**PittFood**: An Android Restaurant Recommender System

By: Song Jin, Chakravarthy Koduru, Steven Nunnally, & Yuwei Zhan

**Introduction**

The PittFood system provides a restaurant recommender system for users near the Pitt campus area. The system is based on an academic yelp dataset, which includes areas surrounding Carnegie Mellon University. After cutting down the dataset to our targeted area, we were left with 286 restaurants and 2255 users who gave 6,008 ratings of those restaurants. We were unable to connect users of our system to their Yelp user account, so this data is the basis for our Collaborative Filtering recommendation technique described below, to avoid the cold start problem. This dataset fills out less than 1% of the user rating matrix, so we definitely share in the sparsity problem experienced by many other popular recommendation systems, like Netflix. Due to this similarity, it was decided to use Singular Value Decomposition for rating prediction, which is just one of Netflix’s more successful recommendation algorithms.

The PittFood system has its own group of users who can add their own ratings of restaurants at registration. These ratings are combined with the yelp dataset described above. Users can then log in on their android phones and ask for recommendations. The system synchronizes with a MongoLab Database online to get the predictions and restaurant information which is updated manually. This information may be displayed in a list or map view and users can decide which restaurant they want to try. Below we will describe the recommendation algorithm in the Algorithm section. Then there will be an in-depth look at the interface in the Interface section, followed by a brief Contribution section detailing the contribution of each member. Finally, we will look at future work for the project to recognize improvements considered in the Future Work section.

**Algorithm**

The algorithm used in this project to make restaurant predictions for users is a Collaborative Filtering method using Singular Value Decomposition (SVD) to create predictions based on users who rated items similarly. This method was learned from documents from the websites of [1],[2],[3]. The idea is that users who agree on previously rated restaurants will agree on ratings for restaurants that they have not yet tried. The algorithm first accesses the online database to obtain the current ratting matrix. Each row represents a different user and each column represents a different restaurant. Based on the dataset described above, this matrix starts off as a 2255x286 matrix. Less than 1% of this matrix is completed so there are many blanks thus far. So the next step is to fill in these blanks with average ratings for each restaurant. If no ratings for a restaurant exists then the average is the average of the maximum and minimum scores for the dataset, or 3 in this case (max is 5 and min is 1). The matrix now contains the information necessary to use SVD to get predictive ratings.

The ratings matrix is decomposed into its SVD components, the U, S, and VT matrices. U is the user eigenvectors comparing similar user features, and VT is the restaurant eigenvectors comparing similar restaurant features. S is a diagonal matrix of the eigenvectors to determine the importance weight of each eigenvectors/feature. These values decrease from top left to the bottom right so that the important features are inherently sorted. These matrices can be further reduced in rank to decrease the computation time necessary to determine the predictive ratings for each user for each restaurant. This is called the rank-k SVD where the k is found by trying every k and choosing the smallest k with the best possible mean squared error (MSE), calculated from the difference between the predicted values and the ratings already given. This technique is called cross-validation. Finding the optimized k is an inefficient process, so it should not be found for each set of new predictions. Once the best k is found the SVD components are reduced to their k diagonals and the predictive ratings can be determined by US1/2\*S1/2VT. The algorithm deletes the old predictions on the database, and uploads only those new user restaurant prediction pairs for the PittFood users (i.e. ids denoted with a “pf\_” prefix) which are not already rated.

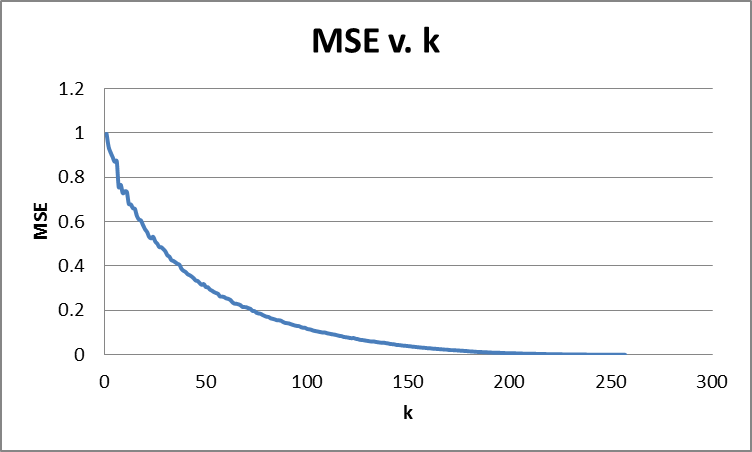


Figure 1: Plot of the MSE over all k values for the original yelp dataset

When testing for the best k on the dataset described above, the best k was 257, which was the maximum rank for S, with a MSE of less than 0.0000008. This shows that there are approximately 257 important features relating the users and the restaurants in the Yelp dataset. Figure 1 shows a plot of the MSE compared to the different k values tested. This decreasing trend is expected and can be advantageous if programming efficiency is determined more important than accuracy. Currently the predictions are made off-line so the efficiency of the program is not important and the k with the lowest MSE is used. As the dataset increases with the system’s users and if we decide to make predictions online, in order to give the most up-to-date suggestions, then the tradeoff between MSE and k should be considered more so that a k of 200 or even 150 may be acceptable from the plot in Figure 1.

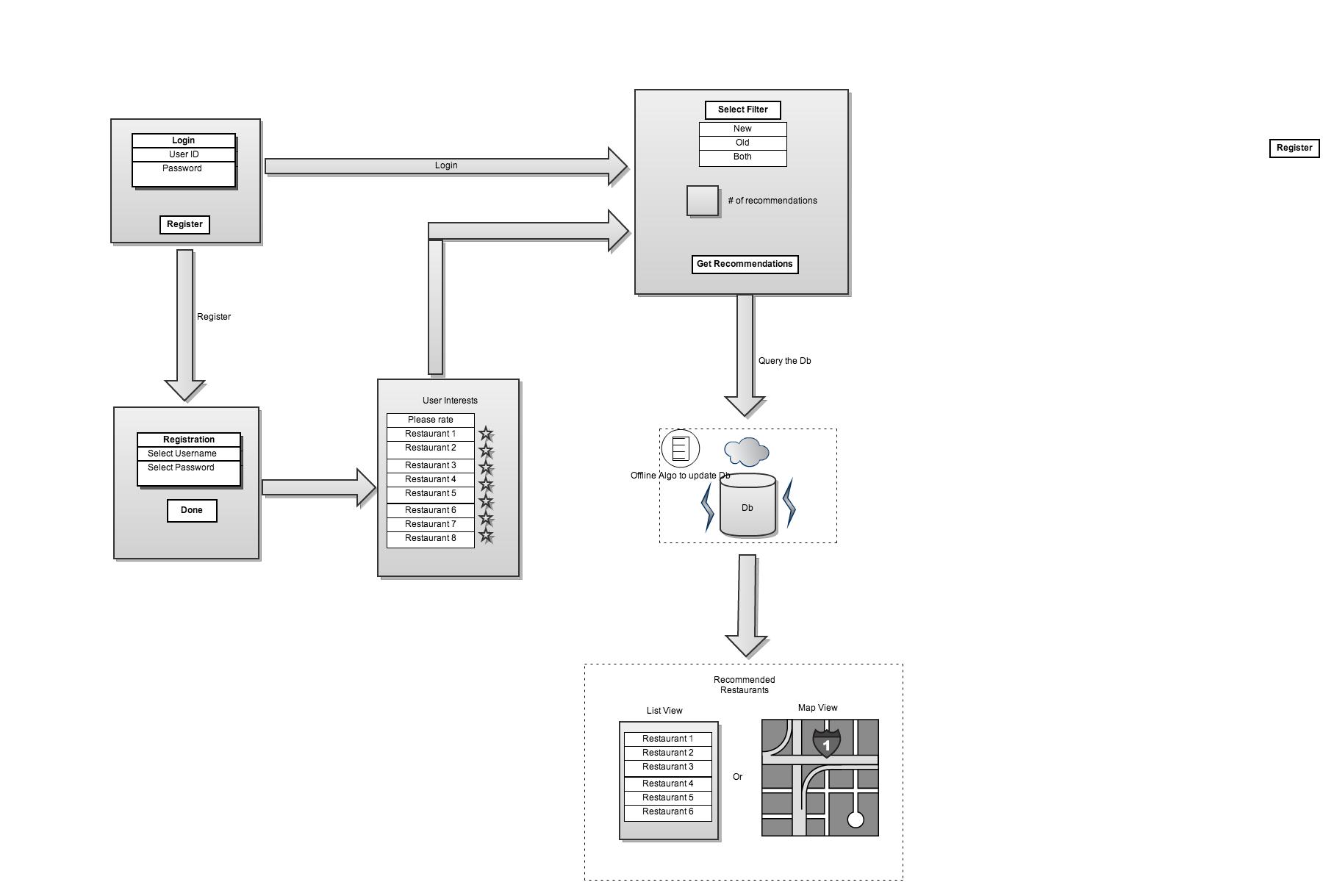
****

Figure 2: General system design and interface flow.

**Interface**

Figure 2 shows the design of the system. The program starts off with login or registration depending on if the user is returning or not. If the user must register then they fill out the information which is added to the MongoLab Database for future logins. Then, to defeat the cold-start problem for a new user, the interface gives the users an opportunity to rate several restaurants. If they do not then they will be given the average ratings for each restaurant as their predicted rating. Now we go to the recommendation screen, which is the same screen the users who login will start.

The recommendation screen allows users to ask for old (restaurants they’ve rated), new (restaurants they have not rated), or a mixture of both types of restaurants for recommendation. Then they also select a number of results they want returned for recommendations. The next screen will show a tabular display of either a map with all of the recommended restaurants shown and available for clicking on for more information, or a list view of the restaurants in order of best recommendation depending on the tab the user selects. All other Figures below show screenshots of each step.

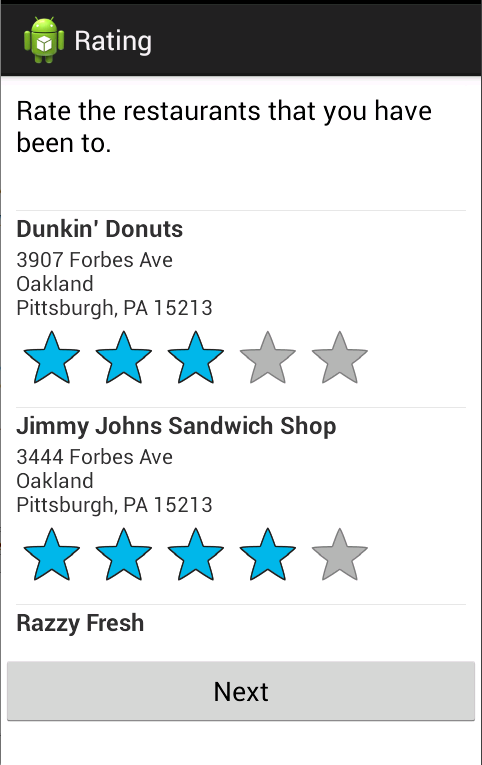
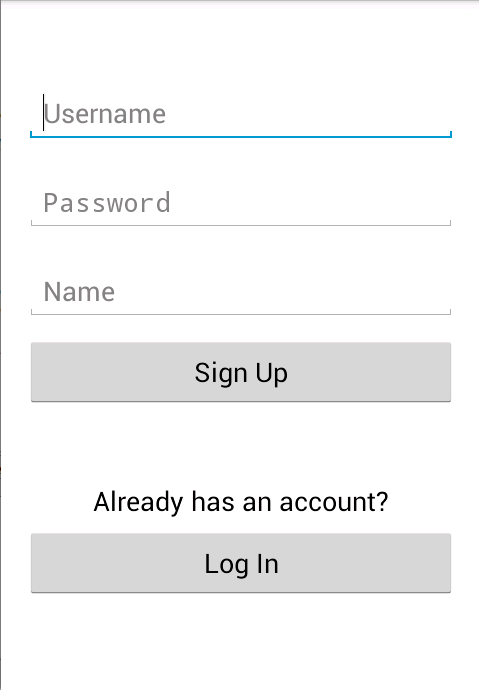
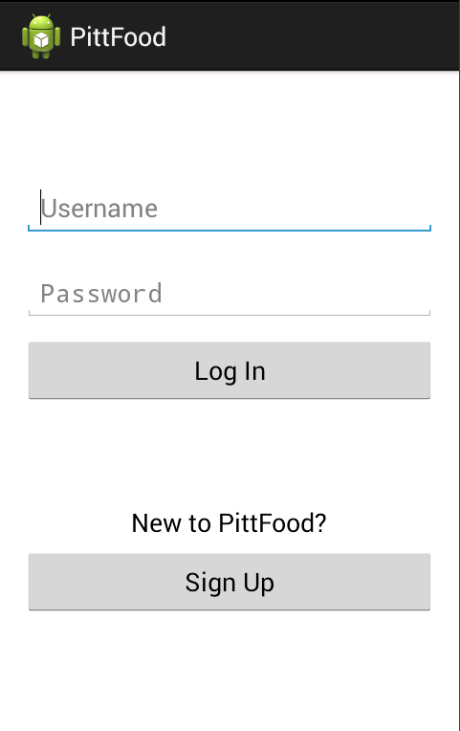
****

Figure 3: The login, registration, and beginning recommendation screens respectively.

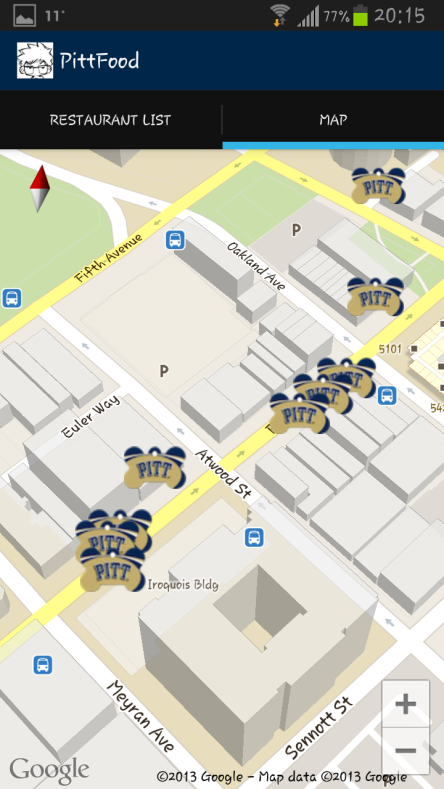
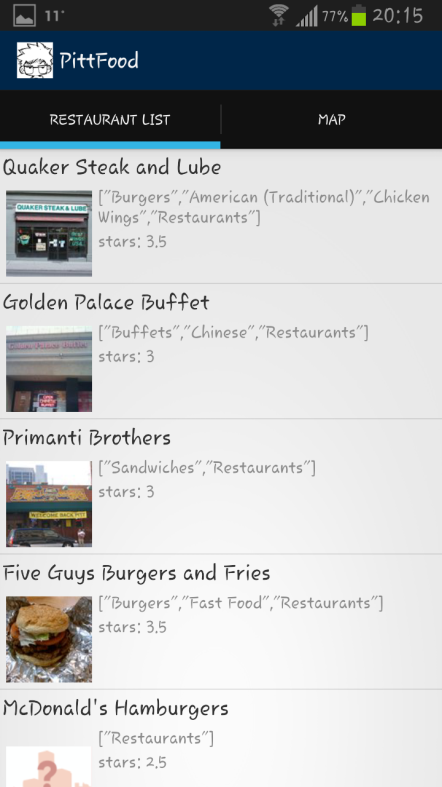
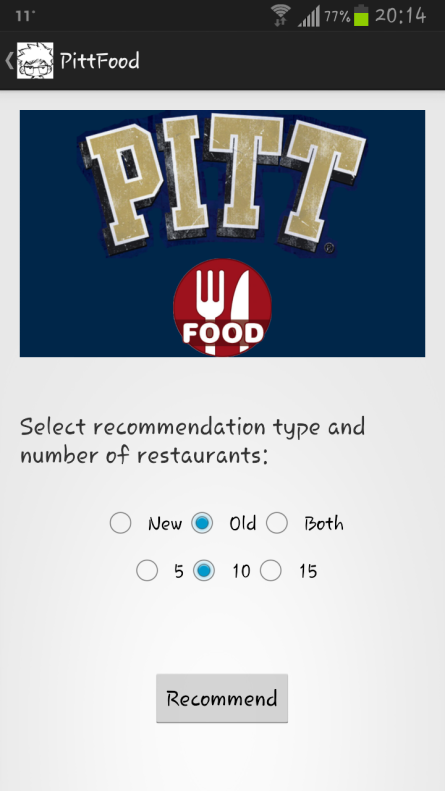


Figure 4: The recommendation, list view, and map view screens respectively.

**Contributions**

Song completed all of the android programming for the recommendation screen and the recommendation view screen, both map and list views. Chakravarthy helped clean the yelp dataset and uploaded it to the MongoLab Database. Steven programmed the offline SVD method for predictions, as well as the training program for k. Yuwei did all of the programming of the login and registration screens. All members were involved in the design and helped each other with problems.

**File Layout**

All of the code can be found on <http://code.google.com/p/pittfood/>. Give your Google email account to [smn34@pitt.edu](mailto:smn34@pitt.edu) and you will be added to the project so that you may download the source. In this home folder there are important documents. DatabaseDesign.txt is the design of the database on MongoLab as well as the username and password to log in (<https://mongolab.com/databases/yelptest>). k.txt is the file containing the k trained from the FoodRecSVDTrain program. Report.docx is this paper and Presentation.pptx is the final presentation given on the last class. AIS\_PittFood\_Android\_build contains the Project that Song wrote for his contribution. PittFood contains the Project that Yuwei wrote for his contributions. FoodRecSVD is the prediction program and FoodRecSVDTrain gives the plot at Figure1 and the k with the smallest MSE for FoodRecSVD to use. RaterParser parses the ratings from the yelp dataset into the online database, although the yelp dataset is removed from the folder because it is too big for uploading to the repository (but can be downloaded at <https://www.yelp.com/academic_dataset> with an appropriate YWSID). EarlyPres contains the presentation material for the first presentation where the idea was first presented to the class in the middle of the semester. Chakravarthy’s work of parsing and cleaning the restaurant data was done by hand and only shown in the online database.

**Future Work**

There are many improvements that we have identified for this system that is outside the objectives of this class. We would like to add restaurant searching and cuisine filtering in order to target certain recommendations or restaurants to rate. We would also like to add the ability for the user to modify ratings in case their preferences change. Currently users can only add ratings at the time of registration and these ratings cannot be changed or added to. This system only gives recommendations off of these original ratings. We would also like to change the format of the recommendations at registration where we give them random restaurants, one at a time, asking what they would rate it or if they have not eaten there. This would loop until we have no restaurants left or some x number of restaurants are rated. This x would be empirically shown to be a minimum x required to receive improved recommendations. We would also like our users to be able to add a restaurant not currently in our database. This would make the system much more usable since businesses change over time and this could also expand our area of coverage over time. Also, with more time, further empirical data should be collected on the recommendation system as cross validation is often biased and not always a great measurement. User studies and questionnaires could be given to determine the effectiveness of the system and other algorithms could be tested to determine which works more efficiently and more accurately. Finally, predictions are only updated when the program is manually ran from a computer. This should be more autonomous, either by scheduling a time to run both the prediction program and the k training program or by linking it to adding new ratings or users so that the predictions contain the most up-to-date recommendations, using all of the ratings data. The other option would be to make the prediction code much more efficient and updating the predictions online, which would take careful consideration of the k accuracy/efficiency tradeoff.

**References**

1. <http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/svd.html>
2. <http://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/>
3. <http://www.slideshare.net/bmabey/svd-and-the-netflix-dataset-presentation>