# *An Open Source Discussion Group Recommendation System.*

***SARIKA PADMASHALI***
***CS297 MASTER'S PROJECT***

## INTRODUCTION

The purpose of this project is to build a recommender system for Yioop. Yioop is an open source search engine, wiki system, and user discussion group system managed by Dr. Christopher Pollett. I will be developing a recommendation system for Yioop where the users will be suggested about the threads or groups they could join based on their user history.

A recommendation system analyzes user behaviour on a website to make suggestions about what a user should do in the future on the website. It basically tries to predict the "rating" or "preference" a user would be giving to an item.

I have never contributed to an open source project. Yioop will be the first open source project that I will be contributing to. Whenever I surfed the web I was always dealing with some recommendation systems and search engines. I always wondered what was the difference between the two — both suggest some items to the users on the web, both do some sort of searching and filtering. Yioop is a search engine and I thought I would be able to understand the difference better if I had a chance to work with Yioop framework and build a recommendation system on top of it. Also, in the previous semester I had done a literature review on recommendation systems and I was very fascinated by the content and the mathematics that went behind it.

In this semester I have implemented a few techniques to get the basics and feel of how the recommendation system works. This project is an attempt to improve the users experience while they are browsing through the Yioop website. This feature would avoid users to hit the search tab and search for threads they like. Rather the threads of their interest will be recommended to them.

I have coded many of the deliverables in python so far. However, I plan to build a recommendation engine in PHP for my final project as the Yioop codebase is in PHP. This semester, for the first deliverable, I implemented a max flow problem in python – FordFulkerson and Push relabel algorithm. The max flow problem can be extended to solve the bipartite matching problem which can map users to threads. For Deliverable 2, I researched the algorithm used to win the Netflix Prize and tried to implement a part of it for the Yelp Dataset. My program predicted what a user would rate a particular restaurant using the

biases. In Deliverable 3, I got an opportunity to work with the Yioop codebase. I will be working extensively on the Yioop codebase next semester, hence it was essential to understand the framework so that next semester I spend more time on improving and building the recommendation model for Yioop and less time in understanding the framework. In Deliverable 4, I tried another technique which used the latent matrix facrtorization to predict the users rating. I have discussed each of these deliverables in detail in the section followed by preliminaries.

**PRELIMINARIES**

Let us take a few examples to help us understand what a recommendation system recommends. Netflix Recommender system suggests a list of movies to users to improve their movie watching expeirence. Whenever you log in to your amazon account, you see "Users who bought this, also bought that" suggestions from amazon based on your purchase history. This is the power of the recommendation systems.

Recommendation systems, when properly built ease the users' experience of the website and also increases the number of hits the website receives. Recommendation systems have proven useful for huge companies like Amazon and Netflix who have a very large expansive client base and have a large number of movies or products which a large number of people haven't even seen or pruchased.

A huge corpus of data depicting users' behaviour is the key to building a good recommendation system as we are trying to find similar users or items to make some suggestions. If we do not have a pattern or similarity between users or products and each item is different and every other person has a different taste when why build a recommender system in the first place. However, there is always a pattern which exists among all the users behvaiour however different or unique a person might be. Same is the case with products or items and in our case — groups.

**TYPES OF RECOMMENDER SYSTEMS**

There are two main types of recommendation systems — Content based filtering and Collaborative filtering. Each system tries to find similarities between users and items. Merging data between different users and items helps these systems gain insights on their patterns.

**Content Based Filtering**

Content based filtering systems focus on the attributes of items. The similarity between items is determined by measuring the similarity between their attributes. Figure 1 is an example of content based filtering. Below is a mapping of users to the items they shopped. We see that the items grapes and watermelon have similar attributes — they both are fruits. When Joe purchases watermelon, we also recommend him to buy grapes. This is a basic example of how content based filtering works.

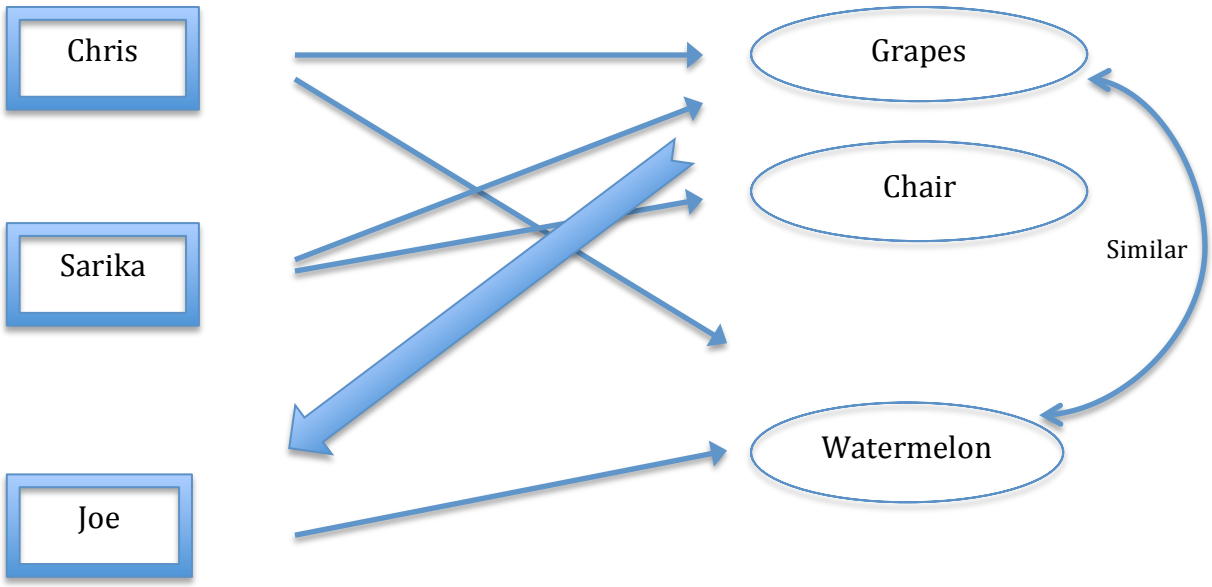


Fig. 1. Content Based Filtering

**Collaborative filtering**

Collaborative filtering tries to find other similar users and suggests items which similar people have liked. It is based on the idea that if two or more users have assessed an item similarly in the past they are likely to assess other items similarly in the future. Let us ask ourselves a few simple questions here: How often have you asked a friend who has a similar taste as yours in movies to suggest a movie on a boring friday night? How often have you asked your friends to come shopping with you because you like the clothes they wear?

Collaborative filtering tries to do the very same thing which you do, infact it acts like your friend here and suggests items for you by mining millions of similar users. Figure 2 gives an illustration of the basic working of collaborative filtering. Chris purchases strawberries, oranges and mangoes, Sarika buys

mangoes and watermelons, while Joe buys watermelon. What else would we suggest Joe? We see that the shopping pattern of Sarika and Joe are similar hence we suggest Joe to buy mangoes.

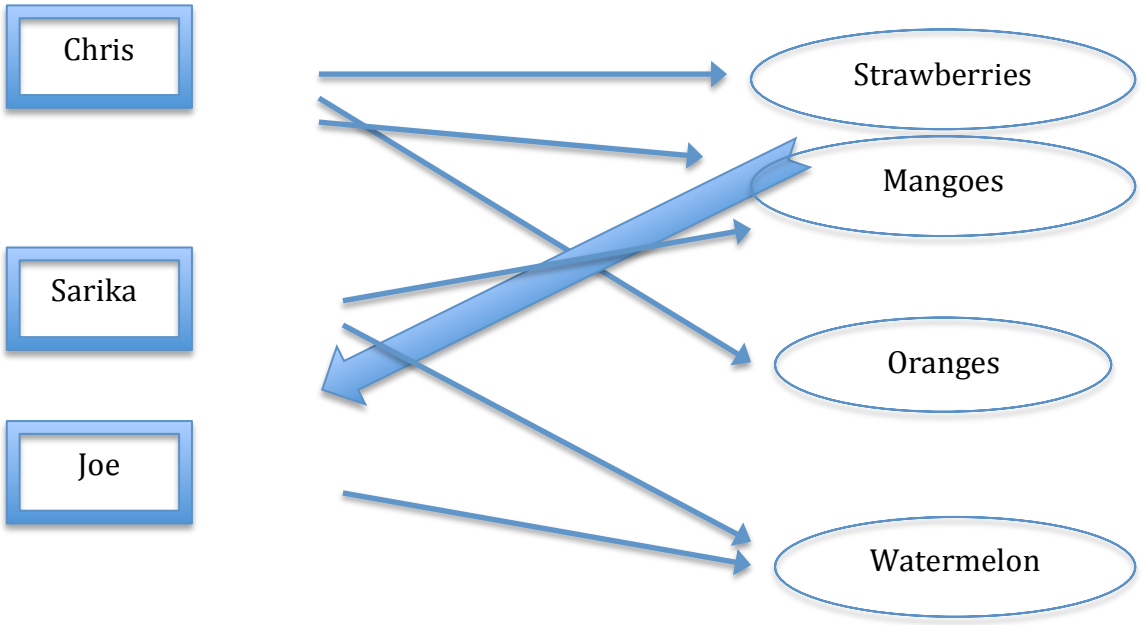


Fig. 2. Collaboratoive Filtering

**A RECOMMENDATION SYSTEM FOR YIOOP**

For my project, I will use collaborative filtering techniques to recommend to users about the threads they might be interested in. I will be suggesting to users a list of groups they might find exciting.

The project will have the following components: I plan to make use of the user history of Yioop and all the activities which each user might have done on the Yioop page — click data, browsing history, threads viewed (if this data is available) and apply some collaborative filtering techniques to recommend new groups to the users.

This semester I have been researching on the collaborative filtering techniques, tried to understand the Yioop framework, read various research papers to find out what would and what would not work for my project.

**Deliverable 1: Max Flow Problem**

The Maximum flow problem involves finding a feasible flow through a single-source, single-sink flow network that is maximum. I can use the bipartite maching problem to map users to threads. We can represent the Yioop dataset as a weighted bipartite graph where edges between users to threads will be weighted by some rating. The recommendation model can be used to predict those weights between users and threads. The edge between a user and a thread can be thought of as how important that thread is to a particular user. Based on these weights we can predict users with threads. However, I will not be using this for my project.

Let us define the maximum flow problem with the help of diagram. Figure 3 is a network with nodes which are labelled 's', 1 through 7 and 't'. Node 's' is the source and node 't' is the sink. It is a single-source, single-sink network. Each edge between two adjacent nodes has some weights assigned to it. Our job is to pass these weights through the network from one node to the other such that the weights are always flow. We call this as the flow. We have to find the maximum flow through the network.

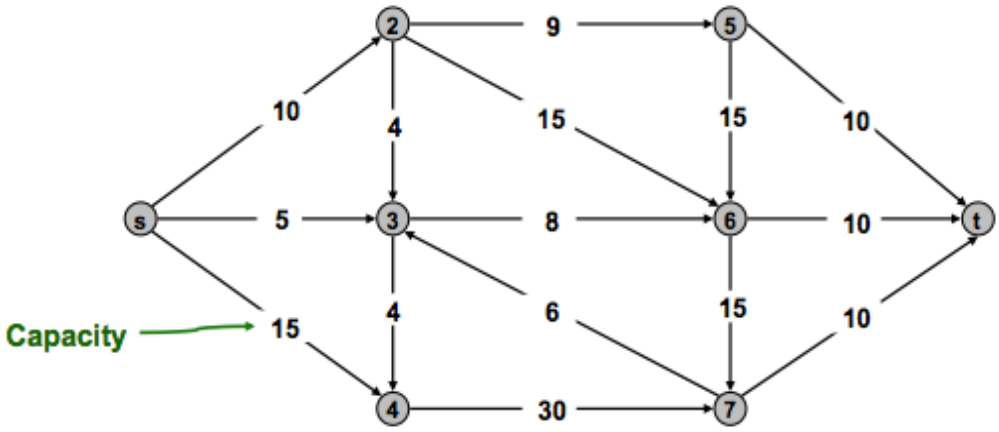


Fig. 3. Network Graph

I have tried to solve the maximum flow problem using two algorithms.

1.) Ford Fulkerson method - This algorithm selects a path in the network and finds the maximum flow through that path. The algorithm then augments on this path chosen and tries to find if some more flow can be passed and keeps doing it till there is no augmenting path available. This is not the best possible algorithm as it certainly depends on the path we have chosen first. However, there are many techniques for selecting the best augmenting path.

```
In [5]: %run "/Users/Sarika/Documents/SEM 3/Masters Project/Deliverable 1/Ford-FulkersonAlgorithm.py"

Enter the filename: graph.txt
This is Graph G:
[0, 16, 13, 0, 0, 0]
[0, 0, 0, 12, 0, 0]
[0, 4, 0, 0, 14, 0]
[0, 0, 9, 0, 0, 20]
[0, 0, 0, 7, 0, 4]
[0, 0, 0, 0, 0, 0]
This is Flow f:
[0, 12, 11, 0, 0, 0]
[0, 0, 0, 12, 0, 0]
[0, 0, 0, 0, 11, 0]
[0, 0, 0, 0, 0, 19]
[0, 0, 0, 7, 0, 4]
[0, 0, 0, 0, 0, 0]
This is Residual f':
[0, 4, 2, 0, 0, 0]
[0, 0, 0, 0, 0, 0]
[0, 4, 0, 0, 3, 0]
[0, 0, 9, 0, 0, 1]
[0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0]
Total flow:  23
```

Fig. 4. Ford-Fulkerson Algorithm Output

2.) Push - Relabel algorithm – This algorithm is very similar to how water flows i.e water always flow from a reservoir at a higher altitude to a lower altitude. In this algorithm, first all the node are assigned some height - source is given a height equal to the number of nodes and all the other nodes are given a height of 0. Our aim is to push the flow from a node at higher altitude – source of height 'n' to a lower altitude – sink at height '0'. The heights of the intermediate nodes keep changing as the algorithm proceeds.

There are two major operations in this algorithm:

Push operation: This operation basically tries to push an overflowing node at a higher altitude to a node at a lower altitude. This operation takes place only if the node has some excess flow than it can carry.

Relabel operation: This operation takes place if there is an overflowing node and it is at a lower or equal height than all its neighbors and has no node to pass the excess flow to. Hence we have to relabel the heights of the current node. The height of the current node is then raised by 1 more than the minimum height among the adjacent nodes.

I have implemented the Ford Fulkerson and the push relabel algorithms and it gives the maximum flow in the network. It also uses doctest module in python to test simple cases.

The maximum flow problem can be extended to solve the bipartite matching problem. A bipartite graph G = (V, E) is a graph in which the vertex set

V can be divided into two disjoint subsets X and Y such that every edge e in E has one end point in X and the other end point in Y.

```
In [4]: %run "/Users/Sarika/Documents/SEM 3/Masters Project/Deliverable 1/Push-RelabelAlgorithm.py"

Please enter the filename: graph.txt

Number of nodes in the network: 6
==================Push-Relabeled(Preflow-push) algorithm ==================
Maximum Flow for the network is:  23
```

Fig. 5. Push-Relabel Algorithm Output

**Deliverable 2: Collaborative filtering (Baseline Predictors)**

In this deliverable, I tried to build a recommendation system for Yelp by implementing a part of the algorithm used by the Netflix Competition winners. I tried to exploit the biases amongst users and restaurants to make predictions. For example: Some users have a tendency to rate all items poorly while some rate all the items generously. Similary for the items, some items have a very good quality as compared to some other items. I have tried to incorporate these biases while predicting a users rating for a particular item.

I have taken the Yelp Dataset and it predicts user ratings for a particular business/restaurant based on user reviews. The dataset was provided by Yelp for public. I have made use of only the Yelp reviews data to predict ratings. The yelp_academic_dataset_review.json file has 2.7M reviews and 649K tips by 687K users for 86K businesses. I have made use of the following features - "business_id", "user_id", "stars" for prediction. I have tried to predict the rating by computing the business/restaurant and user biases by aggregating the data. I have calculated the biases of users and businesses with respect to the overall average rating.

The program calculates the average overall rating followed by the user and restaurant bias. It saves these values in a dictionary. Once this training is done, based on the user and restaurant for which we want to predict, it adds these biases to the average rating to predict a users rating for that particular restaurant.

Example: Suppose we want to predict what user "Sarika" would rate the restaurant "In-n-Out Burger" and the overall average rating = 3.7, the users and the restaurants bias comes out to be 1.7 and -1.2 respectively then the predicted rating would be 3.7 + 1.7 -1.2 = 4.2.

```
In [7]: %run "/Users/Sarika/Documents/SEM 3/Masters Project/Deliverable 2/YelpRecommender.py" 7KoVg5QMjYu8taLFSE7hNA mYSpR_SLPgUVymYOvTQd_Q
User ID entered:  7KoVg5QMjYu8taLFSE7hNA
Restaurant ID entered:  mYSpR_SLPgUVymYOvTQd_Q
Sum of ratings:  10107148
Count of ratings:  2685066
Average Rating of Yelp:  3.76420840307
Predicted Rating:  5.0
```

Fig. 6. Collaborative Filtering (Baseline Predictors) Output

**Deliverable 3: Yioop Register Page Patch**

This particular exercise helped me to get familiar with the Yioop framework. I will be recommending threads to users in Yioop and would be rendering my recommendations on the Yioop page. Hence it is essential to understand the Yioop framework. I have made two patches to get familiar with it.

**Patch 1**

In this patch, I have added client side validation in the register page of Yioop. While a new user gets registered in Yioop he has to fill a form which has the following fields like first name, last name, email address, username, password, retype password and after a user fills these fields the user clicks on the submit button to get registered. The form is validated on the server side and if there are any errors then the user is warned about the errors. This consumes a lot of time and can also have some security issues. I have incorporated client side validation of the above form using .

The following client side validations have been implemented in Yioop:

1.) All the required fields of the form must be filled otherwise the submit button is disabled.

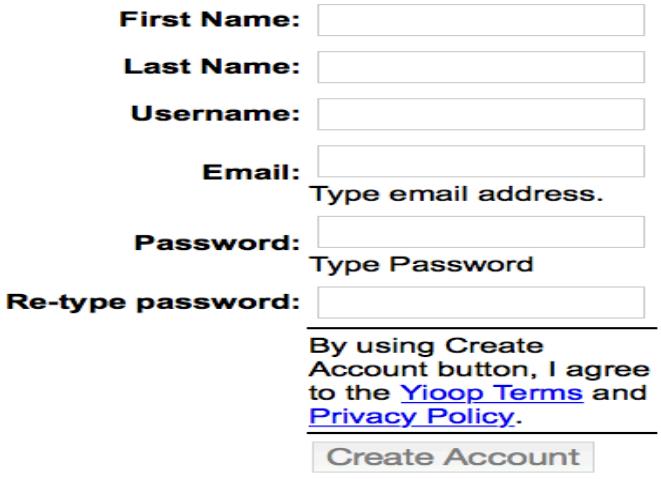


Fig .7. Register Page -Disabled Button Output

2.) If any field is touched by a user which is a required field and the user keeps it empty, the user is immediately warned to fill it.

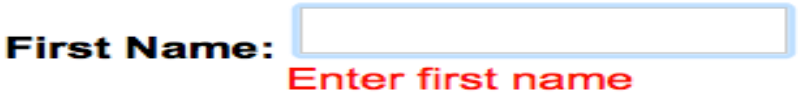


Fig .8. Register Page –Required field Incomplete Warning

3.) The email address field of the form is checked if it is in the proper format. For example: sarika.yahoo.com is an invalid email address whereas sarika.padmashali@yahoo.com is a valid one

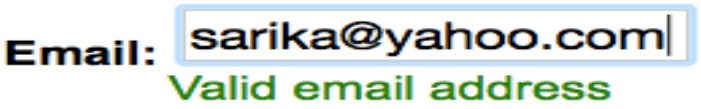


Fig .9. Register Page -Email Address Validation Output

4.) The password field should now have alphabets, numbers and must be at least 8 characters long. It checks the strength and warns the user about the strength of the password.

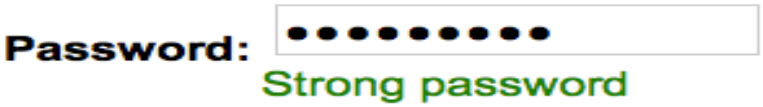


Fig .10. Register Page -Password Validation Output

5.) The retyped password should match to the one typed earlier else it warns the user.

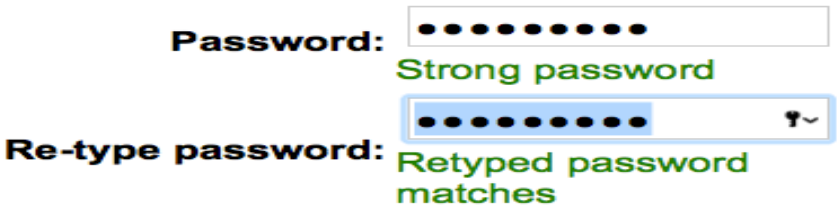


Fig .11. Register Page –Retyped Password Validation Output

**Patch 2**

In this patch, I have tried to incorporate a new filter using which the owner of the group can delete multiple users based on their join date. If a particular user or a group of users have joined a group long back and the owner wants to remove those users from the group then the owner can select all those

users and can delete them just at the click of a button. The joining date is displayed to the owner in the format mm/dd/yy.

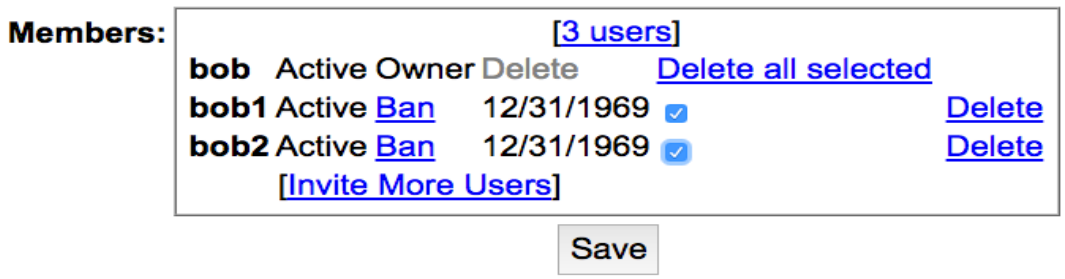


Fig. 12. Delete Multiple Users Based On Join Date


Fig. 13. Delete Multiple Users Based On Join Date (After Deletion)

The two patches in Yioop gave me an opportunity to play with the components, controllers and views part of the framework which I will be using in my project implementation.

**Deliverable 4: Collaborative Filtering (Latent matrix factorization)**

This is another technique which I have implemented and can be used for predicting a users rating for items. Collaborative filtering techniques which make use of user-based and item-based filtering are very intuitive and easy to implement, but it is not as effective as matrix factorization techniques. Matrix factorization techniques are more effective as it finds the hidden features which are responsible for the interaction between users and items.

As the name suggests matrix factorization aims at finding the factors of a matrix such that when you multiply it you get back the originial matrix — if not the original at least something closer to the original matrix.
Once we find these factors, we can now use these factors to predict the missing ratings of the items in the matrix I.e., we can now rate the items which had not been rated by a user earlier.

Matrix factorization basically tries to find the latent features about how users rate items. For example, a bunch of users might give high ratings to a particular restaurant because they like the cuisine served in the restaurant or they like the ambience of the restaurant. If we are able to find these hidden features then we can predict how a user might rate an item because there is

some kind of mapping between the attributes of the users and the attributes of the restaurants. Also the number of hidden features will be much smaller to the number of users/items. Like I said earlier, if every individual has a unique taste or every movie is unique there is no need for having a recommendation system.

I have used the same Yelp dataset. I have sampled the dataset as scalability is one issue which comes with matrix factorization as it is computationally expensive. We have some reviews data from Yelp and each record has some user ratings, user ids and restauran ids associated with it. The user and restaurant ids are in a random string format so I first converted it into some integers and kept that mapping in a dictionary. Now I created a matrix R which will be a matrix of users and restaurant ids. I have kept 0 at the places where the users have not recommended an item. The size of R will be $|U| \times |B|$ where U is a set of users and B is a set of restaurants. We now want to find K latent features. We now have to find the two matrices P and Q such that their prouct is R approximately.

P = ($|U| \times$ K matrix) Q = ($|B| \times$ K matrix)

$R \approx P \times Q^T \approx \hat{R}$

Now from the above equation we can say that each row of P is an association between features and users. Similary, each row in Q is an association between features and restaurants.
Hence now to predict how a user might rate restaurant we just take the dot product .

$$\hat{r_{ij}} = p_I^T \times q_j$$

I have used the gradient descent method to find the matrices P and Q. Gradient descent basically initilizes the matrix P and Q with some random numbers and then tries to find how different the obtained matrix is to R and tries to minimize the error iteratively. To minimize the error we have to know the direction in which we have to modify the current values. We are finding the gradient at the current values by differentiating with respect to both the variables. Once we find the gradient we can multiply it with a constant $\propto$ and add it to the current values. $\propto$ determines the rate at which we want the error approaching the minimum.

I have also used a regularization technique to avoid overfitting by adding a parameter $\beta$. It basically controls the magnitude of the user feature and the restaurant feature such that it approximates R better.

```
In [12]: %run "/Users/Sarika/Documents/SEM 3/Masters Project/100review.py" nEYPahVwXGD2Pjvgkm7QqQ mYSpR_SLPgUVymYOvTQd_Q
User ID entered:  nEYPahVwXGD2Pjvgkm7QqQ
Restaurant ID entered:  mYSpR_SLPgUVymYOvTQd_Q
The predited rating for the user id and restaurant id entered above is 1.39715987861
```

Fig. Collaborative Filtering (Latent Matrix Factorization) output

**WHAT DID NOT WORK?**

Hits Algorithm:

During my research about recommender systems I researched about HITs algorithm which is used for ranking web pages and found that HITs algorithm wouldn't be the best choice for a recommender system. HITs broadly classifies a document into two classes which are authorities and hubs. Authorities include pages which have relevant information whereas hubs are pages that link to authorities. A good authority is linked from many good hubs and good hubs are linked to many good authorities. For web pages each website is linked to other websites and hence we can calcuate the hub and authority scores. However for a recommender system we did not find appropriate mappings for restaurants and users. Hence we thought it wouldn't work well for our system.

**WHAT WORKED?**

I also read a few chapters from whose #1? The science of rating and ranking [1] and I found relevant chapters which I could use in my project. The ranking problem is similar to recommendation problem in many ways. One of the major similarity being – rating which is required both for prediction and ranking.

Massey's method and Colley's method are two algorithms which is used for ranking football teams. I plan to use these techniques for ranking the threads to users in the project. Also, single value decomposition is another matrix factorization technique which could be used in my project.

**CONCLUSION**

I spent this semester researching on the various techniques used for building a good recommender system. I feel this has built the foundation for my project and I know what are the places I need to work on in the next semeter.

For my CS 297, I undertood how a recommendation system works, what works and what would not work for my project. Through my deliverables learnt how can I apply the various techniques to my recommendation system. I also got an opportunity to understand the Yioop framework which I would be working on

and the research papers I read gave me an insight into the recommendation system.

For my CS 298, I will first start by collecting the huge corpus of Yioop data, will go through all the data cleaning and once I have the dataset ready. I will start applying the techniques I used in CS297. One of my deliverable couldn't scale well for millions of records and hence doing some matrix multiplication tricks to get around it would be something which I will be doing during the initial phase of the project. I will evaluate my model and see if I am receiving decent recommendations. The users will then be able to see recommendations of threads in Yioop discussion groups.

**REFERENCES**

[1] Langville, A. N., & Meyer, C. D. (2012). Who's #1?: The science of rating and ranking. Princeton: Princeton University Press.
[2] Pollett, C. "Open Source Search Engine Software!" Open Source Search Engine Software – Seekquarry. Retrieved on 13 Dec. 2016.
[3] Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). Introduction to algorithms. Cambridge, MA: MIT Press.
[4] Gábor Takács , István Pilászy , Bottyán Németh , Domonkos Tikk, Major components of the gravity recommendation system, ACM SIGKDD Explorations Newsletter, v.9 n.2, December 2007 [doi>10.1145/1345448.1345466]