Reddit Rating Prediction

‘We Said It, We Reddit’ : An exploration into optimal rating prediction of reddit resubmissions using sentiment analysis per community, regularised linear regression and collaborative filtering.

Nicolas Carmont Zaragoza
ncarmont@ucsd.edu

Vimanyu Saxena
vsaxena@ucsd.edu

Patrick Salmas
psalmas@ucsd.edu

Task 1:

The dataset we decided to study was Reddit Submissions. Reddit has registered members that submit content to the website, and each member has a unique username. We decided to review the dataset of Reddit Submissions various Reddit Users have created over distinct topics. Each Reddit post is associated with a time it was posted on the website, and it is presented in the dataset in three different ways which are unixtime, rawtime, and localtime. Interestingly, each post is also associated with a score which is defined as the number of downvotes that that post receives subtracted from the number of upvotes. Each submission is also associated with a community or subreddit it was posted in. Reddit users usually attach a picture and add a title to their submission that directly relates to the crux of their post. While analyzing the dataset, we found out that there are 63339 unique users, 16736 unique images, and 868 unique subreddits. As is evident from the numbers, multiple images have been resubmitted by multiple users. Another interesting finding was that 47784 users out of 63339 have just posted once. As can be seen from figure 1 and figure 2, average image scores and average total votes go down as number of resubmissions go up. Naturally, we thought original posts would have the highest rating compared to other posts with the same image that are posted later, but we found that that was not the case and it was entirely random as to which post will get the highest score or the total number of votes. Since posts across the reddit platform can be upvoted and downvoted, we created a new quantifying term called rating which was defined as the number of upvotes divided by the total number of votes that that submission receives. Because we had to come up with a predictive task, we thought that there could be a possible relation between the number of upvotes or downvotes a post gets and some of the properties of each post such as its subreddit, title, or even the time it was posted.

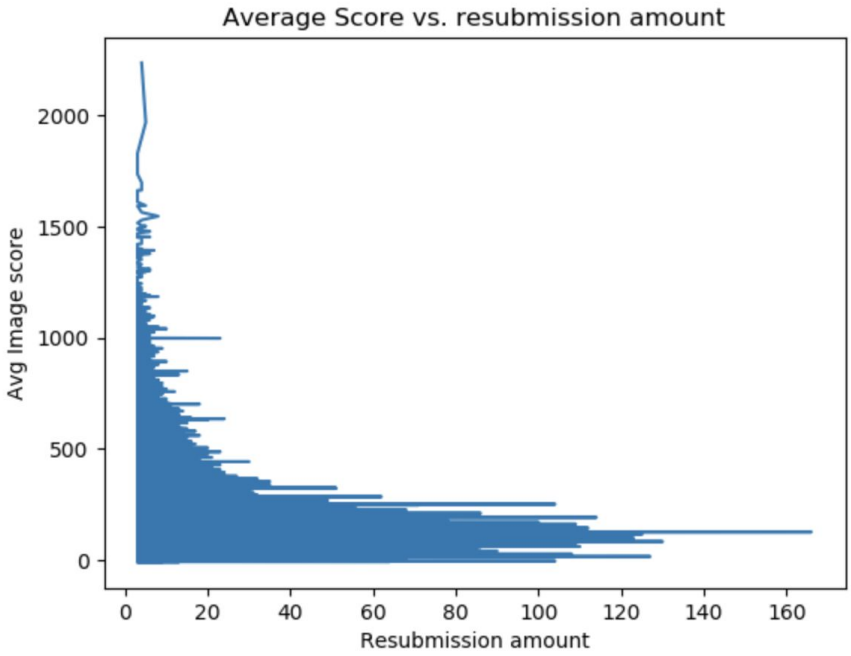


Figure 1 : Average Image Score vs Resubmission Amount

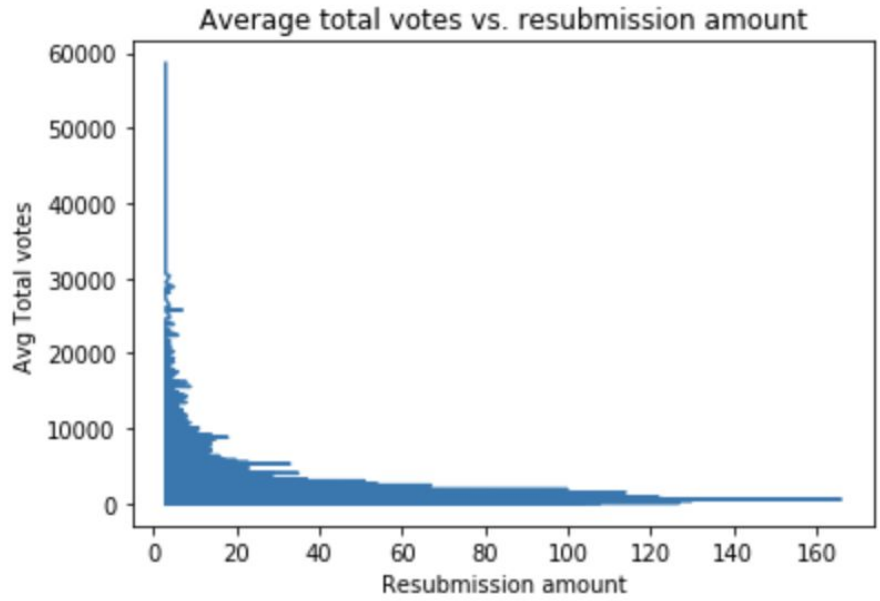
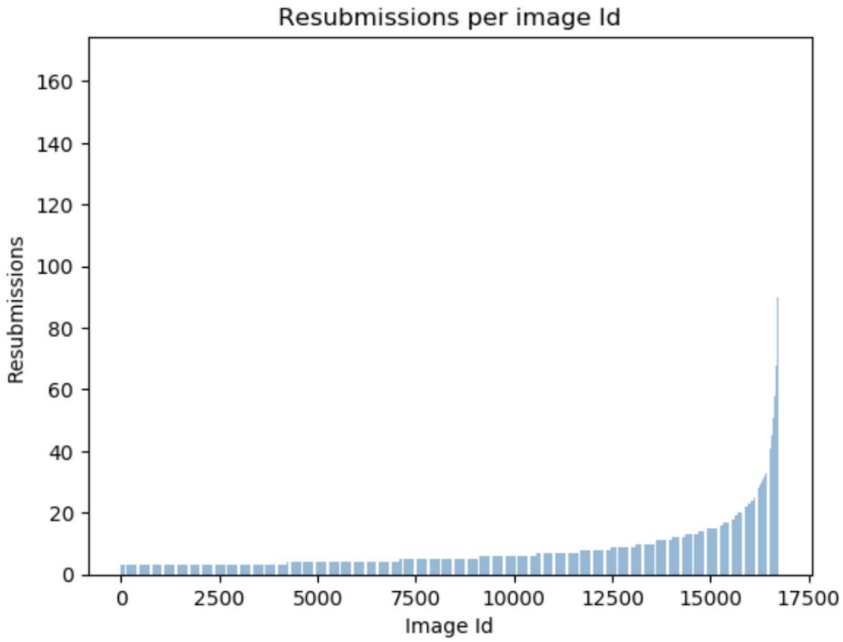


Figure 2: Average Total Votes vs Resubmission Amount




```
Image stats:
        Amount of different images: 16736

        Average amount of image resub: 7.905592734225621

        Top 10 most popular image resubs:
               1.) ImageId: 6037, resub num: 166
               2.) ImageId: 5919, resub num: 130
               3.) ImageId: 174, resub num: 127
               4.) ImageId: 6219, resub num: 125
               5.) ImageId: 996, resub num: 123
               6.) ImageId: 1757, resub num: 122
               7.) ImageId: 7344, resub num: 114
               8.) ImageId: 2285, resub num: 112
               9.) ImageId: 6233, resub num: 111
               10.) ImageId: 17378, resub num: 110
```

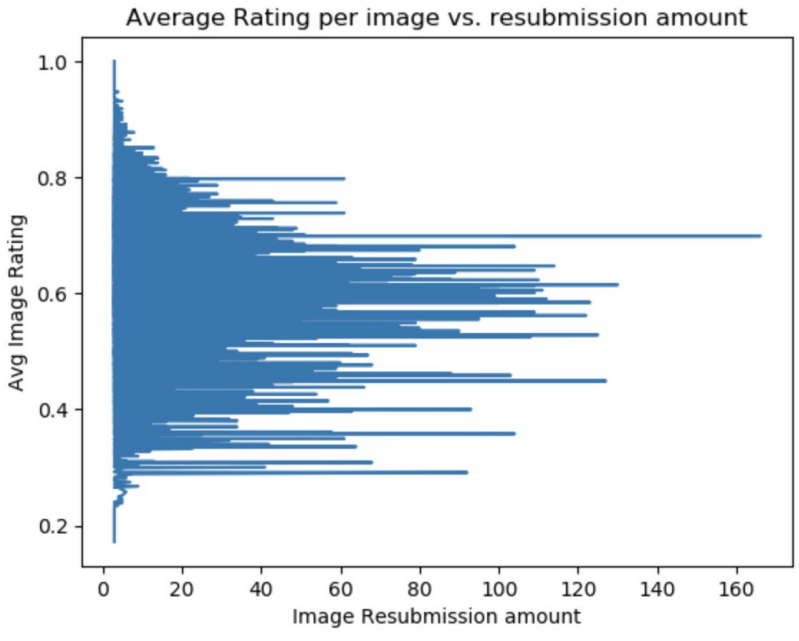


Figure 3(top): Resubmissions per Image ID
Figure 4(middle): Image Statistics
Figure 5(bottom): Average Rating per Image vs Resubmission Amount

Task 2:
Coming up with a predictive task was interesting as our first approach was to predict the score field of each submission. While analyzing models to predict scores, we figured out that score is a mathematical quantity that doesn't do complete justice in explaining the number of upvotes or downvotes of that post. It's a relative term that could have an equal value for two posts that are entirely different. So we created and switched to a different mathematical term called rating that does more justice and takes the ratio of upvotes to the total number of votes. For predicting rating, we came up with a lot of different models that could help us predict rating of each post accurately. Since rating values for the dataset are continuous, we concluded that using Mean Squared Error would be the most efficient way to evaluate our model. For baselines that predict average ratings, we calculate the average ratings associated with each of the unique subreddits. We can also have a different baseline where we calculate average ratings associated with each of the unique Reddit users and images. We can use these averages to predict for Reddit posts if the subreddits, images or users in the training set are seen again in the test set. For assessing the validity of our model's prediction, we can randomly pick 1000 Reddit submissions from the data, compare their predictions with their ratings, calculate Mean Squared Error for those 1000 Reddit submissions and compare it to our original Mean Squared Error. This approach will give us a rough estimate of how well our MSE is doing with our baselines and how we should change them to get to better results. We used quite a few models such as Linear Regression, SVM, Cosine Similarities, and Sentiment Analysis to work with our baselines defined above. We realized that a combination of these models would help us achieve more optimal results. For this dataset, a combination of Linear Regression and Cosine Similarity is being used as our model where Cosine Similarity between a post's user and other similar users is one of our features in linear regression. This is more explicitly described in the next task(Task 3).

For features, we tried a lot of approaches to get the best features to predict rating.
1) Title Prediction or Top Words Features: We performed sentiment analysis over 1000 most popular words and scored titles in the same manner as the reviews in homework 3.

2) Average Rating Per Word Per Community: This is also related to title. For all the words in title, we decided to keep track of all the average ratings per word per community. To process data for this, we looped through the data, extracted words from title of each of the posts, calculated that post's rating and added it towards that word's average rating in that community. For example, if a post's subreddit/community is 'funny' and that post's title is 'troll cat'. Let's say the rating for that post is 'x'. Then we will add 'x' to the average rating of 'troll' and 'cat' under the 'funny' subreddit. These words could appear in a different community, and they will have different averages according to that community.

3) The penultimate feature is the cosine similarity of users where we calculate the weighted average rating depending on the user's cosine similarity with users who submitted the same image. To process data for this, we loop through the data and make two dictionaries. One that maps all users to images and one that maps all images to users. This way we can efficiently create a map between users who posted the same image. We need two dictionaries because having one dictionary that maps users to users takes very long to loop through. Now that we have an efficient map between users. We calculate average similarity between similar users and use that as a feature in our feature matrix for each post using their username field.

4) Another feature is the average rating per hour feature which returns the average rating of all posts submitted at a certain local time (in hours). Similar to what we have done before, for each post, we parse the hour from the rawtime field and map each hour to the average rating associated with it. We calculate average rating by keeping track of the number of times each particular 'hour' has shown up in our training set and using that to average out the sum of all the ratings each 'hour' is associated with.

5) Baseline Feature Average Community Rating: This is described as the average rating associated with each unique community. To get this data, we loop through the training set and map each unique community to their average rating.

6) Baseline Feature Average User Rating: similar to average community rating, but it's for each unique user instead community. Each user is identified from the 'username' field of each post.

7) Baseline Feature Average Image Rating: Calculating average rating associated with each unique image. Each image is identified from the '#image_id' field of each post.

```
Top average hour ratings:
0.) At time 9h average Rating is 0.007927519818799546
1.) At time 12h average Rating is 0.008141592920353982
2.) At time 11h average Rating is 0.008665511265164644
3.) At time 8h average Rating is 0.01057340382269215
4.) At time 7h average Rating is 0.011627906976744186
5.) At time 3h average Rating is 0.01192504258943782
6.) At time 14h average Rating is 0.012059663598857505
7.) At time 5h average Rating is 0.012493492972410203
8.) At time 19h average Rating is 0.012899607403252944
9.) At time 10h average Rating is 0.013342949873782907
10.) At time 6h average Rating is 0.01405152224824356
11.) At time 1h average Rating is 0.014292520247737018
12.) At time 13h average Rating is 0.014323784143904063
13.) At time 0h average Rating is 0.014565126924677487
14.) At time 16h average Rating is 0.014841199168892847
15.) At time 20h average Rating is 0.014978213507625272
16.) At time 23h average Rating is 0.015275707898658718
17.) At time 15h average Rating is 0.015492546039169833
18.) At time 18h average Rating is 0.015557476231633534
19.) At time 21h average Rating is 0.015588235294117648
20.) At time 17h average Rating is 0.016354445435622957
21.) At time 22h average Rating is 0.016526138279932545
22.) At time 2h average Rating is 0.018257694314032343
23.) At time 4h average Rating is 0.02472952086553323
```

Figure 6: Top Average Hour Ratings

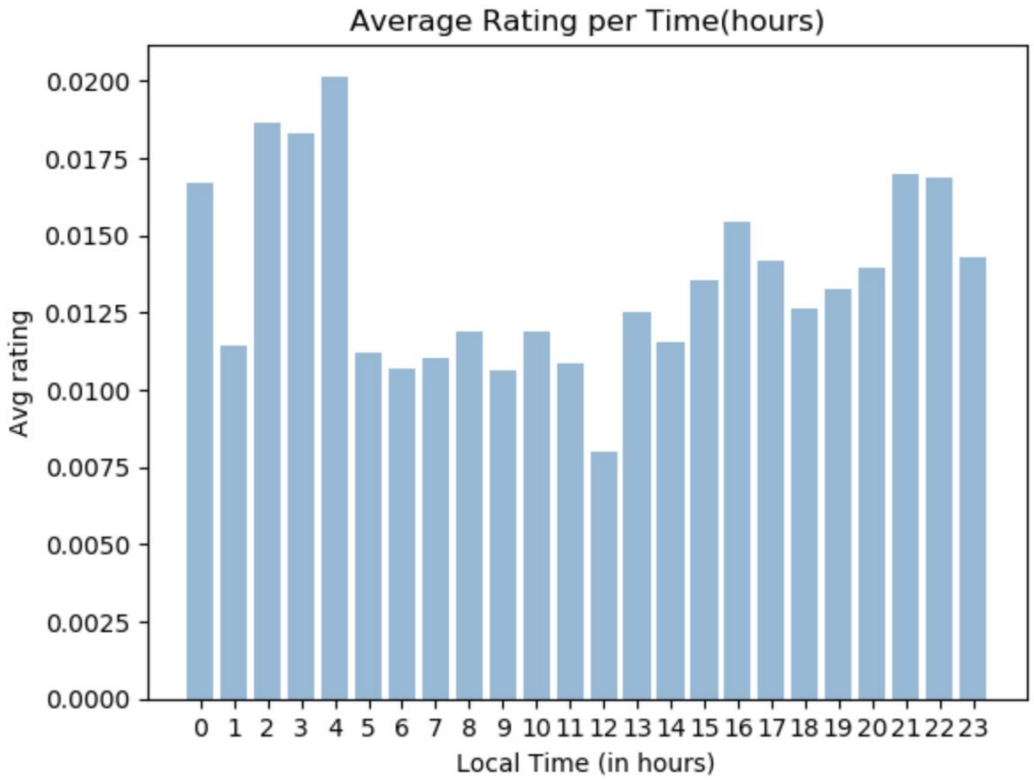


Figure 7: Average Rating Per Time(Hours)

Task 3:

For predicting rating, our most successful model utilized linear regression, combined with other models used for particular features. Some assisting models we used were sentiment over words in submission titles and cosine similarity. Our best linear model consisted of the following feature vector:

[1, top words that appear in title, average rating of words in title corresponding to community, cosine similarity of users, average rating per hour]

We chose the above model for our task because we found it be a strong combination of powerful features. In fact, many of our “failures”, which are described below, aided us in arriving to our final classifier.

Initially there was good reason to believe that a user-user collaborative filtering model was suitable for predicting ratings. This was because our data analysis in part 1 revealed there was high amount of Image IDs submitted by a small amount of users. These niche groups were beneficial for distinguishing between users. Additionally, there was a small amount of Image IDs with very high resubmissions. This provided an appropriate balance between users sharing at least one common image and users sharing similar niche images.

To implement user-similarity we used weighted cosine similarity because it was optimised for continuous predictive tasks, accounted for the similarity weight of users and evaluated user similarity based on the closeness of ratings (not just presence).

Analysing the model, we achieved a low MSE of 0.07410284432886191 on training data, which suggested that user-similarity was a good predictor on seen data and distinguished seen users correctly. Nonetheless, on training data the MSE was much higher at 0.41651326279032924.This was mainly due to the many cold cases encountered with unseen users and images in the sparse test data.

On evaluating the weighted cosine similarity model we found that, due to the sparseness of resubmission data, unseen users and images were not being predicted accurately. To improve on the cold cases we combined the cosine similarity model with a content-based method.

> When unseen users were encountered the average image rating was returned, when an unseen image was encountered the average user rating was returned and when both were unseen the overall average was returned.

Analysing this model, it seemed to perform considerably well on both seen training data (MSE = 0.07410284432886191) and even better on unseen test data (MSE = 0.034245667746484827) .This model appropriately combined the benefits of both collaborative filtering for user preferences and a simple average aggregate for cold cases.

Whilst certainly an improvement, the weighted cosine similarity model wasn't an optimal model alone for accurately predicting submission rating. This was seen as in comparison the cold case aggregate seemed to improve the cosine similarity MSE considerably. This may be due to the sparseness of the reddit resubmission data. Overall we found user-similarity alone was not optimal for predicting rating, but was more when powerful combined with other independent features such as title sentiment analysis and time of day in a linear regressor to improve accuracy.

SVM was unsuccessful in the sense of implementation. We were trying to train a support vector machine to effectively classify a submission as either "well received" or "not well received". "Well received" means above 50% rating, and "not well received" ratings are 50% and below. It became clear that this model yielded little information about the Reddit data we were analyzing. Even if the this classifier is trained very well, we learn little from an image surpassing some threshold. Perhaps SVM could be useful for predicting which submissions fall below are particular threshold and could thusly be considered too offensive to post on Reddit, but this is not a measure or scenario that we are trying to predict.

The last model we used was sentiment analysis over the words in the titles the submissions. Although we consider this model to be an unsuccessful attempt, we learned much more from this approach than any of the previously mentioned "failures". We discovered that analysis of the titles of submissions was actually more important than the images themselves. We found this to be quite shocking since we assumed that image itself would be the most important factor of a submission. This led us to realize that the description, associated with an image, is truly what makes a submission funny, cool, or even interesting. This is something that we perhaps all recognize as being true, but intuitively is not what most people would assume.

These unsuccessful models helped us realize more intelligent approaches. The results from sentiment analysis prompted use it as a feature in our final linear regression model. Sentiment analysis as well inspired the feature that measured average rating of words in title, corresponding to subreddits/communities. We as well learned that the user itself was not very important, and that the image itself not as important as one would assume, but still somewhat useful measure.

Task 4:

The data set utilised for our rating prediction task was the Reddit submissions data set (Himabindu Lakkaraju, Julian McAuley, Jure Leskovec *ICWSM*, 2013). This data focused on resubmitted reddit images specifically with 132,308 total image of which 16,736 were unique. The feature metadata within this set included timestamps (such as date and time), upvotes, downvotes, post title and subreddit (topic category).

As described, the data set comes from a similar paper by McAuley et al. on understanding the interplay between titles, content and communities in reddit resubmissions on predicting reddit submission score. In our investigation we attempted to predict for submission rating rather than score. We thought this was an interesting approach because score itself may not do complete justice as an optimal predictor for reddit submission receival as it is influenced by both the upvote-downvote ratio (or rating) and the total votes magnitude. These two compounded quantities mean it can be difficult sometimes to specifically account for which of the two quantities influences score the most and whether they are constructively or destructively interfering with each other.

In this investigation, we attempted to try and understand rating specifically as a measure of receival and determine what the optimal model for predicting for rating was to hence compare whether this is divergent to the compounded measure of score prediction.

The state-of-the-art techniques thoroughly tested throughout our exploration include content-based recommendation using linear regression, user-based collaborative filtering using cosine similarity, weighted cosine similarity, improved cold-cases using aggregate prediction, SVMs, stopword removal, sentiment analysis by community, and lambda regularisation for overfitting. Our model further explored the use of a final linear regressor which combined successful features and past models such as weighted cosine similarity with theta values relative to the model's performance on training data.

Our findings show that the most successful techniques include sentiment analysis by community, content-based recommendation using linear regression, user-based collaborative filtering using weighted cosine similarity and our final experimental method of using a linear regressor to combine successful models. Some of the results achieved are similar to the paper by McAuley et al. as the model's most successful linear content-based features also include time of day, sentiment analysis of words by community and combining a community model to account for distinct non-language features. Nonetheless, our findings also differ in the sense from their model by employing an experimental technique of combining successful models based on their accuracy which seemed to work effectively and also the use of collaborative filtering using weighted cosine similarity. Whilst our model worked effectively with low MSE, further explorations may investigate the use application of PCA dimensionality reduction to account for potential overlap between our successful features. Furthermore, further exploration into the timing of submissions as explored by McAuley et al. and titles could possibly provide and improvement to the current model.

Other literature attempting to understand similar online content receival includes "Predicting the Popularity of Online Content" (sZabo and Huberman) which details the popularity of posts in similar content sites like Youtube and Digg. The paper similarly finds that time of day is an powerful feature in trying to predict content receival as users tend to be more active online at certain times. Nonetheless the paper also explores the potential feature of days of the week as a strong predictor for content receival, while also the correlation between the initial popularity of

a post and it's overall later rating depending on the nature of the content site. These could make interesting further additions to our model if initial popularity timeframe were to be included as an additional field to the existing reddit data set.

Additionally a similar paper attempting to predict reddit post popularity (Segal and Zamoshchin) similarly found sentiment analysis by community to be a strong indicator for popularity. Our model's use of collaborative filtering with weighted cosine similarity provides an interesting contrast to their approach and a combination between methods. Nonetheless, the use of Tf-idf and bigram sentiment analysis could provide and interesting additional to our explored model.

Task 5:

Results/conclusions:

We concluded that the combination of the most "potent" features, and applying them to linear regression techniques, led to the strongest predictors. We ended up with the following MSE values for our training and test data (random splits of the original data set) for our best linear regressor:

Best feature representation used for linear regression:
[1, top words that appear in title, average rating of words in title corresponding to community, average rating based cosine similarity of users, average rating per hour]

Theta: [0.6315751, 0.1776696, 0.199344, 0.1222177, -0.0386619]

MSE on Train: .02544299
MSE on Test: .02648414

The MSE values from the above linear model, vastly out performed some of our more simple baselines. Our baselines consisted of very basic features, such as average user rating, image average rating, and community average rating. These more naive linear regressors led to much higher MSE rates over the test data. Below are the results from one of our baselines (feature rep: [1, community average rating, user average rating, image average rating]).

Theta: [-0.253089, .1343802, .757587, .5140039]

MSE on Train: 0.011354
MSE on Test: 0.194484

We expected our best model to outperform the more simple ones, but we as well discovered which features were less important, and which features made our model more susceptible to overfitting. We found that community average rating was far too simple to be truly predictive. We also found that more simple features, in general, led to higher rates of overfitting. Even when

using a regularized linear model and adjusting lambda values, we still found this to be the case. Though it should be noted that small lambda values with simple features led to the highest rates of overtraining.

[1, top words that appear in title, average rating of words in title corresponding to community, average rating based cosine similarity of users, average rating per hour]

Theta: [0.6315751, 0.1776696, 0.199344, 0.1222177, -0.0386619]

From analyzing the theta values of our predictor, we were able to determine that all of the above features held relatively high weights. Considering the value range of what we are predicting, and the range of values used for our various features, we found ourselves satisfied with our theta values. We discovered that our most powerful features utilized cosine similarity. The next two most important features both related to the sentiment analysis of submission titles. They we realized in hindsight that perhaps these two features may have caused a degree of overlap between them. If given additional time, we would have considered using PCA to figure out if there were in fact redundant features being used.

We as well tried many other feature representations, some of which were more effective than others. Below are several examples:

- [1, community average rating, user average rating, image average rating]
- [1, community average rating, user average rating, image average rating, month of the submission]
- [1, image average rating corresponding community of post, user average rating, community average rating]
- [1, community average rate, average rating of words in title corresponding to community]
- [1, top words that appear in title]
- [1 , top words that appear in title, image average rating]

Using the above feature representations for linear regression, along with several others variations, we were able to recognize which features were most useful. We learned earlier that performing sentiment analysis over the titles was particularly powerful, so it quickly became clear that this was an important feature. It ended up being vastly stronger than features like community average rating, image average rating, and image average rating corresponding to community.

Experimenting with various feature representations led us to as well understand which features were considerably weak. Unsurprisingly, some of the more naive features, like user average rating, community average rating, and image average rating, were less informative for making accurate predictions. It was apparent that these feature were too simple to yield any useful information. Measuring image average rating, according to community, was another feature that

we considered weak. From analyzing theta values when using this feature we noticed that it consistently held little weight in the overall prediction. This finding opposed our initial expectation. We thought that this feature would account for the submissions that were purposefully miscategorized, but oddly it was not the case.

We believe our model is more successful than various other attempted models because the others were far too simple. We believed that using linear regression in tandem with other models accounted for most of the relevant data. We were also careful to not overcomplicate our final feature representation. Overly complex feature representations led to worse results when measuring MSE.

Sources:

Lakkaraju, H., McAuley, J., & Leskovec, J. (2013). What's in a name? Understanding the Interplay between Titles, Content, and Communities in Social Media(Master's thesis, Stanford University, 2013) (pp. 1-10). Stanford University.

Segall, J., & Zamoshchin, A. (n.d.). PREDICTING REDDIT POST POPULARITY(Master's thesis, Stanford) (pp. 1-5). Stanford.

Szabo, G., & Huberman, B. A. (n.d.). Predicting the Popularity of Online Content(Master's thesis, HPL) (pp. 1-9). HPL. doi:10.1145/1787234.1787254

Appendix:

IMPLEMENTATION OF WEIGHTED COSINE SIMILARITY

```
userCosine = defaultdict(dict)
for user in userSim:
    if(user):
        for u2 in userSim[user]:
            dotProd = 0
            if(u2):
                for i in usersImageSubs[user]:
                    if (i in usersImageSubs[u2]):
                        dotProd+= ratings[user][i]*ratings[u2][i]
                userCosine[user][u2] = dotProd/(math.sqrt(len(usersImageSubs[user]))*math.sqrt(len(usersImageSubs[u2])))


def weightedUserSimScore(user, img):
    rating = 0
    if((user in users) & (img in images)):
        weightedSum = 0
        sumSim = 0
        for (u,v) in userCosine[user].items():
            if(img in usersImageSubs[u]):
                weightedSum+= ratings[u][img]*v
                sumSim+=v
        if(sumSim==0):
            sumSim = 1
        return (weightedSum/sumSim)
    elif (img in images):
        return avgImageRating[img]
    elif (user in users):
        return avgUserRating[user]
    else:
        return avgRating
```

IMPLEMENTATION OF TIME-BASED FEATURE PREDICTION

```
avgRating.sort(key=lambda tup: tup[1])
avgRating.reverse()
avgRatingDict = dict(avgRating)

def avgHourRating(ISOtime):
    feat =[1]
    if ISOtime is None:
        feat.append(0)
    else:
        hour =convertToHour(ISOtime)
        totRatings =dict(ratingPerHour)
        totRating = int(totRatings[hour])
        feat.append(totRating)
    return feat

X = [avgHourRating(d['rawtime']) for d in data]
y = [returnRating(d['number_of_upvotes'],d['total_votes']) for d in data]

halfLenData = int(len(data)/2)
X_train = X[:halfLenData]
y_train = y[:halfLenData]

X_test = X[halfLenData:]
y_test = y[halfLenData:]

theta1,residuals,rank,s = np.linalg.lstsq(X_train, y_train,rcond= -1)
predictionsTrn = [sum(x*theta1) for x in X_train]
predictionsTst = [sum(x*theta1) for x in X_test]
```

Average Rating Per Word Per Community

```
averageRatingPerWordPerCommunity = defaultdict(lambda: defaultdict(int))
sizeOfWord = defaultdict(lambda: defaultdict(int))
for d in training:
    try:
        title = ''.join([c for c in d['title'].lower() if not c in punctuation])
        list = title.split(" ")
        rating = int(d['number_of_upvotes'])/(int(d['number_of_upvotes']) + int(d['number_of_downvotes']))
        for w in list:
            if(w not in stop_words):
                if(sizeOfWord[d['subreddit']][w] == 0):
                    averageRatingPerWordPerCommunity[d['subreddit']][w] = rating
                else:
                    averageRatingPerWordPerCommunity[d['subreddit']][w] = (averageRatingPerWordPerCommunity[d['subreddit']][w]*
                                                                           sizeOfWord[d['subreddit']][w]) + rating

                sizeOfWord[d['subreddit']][w] += 1
                averageRatingPerWordPerCommunity[d['subreddit']][w] = averageRatingPerWordPerCommunity[d['subreddit']][w]/
                                                                      sizeOfWord[d['subreddit']][w]
    except:
        continue
```