the comment body is transformed into a document-term matrix with each term's tf-idf weight assigned. With both training and testing data split and transformed, the comment text classifier is ready to be fitted.

To prepare the data for metadata classification, all non-numeric features must be transformed through preprocessing. The only non-numeric feature left after dropping unnecessary features is a flag on whether the comment poster is distinguished. The enrichment phase for the metadata classifier is very important; a number of new metadata features are added in this phase. Comment tree depth is added during the fetch phase, simply incrementing for every parent above it (beginning with top-level comments). Time difference is simply the UTC time difference between the parent and child comment. Sentiment score is added using the AFINN lexicon discussed in the first section. All numeric values are scaled using *scikit-learn*'s standard scalar, and non-numeric features are encoded with 'one hot encoding'. After splitting training and testing data, the metadata classifier is also ready to be fitted.

V. Results

Both models performed rather well, especially considering the high potential for high bias towards defaulting to the *INSIGNIFICANT* category, as 90% of comments are labeled that way by definition. The comment text classifier, acting as our baseline model, takes a significant longer time to create overall, including preprocessing and training time. This is mainly because tf-idf vectorization is a costly task, requiring the storage and cross-reference of every unique word used. However, creating the metadata classifier is a quick task, as its features are all numeric and quick to transform.

<i>Classifier</i>	<i>Accuracy</i>	<i>Precision</i>	<i>Recall</i>	<i>F₁ score</i>
<i>Comment Text</i>	0.846	0.821	0.846	0.833
<i>Metadata</i>	0.830	0.840	0.830	0.835

<i>Popularity</i>	<i>Actual</i>	<i>Comment text</i>	<i>Metadata</i>
<i>INSIGNIFICANT</i>	17753	18334	17512
<i>NOTABLE</i>	1146	845	1291
<i>SIGNIFICANT</i>	623	413	689
<i>POPULAR</i>	178	111	200
<i>VERY POPULAR</i>	13	10	21

Figure 4. Table with the two classifiers and their respective accuracies, precisions, recalls, and F_1 score after predicting 19713 comments from the test set, and a table comparing the resultant distribution of categorizations.

Seen in Figure 4, the decision tree classifier trained off of comment data performs marginally better than the metadata classifier. However, it appears to have higher bias. Their similar performance is rather surprising, as the comment text classifier has a lot more ‘content’ to train off of. Similarity in performance between comment text classifiers and metadata classifiers is still observed even using different subreddits and timeframes.

<i>Feature</i>	<i>Importance</i>
<i>Time difference</i>	0.419
<i>Character count</i>	0.233
<i>Word count</i>	0.158
<i>Sentiment score</i>	0.131
<i>Depth</i>	0.040
<i>Gilds</i>	0.019
<i>Distinguished</i>	0.000

Figure 5. Feature importance, calculated by a normalized total reduction of criteria by feature, or equivalently the Gini Importance [9], of each feature used in the metadata classifier.

Provided the relatively high accuracy from singular decision tree models, various boosting (ensemble) methods were attempted. Interestingly, both adaptive boosting and gradient boosting led to significantly higher accuracies, but deceptively through high bias, a concern discussed earlier.

More specifically, adaptive boost and gradient boost classifiers ‘cheated’ by listing nearly every comment as *INSIGNIFICANT*. The random forest classifier, an ensemble of multiple decision trees, performed marginally worse than the standalone decision tree classifiers. Furthermore, a grid-search was performed in an attempt to find optimal hyperparameters for the decision tree. The main hyperparameters in question include the maximum depth and the impurity index used. However, the grid search, using *skicit-learn*’s implementation (see [9]), also had a strong tendency to prefer high bias in the resultant decision tree. After taking a closer look, this seems to be a result of the grid search strictly preferring the lowest max depth possible, often oversimplifying comment classification.

Although current optimization methods show an increase in bias, the two decision tree models perform comparatively well, especially considering the increased complexity of a 5-class classification problem. Lamberson et al.’s study used binary classifiers throughout their work, consisting of standalone NB and SVM classifiers, and a combination of the two. Despite simplifying the problem immensely through binary classification, their best prediction accuracy, for just two categories, barely broke 60% [5].

VI. Conclusion and Future Work

The use of decision trees and relatively good performance uncovers interesting insight. Most interestingly, the metadata classifier’s performance is very close to the baseline (comment text) classifier’s performance. This was very surprising, provided that the general assumption would be that a comment’s popularity should be based on the content itself. The metadata classifier turns this idea around, showing that sufficient performance may be attained using *only* comment metadata. In other words, one could reasonably predict a comment’s popularity without ever needing to read the comment itself; instead only basic information like team posted, overall

sentiment, comment depth, etc. is sufficient. More specifically, according to the calculated feature importance for the metadata classifier in Figure 5, the time delta between the parent and child comment has the highest importance. This means that the time difference feature, overall, allows for the highest decrease in *impurity* within the tree. This could be further interpreted as one being able to gain the most net information splitting by the time difference as opposed to any other feature overall during categorization.

Despite performing well with the tests we performed, and relative to previous comment upvote predictors in literature, there is a lot more that can be done. In future steps, we would like to:

1. Add further constraints on data to prevent high bias seen earlier with adaptive boosting and gradient boosting
2. Use models to further understanding of individual subreddits
3. Test models on data from different subreddits to see if accurate predictions corresponds with expected subreddit 'similarity'
4. Devise and test different popularity categorization criteria

VII. References

- [1] C. Buntain, J. Golbeck. *Identifying social roles in reddit using network structure*. WWW '14 Companion. 2014. doi:10.1145/2567948.2579231
- [2] S. Faraj, S. Johnson. *Network Exchange Patterns in Online Communities*. Organization Science. 2010, Dec. 29. doi:10.1287/orsc.1100.0600
- [3] C. Nguyen. (2018, May 30). *Reddit beats out Facebook to become the third-most-popular site on the web*. [Online]. Available:
www.digitaltrends.com/computing/reddit-more-popular-than-facebook-in-2018/
- [4] A. Reevesman. (2018, Dec. 31). *Predicting Reddit Comment Upvotes with Machine Learning*. [Online]. Available:
towardsdatascience.com/predicting-reddit-comment-karma-a8f570b544fc
- [5] D. Lamberson, L. Martel, S. Zheng. 2014. *Hacking the Hivemind: Predicting Comment Karma on Internet Forums*.
- [6] J. Leskovec, A. Krevl. (2014, June). *SNAP Dataset: (Stanford) Large Network Dataset Collection*. [Online]. Available: <http://snap.stanford.edu/data>
- [7] B. Boe. *PRAW: The Python Reddit API Wrapper*. (2012). [Online]. Available:
<https://github.com/praw-dev/praw/>
- [8] F. A. Nielsen. (2011, May). *A new ANEW: Evaluation of a word list for sentiment analysis in microblogs*. Proceedings of the ESW2011 Workshop on 'Making Sense of Microposts': Big things come in small packages. vol. 718, pp. 93-98.
- [9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research. vol. 12, pp. 2825-2830.
- [10] C. D. Manning, P. Raghavan, H. Schütze. "Scoring, term weighting and the vector space model", in *Introduction to Information Retrieval*. Cambridge, United Kingdom: Cambridge University Press. 2018, ch. 6, pp. 109-133.