Meetup.com Recommender System

The Problem

Everyone has experienced the struggle of being unable to make simple decisions throughout their daily lives. Whether this is choosing where to eat on Yelp, what to buy on Amazon, or even who to connect with on Facebook or LinkedIn, many of us spend an enormous amount of time going through catalogs before making a decision. In fact, most of us spend a majority of our time parsing through catalogs for new and exciting options to try. Luckily, there are recommendation systems that can help us simplify our search and decision making. In short, recommendation system provides an individual a handful of choices from an existing pool of options that caters to users' preference. For this project, the goal is to build a recommendation system and recommend users to products they may like. Specifically, the recommendation system will be developed for Meetup.com, a socializing website that connects individuals by allowing them to join groups and attend events regarding a subject that they have or may have interest in. The idea will be to recommend users to join and participate in other groups that they have yet to discover.

Recommendation Systems

Recommendation system can be separated into two approaches, content-based and collaborative filtering. Each approach has their own strengths and weaknesses. Both approaches will be briefly discussed in this section.

Content-Based

Content-Based recommendation system focuses on developing a profile for every item using a set of features to characterize all items in the catalog. In doing so, the recommendation

system can compare items that are similar and recommend new ones to the user. The idea is if a user liked a particular item, the user will also like an item similar to it. For instance, movies are often characterized by the genres, director, and actors/actresses. If users liked movies that were directed by Steven Spielberg or movies that has Leonardo DiCaprio as an actor, then more movies that were directed by Steven Spielberg or movies that involved Leonardo DiCaprio will be recommended.

Strengths

One advantage content-based has over collaborative filtering is that it doesn't suffer from the cold start problem. The cold start problem stems from the fact that in collaborative filtering, recommended items are based on ratings or feedback from other users who have interacted with that item. If a new item is added to the catalog, it will take some time before the new item gets enough feedback to be recommended. With content-based, any new item added to the catalog can still be recommended to users as long as they share similar features that a user likes. Also, a content-based approach is transparent for an explanation. Recommendations to the user can be easily explained and traced back due to these similar features.

Weaknesses

A pitfall with this approach is that it doesn't provide any surprises to the user. Any item recommended to a user will be similar to the ones the user has dealt with in the past. As a result, if a user is looking for something unexpected, content-based will be hard to deliver. Another issue with content-based is that it may be difficult to gather features to characterize every item in a catalog.

Collaborative Filtering

Collaborative filtering can be separated into two methods, memory-based and model-based. Memory-based collaborative filtering uses a similarity measure to determine how closely related users are to each other and recommends items to a particular user that a similar user has liked. For example, if user A and user B have seen and liked a set of movies, then user A and user B are said to be similar. As such, user A will be recommended movies that user B has liked that user A has not yet seen and vice versa. To expand the recommendations, user B can be increased to Top-N similar users, where N is the number of similar users, providing a neighborhood of N similar users. The concept of this approach is that users that have similar preferences now will also have similar preferences in the future. One key difference from content-based is that collaborative filtering does not require any knowledge about the item, just how other users have responded to it.

Model-based collaborative filtering still captures the essence of collaborative filtering by using only feedback on items and nothing about the item itself but has a more machine learning approach. There is a cost function to minimized and parameters to tune. Although memory-based is more intuitive and easier to implement, it's not scalable like the model-based. This will be explained further in Implementation of Recommendation System section.

Strengths

There isn’t any need to create a set of features to characterize all items in the catalog. As a result, it can be easily implemented.

Weaknesses

As stated above, collaborative filtering suffers from the cold start problem. Without the right amount of user feedback on an item, that item may never be recommended at all. Also,

there isn't much transparency. Items that are recommended can only be traced back to what similar users have liked in which some cases can be a bit random.

In the end, determining what approach to use depends on what information is present and what problem is being solved.

Data Wrangling

To develop a recommendation system for Meetup.com, a dataset uploaded onto Kaggle gathered from Meetup's API was used. The dataset contains information on groups, events, and members in these three cities: San Francisco, New York City, Chicago. As the data was separated into many CSV files, each one was explored. While some tables contained useful information for this task, some tables were redundant. As a result, only the following tables were used: members, groups, category topics, group topics. These tables were joined in various ways to build the utility matrix that will help develop the recommendation system. Also, since the data only contains users and groups for three cities, the data was partitioned as such.

To develop a recommendation system using content-based approach, the groups, group topics and category topics tables were joined together on unique group id. These topics will be used as the features to characterize what the group is about. Any other information from these tables is dropped. In the end, the final table will consist of distinct groups in the rows and specific topics in the columns. Each element in this table has a binary representation where a one will indicate that the group is characterized by that tag and a zero otherwise.

For collaborative filtering, the members table was used to create a utility matrix in a user by group format. As collaborative filtering doesn't require any information about the item but only the users' feedback and interactions with the item, any other information is dropped. In this matrix, each element contains a binary representation to indicate whether a user has interacted

with that group. Another way to represent the interactions is to use the time delta in months between the users last visited timestamp and the users joined timestamp. The last visited timestamp records the last time the user visited the group page on Meetup. The joined timestamp records the time the user joined the group. The bigger the difference indicates the user has a greater preference and liking for this group. Both of these representations will be used.

Exploratory Data Analysis

To develop a better sense of the data, exploratory data analysis was performed on the dataset. As stated above, the dataset only contains members and groups from San Francisco, New York, and Chicago. Each city has a total of 8576, 4574, and 3180 groups respectively. These groups range from various categories such as Career/Business to Sports. Figure 1 shows a bar graph of the number of groups per category. It can be seen that the top three categories are Tech, Career/Business, and Socializing for all three cities. Of all of these groups, Figure 2 shows the Top 30 Most Popular Groups by the number of members participating in the group. From the title of the groups, they range from Tech, Socializing, to Outdoor/Adventures.

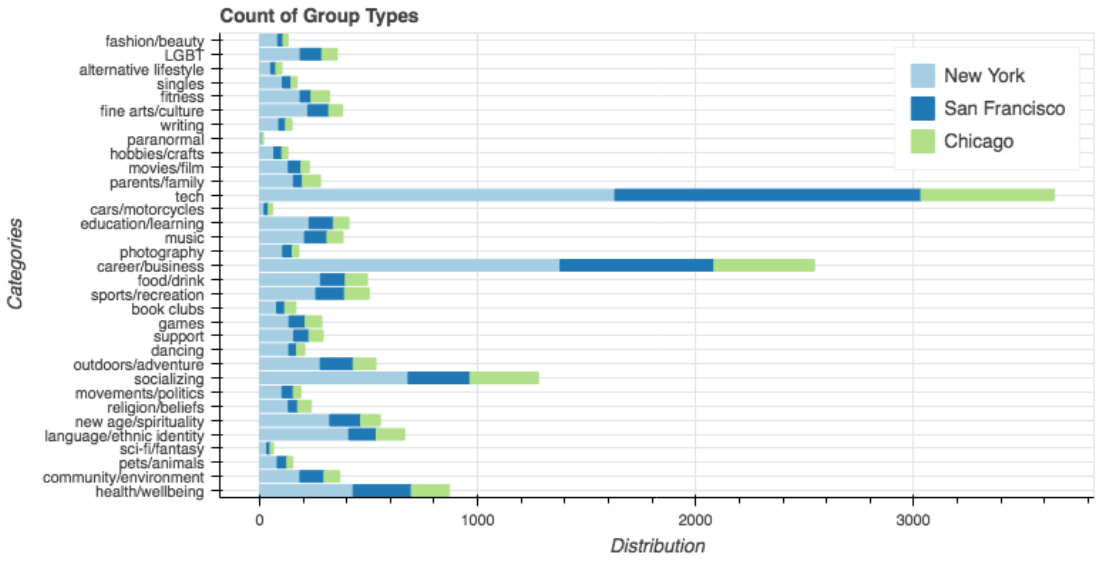


Figure 1 Distribution of Groups by Category

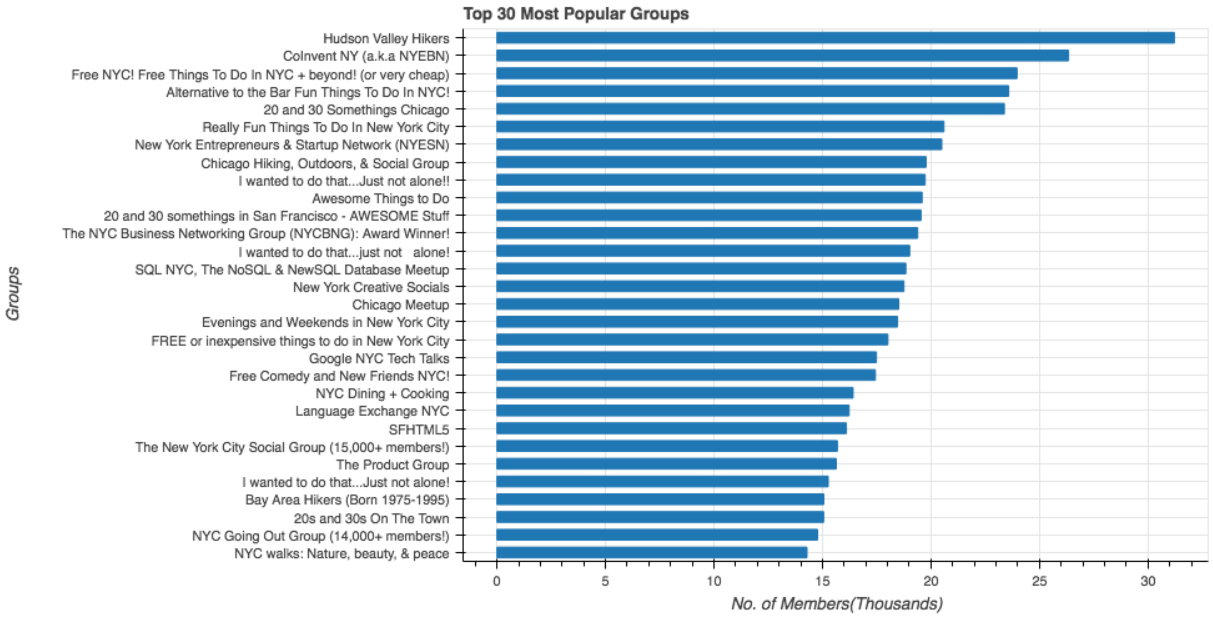


Figure 2 Top 30 Most Popular Groups

One question that arises after seeing Figure 1 was when did these top categories become popular and were these trends similar in all three cities? This question can be addressed by plotting the increase in groups for the top three categories by year. From Figure 3, it can be seen that all three cities seem to have the same trends in terms of the rise in all three categories. For San Francisco, it seems the number of groups in the Tech category continues to dominate. This is consistent with the fact that San Francisco is located close to Silicon Valley and the tech hub. As for the other two cities, New York and Chicago, the increase in socializing and career/business groups seem to be the bigger trend in recent years. This shows that New York and Chicago are more diverse than San Francisco in terms of people interests.

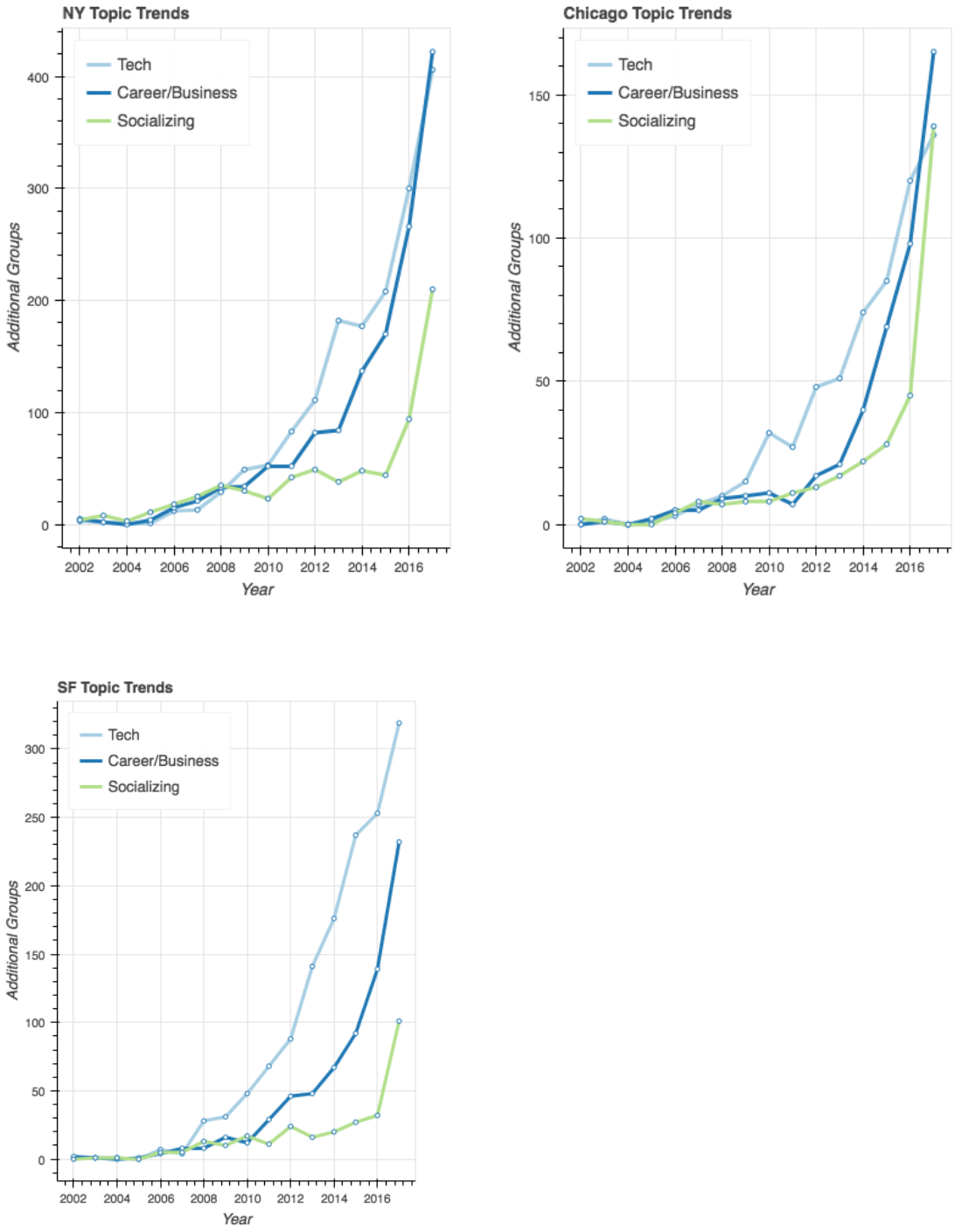


Figure 3 Group Trends

Aside from the most popular groups and categories, couple other questions that can be asked are which group were the most active in the past 7 years? How does this compare with the most

popular groups? These questions were addressed after plotting Figure 4 which shows the number of events a group has hosted in the past seven years. It can be seen that the most active group of these three cities was Chicago Toastmasters with a total of 1061 events hosted. Comparing Figure 4 with Figure 2 shows that being the most popular group did not equate to being the most active group. The only group that was part of these two lists was the Chicago Meetup group.

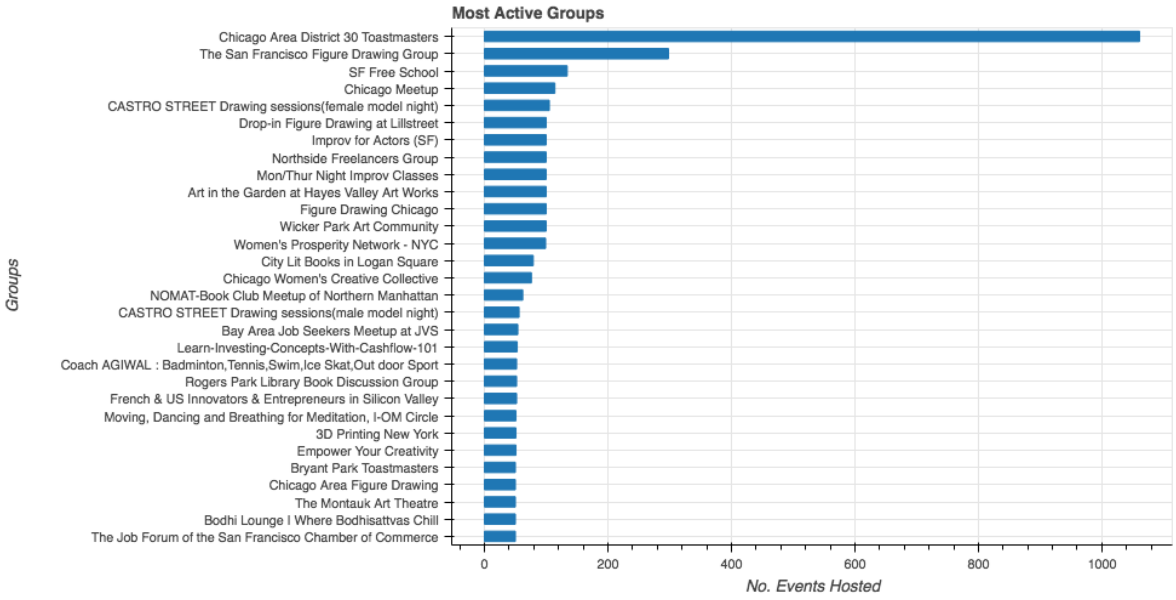


Figure 4 Active Groups

Lastly, to see user activity for all three cities for each year, Figure 5 plots a line graph that shows the number of members that joined a group during that year. It can be seen that the year of 2014-2015 had the most user activity for all three cities for Meetup.

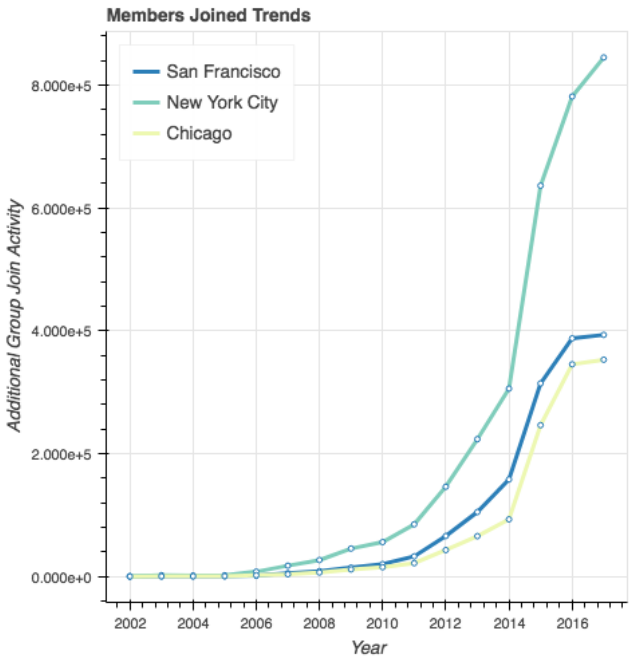


Figure 5 Group Activity Trend

Statistical Inference

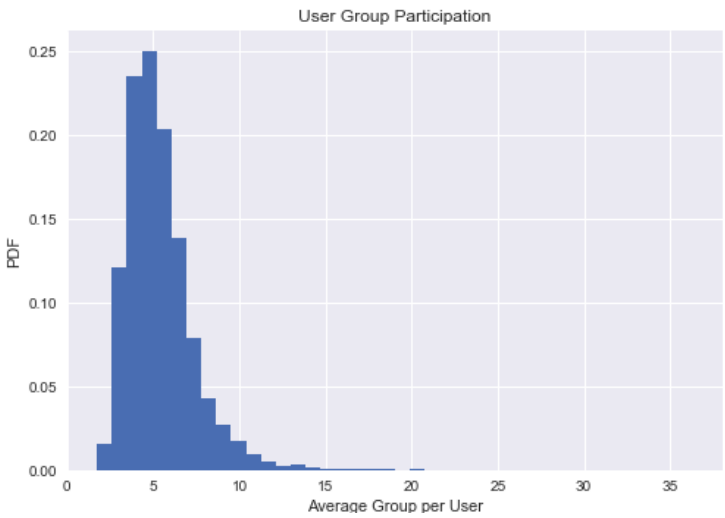
Recommendation system provides a handful of choices from a catalog that caters to the user's preference and taste. The goal in doing so is to maximize user experience with the service that will keep the users active. To determine whether a recommendation system is increasing activity, a benchmark will be needed. This can be done by finding the current average groups a user participates in.

Given that the dataset at hand only contains users from three cities, the true population average of groups a user is in cannot be determined. However, this value can be estimated using the sample mean as the estimator. Using this estimator, a sampling distribution of the sample mean can be plotted. The mean of this sampling distribution can then be used as an estimate for the average number of groups a user is in. Once this estimate is determined, a hypothesis test can be performed using a one a sample t-test to validate this estimate. The average number of groups a user participates in was determined to be 5.

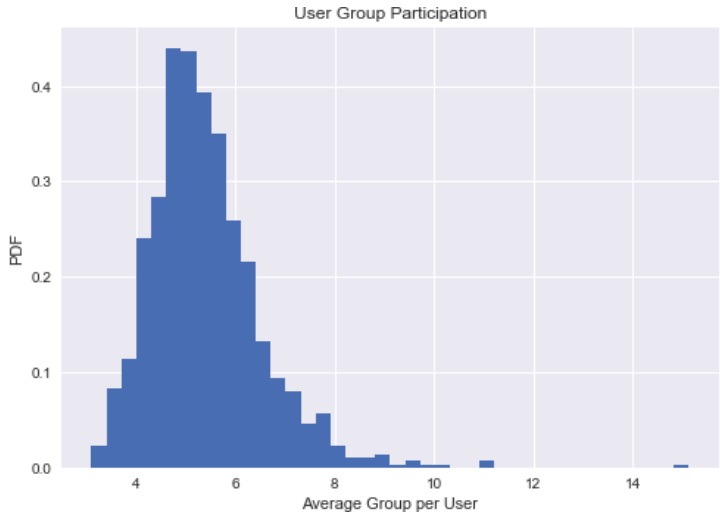
To begin the hypothesis testing, the null and alternative hypotheses are stated. The null hypothesis will be that the average number of groups that a user participates in is 5. In contrast, the alternative hypothesis is that the average number of groups that a user participates in isn't 5. With the significance value set to .05, if the calculated p-value from this test is less than the significance value, then there will be strong evidence to reject the null hypothesis and accept the alternative hypothesis. Otherwise, the null hypothesis remains to hold.

This hypothesis test was done twice. For the case when sample size N was set to 30, the calculated t statistic was .05787 and the p-value was determined to be .95. This means that assuming the null hypothesis is true, the probability of getting a sample mean of 3.4 is 95%. For the case when the sample size N was equal to 100, the t statistic was -1.717 and the p-value was calculated to be .09. In both cases, the results conclude that there isn't enough evidence to reject the null hypothesis and that the average number of groups a user participates in is 5.

Figure 6 shows the sampling distribution of the sample means for both cases. It can be seen that there is a positive skew in both distributions. However, as the sample size increases, the positive skew seems to decrease which is consistent with the central limit theorem which states the sampling distribution converges to a standard Gaussian distribution as the sample size continues to increase.



N = 30



N = 100

Figure 6 Sampling Distributions

In summary, the goal of this project is to develop a recommendation system using data from Meetup.com to personalize and recommend users groups that they might be interested in. It was found through statistical inference that the average number of groups that a user participates in is 5. Perhaps this number can increase with the help of a recommendation system.

Implementation of Recommender Systems

As mentioned above, there are two main approaches to building a recommendation system, content-based and collaborative filtering. For this project, both of these approaches were implemented.

Content-Based

The content-based approach focuses on developing an item profile for each item in the catalog where selected features are used to characterize items. For this data, the attributes of the

groups will be the topic and category tags. These tags help describe what each group is about. These tags vary from 80s-dancing, 90s-music, to sports or pets-animals. Using these tags as features, each group can be treated as a row vector where a cosine similarity measure can be used to determine how similar each items are by the angle of these row vectors.

$$cos(x,y) = \frac{(x \bullet y)}{||x|||y||}$$

Items most similar to another will be recommended. There are other measures that can be used to find similarities such as Jaccard Similarity or the Pearson correlation. Once the scores are computed amongst all groups, the 5 most similar groups are recommended. The following shows a couple of the recommendations given an item. The list of groups on the left was recommended when the item NYC Pit Bull Group was inputted and the list of groups on the right was recommended when Alternative Health NYC was inputted. As shown below, the recommended options are within the same category.

Harlem Meer Mutts Club
Brooklyn disk dogs
NY & NJ(Waterfront) Miniature Pinscher Meetup
Me & My Best Friend Hiking Adventures
THE NEW YORK CITY MALTESE MEETUP GROUP

Recommendation for NYC Pitbull Group

The New York Chakra Healing Meetup
Living Energy- Global & Local Wellness Community
The Herbalists Meetup Group
Say YES to Your Life, Manhattan!
Cosmos Tree

Recommendation for Alternative Health NYC

Collaborative Filtering

Memory-Based

The memory-based collaborative filtering method is similar to the content-based approach above. The difference here is instead of calculating the cosine similarity scores between groups based off of topic and category tags, the similarity score is used to compute a

handful of similar users based on their interactions. Groups are then recommended base on what these neighbor users have joined that the input user has not. To do this, the Nearest Neighbors method with the metric parameter set to cosine similarity from sklearn was used. The images below show a sample of recommended items given a user id.

member_id	city_x	group_id	city_y	group_name
65	San Francisco	2701562	San Francisco	GoSF
65	San Francisco	14177122	San Francisco	Sourcegraph Tech Talks
65	San Francisco	14638342	San Francisco	San Francisco CoreOS Meetup

User ID

```
                                Wercker SF
                    Bay Area Mesos User Group
                              Bot Builder MeetUp
The San Francisco Redis Meetup Group
                              Docker San Francisco
                              San Francisco Perl
                              SF Data Engineering
                                DART
```

Recommended Groups

Compared to the results from content-based, the recommended groups on the right contains a different variety of groups despite a majority of them being similar.

Model-Based

For model-based collaborative filtering, there is a difference between how user-item interactions are represented. It can be either explicit feedback or implicit feedback. Explicit feedback is a type of rating where users explicitly show their preference towards an item by rating them on a scale. For example, Netflix has a 5-star rating scale for users to rate each movie they have seen. This gives the system an idea what items are liked and disliked by the user depending on how high or low the ratings are. In contrast, implicit feedback doesn't provide a scale for explicit rating towards an item. This type of feedback only guarantees that the user has interacted with the item and their preference towards the item. It does not show whether or not a user liked the item or hated it. Implicit feedback could be the number of clicks on a website, the

quantity an item was bought, or the amount of time spent on a page. Note that different types of feedback result in different cost function. For the Meetup dataset, there is no definite way to tell whether users liked the group that they joined or that they hated some groups they didn't join. Two sets of implicit feedback will be used to implement the model-based collaborative filtering. The first implicit feedback is the binary representation that indicates whether a user has joined a specific group. The other is a timedelta representation in months between the time the user joined the group to the time the user last participated in the group.

The main idea behind implicit collaborative filtering recommendation system is to use matrix factorization on the utility matrix. Figure 7 provides an image to visualize this. In Figure 7, R is an M by N sparse matrix where M is the number of users and N is the number of items. Matrix factorization allows R to be broken down into two smaller dense matrices U and V: one with dimensions (M x K) and the other (K x N), where K is the number of latent features, such that the product of these two smaller matrices U and V can approximate the original utility matrix. Similar to dimensionality reduction, these latent features don't represent anything meaningful but provide a nice way of representing each user and item in the same vector space.

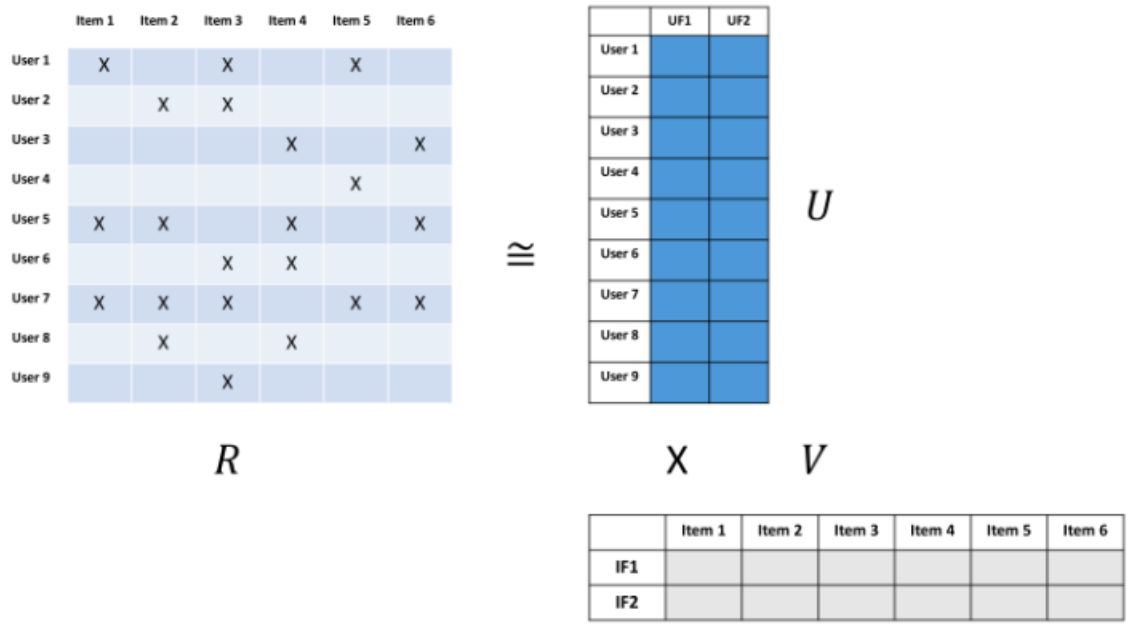


Figure 7 Matrix Factorization Diagram

To find the best U and V where the product of these two matrices give the best approximation of R, Alternating Least Squares (ALS) can be used after randomly initializing U and V. ALS is an iterative optimizing process that minimizes the following cost function

$$\min_{x_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

by holding each vector x_u in matrix U fixed and minimizing each vector y_i in matrix V and then alternating to holding each vector y_i in matrix V fixed and minimizing each vector x_u in matrix U. This iterating process repeats until convergence. As stated in Reference [d], the p_{ui} is a binary representation which indicates the preference of user u towards item i and is derived by binarizing r_{ui} values

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

where r_{ui} is the original values used to indicate an interaction between users and items. c_{ui} represents the confidence for a particular preference and is defined to be $c_{ui} = 1 + \alpha r_{ui}$. The greater the interactions r_{ui}, the more confidence the system knows user u prefers item i. α is the learning rate that determines how fast the confidence increases as r_{ui} increases. As a result, a huge penalty will be given if a preference with high confidence is approximated incorrectly while a preference with low confidence will have small effects to the overall cost. As this method stems from Reference [d], refer to it for more a detailed math explanation. After convergence from ALS, a dot product between matrices U and V can show the predicted feedback for specific user item interaction even if there weren't any prior interactions in the original utility matrix.

To implement this recommendation system proposed in Reference [d], Ben Fredickson developed a library called implicit that helps train the latent factor model to provide recommendations. As such, this library was used to train two models using the binary interaction representation and the timedelta interaction representation. To test the library, a simple model

was trained with a binary interaction utility matrix along with the following hyperparameters: alpha $\alpha = 15$, regularization $\lambda = .01$, and latent factors n = 20. The results are shown below.

Recommended Groups **Joined Groups**

The list of groups on the left are groups that were recommended to a specific user while the list of groups on the right groups the user has already joined. From the list of names on the right, it can be seen that this user participates in many tech groups along with a couple of social groups which the recommended groups capture. All in all, intuitively, this recommendation system using the library seem to work fairly well. However, to evaluate the recommendation system in a more pragmatic way, there are two methods that can be used, Precision @ k and Mean Percentile Ranking (MPR).

Before going any further, let the term relevant items be defined as items that are already known in the utility matrix that the user has interacted with and recommended items are defined as items that are recommended by the recommendation system. The idea is to hide certain interactions users had, train the model with these hidden interactions, and see if the model can recommend items that were previously hidden. Similar to the Precision metric used to evaluate classification models, Precision @ k evaluates the recommendation system by counting how many items that were recommended relevant out of all the items that were hidden. As such, users who have joined in at least 20 groups were randomly chosen to have some of their interactions hidden. Hiding interactions from users who only participated in a few groups are too costly as there isn't much information for that user to begin with. Once a certain percentage of interactions were hidden, a model was trained using this utility matrix and parameters were tuned to find the best set of parameters that recommended the most relevant items. The best results for a binary representation was 18.6% with the following parameters: alpha $\alpha = 1$, regularization $\lambda = .001$, and latent factors n = 80. The best results for the timedelta representation was 11.2% with the following parameters: alpha $\alpha = 1$, regularization $\lambda = .001$, and latent factors n = 40. The suspected reason for why the timedelta representation is a lot lower than the binary representation is due to the fact that each interaction with the timedelta representation provides a different confidence level for each item. If an interaction that was hidden was of high confidence, it may be hard to retrieve that information compared to the binary representation where all confidence levels are the same throughout all interactions. The downfall of using this metric is that it doesn't tell the whole story of the recommended items. The idea for a recommendation system is to recommend users items that they will have not yet seen while still

having a high preference for. Precision @ k does not give any information on other items or that the relevant items were even liked in the first place.

Luckily, MPR does. MPR ranks all items in the catalog for each user. The idea is items that users have a high preference for should be ranked in the top while items with low preference should be ranked near the end. Depending on where the selected item is placed in the rankings, a percentile is given towards the item. Percentile of 0% indicates the item is most likely to be recommended to the user while a percentile of 100% means that this item is unlikely to be recommended to the user. Using this notion, every user with hidden interactions had all items in the catalog ranked and the percentile ranking for the hidden items was calculated for each user. If recommendations were assigned at random, the expected value of the MPR is 50%. Items recommended based on popularity to users yields an MPR of approximately 20%. As such, retrieving an MPR less than 20% should be the baseline as anything greater than 20% is no better than recommending popular items. To verify this, MPR was calculated for all test users with the 5 most popular groups by members. The calculated MPR was around 18.6% which is close to the approximation. Nonetheless, training the latent factor model using the parameters found from Precision @ k where alpha $\alpha = 1$, regularization $\lambda = 10$, and latent factors n = 10, MPR was calculated to be 12.8% for the given set of users with hidden interactions. As this is below 20% mark, the model performs better than a popular recommendation system. To further test this recommendation system, it would need to be deployed online to perform an A/B test to see if it is increasing usage from users.

In summary, the goal of this project was to develop a recommendation system using Meetup dataset. For this project, both content-based and collaborative approaches, memory-

based and model-based, were implemented. Content-based focuses on developing a profile for all the items in the catalog while collaborative filtering uses user feedback. The model-based approach using matrix factorization was able to produce a recommendation system that performs better than a most popular recommendation system.

References

[a] CS50, director. *Recommender System*. *YouTube*, YouTube, 10 Nov. 2015, www.youtube.com/watch?v=Eeg1DEeWUjA.

[b] Banik, Rounak. "Recommender Systems in Python: Beginner Tutorial." *Recommender Systems in Python: Beginner Tutorial*, 16 Jan. 2018, www.datacamp.com/community/tutorials/recommender-systems-python.

[c] Das, Shuvayan. "Beginner Guide to Lean Content Based Engines." *Beginner Guide to Lean Content Based Engines*, 11 Aug. 2015, www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/.

[d] Hintz, Jeremy. *Matrix Factorization for Collaborative Filtering Recommender Systems*. University of Texas, Austin, 2015, pp. 1–5, *Matrix Factorization for Collaborative Filtering Recommender Systems*.

[e] Hu, Yifan, et al. "Collaborative Filtering for Implicit Feedback Datasets." *2008 Eighth IEEE International Conference on Data Mining*, 2008, doi:10.1109/icdm.2008.22.

[f] Johannsdottir, Agnes. "Implementing Your Own Recommender Systems in Python." *Implementing Your Own Recommender Systems in Python*, 17 June 2017, cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html.

[g] Kohler, Victor. "ALS Implicit Collaborative Filtering." *Medium*, 23 Aug. 2017, medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe.

[h] Malaeb, Maheer. "Recall and Precision at k for Recommender Systems." *Recall and Precision at k for Recommender Systems*, 13 Aug. 2017, medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54.

[i] Mander, Peters. "Implicit Matrix Factorization for Recommender Systems of Smaller Scale e-Retailers." *Tilburg University*, 2017.

[j] Ricci, Francesco, et al. *Recommender Systems Handbook*. Springer, 2015.

[k] Rosenthal, Ethan. "Data Piques." *Data Piques Full Atom*, 19 Oct. 2016, blog.ethanrosenthal.com/2016/10/19/implicit-mf-part-1/.

[l] Schenkel, Jonas F. "Collaborating Filtering for Implicit Feedback." *University of Oslo*, 2017.

[m] Steinweg-Woods, Jesse. "A Gentle Introduction to Recommender Systems with Implicit Feedback." *A Gentle Introduction to Recommender Systems with Implicit Feedback – Jesse Steinweg-Woods, Ph.D. – Data Scientist*, 30 May 2016, jessesw.com/Rec-System/.

[n] Tomur, Ankur. “Content Based Recommendation System in Python.” *Content Based Recommendation System in Python*, 14 Aug. 2017, medium.com/@tomar.ankur287/content-based-recommender-system-in-python-2e8e94b16b9e.