Project: PopCore

Name: Ke Wang (kw427)
Advisor: Dan Cosley

1. Goal

Social Recommendations
The goal of the project is to get familiar with building a social recommender and implement algorithms for recommendation and visualization. This involves understanding preference data from users of our social recommender system (PopCore), thinking of creative ways to visualize preference data, modeling or designing algorithms for recommendation, and finally, implementing some of them on the live system.

2. Preparation

Read through all previous code to get familiar with the overall structure and the data retrieved from Facebook.
Applied typical machine learning algorithms to find interesting patterns from the data.
Given these initial observation and result, came up with a plan about my general goal on the PopCore project throughout this semester.

3. Implementation:

Since this is more like a research project and there is no certain result that I could expect, I had to try different approaches to find the most important features for a recommendation system. I have used classical statistical approach like kNN, k-means clustering. By analyzing the results iteratively, I have figured out some potential features.
The most important take-out from my project is that the timeline feature may act as an important factor in recommendation. The timeline stands for the relationship between like date and release date, or the relationship between like date and different users.

a) input data format:

There are two kinds of formats, one is <user_id, item_id, release_time, like_time> and the other is <user_id, friend_id, item_id, release_time, like_time>. The latter one not only specify the friendship but also information about the friends' like.

b) two types of TIMELINE feature:

1) Time difference between like_time and release_time

- **Filtering the data**

Quite a lot of items were released in early days while Facebook launched in 2004 and user can only click like button of the items after that. In that case, those items differs heavily in the releasing date while the liking date may be similar (e.g. user like the item on the day he joined facebook)

Use the items that were released after the users' join date (say the first like_time). Then we can analyze how users will response the newly released items.

- **Each item**

Each item has different like_time for different users. We define the new variable as time_diff = like_time - release_time. Then for each item, we can get the list of time_diff of different users, and calculate the average time difference between like_time and release_time. That is, each item has a corresponding tuple as <release_time, avg(time_diff)>. We draw the figure of the tuple, and do k-means clustering of similar items.

- **Each user**

Each user would like different items at different time. Just as above, we define the new variable as time_diff = like_time - release_time. Then for each user, we can get the list of time_diff for different items, and calculate the average time difference between like_time and release_time. That is, each user has a corresponding tuple as <user_id, avg(time_diff)>. We draw the figure of the tuple, and do k-means clustering of similar users.

- **Plotting**

Plotting can show how frequently a user would like an item as well as how users' behaviors differ for the same item more intuitively. Here I only put one figure of the tuple of <user_id, avg(time_diff)>.

Sorting the time_diff in descending order and hash the user_id to the range from 1 to the number of all users. We can draw the figure as below:
x-axis represents the hashed user id (which is hashed to 1- 3396), and the y-axis represents the sorted average time_diff of each user (which is 1933.68 days to 1.25 days).

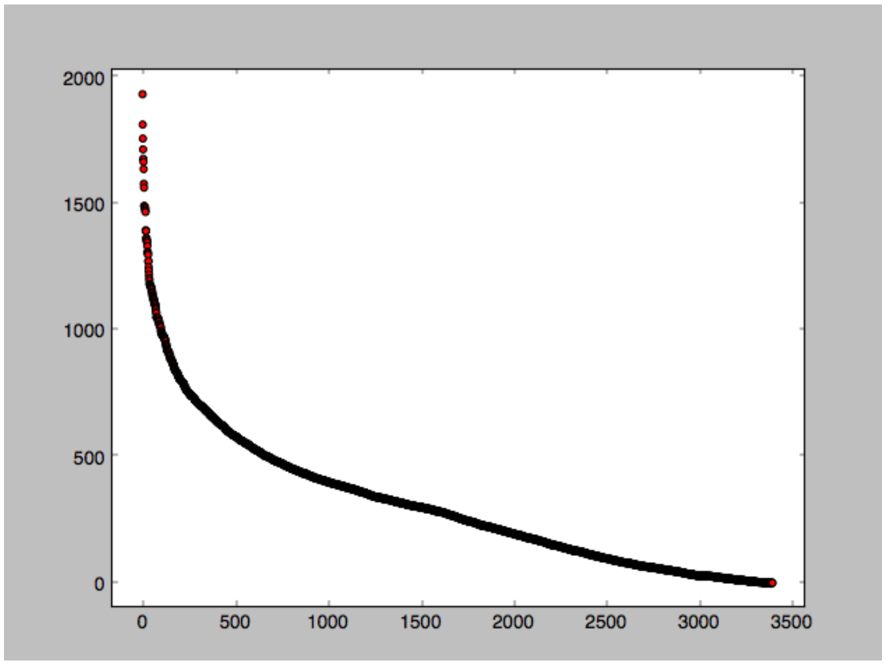

Figure 1: user average time difference

2) Time difference between like_time of a pair of friends:

- **Background:**

A user will influence or be influenced by his friends' like or dislike. We can use this as one factor to do recommendation.

We may say user A likes item1, then his friend B likes item 1 within a small time period. It is probably that B likes this item because A likes it (maybe A “influence” B on item 1). Based on this idea, we can find how a user can influence his friends' behavior.

- **Friend Pairs:**

We can find all the pairs of friends, and lists of their liking items.

We may not need to filter the data this time because we only depend on the time_diff between the friends' like_time regardless of the item release_time. Here I put the figure of the tuple of <friend_pair_id, avg(time_diff)>.

Sorting the time_diff in descending order and hash the friend_pair_id to the range from 1 to the number of all friend pairs. We can draw the figure as below:
The x-axis represents the friend pairs id (which is hashed to 1-237) and the y axis is the sorted average time_diff (1-2.98) because I set the time bound as 3 days.

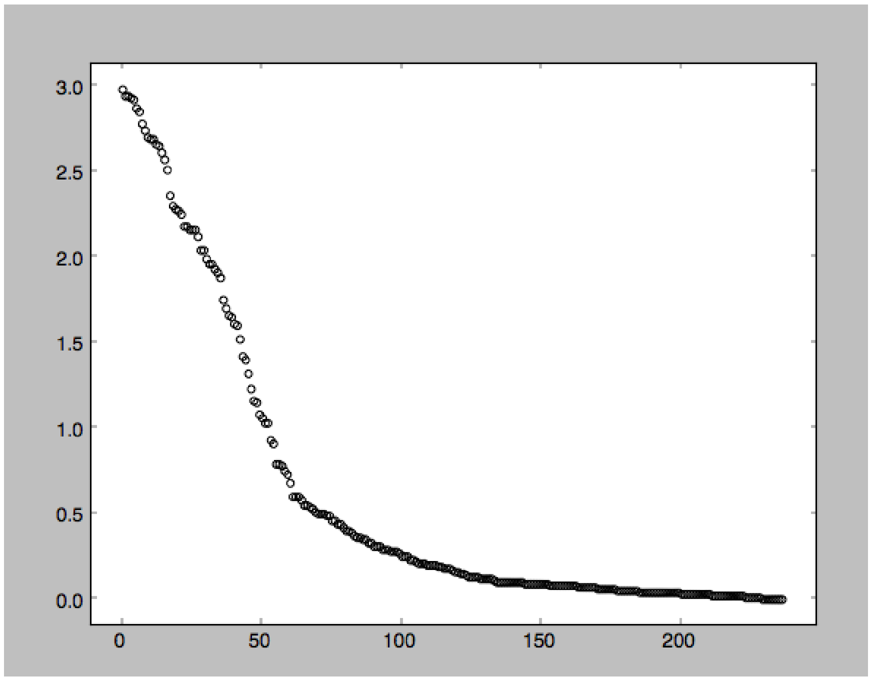

Figure 2: friend pair average time difference

- **Rank the friends of each user:**

For each user, get a list like [uid] : {<f_1,1/t_1>,...,<f_n,1/t_n>}, where t_i represents the average time_difference between the user and friend f_i . Use the inverse of the time difference since the larger the time difference, the smaller probability one would influence another. Finally output dictionary as: [uid]: {<f_1,p_1>,...,<f_n,p_n>}

Within the friends, we can normalize the probability as

$$p_i = \frac{p_i}{\sum_i p_i}$$

- **"Influence" rate between friend pairs:**

For each friend pair, not all common items meet our time period requirements, which means these items are not "influenced" by the friends. So

$$\text{"influence" ratio} = \frac{\text{\# of required common items}}{total\ \#\ of\ common\ items}$$

- **"Influence" proportion:**

For each user, we care about how many of their liking items are "influenced" by their friends. We suppose if the user has large common items with his friends, it may be more likely that he will have more items "influenced". So we define the "influence proportion" as

$$\text{"influence" proportion} = \frac{\#\ \text{of influenced item}}{total\ \#\ of\ user's\ like}$$

- **Jaccard Coefficient:**
  For each friend pair:

$$Jaccard(U1, U2) = \frac{|U1 \cap U2|}{|U1 \cup U2|}$$

We also have average time_diff for each friend pair <pair_i, t_i>, this time_diff can act as the score that is a globally variable so that the score can be compared across the friend pairs. We normalize the time_diff as

$$t_i = \frac{t_i}{\sqrt{\sum t_i{}^2}}$$

- **Plotting:**
  We try to plot the Jaccard Coefficient and "influence" proportion in the same figure to show the similarity or difference between them.

x-axis represents the hashed friend_pair_id;
y-axis is the value of the two ratio where red is normalized score of "influence" proportion, and blue is Jaccard Coefficient.

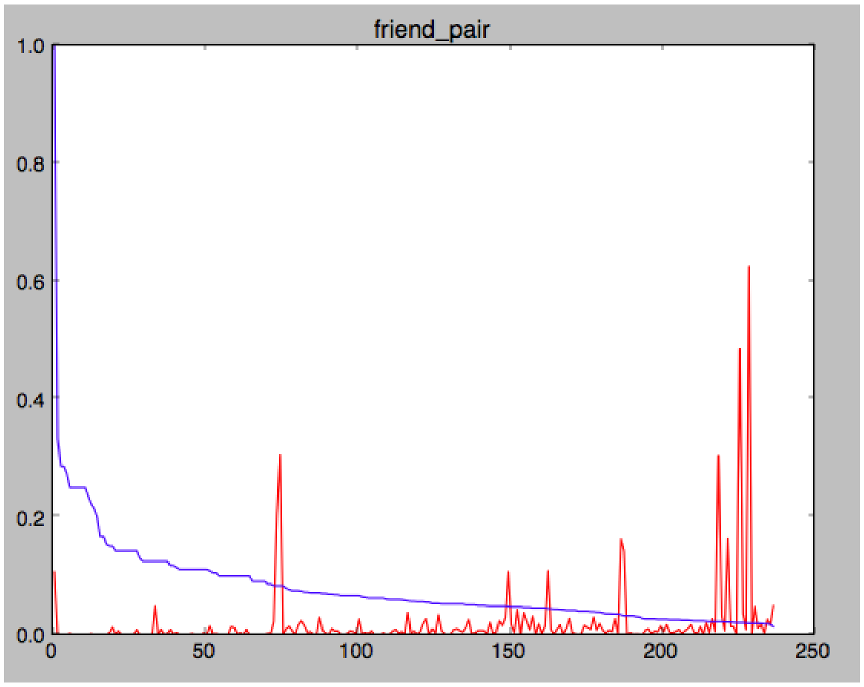

Figure 3: relationship between Jaccard Coefficient and "Influence" Proportion

- **Result:**

From our results, we have shown that the timeline feature is correlated among a group of friends, therefore we could use the timeline feature to give recommendation to users based on their friends. The larger the Jaccard Coefficient, the smaller the normalized score.

It is also possible to assign a weight to this feature and added it into the original recommendation system, making it more efficient.

4. Conclusion

I believe my work in the PopCore project throughout this semester is successful. Our results have indicated a new feature (timeline) that can be used to improve the recommendation system. At the same time, I really learned a lot of new stuff, both technically and mentally, and gained precious experience.

5. Acknowledgement

I really want to thank my advisor Prof. Cosley who has given me quite a lot of suggestions and instructions on the overall project as well as quite useful feedbacks to help me improve my work. I also want to thank Amit who is my mentor on PopCore. We meet every week to go over what I have done during the past week. Specifically, we have focused on the measurement of the results generated by our approach, the feasibility of the idea and method we used, and modification and renovation of existing algorithm or legacy code.