Spoons : Restaurant Recommendation for Groups

Donsuk Lee
California Institute of Technology
B.S. in Computer Science, '16
dllee@caltech.edu

Joon Sik Kim
California Institute of Technology
B.S. in Computer Science, '17
jkim5@caltech.edu

ABSTRACT

While there are many restaurant recommendation services for individuals, it is difficult to find one that takes into account the different preferences of a group of users. Spoons is a new web-based recommendation service that suggests restaurants to groups. The system features intuitive interface that users can easily interact with and incorporates machine learning techniques to recommend restaurants that suit best for a group. The application is currently deployed on Heroku, at http://spoonsproject.heorkuapp.com.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Web application, Restaurant recommendation

Keywords

Recommendation, Django, Machine Learning, Similarity

1. INTRODUCTION

Individuals, nowadays, often face overwhelming options of items to choose from. Finding desired products in the flood of information may be a tedious and time consuming task for the individuals with limited knowledge. Recommender systems are developed to help users decide easily and accurately by suggesting a list of items tailored to their tastes.

Recommendation can also be a key to acquiring a stable user base for web-based services. Giant online services such as Amazon, Yelp, Netflix, Facebook, etc. all try to come up with better product recommendations for the users, as the accuracy of the recommendation now became a virtual indicator of whether the user will keep using the service or not in the long run. The better the recommendations are, the more likely the users will revisit the page again with satisfying product reviews.

There are many approaches to designing recommender systems, mostly involving machine learning techniques. The two widely used techniques are content-based recommendation [7] and collaborative filtering [8, 9]. Content-based methods suggest items that best match a user's interest by measuring the similarity between the user's profile information and the description of the items. Theses algorithms are relatively easy to implement and do not suffer from data sparsity problem, although the prediction may not be very accurate. On the other hand, collaborative filtering algorithms use user-item matrix to calculate similarity between users or items. Under the assumption that similar users share similar tastes, the calculated similarity is then used to predict the ratings of items that a user has not yet rated. The more users rate items, the more accurate the recommendation becomes. However, the collaborative filtering algorithms often suffer from cold-start problems.

To date, recommendation has been used for various items, including movies [10] and news [11]. Most recommendation services mainly focus on suggesting items to individual users. However, it is more appropriate for some domains in which several people participate in an activity to make recommendation for groups. Watching movies is a good example where group recommendation is more useful [6]. Similarly, a restaurant recommendation service for groups of people can be more helpful for the users planning to dine with friends or family.

In this project, we tried to aim a different approach for recommendation: instead of focusing on individual recommendation, we tried applying recommendation in group context, especially for restaurants.

Consider a scenario: Don, David, Peter, and Paul are planning to eat out for tonight. They want to select a restaurant that can satisfy the most of them. To do so, they will have to contact each other to rule out some restaurants and figure out the preferences of each other. Peter calls and texts David to ask, but David is not answering. Spoons helps to reduce such tedious and rather complicated process into a quick and fun process, where one person can take care of the rest's preferences without contacting them each in person.

2. RELATED WORKS

Restaurant recommendations are prevalent in many services like Yelp, TripAdvisor, OpenTable, UrbanSpoon, Zagat, etc. They use users' rating/review history, location, or other preference features to predict the users' likes, and show them for the users to reference in real-time. Our project distinguishes itself from these recommendation web services that it has a group recommendation feature – other services are limited to individual recommendations.

There are some research done in this area as well. In [1] the authors discuss some group recommendation services in various areas, like Travel Decision Forum for travel planning, G.A.I.N for news information, Adaptive Radio for music, etc. and explains some preference aggregation algorithms for group recommendation to work. In [2] the authors explain the expanding concept of individual recommendation algorithm using collaborative filtering and content-based recommendation in to group context, and furthermore, into a social network context. In [3] the authors use a technique called information matching to tackle the group recommendation problem in a probabilistic approach.

3. FRONT-END FEATURES

Figure 1 shows overall picture of how the application uses external information for the recommendation. Login authentification uses Facebook API, and users can log in with their Facebook account only. From the account information, Spoons retrieves friend information used for group recommendation. Users input their likes and preferences in the application, and based on the input, the application recommends the restaurants the user might like. Restaurant information is retrieved from the pre-populated database, which is populated using Yelp API and crawlers (explained in Section 4).

For the front-end design, we used Bootstrap with some tweaks on CSS and Javascript.

3.1 Sign-up page

Users can log in using his or her Facebook account. Facebook login provides convenience in various ways, for instance, it is easy to connect to the friends using the application for group recommendation. When the user is logging in for the first time, it will have a sign-up page, which will ask the user to check the categories of food he or she likes. This will be the starting point of the recommendation – users then can continue using the app, like the restaurants, and build the personal profile in more detail for better recommendation.

3.2 Search / Like

Each restaurants are introduced in a card, with a thumbnail image retrieved from Yelp and other relevant information at the bottom, as shown in Figure 2. Searching is also possible for restaurants in the database. The Like button in the card will add the restaurant to the user's profile as a liked restaurant (shown in the next section). Some key information is listed on the card, including price range, average rating, categories, ambience and attire. For more information about the restaurant, user can click on the name of the restaurant, then it will redirect to the restaurant page on Yelp.

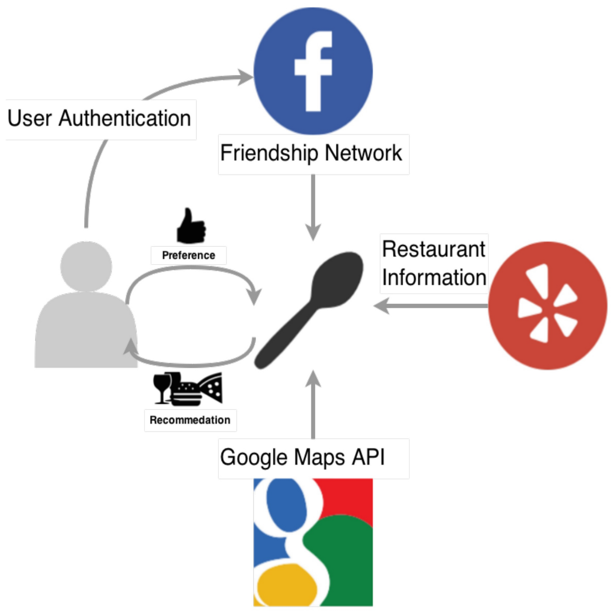

Figure 1: Overall Architecture of the Website. Spoons use Yelp for restaurant information, Facebook for friend and user login information.

3.3 Profile Page

Profile page can be set up using the upper right corner button displaying the user's name. It contains the categories the users checked at the top, and the restaurants they liked at the bottom. They can unlike the restaurants whenever they want by pressing Unlike button on each restaurant cards. The liked restaurants will be used for recommending similar restaurants to the user.

3.4 Add Friend / Group Recommendation

Using Facebook API, the application allows the user to form a group with friends who also use this application. This is done by clicking a group button at the upper right corner of the main page. It will show a check box of friends that are currently using the app. Check the friends and click Go to redirect to the group recommendation page, where it will analyze the users' preferences and recommend the best restaurants for the group context. There is also a filter button, which filters out certain restaurants that meet the criterion people all want in addition. So it can take some external and situational preferences into account as well (Figure 3).

4. BACK-END DETAILS

Back-end details are visualized in Figure 4 with illustration of flow of information. Overall, the database and user interface is connected via three main individual recommendation algorithms (simple show, similar show, and learn show), and one group recommendation algorithm. Restaurant information stored in the database is extracted, converted into vector form suitable for computations through `vectorize()` function, and the vectors are fed into the recommendation algorithms.

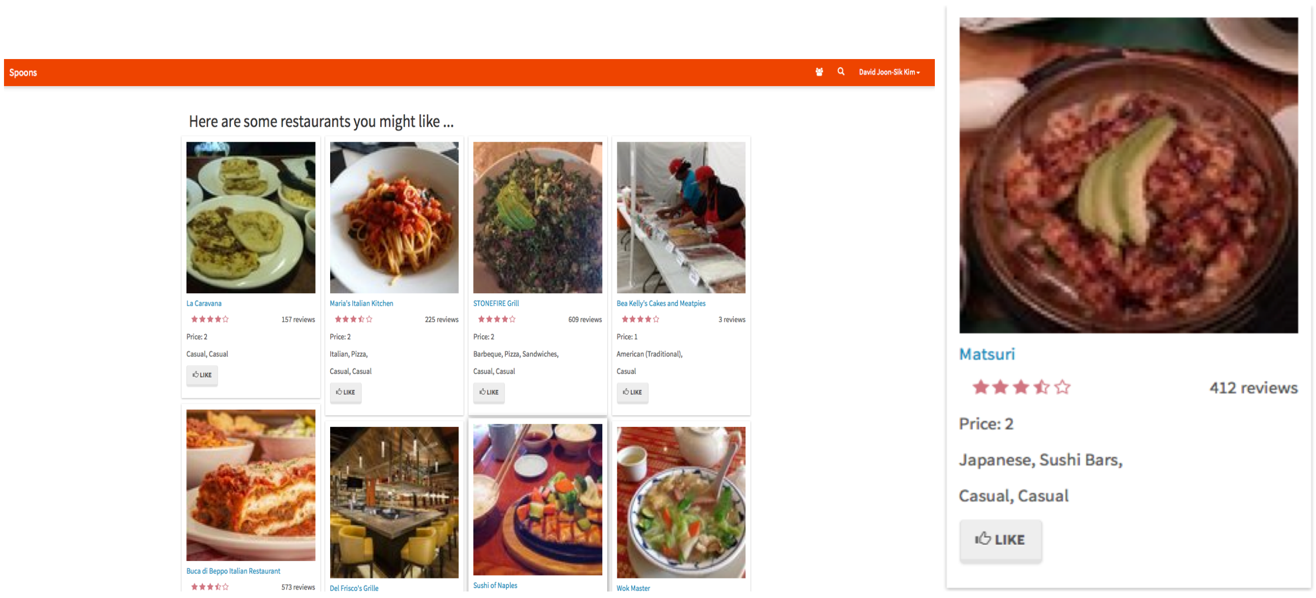

Figure 2: Main Page (Individual Recommendation Page) UI and Restaurant Cards. Main UI as search tab and group recommendation tab at the upper right corner of the website, along with personal profile tab. Restaurant cards contain a link to the Yelp page, stars, number of reviewers, price range, categories, ambience/attire information, and a Like button.

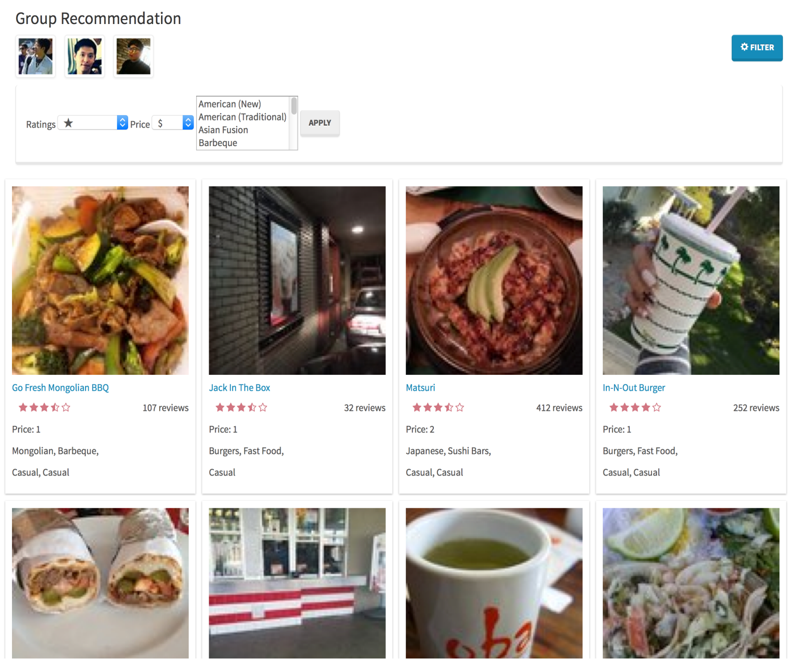

Figure 3: Group Recommendation in action. The users' Facebook profile pictures are shown along with filters they can apply to the recommended restaurants, and the result is shown below in card format.

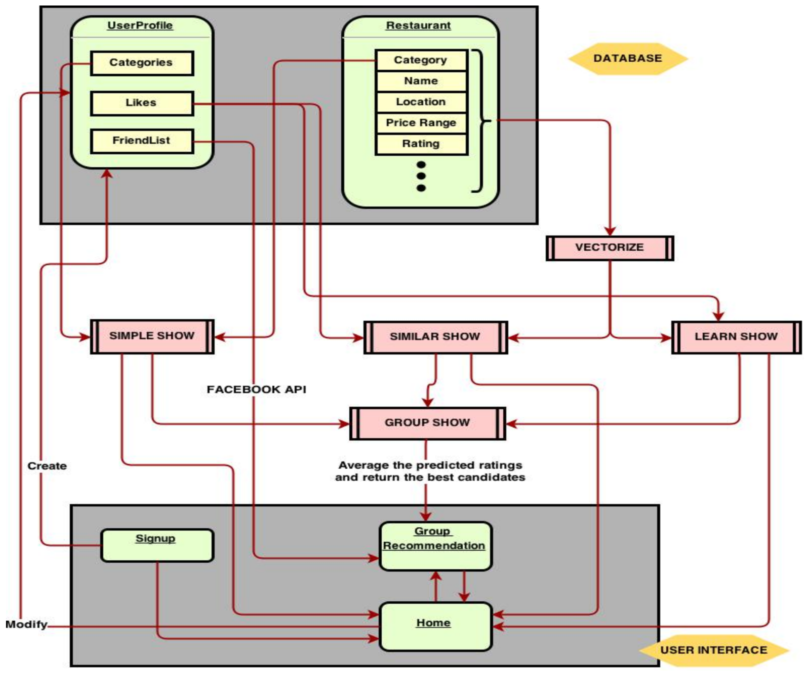

Figure 4: Flow of Information. There are three different individual algorithms in the middle portion connecting the user interface and the database, and one group recommendation algorithm that makes use of the three individual algorithms. The vectors used for each algorithms are created by vectorize() function that calls restaurant features from the database.

4.1 Web Framework

For the web framework, we used Django 1.8. This web framework was especially helpful in the process of development as the front and back-end connection was made easy with the language we were most familiar with, Python.

4.2 Database Setup

Initial database required restaurant information in detail to provide the users with recommendations. So we decided to first focus on Pasadena only, and expand later on. We used Yelp API and crawled data from Yelp to populate our initial database of restaurants. Since Yelp API provided limited number of features for individual restaurants, we needed build a crawler to manually get some additional features from the web page directly, like ambience, noise level, waiter service, etc.

We used sqlite3 for testing and PostgreSQL for deployment.

4.3 Testing Different Learning Algorithms

For this web application to work, we needed a working individual recommendation algorithm that could take in the user information and then generate candidates of restaurants that reflect the user's preferences. To tackle the learning algorithm in general, we first used the data set publically distributed by Yelp for the Yelp Data Set Challenge in 2015, which consists of 1.6 million reviews and 500,000 tips by 366,000 users for 61,000 businesses. Within that data, we extracted 21,892 businesses that are categorized as restaurants (initially businesses contained various types like theater, groceries, gas stations etc.).

Table 1: Model Training / Testing Result (RMSE)

	Decision Tree	AdaBoost	Random Forest
Parameters	Depth 7	Depth 3, 300 classifiers	300 classifiers
In-sample RMSE	0.673	0.655	0.419
CV RMSE	0.687	0.697	0.699

Using the data, we first tried collaborative filtering for users versus restaurants to predict the users that might like certain restaurants. We used the user's reviews as the indicator that the user likes the restaurant. However, the matrix for the collaborative filtering was very sparse, because most of the users did not leave reviews on Yelp. With big sparsity, collaborative filtering did not work properly, so we had to try a different approach.

Then we decided to focus on content-based recommendation, which uses the restaurant's traits to recommend. We vectorized each restaurant's features into 150-dimensional vector where each element indicates whether certain feature is True or not (binary). We ran different training models (random forest regressor, decision tree regressor, AdaBoost regressor) on the aggregated vectors to find the predicted ratings for each restaurants.

The decision tree training yielded the lowest 5-fold cross-validated root mean square error (RMSE) of 0.687, with appropriate parameter handling (Table1).

4.4 Individual Recommendation

These are the set of algorithms used for individual recommendation of restaurants. Based on the information provided in the user's profile, these algorithms generate a list of restaurants that the user may like. The situation when each of the algorithm is used is different, as it will be explained in detail.

4.4.1 Simple Show Algorithm

Simple-Show algorithm consists of generating list of restaurants based on the categories the user chose at their sign-up page. It filters out restaurants in that category and orders them based on their average ratings. This is used to solve the cold-start problem for individual user preference – when the user have no or just a few restaurants liked, then we do not have any data to recommend from. So we simply take a naive approach using the categories user liked.

Algorithm 1 Simple Show Algorithm

```
 1: procedure SIMPLE_SHOW(user)
 2:     LST ← (empty list)
 3:     for r in user.likes do
 4:         if r.category ∈ user.liked_cateogory then
 5:             Add (r, rating) to LST
 6:         end if
 7:     end for
 8:     LST ← sort(LST) based on rating
 9:     return LST
10: end procedure
```

4.4.2 Similarity Ranking Algorithm

When the user has liked enough restaurants, we use similarity ranking algorithm to generate candidates. We first compute the average vector that reflects the user's liked restaurants by summing all vectors component-wise and dividing it by the number of restaurants liked. Then we compute similarity in the following way (cosine similarity), and then sort the restaurants in descending order to create a list.

$$sim(U_i, V_i) = \frac{\sum_{l=1}^{k} u_{i,l} v_{j,l}}{\sqrt{\sum_{l=1}^{k} u_{i,l}^2}\sqrt{\sum_{l=1}^{k} v_{i,l}^2}} \quad (1)$$

where U_i and V_i are simply vectors representing different restaurants, and $u_{i,l}$ and $v_{i,l}$ are lth component of the vector.

The computed value becomes the similarity score of the user's average vector and a new restaurant from the database – the bigger, the more similar. These scores will be used later for group recommendation.

Algorithm 2 Similarity Ranking Algorithm

```
 1: procedure SIMILAR_SHOW(user)
 2:     LST ← (empty list)
 3:     avg ← average vector of user.likes vectors
 4:     for r in restaurant_db do
 5:         sim ← sim(avg, r)
 6:         Add (r, sim) to LST
 7:     end for
 8:     LST ← sort(LST) based on sim
 9:     return LST
10: end procedure
```

4.4.3 Learning Algorithm

When the user has liked sufficient number of restaurants for learning to take place, we use this algorithm. Using decision tree regressor trained from the user's list of restaurants, we predict the ratings of each restaurants, and then sort them based on the predicted ratings.

Algorithm 3 Learning Algorithm

```
 1: procedure LEARN_SHOW(user)
 2:     LST ← (empty list)
 3:     clf ← Trained DecisionTreeRegressor using user.likes
 4:     for r in restaurant_db do
 5:         rate ← clf.predict(r)
 6:         Add (r, rate) to LST
 7:     end for
 8:     LST ← sort(LST) based on rate
 9:     return LST
10: end procedure
```

So overall, the individual recommendation algorithm is shown in Algorithm 4.

Algorithm 4 Individual Algorithm

```
1: procedure INDIVIDUAL_REC(user)
2:     if user.numOfLikes < 5 then
3:         return SIMPLE_SHOW(user)
4:     else if user.numOfLikes < 20 then
5:         return SIMILAR_SHOW(user)
6:     else
7:         return LEARN_SHOW(user)
8:     end if
9: end procedure
```

4.5 Group Recommendation

Group recommendation usually comes in two different types: first is to create a pseudo-user that represents the taste of the whole group, and second is to generate candidates for individual users and then aggregate the list to come up with a list for the group [6].

4.5.1 Pseudo-User

This approach first uses the individual preference to build up a representative user preference for the group, and then recommends to that user. However, this approach can result in an outlying preference in the end because some users who liked many restaurants will contribute more in building representative preference. And because of that, it is possible that each member will not be given a same weight on contributing to the representative preference.

4.5.2 Aggregation

This approach first recommends individually, and then aggregates the lists to create a recommended list for the group. This has some advantages over creating a pseudo-user, because the recommended results can be well-reasoned, as the recommendation is directly taken from the user's personally recommended list. Aggregation can be done in several ways, but the most common one is to take the average approach so that the users can all benefit if some sense. This is to maximize the overall satisfaction of the group members, giving each of them a same weight.

Group recommendation algorithm we used is an aggregation of individual recommendations with average (maximizing satisfaction) approach. Using individual recommendation algorithm, we first generate candidates for each users in the group and compute the scores of each restaurants for each users. Then we aggregate them and compute the average scores for each restaurants, and then re-rank them based on the average scores.

Algorithm 5 Aggregation Algorithm

```
1: procedure GROUP_REC(group)
2:     LST ← (empty list)
3:     for each user $u_i$ in group do
4:         Add INDIVIDUAL_REC($u_i$) to LST
5:     end for
6:     LST ← sort(LST) based on average scores
7:     return LST
8: end procedure
```

4.6 Recommendation Using Social Context

Recommendation algorithm can definitely stand alone in this application, but since we are dealing with Facebook account information and friend lists, we thought about adding the social context to the recommendation. This exploits the fact that close friends, who share common experiences, tend to share similar tastes for food as well. Under this assumption, we can use the social network to come up with a recommendation for individuals.

First, we locate a user in the network. Then we find clusters within the network, and some influential nodes for each cluster. This can be done using degree centrality, for simplicity, and select the users with the biggest degree. Then recommend the restaurant these influential nodes for the cluster liked.

However, the problem with this algorithm is that there is always a possibility that the influential user may not share same preference with the user in that cluster. So we must limit the number of restaurants recommended in through this method, or somehow increase the probability that the suggested restaurant in some sense fits with the user's preference. This can be done by for each suggested restaurants from the influential nodes, computing similarity using the average feature vector from the individual user, and returning the top few candidates, or by setting up a threshold of similarity scores to filter out only the restaurants with the higher score.

5. FUTURE WORKS

For this project, we focused on the restaurants in Pasadena, because building up initial data generally was not easy – Yelp API did not provide detailed information of the restaurants, and crawling additional features from the web page directly was limited. So the on-going work is the scalability issue – how to include restaurants in other cities, automatically if possible, and how to organize the database more efficiently for increased number of restaurants.

Also the recommendation algorithm in social context is something we wanted to try but did not have much time to do, as it requires a base of many test users composing a big social network. We want to validate the effectiveness of social recommendation algorithm once the website gains enough user-base.

6. ACKNOWLEDGMENTS

We would like to thank Professor Adam Wierman for active feedback and overall support for the project, and Professor Yisong Yue for the initial advice on the concept and design of the website.

7. REFERENCES

[1] A. Jameson, B, Smyth. *Recommendation to Groups.* 2007

[2] R. Zafarani, M. A. Abbasi, H. Liu. *Social Media Mining: An Introduction.* 2014

[3] J. Gorla, N. Lathia, S. Robertson, J. Wang. *Probabilistic Group Recommendation via Information Matching.* 2013

[4] Django Documentation. https://docs.djangoproject.com/en/1.8/

[5] Bootstrap Documentation. http://getbootstrap.com/components/

[6] M. O'Connor, D. Cosley, J. A. Konstan, J. Riedl. *PolyLens: A Recommender System for Groups of Users*

[7] G. Adomavicius, A. Tuzhilin. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.* Knowledge and Data Engineering, IEEE Transactions on, 17(6), 734-749, 2005.

[8] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. *Collaborative filtering recommender systems., The AdaptiveWeb*, Springer, 2007, pp. 291-324.

[9] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. *Item-based collaborative filtering recommendation algorithms*, Proceedings of the 10th international conference onWorldWideWeb, ACM, 2001, pp. 285-295.

[10] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, J. John. *MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System*, IUI'03, January 12-15, 2003.

[11] A. S. Das, M. Datar, A. Garg, S. Rajaram. *Google news personalization: scalable online collaborative filtering*, Proceedings of the 16th international conference onWorld-WideWeb, ACM, 2007, pp. 271-280.