Subreddit Recommendations within Reddit Communities

Vishnu Sundaresan, Irving Hsu, Daryl Chang

Stanford University, Department of Computer Science

ABSTRACT: We describe the creation of a recommendation system for *Reddit*, a social news and entertainment site where community members can submit content in the form of text posts, links, or images. Our dataset consists of 23,091,688 votes from 43,976 users over 3,436,063 links in 11,675 subreddits. Using this network, we constructed a weighted graph of subreddits, partitioned it into 217 distinct subcommunities, and created both an item-based and user-based recommendation algorithm. Given a user and a subreddit, our algorithm recommends to the user novel subreddits within the same subcommunity. User-based recommendation was found to outperform item-based recommendation.

1. INTRODUCTION

Reddit is a social news website where users can submit links to content on the web. These links can then be upvoted or downvoted by other users, influencing the prominence of the posts. Reddit entries are organized into *subreddits*, which are custom-made subforums for specific areas of interest.

When viewing a subreddit today, a user is encapsulated in a particular subforum with no links or recommendations to similar content. In a world where suggestions and links to more information are almost essential to growth and discovery, it is astounding that Reddit (a site with over 5 million hits a day and 174 million unique visitors a month) does not have a tool to connect its communities.

We will be creating a recommendation feature for Reddit, using data to create a 3-step process that will be able to take a user and give suggestions as to other related communities based on general user behavior. The first step will involve creating a large weighted graph with edges between different subreddits, then clustering these nodes and categorizing them to separate communities, and finally classifying a user under one of these communities and giving them recommendations to other subreddits that they are likely to be interested in within that community.

2. PRIOR WORK

2.1 "Navigating the massive world of Reddit: Using Backbone Networks to Map User Interests in Social Media" (Olson and Neal)

Olson and Neal [1] showed that techniques for network backbone extraction and community detection can be used to map and navigate interest networks in social media. This ultimately facilitates the organization of users into specific interest groups. In this work, an interest map was built for Reddit, using a dataset of over 875,000 active users that post to over 15,000 distinct subreddits. The interest map was built by viewing the Reddit network as a hierarchical map of all user interests in the social network. Using the backbone extraction algorithm, the network was divided into 59 *interest meta-communities*, or groups containing similar interests. The backbone extraction algorithm preserves edges whose weight is statistically significant compared to a null model where edges are assigned weights uniformly at random.

An extension to the backbone algorithm is used in our work to first create a weighted graph of subreddits. This allows us to restrict the recommended subreddits to the community of the user's current subreddit. Furthermore, a shortcoming to this work is that the graph is still not divided into distinct clusters of communities that are related. This can be achieved by applying the Louvain algorithm to cluster the graph recursively, and is discussed in the algorithms section of this paper. The output of the above algorithm will result in several subgraphs containing subreddits in the same community, which will then be used to make recommendations for a user inside one of these communities.

2.2 "Finding and Evaluating Community Structure in Networks" (Newman, Girvan)

Newman and Girvan describe several algorithms for determining the number of communities, as well as the structure [2]. Each method, however, utilizes a basic approach of removing an edge that has the highest likelihood of being an inter-community edge, and then recalculating the next highest such edge. The three methods proposed are shortest-path betweenness, which determines the most used edges through the number of shortest paths passing through each edge; resistor networks, which ranks edges according to the minimum potential lost over each edge when treating the graph as a circuit; and random walks, which, similar to shortest-path, determines which edges are passed over the most on a random walk between two nodes.

One shortcoming of these algorithms, however, is the assumption that all edges are unweighted. In our graph, it is clear that edges with less weight, in comparison to other outgoing edges for that node, are more likely to be edges connecting communities, as opposed to connecting nodes within the community. This is an aspect that can potentially increase the performance of our algorithm from O(mn), where m is the number of edges, and n is the number of nodes in the subreddit graph. However, the main purpose of this algorithm to create clusters of subreddit is to improve the performance of the recommendation algorithm described in Section 5.

2.3 "Evaluation of Item-Based Top-*N* Recommendation Algorithms" (Karypis)

User-based and item-based algorithms are two popular methods to make recommendations within a bipartite graph. User-based algorithms first identify the k most similar users to the active user, and then recommend the N items most frequently used by those users. In contrast, item-based algorithms first compute item-to-item similarities, and then recommend the N items most similar to the active user's existing items. Karypis demonstrated that item-based algorithms outperform user-based ones in both performance and recommendation accuracy [3]. Additionally, unlike user algorithms, item-based algorithms can be used when the historical information for the user is limited.

However, Karypis' item-based algorithm suffers from two main weaknesses. First, it has O(m^2n) runtime, where m is the number of items and n is the number of users, since $m(m\text{-}1)$ similarities need to be calculated, each with up to n computations. Since Karypis used a sparse dataset, the *de facto* runtime was reduced by using sparse data structures and only computing similarities between items that shared a user. Still, this optimization may not scale to extremely large datasets. Second, item-based algorithms may not provide truly personalized recommendations, as user-based algorithms do. For example, an active user who exhibits behavior similar to that of an extremely small subset of users would receive more personalized recommendations with a user-based algorithm.

Item-based algorithms may be improved by incorporating aspects of user-based algorithms. Karypis suggests first identifying a large neighborhood of similar users and then computing item similarities based on this subset of users. Reducing to a subset of similar users would combine the personalization of user-based recommendation algorithms with the performance of item-based ones, and would further improve runtime by making the item vectors more sparse.

3. PROBLEM STATEMENT

On a high level, we would like to provide relevant subreddit recommendations for a user on a given subreddit. Going into specifics on what recommendations we are serving, imagine you are a user on a specific subreddit, i.e., Stanford. You want to get recommendations for other subreddits similar to Stanford, such as StanfordCardinal or StanfordHousing. To do so, we will take advantage of previous user data from yourself in that subreddit and create a subset of users with similar interests.

4. DATA SET

We have a public dataset of reddit votes that consists of 23 million votes from 44,000 users over 3.4 million links in 11,675 reddits. This implies an average of 525 votes per user, which would be sufficient to determine user-to-user similarity in the user-based part of our algorithm, which is described in more detail below. Each vote can be an upvote or downvote and is associated with a user, link, and subreddit.

5. ALGORITHM

In order to restrict the recommended subreddits to the community of the user's current subreddit, we first create a weighted graph of subreddits [1]. To create the weighted graph, we use the data set above without the column on voting activity, removing any duplicate (user, subreddit, link) entries as necessary. This results in there being only unique (user, subreddit) pairs. Weighted edges are then created between subreddits, with each edge (*u*, *v*) having a weight equal to the number of shared users between subreddits *u* and *v*.

The number of edges in the weighted subreddit network is then reduced to the edges that represented a significant user overlap between two subreddits. This is done by making the original undirected graph into a directed graph, with two edges for each original edge. Then, each edge has its weight set to the percentage of users that the source node subreddit shares with another subreddit. If the weights of both directed edges in a pair of nodes fall below a certain threshold α (5×10^{-4}), we remove the edges from the graph.

Note that at this point, the weights in the graph from [1] are no longer used. Once we have created this graph of all subreddits with edges linking the related ones, we apply the Louvain algorithm to cluster the graph into distinct communities, such as NBA, NHL, NFL, and MLB. The Louvain method optimizes the Modularity Q of partitions obtained, where Q is a scalar value between -1 and 1 that measures the density of links inside communities as compared to links between communities [5]. Specifically, modularity is defined as

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{i,j} - \frac{k_i k_j}{2m}\right]\delta(c_i c_j), \quad (1)$$

where $A_{i,j}$ represents the edge weight between nodes *i* and *j*, $k_i = \sum_j A_{i,j}$ is the sum of the edge weights with *i* as one of the vertices, $m = \frac{1}{2}\sum_{i,j} A_{i,j}$ is the total number of edges in the graph, c_i is the community that node *i* is assigned to, and δ is a delta function where $\delta(c_i c_j)$ is 1 if *u* = *v* and 0 otherwise.

The Louvain method attempts to optimize the modularity of a partition by iteratively repeating two phases. First, each node in the network is assigned to its own community. Then, for each node *i*, each neighbor *j* of *i* is considered and the change in modularity that would take place by removing *i* from its community and by placing it in the community of *j* is calculated.

The change in modularity resulting from moving an isolated node i into a community C is computed by:

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right], \quad (2)$$

where $\sum_{in}$ is the sum of all the weights of the links inside C, $\sum_{tot}$ is the sum of all the weights of links incident to the nodes in C, $k_{i,in}$ is the sum of all the weights of the links from i to the nodes in C, k_i is the sum of all the weights of the links incident to node i, and m is the sum of the weights of all the links in the network. This process is applied repeatedly and sequentially to all nodes until no modularity increase can occur.

In the second phase, the algorithm aggregates all of the nodes in the same community and builds a new network whose nodes are the communities found from the first phase. These two steps are repeated iteratively until a maximum network modularity is obtained.

The process of obtaining the best partition is provided by the *best_partition* method in community.py. Our algorithm then continues to recursively cluster these communities until we reach a set of communities each of a reasonable size. This is meant to improve the performance of the *k*-nearest neighbors algorithm described in [3] by drastically reducing the number of nodes on which to compare, as well as prevent an unrelated subreddit that has common users from being incorrectly recommended. The process of clustering the graph into subgraphs was done in partition.py.

After the initial partition, we put it into a queue, which then enters a while loop until nothing remains. At each stage in the loop, we pop the top graph from the queue, partition it, discard any extremely small partitions (under a threshold of 10 nodes), and add any partitions of size less than 100 (or that cannot be further partitioned). The remaining partitions are then converted into subgraphs, and put in the back of the queue to be later re-partitioned.

ALGORITHM 1: Graph Partitioning Algorithm

```
Input: Graph $G$
Output: Final list of all clusters $F$
1: Queue $Q \leftarrow \varnothing$
2: $Q.Enqueue(G)$
3: while ¬$Q.IsEmpty()$ do
4:    current subgraph $c \leftarrow Q.Dequeue()$
5:    $P \leftarrow community.GetBestPartitionLouvain(c)$
6:    $clusters \leftarrow \{\varnothing\}$
7:    for each subreddit $s \in P$ do
8:       $clusterID \leftarrow P[s]$
9:       if $clusterID$ is not in $clusters$ then
10:          $clusters[clusterID] \leftarrow [s]$
11:      else
12:         append $s$ to $clusters[clusterID]$
13:         continue
14:      endif
15:   endfor
16:   if only 1 cluster in $clusters$ then
17:      append $c$ to $F$
18:      continue
19:   endif
20:   for each $clusterID$ in $clusters$ do
21:      $subgraph \leftarrow$ all nodes in $clusterID$
22:      if $subgraph$ has less than 10 subreddits
23:         continue
24:      else if $subgraph$ has less than 100
25:         append $subgraph$ to $F$ (size
26:      else
27:         $Q.Enqueue(subgraph)$
28:      endif
29:   endfor
30: endwhile
```

The maximum partition size of 100 was picked because it was on the order of $\sqrt{N}$, (where N is the original graph size), which is meant to reduce the complexity of the *k*-nearest neighbors algorithm for our recommendation aspect. After running the clustering, we discarded 1395 nodes due to extremely small partitions, and had a total of 217 separate subgraphs each ranging from 10 to 176 subreddits each.

Overall, the average subgraph size was found to be 46.793, and the most common size was 12 nodes, with 10 clusters. The distribution of cluster sizes is shown in Figure 1.

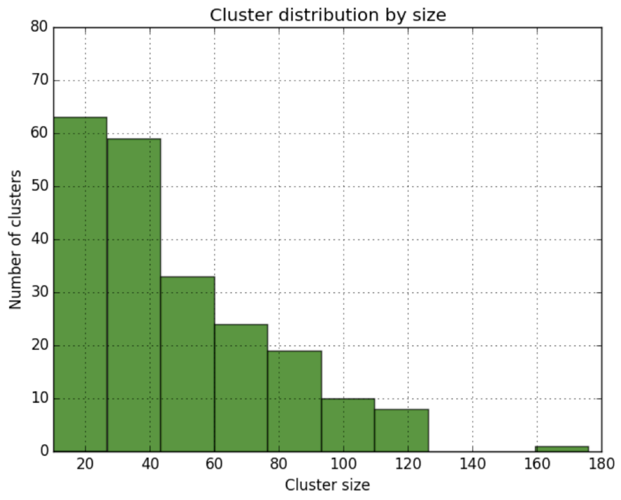

Figure 1. Histogram of network cluster sizes.

The output of the partitioning algorithm resulted in several subgraphs containing subreddits in the same community, which was used to make recommendations for a user inside one of these communities. An example cluster of size 46 is shown in Figure 2.

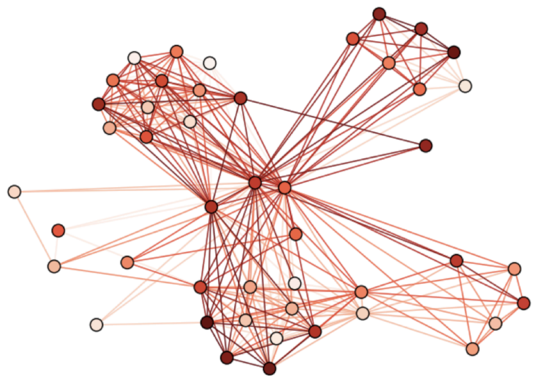

Figure 2. Visualization of sample reddit subgraph obtained after running partitioning algorithm.

Because the subreddit IDs are given as salted hashes in the dataset, the individual nodes are not labeled in the example subgraph. The nodes are positioned using a spring layout, and thus variations in the edge lengths between adjacent nodes can be disregarded. Additionally, the intensities of the node colors are independent of the node degrees, and correspond to a color mapping that is cycled through when plotting the graph to allow for better readability.

We created a baseline recommendation algorithm in order to gauge the performance and accuracy gains from later improvements to the algorithm. The baseline is an item-based recommendation algorithm that calculates the similarity between the current subreddit S and each subreddit candidate C in the subreddit community of S. The similarity can be calculated using any of the methods used by Karypis [3], including cosine similarity and conditional probability. After calculating similarities between S and each candidate C, the algorithm takes the top 3 subreddits as recommendations. The runtime of item-based similarity is thus reduced from $O(m^2n)$ to $O(c^2n)$, where m is the number of total subreddits, n is the number of users, and c is the number of subreddits in a cluster. In this case, the runtime was decreased by approximately six orders of magnitude, since $m = 66{,}000$ and $c \leq 100$.

Next, we created a user-based recommendation algorithm for comparison with the baseline. The user-based algorithm only considers the set of users U that have been active in the current subreddit S. It then vectorizes each user u in U as a feature vector representing that user's voting history within subreddit S. Specifically, we keep track of how many times each post (represented as a link in the dataset) that user u has voted on.

The data structure used was a nested dictionary D mapping from users to dictionaries of (subreddit, link) key-value pairs, where

D[*u*][*s*][*l*] gives how many times user *u* has voted on link *l* in subreddit *s* (Figure 3).

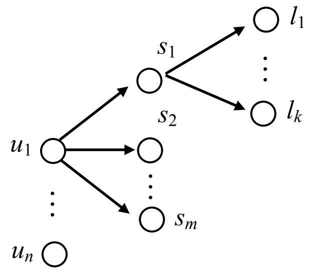

Figure 3. Schematic of the dictionary mapping from users, to subreddits, to links (posts). In the example above, there are *n* users, *m* subreddits, and *k* links.

Then, for all other users *u'* in subreddit *s*, the algorithm computes the cosine similarity between the vectors *D*[*u*][*s*] and *D*[*u'*][*s*] to find the top *k* most similar users to the current user. For two vectors $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$, the cosine similarity is given by:

$$sim = \cos\left(\overrightarrow{v_1}, \overrightarrow{v_2}\right) = \frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{\left\|\overrightarrow{v_1}\right\|_2 \left\|\overrightarrow{v_2}\right\|_2} \qquad (3)$$

Let *c* be the cluster ID of the cluster that subreddit *s* is located in. A dictionary, *commonSubs*, is then used to keep track of the most common subreddits within *c*. Here, the most common subreddits are defined as subreddits that are shared by the most users (within the *k* similar users). To populate *commonSubs*, for each of the *k* most similar users k_i, we go through each of its subreddits s_j and increment *commonSubs*[k_i][s_j] if s_j is part of *c*. The top *n* most common subreddits in *commonSubs* are then recommended. In this way, we reduce computation time of user-based recommendations by reducing both the number of users to be compared and the dimensionality of the user feature vectors.

6. EVALUATION METRIC

In order to evaluate the relevance of the recommendations, we tested our algorithms on *power users*, defined as users who have voted in at least 30 posts within each of at least 40 subreddits. These threshold values are called the *post threshold* and *subreddit threshold*, respectively. The threshold values were determined after running preliminary tests that first varied the number of posts μ and then the number of subreddits η, and recording the values of μ and η that maximized the precision. For a given user *u*, the precision *P* is defined as the proportion of recommendations that are actually subreddits in which the user is active.

$$P = \frac{\left|\left\{recommended\ subs\right\} \cap \left\{subs\ of\ user\ u\right\}\right|}{\left|\left\{subs\ of\ user\ u\right\}\right|} \qquad (4)$$

For a given power user, we randomly chose an active subreddit and used the item-based and user-based recommendation algorithms to make recommendations. Then, we computed the precision of the recommendations.

We also created an interface to run experiments on the user-based algorithm. These experiments involved varying parameters such as the number of similar users *k* that we get recommendations from, along with the percentage of similar users that must be active in a subreddit candidate *C* in order to recommend that subreddit.

7. RESULTS AND IMPLICATIONS

7.1 ITEM-BASED RECOMMENDATION

The baseline item-based recommendation achieved 53.6% precision when making 2 recommendations.

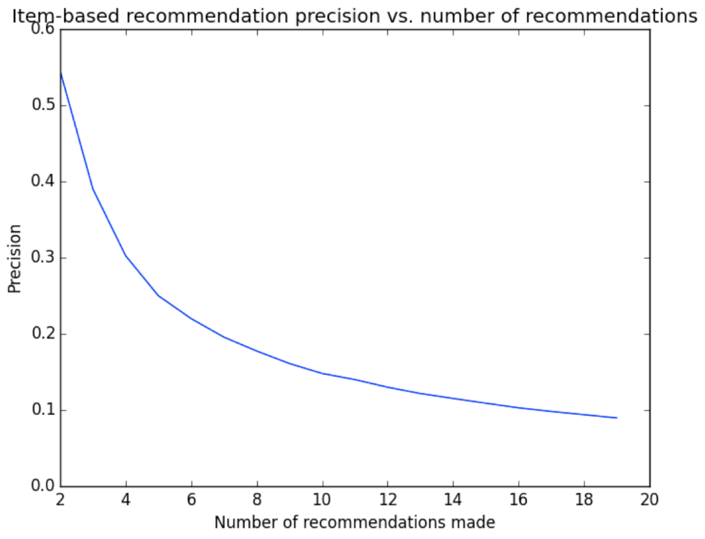

Figure 4. Precision of item-based algorithm vs. number of recommendations made.

It is interesting to point out that this graph follows the structure of an exponential decay graph. Here, we picked recommendations made to be 2 to report when testing our baseline as it resulted in the highest precision. Only recommending 1 subreddit would obviously have resulted in a higher precision based on the trend of the graph, but we set a lower limit of 2 in order to put a cutoff on the quality of our product.

7.2 USER-BASED RECOMMENDATION

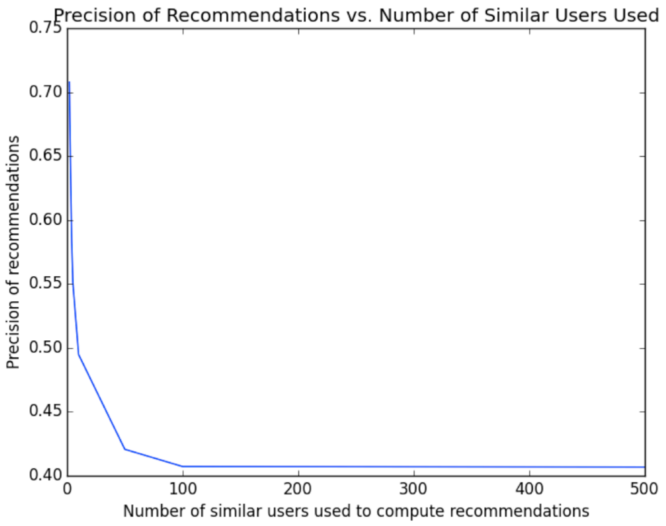

Figure 5. Precision of user-based algorithm vs. number of similar users (*k*) used to compute recommendations.

High: 70.8% for 2 recommendations
Low: 40.7% for 500 recommendations

The graph in Figure 5 shows the number of similar users used to determine which recommendations to make, and how significant with respect to these users the most common subreddits are. In order to pick the optimal point to decide on the number of subreddits to recommend, we cannot simply choose the highest point (at recommendations = 1, not shown) because this number is simply a cap on the maximum number of recommendations that can be shown and not the absolute number to be shown. If a recommendation is poor (not in the current cluster, or too low of significance), we do not want to show it. Additionally, there can be 2 recommendations that have a high score, and the lower one would not be shown.

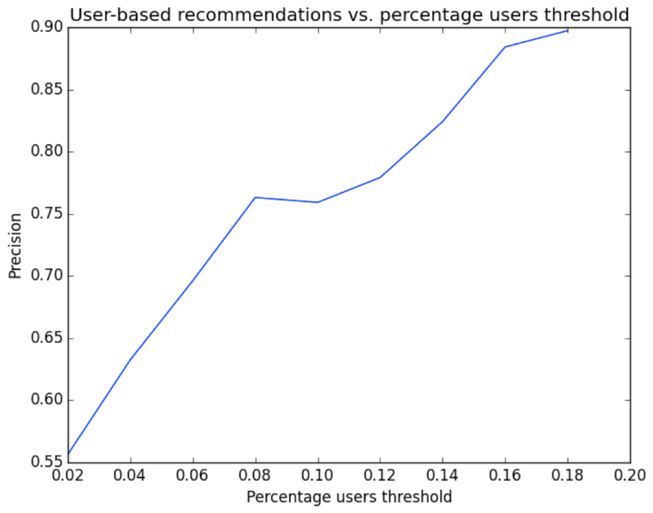

Figure 6. Precision of user-based algorithm vs. percentage of users who must be active in a subreddit candidate.

High: 89.7% precision with 20% threshold
Low: 55.6% precision with 2% threshold

In Figure 6, the precision increases almost monotonically as a function of threshold, while also tapering off around 90% precision. This shows that as we increase the relative cutoff for the strength of our recommendation, the accuracy also increases, which is what we would expect. Note that while it may seem simply to choose the highest threshold to use in tests, this also results in a fewer number of recommendations, leading to noise and high variation in our results.

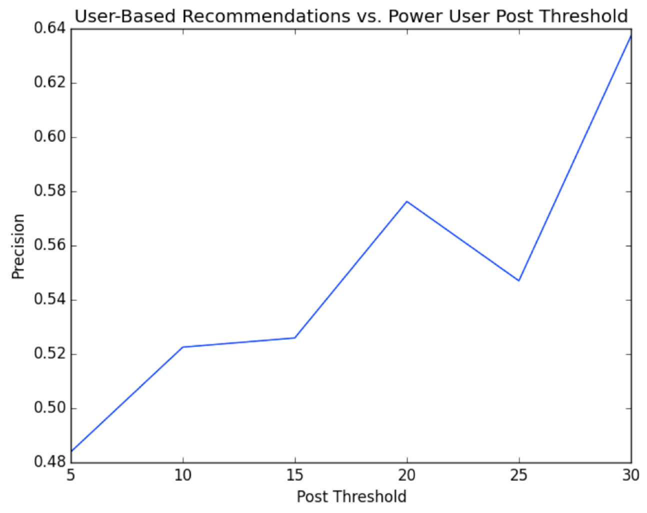

Figure 7. Precision of user-based algorithm vs. threshold for number of posts. The percentage of users required to be active in a subreddit candidate is *p* = 0.05, and the number of similar users is *k* = 500.

Post Threshold	# Power Users	Precision
5	2299	0.4839
10	1339	0.5225
15	940	0.5259
20	716	0.5763
25	582	0.5470
30	482	0.6373
35	389	0.5235
40	334	0.6224

Table 1. Precision of user-based algorithm vs. threshold for number of posts.

Figure 7 shows the precision increase almost monotonically as the threshold on the number of posts required for each power user increases as we would expect. If we look at table 1 though, we see that the number of power users resulting from such an increase also decreases. We do not want to have the resulting number of power users to be too low and thus nullify our cross validation. After a threshold of 20, we see that the precision begins to vary quite consistently, indicating that the number of power users may be low.

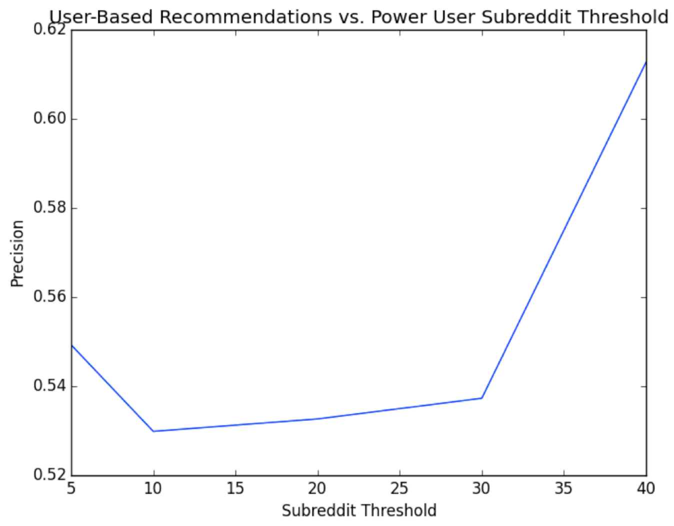

Figure 8. Precision of user-based algorithm vs. threshold for number of subreddits for power users. Here, *p* = 0.05, and *k* = 500.

Subreddit Threshold	# Power Users	Precision
5	5904	0.5493
10	3236	0.5299
20	1147	0.5327
30	482	0.5373
40	201	0.6127

Table 2. Precision of user-based algorithm vs. threshold for number of subreddits.

Figure 8, unlike 7, shows the precision slightly drop as the subreddit threshold increases before the small number of power users at a threshold of 40 causes a large variation. It is interesting that the graph behaves this way, considering that if a power user is involved in many more subreddits, the chance of a correct prediction drastically increases. This leads us to conclude that our recommendation system does indeed give quality recommendations, as opposed to a random subreddit that happens to be visited by a power user.

8. DISCUSSION

As our results show, we achieved a 53.6% accuracy using an item-based algorithm similar to Karypis [3], except with a dimensionality reduction on the number of subreddits we needed to compute the similarities between, resulting in an increase in runtime efficiency. The community-clustering algorithm allowed for this improvement, as well as increasing the precision of our predictions when compared to their results.

Our user-based recommendation algorithm using feature vectors of all posts the user was involved with in the current subreddit resulted in a 70.8% precision in recommendations. We were very pleased at this result, verifying our initial hypothesis that a more personalized recommendation system that took into account specific voting behavior would outperform a more general item-based system.

Karypis [3] noted that a user-based algorithm would suffer from large runtimes and a worse overall result, but with the use of dimensionality reduction from the community clustering as well as insight into how a *power user* should be defined, we were able to outperform the item-based system as well as increase the runtime performance. Conceptually, using similar users to provide a personalized aspect to a recommendation seems much better than a generic one. By identifying habits of users that also share interests with you, we are able to then recommending what they are interested in. We are able to give a different set of recommendations to each individual user, as opposed to the same set of subreddits for each user in a particular subreddit.

One change that we made in the middle of our project was to revisit our assumption on what characteristics the *power user* should hold. Initially, we thought that a simple threshold on the number of subreddits such a user was active in was sufficient, with a large number being better. After implementing our algorithm, we quickly realized that while testing, the random data point chosen could result in in an extremely sparse feature vector. This would result in similarity scores being very small in magnitude, thus yielding almost random and unconfident recommendations. We decided to add another threshold for the number of posts a particular *power user* must have voted in as well, producing much better results.

One large aspect of our project that may seem to bias our results is the concept of a *power user* to test our algorithm. While this may seem to artificially inflate our results, it is the most reliable means of testing our algorithms and reflects the difference between testing and the goal of our project. Our final "product" would be aimed at a target audience of users only involved in 1 or 2 subreddits. With this set of users, the precision and quality of our product would be determined by user-generated feedback or click-through rate, as opposed to cross checking the values with known activity. By setting a threshold on the number of similar users to the test user that are involved in a recommendation, we are showing that it is extremely likely that the user will find our recommendation useful. Additionally, we are currently limiting the feature vector to just the subreddit the current user is on, while for a final product we could incorporate all posts from all subreddits in the current subreddit's community that we partitioned. This would essentially mean using all of the user's data instead of limiting the scope to the current subreddit, which could increase the quality of recommendations even further.

9. CONCLUSION

The difference between item-based and user-based recommendations is very clear. As noted in Karypis [3], the item-based algorithm is the same for each subreddit and is able to be preprocessed leading to a very fast runtime when testing. Where our project yielded better results though, is in the performance and runtime of the user-based algorithm where it yielded higher a higher precision and viable runtime duration when compared to the item-based result.

By partitioning our overall graph into communities of a reasonable size, we were able to improve upon the Karypis [3] paper. The result was a faster runtime for both algorithms as well as a much more personalized user-based system that yielded recommendations that were more related to the current subreddit. While it is still not possible to preprocess the feature vectors for instant comparisons of the user-based algorithm due to memory restrictions, our community partitioning allows the runtime of generating similarity scores to be reduced enough to make the computation viable in a short amount of time. Overall, we were extremely satisfied with the results of our project and hope to implement further functionality that will allow it to be used as a real product with real users.

10. REFERENCES

1. R.S. Olson and Z.P. Neal. Navigating the massive world of reddit: Using backbone networks to map user interests in social media. *CoRR*, abs/1312.3387, 2013.

2. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review, E 69(026113), 2004.

3. G. Karypis. Evaluation of Item-Based Top-*N* Recommendation Algorithms. Proceedings of the tenth international conference on Information and knowledge management. 247-254, 2001.

4. V.D. Blondel et al. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P10008.

5. P.D. Meo et al. Generalized Louvain method for community detection in large networks. *CoRR*, abs/1108.1502.

6. M. A. Serrano, M. Boguna, and A. Vespignani. Extracting the multiscale backbone of complex weighted networks. Proceedings of the National Academy of Sciences of the United States of America. 106(16): 6483-6488, 2009.

[†]All authors contributed equally to this work.