Log Report

Anish Kelkar
apk2129@columbia.edu

Divi Khanna
dk2745@columbia.edu

Pranav Bhalla
pb2538@columbia.edu

Richa Singh
rs3375@columbia.edu

Abstract

This log report details various phases of how our analysis progressed over a period of 3 weeks working on this project. We were motivated by various research studies done on the datadump of StackExchange, and used many of these to develop an intuition of a recommendation system for this collaborative Q&A platform.

Contents

1 Motivation

As graduate students we are frequent users of Stack Exchange and this idea was motivated by our personal experience while using this site. Other Q&A websites such as Quora, Yahoo!Answers, Answers.com follow similar format like Stack Exchange and each one has its own special area of focus. There was an interesting feature we came across at Quora, where the site asked you to select your area of interest and then invited you to answer specific questions to those interest. As Columbia University students residing in New York, many times we are directed towards questions seeking more information on Columbia and New York. This feature of Quora was what gave us this idea to devise a recommendation system for Stack Exchange.

Stack Exchange has always relied on its reputation system to encourage users to participate more in their Q&A format. The reputation system has become a hallmark in the space of Stack Exchange, where many top users (with reputation scores as high as 672,426) now showcase these scores as an added certification to their expertise. The whole network of Stack Exchange website began with Stack Overflow, a platform where programmers seek solutions to their development and debugging efforts. Till today, Stack Overflow remains the dominant network in the family of Stack Exchange sites.

1.1 Objectives

Our initial idea was to match every new question that comes on Stack Exchange to users who will be more interested to answer that question. The objective was to ensure that a new questions is recommended to somebody who has the capability and willingness to answer that question. This system will improve the quality of answers, reduce the time it takes to get answers, and minimize the number of unanswered questions on Stack Exchange.

2 Initial Phase

- We began by studying the data schema for Stack Exchange datadump and chose the smaller site *Sustainable Living*[1] for running our results.

[1]Sustainable Living Stack Exchange is a question and answer site for folks dedicated to a lifestyle that can be maintained indefinitely without depleting available resources.

- As each site of Stack Exchange follows the same schema, our aim was to implement the recommendation system for *Sustainable Living* and then test the results on bigger sites like *Cross Validated*[2]
- Larger sites like *Stack Overflow*[3] have big datasets and the analysis can only be completed by running the recommendation system over a Hadoop cluster.

2.1 Datadump Schema

Tables	Attributes
Posts	Id, PostTypeId, AcceptedAnswerId, ParentID, CreationDate, Score, ViewCount, Body, OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivityDate, Title, Tags, AnswerCount, CommentCount, FavoriteCount, CommunityOwnedDate
Users	Id, Reputation, CreationDate, DisplayName, LastAccessDate, WebsiteUrl, Location, AboutMe, Views, UpVotes, DownVotes, ProfileImageUrl, AccountId, Age
Comments	Id, PostId, Score, Text, CreationDate, UserDisplayName, UserId
Badges	Id, UserId, Name, Date
Post History	Id, PostHistoryTypeId, PostId, RevisionGUID, UserId, UserDisplayName, Comment, Text
Post Links	Id, CreationDate, PostId, RelatedPostId, LinkTypeId
Votes	Id, PostId, VoteTypeId, UserId, CreationDate, BountyAmount

Table 1: Schema for Stack Exchange

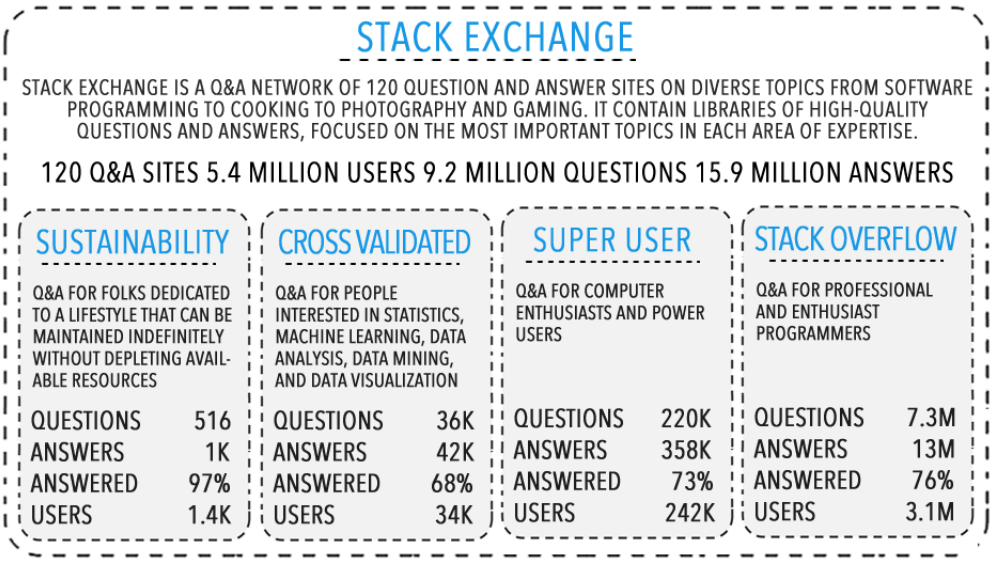

Figure 1: **Different Exchanges on Stack Exchange**

2.2 Recommender System

Recommender systems typically produce a list of recommendations in one of two ways - through collaborative or content-based filtering. Collaborative filtering approaches build a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users; then use that model to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. These approaches are often combined into

[2]Cross Validated is a question and answer site for people interested in statistics, machine learning, data analysis, data mining, and data visualization.

[3]Stack Overflow is a question and answer site for professional and enthusiast programmers.

a Hybrid Recommender System that could be more effective in some cases.

A hybrid recommender system is one that combines multiple techniques together to achieve some synergy between them.

- *Collaborative filtering:* The system generates recommendations using only information about rating profiles for different users. Collaborative systems locate peer users with a rating history similar to the current user and generate recommendations using this neighborhood.

For our dataset, we used a collaborative filtering approach to cluster users into communities. The idea is to create a community of users who are interested in a particular group of topics. For example- it is intuitive to see that on Cross Validated there must be a group of users who specialize in topics of visualizations, and another group who specialize in machine learning algorithms. There may be some overlap between users in both groups, which can impact the efficiency of recommendations that are based on community clusters.

- *Content-based filtering:* The system generates recommendations from two sources: the features associated with products and the ratings that a user has given them. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on product features.

We have used number of questions answered by a user on a particular tag as proxy of their preference for answering a new question in a tag community. We create a linear model that accesses the reputation score and frequent participation in answering questions to come up with a regression model that fits on test data to come up with results on whether a user should be part of the recommendation list or not.

3 Intermediate Phase

3.1 Network Visualization

3.2 Various Network Representations

- We have used the CRAN package igraph [1] which contains routines for simple graphs and network analysis. igraph can handle large graphs very well and provides functions for generating random and regular graphs, graph visualization, centrality indices and much more.
- While working to better visualize the data we generated various graphs, which we think will be interesting to put in this log.

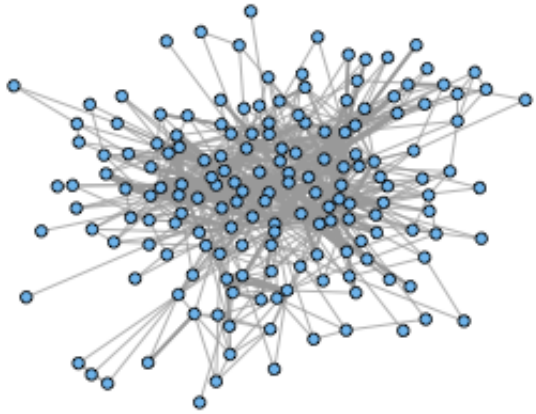

Figure 2: **Tag Network Visualized using Fruchterman-Reingold Algorithm**

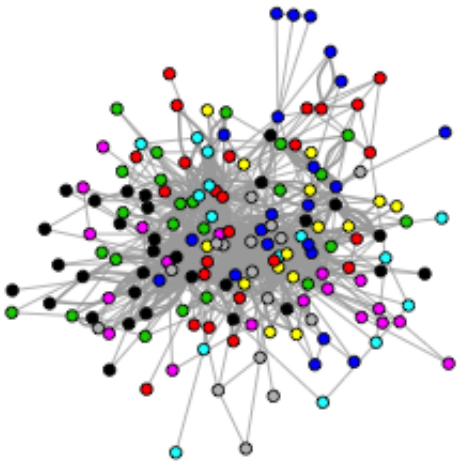

Figure 3: **Same Network, but this time the same colored notes belong to one community**

- Certain graphs had too much information in them due to the sheer size of data. We scaled down data by selecting only prominent clusters for better visualizations, but larger graphs were maybe just nice to look at!

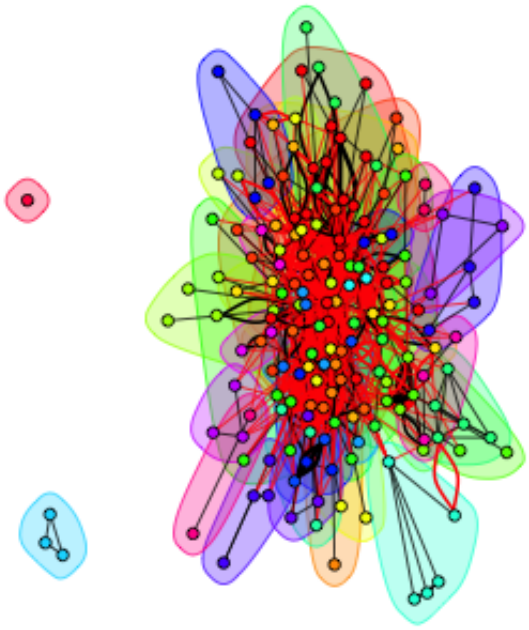

Figure 4: **Clustering all communities in the dataset, hard to see the boundaries in high-dimensional data**

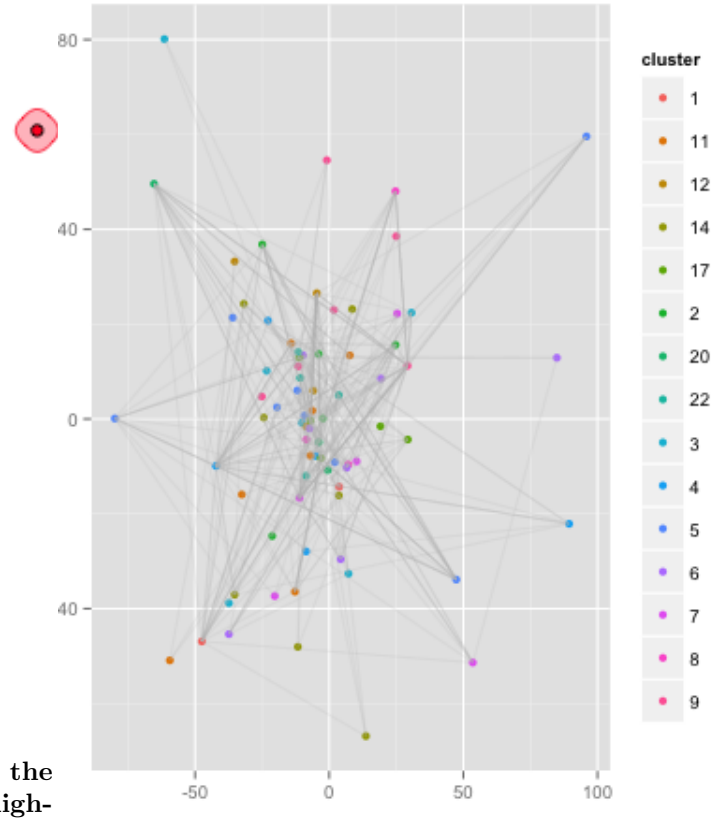

Figure 5: **ggplot connecting large communities**

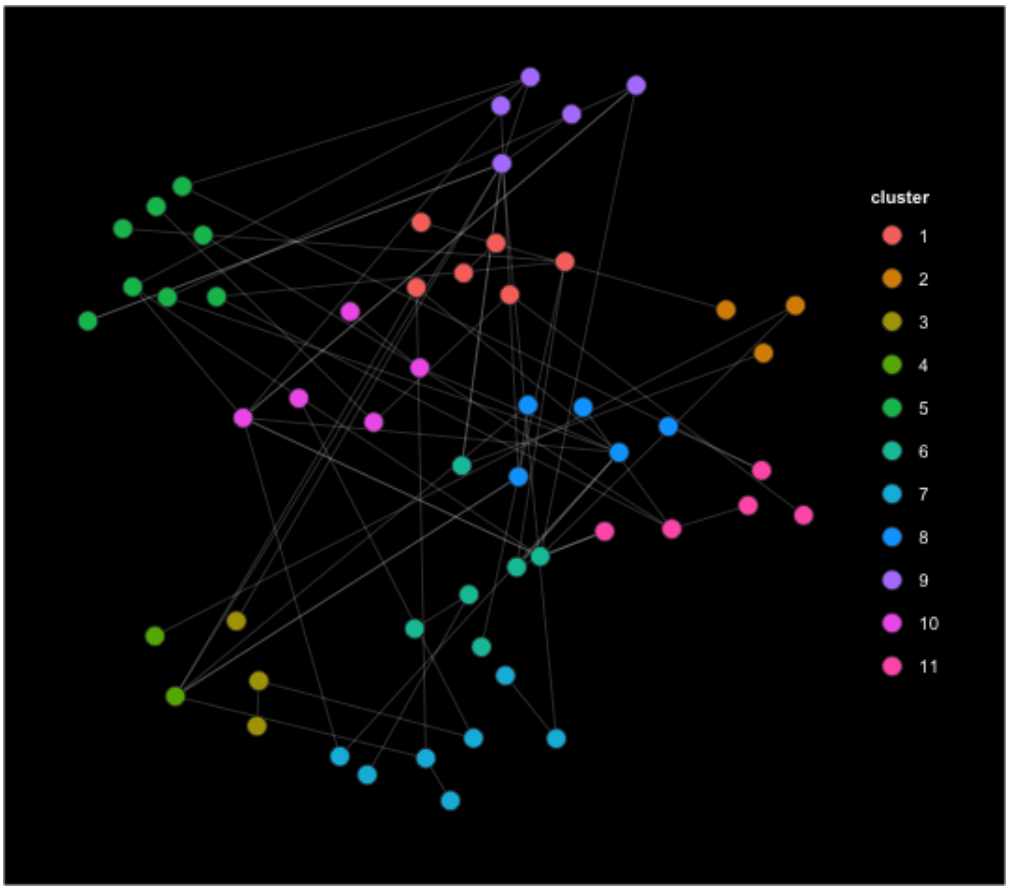

Figure 6: **Large tags plotted by closeness**

3.3 Community Detection Algorithms in igraph

- The igraph package has 8 community detection algorithms that we tested on our dataset to select the one that gave better results for our need. We have chosen infomap.community for its ability to give relevant clusters that justify the tradeoff between number of clusters (C) and size of a cluster (S).
- Our objective was to create communities for 174 unique tags in the Sustainable Living dataset based on pairwise co-occurance of tags in questions.

3.3.1 Summary of Algorithms

1. *edge.betweenness.community* is a hierarchical decomposition process where edges are removed in the decreasing order of their edge betweenness scores (i.e. the number of shortest paths that pass through a given edge). This is motivated by the fact that edges connecting different groups are more likely to be contained in multiple shortest paths simply because in many cases they are the only option to go from one group to another. This method yields good results but is very slow because of the computational complexity of edge betweenness calculations and because the betweenness scores have to be re-calculated after every edge removal.

```
membership: 55 communities
modularity: 0.3287658
```

2. *fastgreedy.community* is another hierarchical approach, but it is bottom-up instead of top-down. It tries to optimize a quality function called modularity in a greedy manner. Initially, every vertex belongs to a separate community, and communities are merged iteratively such that each merge is locally optimal (i.e. yields the largest increase in the current value of modularity). The algorithm stops when it is not possible to increase the modularity any more, so it gives you a grouping as well as a dendrogram. The method is fast and it is the method that is usually tried as a first approximation because it has no parameters to tune. However, it is known to suffer from a resolution limit, i.e. communities below a given size threshold will always be merged with neighboring communities.

Error!: fast-greedy community finding works only on graphs without multiple edges

3. *walktrap.community* is an approach based on random walks. The general idea is that if you perform random walks on the graph, then the walks are more likely to stay within the same community because there are only a few edges that lead outside a given community. Walktrap runs short random walks of 3-4-5 steps (depending on one of its parameters) and uses the results of these random walks to merge separate communities in a bottom-up manner like fastgreedy.community. Again, you can use the modularity score to select where to cut the dendrogram. It is a bit slower than the fast greedy approach but also a bit more accurate.

membership: 16 communities modularity: 0.41396

4. *spinglass.community* is an approach from statistical physics, based on the so-called Potts model. In this model, each particle (i.e. vertex) can be in one of c spin states, and the interactions between the particles (i.e. the edges of the graph) specify which pairs of vertices would prefer to stay in the same spin state and which ones prefer to have different spin states. The model is then simulated for a given number of steps, and the spin states of the particles in the end define the communities. The consequences are as follows: 1) There will never be more than c communities in the end, although you can set c to as high as 200, which is likely to be enough for your purposes. 2) There may be less than c communities in the end as some of the spin states may become empty. 3) It is not guaranteed that nodes in completely remote (or disconencted) parts of the networks have different spin states. The method is not particularly fast and not deterministic (because of the simulation itself), but has a tunable resolution parameter that determines the cluster sizes. A variant of the spinglass method can also take into account negative links (i.e. links whose endpoints prefer to be in different communities).

Error!: Cannot work with unconnected graph

5. *leading.eigenvector.community* is a top-down hierarchical approach that optimizes the modularity function again. In each step, the graph is split into two parts in a way that the separation itself yields a significant increase in the modularity. The split is determined by evaluating the leading eigenvector of the so-called modularity matrix, and there is also a stopping condition which prevents tightly connected groups to be split further. Due to the eigenvector calculations involved, it might not work on degenerate graphs where the ARPACK eigenvector solver is unstable. On non-degenerate graphs, it is likely to yield a higher modularity score than the fast greedy method, although it is a bit slower.

membership: 9 communities
modularity: 0.3860806

6. *label.propagation.community* is a simple approach in which every node is assigned one of k labels. The method then proceeds iteratively and re-assigns labels to nodes in a way that each node takes the most frequent label of its neighbors in a synchronous manner. The method stops when the label of each node is one of the most frequent labels in its neighborhood. It is very fast but yields different results based on the initial configuration (which is decided randomly), therefore one should run the method a large number of times (say, 1000 times for a graph) and then build a consensus labeling, which could be tedious.

membership: 7 communities
modularity: 0.3186567

7. *infomap.community* is based on information theoretic principles; it tries to build a grouping which provides the shortest description length for a random walk on the graph, where the description length is measured by the expected number of bits per vertex required to encode the path of a random walk.

membership: 38 communities
modularity: 0.2016404

4 Infomap Community

The problem of finding the best cluster structure of a graph can be seen as the problem of optimally compressing its associated random walk sequence. The goal of Infomap is to arrive at a two-level description that exploits both the network's structure and the fact that a random walker is statistically likely to spend long periods of time within certain clusters of nodes. More specifically, we look for a module partition M (i.e., set of cluster assignments) of N nodes into m clusters that minimizes the following expected description length of a single step in a random walk on the graph:

$$L(M) = q \curvearrowright H(\mathcal{Q}) + \sum_{i=1}^{m} p^i \circlearrowright H(\mathcal{P}^i)$$

This equation comprises two terms: first is the entropy of the movement between clusters, and second is the entropy of movements within clusters, both of which are weighted respectively by the frequency with which it occurs in the particular partitioning. The specifics are detailed in [2]. Ultimately, equation serves as a criterion for a bottom-up agglomerative clustering search. The implementation provided by [2] uses above equation to repeatedly merge the two clusters that give the largest decrease in description length until further merging gives an increase. Results are further refined using a simulated annealing approach, the specifics of which can be found in [2]. Infomap Community is the best algorithm for network clustering and hence we decided to use it for implementation of our system.

5 Implementation Stage

5.1 Creating data structures

Our initial idea implemented a naive recommendation system traversing for tags in the entire dataset, considerably increasing the time to run the algorithm. In order to reduce the system run time we played with data.frame and list() objects in R, which are better for memeory mamagement than writing for loops. We can still improve much more on the run time, and this would be an essential step before we decide to test this system for larger datasets like *Super User* and *Stack Overflow*.

Question Id	User Ids
8	115 ,69, 5, 22
5	130, 85, 66, 301

Table 2: Question Ids and User Ids

Tag	Question Id
Husbandry	1,2,3,341
Insects	1,412,844,911

Table 3: Tags and Question Ids

Tags	Users Ids
Husbandry	130,85,185,130,66,10
Insects	719,99,549,91,130,244,130,228,20,98

Table 4: Tags and Users

Community	Tags
14	permaculture, fungal-innoculation, pets, defining-sustainability, crop
10	preservation, reuse, waste- minimisation, craft, woodwork, baby-care

Table 5: Community and Tags

Community	Users Ids
16	16, 48, 623,96, 96, 623,96, 361,48, 40, 623,96, 85, 400,318,48, 404,27, 623,96, 623,96
19	19, 462,659,96, 486,318,48, 404,98, 400,318,48, 404,282,462,462,462

Table 6: Community and Users

5.2 Testing the recommendation system

At this stage we had a bunch of working community detection algorithms that we wanted to test out. So we created a function that takes a tag as input and returns all the tags in it's cluster. For example, in case "R", "R-Studio" and "Data-Science" belong to the same cluster, our function given input any one of the 3 tags would return all 3 of them. Initially we used two different ways to test the accuracy, neither of them used clustering so that we would have a baseline to compare against when we used the cluster of tags.

First, we choose those users as potential answerees who had given accepted answers to all the tags in a new question. Suppose a new question came with the tags "data-science", "R", "clustering". We pick only those users who have previously answered questions on all 3 tags.
The second way, we pick users who have given accepted answers to atleast one of the tags in the new question. For the above example, we pick users who have answered questions on either "data-science" or "R" or "clustering" or more than one. Obviously, we get more users with the second algorithm than the first and we would have more users to recommend this question to. Hence we usually get a much better accuracy at finding the person who actually answered the question.

Till this point we have not used our clustering anywhere in finding users. The third testing algorithm we implemented expands the tags for a new question using the function mentioned at the start of this section. And then picks the users who have answered atleast one of the expanded tags.
For the above example, the intial tags "data-science", "R", "clustering" could be expanded to include tags such as "big-data", "community-detection", etc. We pick users who have answered questions on any one or more of these tags. Again, here the number of users would be much larger that first or second testing algorithms, since it would have more tags to find users against.

Validation Method	Accuracy
Users who answer all the tags	2.4%
Users who answer atleast one tag	51.2%
Users who answer atleast one tag from expanded list	63.4%

Table 7: Basic validation methods accuracy on Sustainibility data

5.2.1 Testing on a larger data set

Upto this point we were performing all testing on a smaller dataset (sustainibility.stackexchange). We moved on to trying it on a large dataset (stats.stackexchange) which is almost 10 times the earlier dataset (more than 10k questions). The first problem we ran into was cleaning the data and loading it up to R, the same cleaning code that ran in minutes on the smaller dataset took hours on the bigger dataset. Once the code was cleaned and ready, we ran the clustering algorithms on the bigger dataset. Of our chosen community detection algorithms (leading eigenvector, edge betweeness, walktrap and infomap) only walktrap and infomap converged on the larger dataset, the others simply never finished clustering even after hours. And after observing the clusters made by these two, we saw that the Infomap algorithm was much better at clustering on this dataset. Walktrap had clusters with more than 50 tags, which as we found further in testing, would lead to very large user lists after performing the matching.

Validation Method	Accuracy
Users who answer all the tags	30.2%
Users who answer atleast one tag	72.4%
Users who answer atleast one tag from expanded list	89.2%

Table 8: Basic validation methods accuracy on Stats data

After creating the clusters we ran the valdiation algorithms, for all the 3 methods described in the previous section (match all tags, match atleast one tag and match expanded tags). We got very good accuracy for the expanded tags (close to 90%), but the problem was we ended up recommending to lots of users on the expanded list. It would not be feasible in the real world to recommend an unanswered question to too many users. Hence we needed to come up with either a penalty term that would weigh down our accuracy when recommending to too many users or to rank the users so that we could find the top users to recommend the questions to.

5.2.2 Penalty Term

As an initial penalty term, we thought of weighting the accuracy between 0 and 1. Earlier we awarded a 0 score if we didn't find the actual answeree in the list and 1 if we did find. But this way we do not limit the number of users we recommed to and encourage a system that will recommend to lots of users. So we decided to not award a 1 in case we find a match, but weigh down the score based on the number of users.

The penalty mechanism works like this, in case you dont find a match we give a score of 0. In case we find a match, if you have only picked only 10% of the users you still get a score of 1. But if the number of users matched exceeds 10% of the total users you get a score less than 1. We keep decreasing the scores linearly from 1 to 0, which is the case when we recommend to all users. So recommending to all users is equivalent to being inaccurate. A formula is warranted here.
¡formula here¿
There is no reason behind choosing 10% as the limit, it is just a number we used to test out penalty term. We dropped this penalty measure later in favour of ranking users and then truncating them. Our accuracy when down quite a bit after implementing this penalty term, which was expected since most of our user lists for a new question exceeded the 10% threshold.

5.2.3 Ranking users

Instead of using a penalty term we can rank the users and then pick the top few percent to recommend the questions to. This is a more appropriate way of limiting users as compared with a "penalty term on the acuracy" since we have some metrics using which we can rank users. To rank the users we used a tag score.

We initially create a hash where we store the number of accepted answers a user has given per tag. We used this later to assign a score to each user for a new question, where he gets points (the number of questions he has answered on this tag) for each tag he has answered previously. Using this score, we create the rank for users, order them and pick the top 10%. Again the number 10% is choosen arbitrarily, we had no idea of what a good percentage would be. We didn't get a good result for the Sustainibility dataset, but even after limiting each list to the top 10% best ranking users, we got a pretty good accuracy rate on a bigger dataset (close to 50% on the stats exchange data set). That meant that our method of ranking users was working well in finding the actual answeree of a question.

Validation Method	Accuracy
Ranked Users (truncated)	7.3%

Table 9: Ranked validation on Sustainibility data with truncating length set to 10%

5.2.4 Multiple scoring criteria

We also wanted to incorporate a users reputation into ranking them, as it seems the main reason behind the drive in Stack Exchange. But reputation is a static measure that does not change across questions, hence we were not sure if it would be helpful in finding the actual answeree. We studies the various relations

reputation had with other prominent variables in the dataset, to come up with a way that is a good measure of scoring criteria for users.

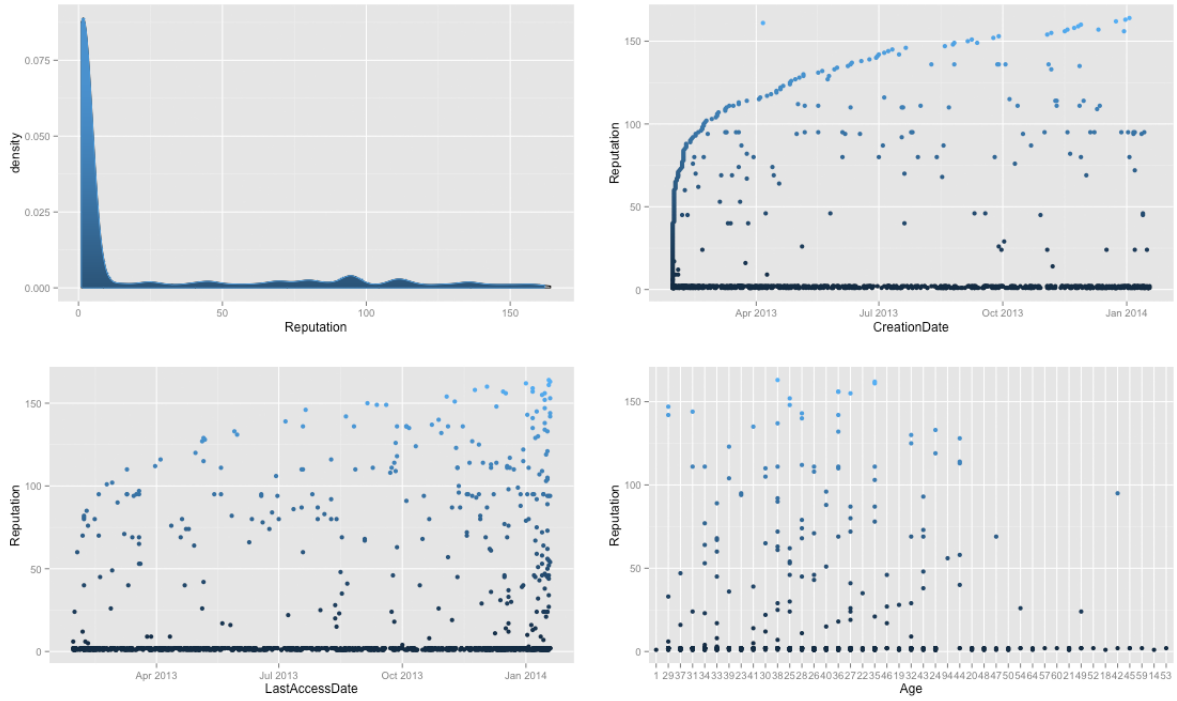

Figure 7: **Relating reputation to create date of user, last access date of user and age of user**

Reputation showed a logarithmic relation with creation and last access dates, and we inferred this pattern as an indication of logarithmic increase in reputation. Simply put, reputation rises faster in the initial activity period of the user. After a certain reputation level is reached, the rise is slower. Something that is quite intuitive as a property of large numbers.
So we decided to model the two features of reputation and tagscore to get a better scoring criteria for users' ranking and used linear regression to find the weights for reputation and tag score. We got a set of weights for reputation and tag score after applying the regression. Using this we again applied a weighted ranking (combination of tag score and reputation weighted by weights from linear regression) on the training data and then ordered the users based on the rank. For the questions in which we found the actual anweree (which is every question since we ran this on the training set) we found the ranked position the actual answeree had. For example, see the below rank ordering for user ids for some question.

400, 31, 76, 890, 113, 73, ...

Now suppose the actual answeree was 76, then the position we found the actual answeree at is the 3rd. Similary we find this value for each question in the training set and take a mean to find the optimum truncating length for the test data. Till this point we have been arbitrarily chosen the truncating length for the user list, be it when we used the 10% penalty term or top 10% ranked users, we have not had a good measure of the truncating length.
This whole procedure of running the linear regression on the test data to find weights for reputation and tag score, and then again running the algorithm over the training set to find the truncating length, is a very lengthy operation that takes a lot of time. Hence we couldn't run it on the "stats exchange" data set. But for the "travel exchange" data set, we ran the whole procedure, we got a negative weight for reputation, which implied that maybe the highest reputed users don't always answer the question, but this hypothesis needs to be verified separately. For the time being we included the reputation when ranking the users.

6 Final Stage

Given our inability to test the model of large datasets, we soon reailzed that we need a better model that is scalable across dataset of any size. We improved our algorithm by playing with different data structures and creating lists() on which operation running time was considerably lower.

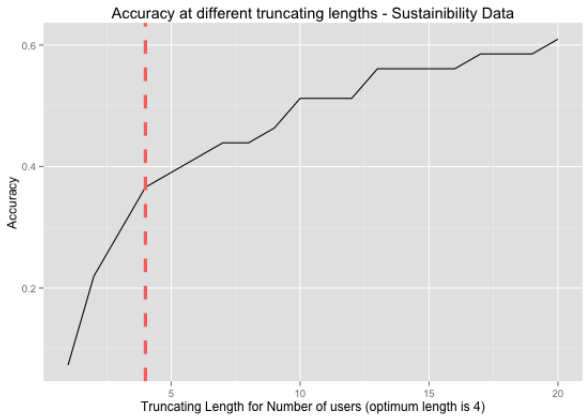

Figure 8: **Accuracy v/s Truncating Length for Sustainibility**

Again, the above figure is for the Sustainibility data set, for which our regression and subsequently running the algorithm on the test data gave a truncating length of 4, which gave an accuracy of about 36%. For the travel data set, we obtained a truncating length of 9, which is pretty small considering the number of possible users for the travel data set was 447. Using this truncating length and weighted combination of tag score and reputation for ranking we obtained an accuracy of 50%. We felt this was a significant result considering we recommend a new question to a maximum 9 people and 50% of the time we find the person who actually answered it.

Validation Method	Accuracy
Ranked Users (truncated)	36.6%

Table 10: Ranked validation on Sustainibility data with truncating length set to 4

*Everywhere in tables we provide accuracy for the Sustainibility data set, since that is the biggest data set where we were able to run all our algorithms. In the text we have discussed details about bigger data sets.

7 Link to code

https://github.com/pranavbhalla89/ads_final_proj

References

[1] igraph: Network analysis and visualization. `http://cran.r-project.org/web/packages/igraph/index.html`, 2014.

[2] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.