Recommendations For Reddit Users

Avideh Taalimanesh and Mohammad Aleagha
Stanford University, December 2012

Abstract

In this paper we attempt to develop an algorithm to generate a set of post recommendations for users of the social news website Reddit given their prior voting history. We attempted three variations of K-means clustering. We first attempted to cluster users simply based on their voting record and then attempted to cluster users based on attributes of the posts they had voted positively on. Both of these approaches produced very large recommendation sets with poor to moderate recall. Finally we attempted to cluster posts based on keywords appearing in the title and observed much higher recall but lower precision as the recommendation sets that were produced were generally much larger. In all three cases we found that the input data was sparse and quite large and would require a significant amount of pruning if these algorithms were to be used in a practical setting. We also found that the sets of recommendations that were generated were often very large and that some heuristics would need to be applied to reduce their size while attempting to preserve the quality of the recommendations.

1 Introduction

1.1 Background and motivation

Reddit is a social news website where users can submit content and have other users comment and vote (up or down) on their submissions. Since 2005, Reddit has grown into a huge community of very active users; in the month of October (2012) alone, Reddit saw 46,839,289 unique users who viewed 3,832,477,975 pages[1]. With so many pages, discovering new and interesting content can be very challenging. One way the website has been able to recommend content to its users is by letting them subscribe to subreddits. A subreddit is essentially a community focused on a specific topic such as science or music. Recommendations are then made based on the top voted posts within the subreddits a user is subscribed to. Despite this, users still often find it difficult to find content they are truly interested in. In 2010, Reddit gave its users the option to make their votes publicly available and later released some of that voting data for research purposes[2]. We propose to use this data to generate recommendations for users based on their voting history.

1.2 Data preparation

The format of the publicly available data is simple; each entry consists of a user id, a post id and an up or down vote (+1 or -1). We were able to obtain a total of 7,405,561 votes consisting of 31,553 distinct users voting on 2,046,401 distinct posts. In addition to this voting data, Reddit has a public API[3] which allows us to make a request for a particular post id and obtain certain metadata about the post as a json string. This metadata includes among other things the posts originating domain, the subreddit the post belongs to as well as the title of the post. For the purposes of this research project and to make operating on the data feasible with the

[1] http://www.reddit.com/about

[2] http://www.reddit.com/r/announcements/comments/ddz0s/reddit_wants_your_permission_to_use_your_data_for/, http://www.reddit.com/r/redditdev/comments/bubhl/csv_dump_of_reddit_voting_data/

[3] https://github.com/reddit/reddit/wiki/API

computational resources available to us, we limited our efforts to a set of 1,000 users[4] voting on 174,886 distinct posts. We wrote a series of scripts in Java to parse the voting data, make requests to Reddit's servers for metadata and to build the input data (design matrices) for our learning algorithms.

1.3 Overview of approaches

We will attempt to tackle this problem using a few variations of K-Means clustering. Our first attempt will be to cluster users simply based on posts they've voted on in the past. The intuition behind this approach is that users who vote similarly on the same set of posts will likely share similar interests. We can leverage this fact to generate recommendations based on posts up-voted by similar users. Our second attempt will again be to cluster users, but this time based on certain attributes of the posts they've voted on, namely originating domain and the subreddit the post belongs to. This will give us a slightly coarser view of a user's interests compared to the first approach but will require a much smaller feature vector that will not grow every time a new post is submitted and will not be as sparse. As before, we can use the clustered users to generate a set of recommendations. The final approach will be to cluster posts rather than users based on keywords appearing in the title of the post. The content of a post can be anything from a news article to a video or even an image but all posts invariably have a title. What's more, Reddit actively encourages its users to give meaningful descriptive titles to their posts[5]. Once posts are clustered based on keywords, we can identify those clusters which contain posts up-voted by a user and use the set of posts from those clusters to generate recommendations.

2 Methodology

2.1 Approach 1: Clustering users based on votes

The feature vector in this approach consisted of all posts[6] and the values each feature could take on were -1, 0 or +1 (down-vote, no vote and up-vote respectively). We ran k-means on 95% of the data (950 users) with k set to 10, 25, 50 and 100. Once clustering was achieved we then, for each of the remaining users u_i, did the following to generate recommendations:

i) We withheld 10% of up-votes from user u_i

ii) With the remaining votes for u_i, we found the set U_i of users in the same cluster as u_i and constructed the set P_i of all posts up-voted by the users in U_i.

iii) We then filtered the set P_i to remove posts u_i had already voted on to obtain a set of recommended posts R_i (in practice, we could also then rank the posts in R_i by popularity (most up-votes) and then only show the user the top t posts).

iv) We then tested our recommendations using the 10% of withheld up-votes and assigned a score S_i which is (# of withheld up-votes for u_i that appear in R_i) / (# of withheld up-votes for u_i).

2.2 Approach 2: Clustering users based on attributes of posts up-voted

The feature vector in this approach consisted of the originating domain of the posts[7] as well as the subreddits they

[4] We decided on limiting our dataset to 1000 users after our ip was blocked by Reddit for making too many requests in a short time period. The Reddit team was kind enough to unblock us once we promised to slow down our requests.

[5] http://www.reddit.com/help/faq

[6] Here we left out posts having only one vote as they provided no valuable information, and were left with 31,833 posts.

[7] 30,373 posts were considered (these were the posts up-voted by the considered users), with

belonged to. The values each feature could take on were the sum of up-votes by a user for posts having those attributes. For example:

	domains		subreddits	
	youtube	imgur	music	funnypics
u1	5	7	1	1
u2	2	0	4	3

As before, we ran k-means on 95% of the data (950 users) with k set to 10, 25, 50 and 100. Once clustering was achieved we then repeated the steps (i) to (iv) from 2.1 to obtain a set of recommendations R_i and a score S_i for each user u_i.

2.3 Approach 3: Clustering posts based on keywords in the title

For this approach, rather than clustering users, we clustered the posts[8] themselves based on keywords found in the title of the posts. To generate the dictionary of words, we ran Porter's stemming algorithm [1] on the set of words present in the titles of the posts. To further trim down the dictionary, we removed a set of standard stop words such as "the" and "of" [2]. We then generated the feature vectors for each post from this dictionary[9] where the value of a feature was the presence (1 or 0) of the given word in the title of that post. We then ran k-means on all posts with different values for k. Once clustering was achieved we then, for each of a small set of users u_i (50), did the following to generate recommendations:

i) We withheld 10% of up-votes from user u_i

ii) With the remaining votes, we found which clusters the remaining up-voted posts from u_i belonged to. From these clusters $k_{i,j}$ we constructed the set P_i of all posts belonging to $k_{i,j}$.

iii) We then filtered the set P_i to remove posts u_i had already voted on to obtain a set of recommended posts R_i.

iv) We then tested our recommendations using the 10% of withheld up-votes and assigned a score S_i which was (# of withheld up-votes for u_i that appear in R_i) / (# of withheld up-votes for u_i).

3 Results and Analysis

3.1 Initial observations

Upon generating the design matrix for our first algorithm, it quickly became obvious that the data was extremely sparse. Of all the posts being considered, a given user had seen and voted on a fraction of 1% of them. This is not unexpected given the huge number of new posts that are submitted to Reddit on a daily basis. In addition, the dimensions of this design matrix (1000 x 31,833) were quite large (and would be expected to grow much larger as time goes on) since the feature vector was made up of the vote for every post under consideration.

The design matrix for the second algorithm was slightly less sparse as there was substantial overlap of domains and subreddits between posts. The dimensions of this matrix (1000 x 28,605), while also quite large, were more manageable and would not be expected to grow indefinitely as the number of domains and subreddits will remain relatively constant over time.

The design matrix for the third algorithm would have grown to be extremely large had we continued to consider all posts voted on by 1000 users, not due to the size of the feature vector (the dictionary would have had 22,547 words) but simply due to the number of posts to be clustered (34,764 posts). We opted to perform this

---

27,488 different domains and 1,117 different subreddits

[8] 5,397 posts were considered (these were the posts up-voted by the considered users)

[9] We ended up with a dictionary of 8,880 words

clustering for only 50 users (resulting in 5,397 posts and a dictionary of 8,880 words). This was still rather computationally expensive and anecdotally took very long to run.

3.2 Results

	$k = 10$	$k = 25$	$k = 50$	$k = 100$
Avg. S_i	0.7100	0.6636	0.6004	0.3866
Avg. $\|R_i\|$	21,884	21,369	19,155	11,819
*R-ratio**	0.6875	0.6713	0.6017	0.3713
*Q-score***	1.0328	0.9885	0.9977	1.0413

Table 1: Results for approach 1

	$k = 10$	$k = 25$	$k = 50$	$k = 100$
Avg. S_i	0.1490	0.1299	0.0717	0.0517
Avg. $\|R_i\|$	17,179	8,737	5,789	3,053
*R-ratio**	0.5656	0.2877	0.1906	0.1005
*Q-score***	0.2633	0.4516	0.3760	0.5148

Table 2: Results for approach 2

	$k = 10$	$k = 25$	$k = 50$	$k = 100$
Avg. S_i	0.9969	0.9815	0.9785	0.9508
Avg. $\|R_i\|$	4,334	3,734	3,678	3,084
*R-ratio**	0.8030	0.6919	0.8030	0.8030
*Q-score***	1.2414	1.4187	1.2184	1.1840

Table 3: Results for approach 3

* *Avg |Ri| / |*All posts*|*
** *Avg Si / R-ratio*

3.3 Analysis

One key fact that must be kept in mind is that the data available to us is in no way complete in the sense that a user's preference is only known for a very small number of posts. Therefore the scores we've assigned to the various recommendation sets we've generated will give us an intuition about the approach taken but do not entirely reflect the quality of the recommendation set (had a user happened to have seen more posts, they may have up-voted those present among the recommendations).

The two metrics of interest when evaluating the approaches we've taken are the score S_i and the size of the recommendation set relative to the number of posts considered which we'll call the *R-ratio*. We want to maximize the average S_i while minimizing the size of the recommendation sets so we'll compute another score Q which we'll define as *Avg.* S_i */ Avg. R-ratio.*

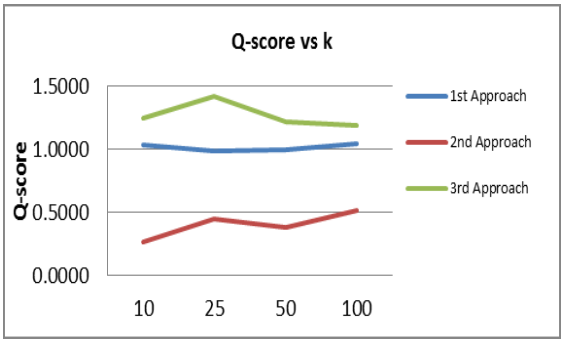

Figure 1

We can see from the results that the approach which had the highest *Q-score* was the 3rd approach which, although it generated fairly large recommendation sets, showed a much higher recall with the highest average S_i scores. The 2nd approach did the worst out of the three approaches with both large recommendation sets as well as low average S_i. The 1st approach simply did not have enough data to adequately cluster users and what we observed was usually the formation of one very large cluster containing most of the users with the rest of the clusters containing a very small number of users. This resulted in decent S_i scores for the users in the large cluster (if most other users are in the same cluster as you, chances are one of them will have up-voted an article you up-voted) but very large recommendation sets.

4 Conclusion

4.1 Input data

We found that it was very difficult to generate good recommendations with only a very limited amount of data about each

user's preferences. In the 2 methods we used which clustered users based on voting history, we found that in some cases it was simply impossible to recommend all articles that a user had up-voted because no other user in the set had up-voted that article. The sparseness of the feature vectors aside, the sheer size of the sets we would have needed to operate on (number of users and number of posts) would not have been possible had we wanted to cluster all Reddit users. It is obvious that to use any of these algorithms in practice would require significant pruning of the data such as segmenting users based on some attributes (subreddit subscription, geographic location, etc.) and then running the algorithms on each segment. Another factor to take into consideration is the age of a post; to further trim down the data, posts older than a certain threshold could be left out (stale posts are not valuable recommendations anyway).

4.2 Recommendation sets

Another difficulty we encountered was producing reasonably sized recommendation sets. Even if we can produce all of the posts a user could ever be interested in, if they are hidden in a gigantic set of recommendations the user will never find them and we haven't done much to improve the experience. We could use some heuristics to trim down the size of the recommendation set at the risk of losing a few good recommendations. One heuristic could be, as mentioned in the previous section, to omit posts which are more than a few days/weeks old altogether as content goes stale over time. Another approach could be to not trim down the recommendation set at all but rather present the posts to the user in an order which we think would make the best recommendations be the easiest to find. One way to achieve this would be, for instance, to order the posts by popularity (most up-votes).

4.2 Future Work

Aside from the improvements to the input data and the post-processing of the generated recommendations outlined in the previous sections, more work could be done to improve the clustering algorithms themselves. Given our best performing algorithm (clustering posts based on keywords), one easy improvement would be to include the subreddit and originating domain of the post in the feature vector along with the dictionary of words. Another possible improvement would be to assign a score to each selected cluster for a user based on the ratio of down-voted to up-voted posts that clusters contains and select the ones with the highest scores rather than select them all to generate recommendations.

5 References

[1] M.F.Porter, "An algorithm for suffix stripping", Originally published in Program, 14 no. 3, pp 130-137, (July 1980)

[2] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li., "RCV1: A New Benchmark Collection
for Text Categorization Research" (2004), Journal of Machine Learning Research 5 (2004) 361-397