Eating Mate Recommendation for Yelp Users

Hao Xu, Zehao Song, Yutong Tie
Department of Computer Science
Columbia University
{hx2208, zs2324, yt2549}@columbia.edu

Abstract—**Yelp has been more and more prevalent in making recommendations for people's daily life, like where to eat or where to shop, however, little has been done regards making recommendations for who they could go with. We proposed that eating mate could be recommended based on the review history of each user. Therefore, we collected review data from Yelp to make eating mate recommendations, which included a collaborative filtering of the review data, linear regression on a single user and business data, and finding K nearest neighbors of each user. The experiment on data points confirms that it is a good way to make recommendations for eating mate. Since the current experiment is based on review data on partial cities in the United States, more tests are needed for other cities, and in order to have a better representation for recommendation, a User Interface is needed in the future.**

Keywords-Yelp; eating mate; recommendation; collaborative filtering

I. INTRODUCTION

Yelp is a platform about sharing reviews and making recommendations. On the one hand, users of Yelp could submit reviews on different shops and restaurants using a one to five-star rating. On the other hand, Yelp would make recommendations for users based on their past review history. McNichol states Yelp has become "one of the most import sites on the Internet" [1]. According to the statistics provided by Yelp on 2016, it had about 102 million total reviews and a monthly average of 92 million unique visitors in the second quarter of 2016 [2].

Though Yelp has strong functionalities in helping people find a place to eat, it has less tricks in finding people an eating mate. It is a great business loss since under most circumstance, people would not like to go eating alone, no matter how good the business is.

Based on this situation, this project is intended to make recommendations for people who share similar preferences to have meal together based on each individual's review history. We are dedicated to dig into each user's tastes which hides behind their reviews, and hopefully, to analysis taste distributions of each city.

Finally, the application should work like: 1). For a new user, we would like this user answer some question so we can know about his preference and make eating mate recommendation based on his answers. 2). For an existed user, we would make recommendation directly based on his rating history. The final out would be several eating mates and corresponding restaurant.

II. RELATED WORKS

I do not think there is any previous work that makes recommendation for eating mates, however, "people you may know" is a section that has been widely implemented in social websites. Besides, most E-commerce websites, like Amazon and eBay, make recommendations for goods that people would like to shop. These kinds of recommendation systems are in different tunes rendered with equal skill as eating mate recommendation. I would list a few websites and how they make recommendations in the following paragraphs.

Facebook [3] is dedicated to connect everyone who is using their website. According to Vignesh's answer on Quora [4], Facebook makes friend recommendations on following 5 aspects: 1). Right after registration, every user would be asked to fill in a questionnaire about his background, like education, region, interest, etc. Based on that information, Facebook would recommend people who went to same school or work at the same place to that user. 2). Once has some friends, Facebook would recommend one's friends to him. Say if A and B are friends, B and C are friends, then Facebook would recommend A to C and C to A at the same time. 3). Facebook would recommend two people in the same group to be friend. For example, if user A is in Columbia Engineering group and user B is also in that group, then Facebook would recommend A to B and B to A at the same time. 4). Facebook would also recommend two people like

same public page to be friend. For example, if user A likes Bayern Munich and user B likes Bayern Munich as well, then Facebook would recommend them to be friend, however, if A likes Bayern Munich and B likes Borussia Dortmund, it might not recommend them to be friend due to the counter relationship between these two teams.

LinkedIn [5] is designed for people to build up their career circles and trying to help people in job-hunting. Linkedin makes recommendations for their user based on following 3 aspects: 1). Like Facebook, LinkedIn makes friend recommendation based on users' basic information, like education, region, etc. It would try to connect users from same school, same area or same company. 2). Unlike Facebook, LinkedIn is intended to build up a more real network among people than Facebook, which guarantees the quality of connections in LinkedIn. LinkedIn would import contacts via each users' email address and recommend friends' connections to each user.

Amazon's [6] recommendation starts from finding a set of customers who have items overlap with the user's purchase and rate history. Then it uses different algorithms aggregates items from those customers, remove the items that has already been purchased or rated, and recommends items that are left to the user. There are basically 2 algorithms being applied in Amazon's recommendation. 1). Traditional collaborative filtering. Each customer is represented as an N-dimensional vector. For example, there are two users A and B, the similarity of A and B is calculated by cos(A, B). 2). There are many attributes that described a specific user and the recommendation is based on different cluster that the user belongs to. Therefore, the recommendation problem is simplified to a classification problem. Amazon uses those attributes to divide customers into different clusters.

In our project, we choose to do similar as what Amazon does. We represent each user in Yelp as feature vectors and make recommendations based on the nearest neighbor of that user.

III. System Overview

In this section, we would ramp up on how we scarp data from Yelp, how our recommendation system works and how the dataset looks like.

Data Collection.

In order to run scarpy, we would need to generate the URL that contains the data we want. In this case, we need to get business_id for each city. Yelp provides its Search API for developers to search business_ids that they are interested.

Sample Request:

```
https://api.yelp.com/v2/search?term=food&location=San+Francisco
```

This is how the sample request looks like. In this request, we are fetching business ids in San Francisco that sell food. We fetch data by changing corresponding attributes in the URL. The response from Yelp server is a JSON object. Below is a sample response:


```
"businesses": [
    {
        "categories": [
            [
                "Local Flavor",
                "localflavor"
            ],
            [
                "Mass Media",
                "massmedia"
            ]
        ],
        "display_phone": "+1-415-908-3801",
        "id": "yelp-san-francisco",
        "image_url": "http://s3-media3.fl.yelpcdn.com/bphoto/nQK-6_vZMt5n88zsAS94ew/ms.
        "is_claimed": true,
        "is_closed": false,
```


This JSON object contains the import information we want, like business id, name, geo location, etc.

After getting enough number of businesses, we start to gather the users' reviews of each business by a python crawler implemented via Scrapy Framework.

Below is how we generate the target URL and fire the request:

```
urls = []
with open('yelp_scrapy/data/%s.jl'%input_file) as f:
    for line in f:
        r = json.loads(line)
        urls.append("https://www.yelp.com/biz/%s"%r[0])
f.close()
for url in urls:
    yield scrapy.Request(url=url, callback=self.parse_review)
```

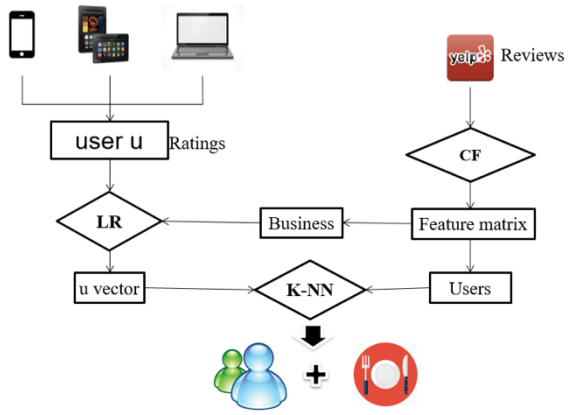
The code above reads lines in generated business id lists in previous step and append it to regular Yelp URL and fire the URL. Then the code parses the HTML code and find the reviews we want in each single page to collect review data. For more detailed

code, take a look at our Github: http://www.gethub.com/tsszh/yelp_recommendation/blob/master/scrapy/yelp_scrapy/spiders/review_spider.py

Workflow.

After collecting large enough data, we start to analysis the data and make recommendation for each user. The whole process is as follow:

Figure 1 Eating Mate Recommendation System Overview



The analysis starts from the review data we collected from Yelp.

Firstly, we do a collaborative filtering(CF) over the review data (user_id, business_id and rate) to generate the feature matrixs. Save both business feature & user feature matrix for future usage.

In principle, to make recommendation for eating mates, we just need to select users with the minimum distance in terms of user features, which is a typical k-nearest neighbor (K-NN) algorithm.

However, for new users, which is not included in the training sets, the user feature vector is not available. To get the feature vecotr of new user, a typical linear regression algoritm is applied to the users' rating history and corresponding business feature matrix. After getting the best estimation of the user feature vector, apply K-NN algoritm to find similar users from preivious User Feature Matrix. Then, we got the eating mates.

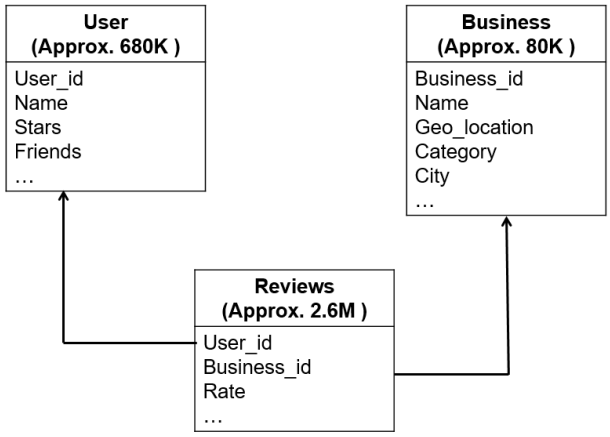
After getting the eating mates, we still need to recommend the restaurants for them. The method is rather simple, taking advantage of the user vectors and business matrix to predict the rating for all business and select the ones with highest ratings. Then, find the shared restaurant between eating mates.

Besides, we could predict the taste or favor of a perticular user. The basic idea is to calculate the categories with the highest term frequecy (Chinese, Pizza, etc…) asscoiated with the user.

Dataset Overview

There are mainly 3 tables in our database system. They are User data, Business data, and Reivew data.

Figure 2 Dataset Relation Schema Overview



There are about 680K user tuples, 80K business tuples and 2.6M review tuple in total has been collected so far in order to make recommendations.

In User table, each user is assigned a unique key, which is user_id (primary key). Other information about each user, like name, his friend, his location, etc. has also been included in the table defined above.

In Business table, each business is assigned a unique key, which is business_id (primary key). Other information about each business, such as name, geolocation, category, etc. is also included in the table defined above.

In review table, each review record is defined by a combination of user_id and business_id (primary key). For instance, tuple (1, 2, 3, …) means that user whose id is 1 gives business whose id is 2 a rate 3.

Other information like rate, comment, timestamp, etc. is also included in the table.

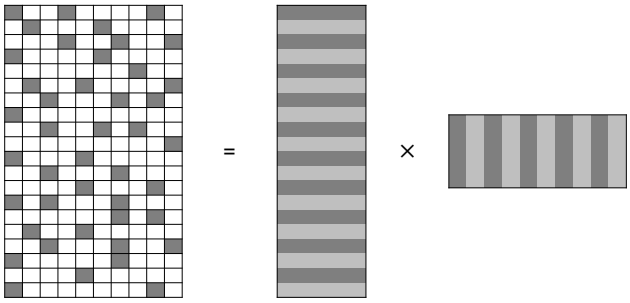
IV. ALGORITHM

Show algorithm and tools you have used to solve your problem.

1. Collaborative Filtering (CF)

As shown in Figure 3, the input is the Review Sparse Matrix, each row stands for a specific user and each colum represents a specific business. For each cell, the value is the rating in float type if there is a review for corresponding user_id & business_id, or zero otherwise.

Figure 3 Illustration of Collaborative Filtering (CF)



The goal of the algorithm is to decompose the Review Matrix ($R^{n\times d}$) into two Rank-k matrix, User Matrix ($R^{n\times k}$) and Business Matrix ($R^{k\times d}$), with minimum square error,

$$min \sum_{R_{ij}\in Review} \left(R_{ij} - \langle U_i, B_j\rangle\right)^2$$

in which UB is the most similar matrix compared to Review Matrix.

Each row of User Matrix stands for a particular user vector (U_i). Each column of Business Matrix stands for a particular business vector (B_j). After getting these two feature matrix, it would be simple to calculate the predicted rating of User i on Business j.

$$R_{ij} = \langle U_i, B_j\rangle$$

To get User & Business Matrix, we use **Alternating Least Squares (ALS)** algorithm. The basic idea is:

(1) Suppose Business Matrix is given. Each row of User Matrix can be calculated via simple Linear Regression.

(2) Suppose User Matrix is given. Each row of Business Matrix can be calculated via simple Linear Regression.

(3) Alternate between these two simple problem.

```
Initialize u_i ∈ R^k and v_j ∈ R^k
for t = 1, 2, ..., NumIter:
    For each user i = 1, 2, 3, ..., n:
        u_i = argmin Σ_(i,j)∈Ω(⟨u, v_j⟩ − R_ij)^2
    For each business j = 1, 2, 3, ..., d:
        v = argmin Σ_(i,j)∈Ω(⟨u_i, v⟩ − R_ij)^2
```

Actually, Spark provide this model-based recommendation algorithm.

```
from pyspark.mllib.recommendation import ALS,
MatrixFactorizationModel, Rating

ALS_setting = {
    'rank': 20,
    'numIterations': 20
}

model = ALS.train(
    ratings,
    ALS_setting['rank'],
    ALS_setting['numIterations']
)

vBusiness = model.productFeatures()

vUser = model.userFeatures()
```

By using Spark Library, CF-ALS becomes straight-forward. There are two parameters in the model to be tuned, rank & numIterations. Rank is the dimension of User & Business Vector. NumIterations is the number of iterations in ALS. The choice of these two parameters will be discussed in Experiment Results Seciton.

2. Linear Regression (LR)

Input: Review history of unknown user ($b \in R^m$) and Corresponding Business Feature Matrix ($A \in R^{m \times k}$). Note, m is the number of reviews given by user, which might be smaller than k, rank of the user vector. If $m < k$, it is impossible to get reasonable LR result. Thus, we only provide recommendation for users who rates at least k restaurants.

Output: User Feature Vector of unknown user ($u \in R^k$).

Objective: $\min\|Au - b\|^2$

Algorithm I: Ordinary Least Squares

Direct Mathematic Calculation via:

$$u = (A^T A)^{-1} A^T b$$

Pros: Accurate, Simple, Fast for small matrix

Cons: Slow for large matrix

Algorithm II: Gradient Decent

Iteration method. Provided by Spark Library.

Pros: Fast for large matrix

Cons: Less accurate for small matrix

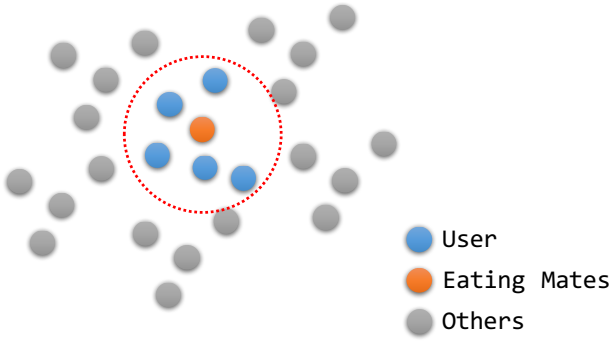
The number of reviews given by user is limited. Normally, $m < 100$. Obviously, ordinary least squares is mush suitable for this circumstance.

```
import numpy as np

vb, mA = getMatrix(compactM)
    vu = np.linalg.inv(mA.T.dot(mA)) \
        .dot(mA.T) \
        .dot(vb)
```

The calculation is direct and simple with the help of numpy library.

3. K-Nearest Neighbor (kNN)

Figure 4 Illustration of K-Nearest Neighbor Algorithm



After getting the user's feature vector, we could give the recommendation by finding the most similar users (minimum norm-2 distance) in User Feature Matrix. Spark doesn't have default library for K-NN algorithm. So, we implemented the naïve K-NN by ourselves in Spark.

```
def knn(vUser, vu):
  return vUser.map(lambda r: (
           r[0],
           np.linalg.norm(vu-r[1]
         )))
         .top(k, key=lambda r: -r[1])
```

4. Restaurant Recommendation

Input: User Vector & Business Matrix

Output: List of recommended restaurants

(1) Predict the rating for all restaurants by

$$R_j = u^T \cdot U_j$$

(2) Select the top 10 restaurants with highest ratings

(3) Select the top shared restaurants for eating mates as the recommendation

```
def recomd(vBusiness, vu, num=10):
  return map(
           lambda r: r[0],
           vBusiness.map(
```

```
            lambda r: (
              r[0],
              np.array(r[1]).dot(vu))
            ).top(num, key=lambda r: r[1])
        )

def sharedRecom(vu1, vu2):
  pool = set( recomd(vBusiness, vu1) )
  return filter(
           lambda r: r in pool,
           recomd(vBusiness, vu2)[:1]
         )
```

5. Taste Analysis

Input: User Vector & Business Matrix

Output: List of (Category, Weight)

```
from operator import add

def getTaste(vBusiness, vu):
  rest = recomd(vBusiness, vu, num=1000)
  taste = sc.parallelize(res)\
          .flatMap(
            lambda r: business_dict[r][1]
          ).map(
            lambda r: (r, 1)
          ).reduceByKey(add)\
          .takeOrdered(
            50, key=lambda r: -r[1]
          )
```

(1) Get top 1000 recommended restaurants

(2) Count the term frequency of each category associated with these restaurants

(3) Select the Top 50 categories.

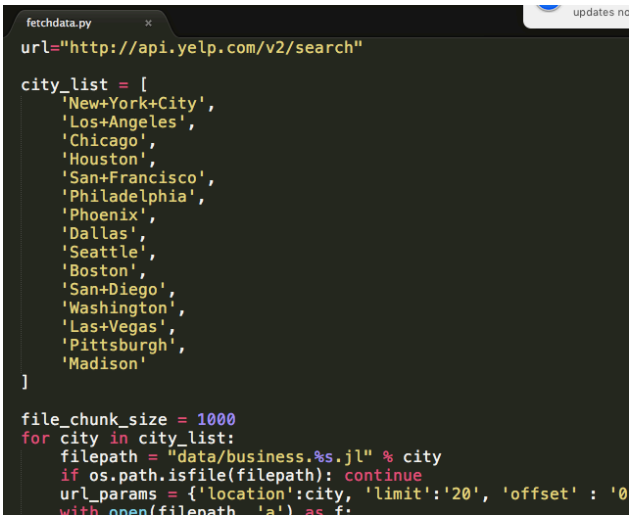
V. SOFTWARE PACKAGE DESCRIPTION

Describe the software package that is going to be open sourced. Show some screen shots of how user use it and or UI.

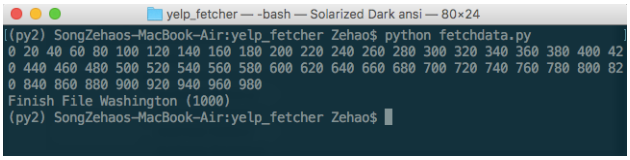
1. Yelp API

Path: scrapy/yelp_fetcher/fetchdata.py

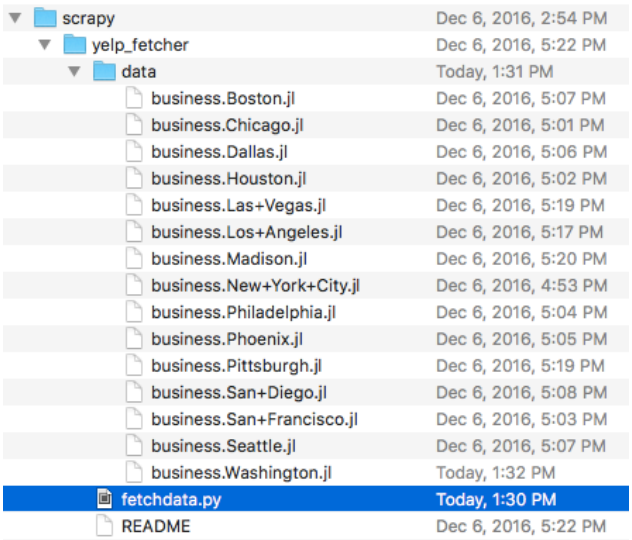
First, add the city name to the **city_list**, eg, “Washington”.



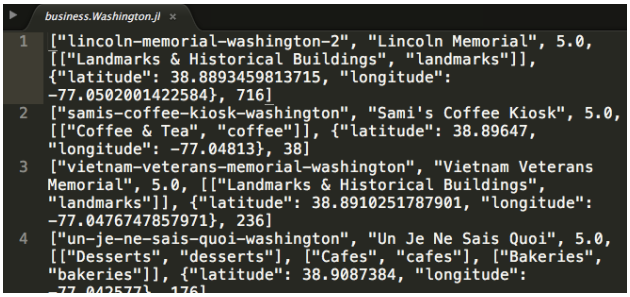
Second, run the scripts with python 2.



Now, the results will be stored in “data/business.Washington.jl”



Here is the results:

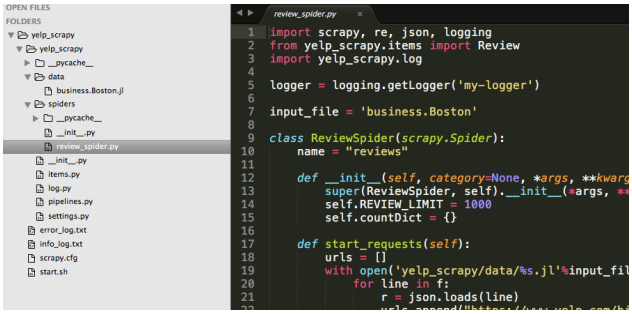


Each line is a record of business, including business_id, business_name, overall rating, categories, location and review count.

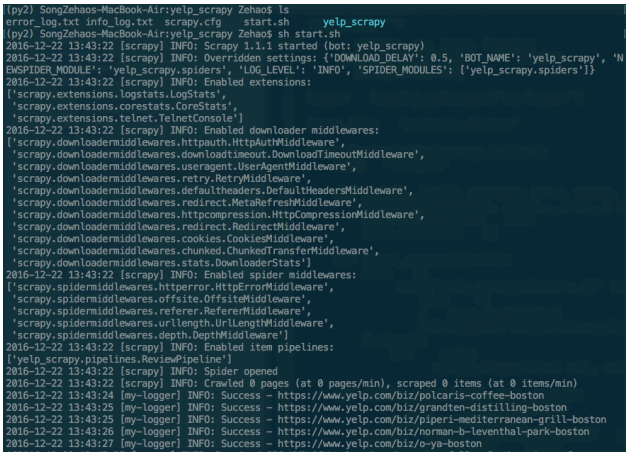
2. Review Spider

Path: scrapy/yelp_scrapy

First, copy the business list file collected in the previous step into “scrapy/yelp_scrapy/data” folder and change the input_file variable to proper path.

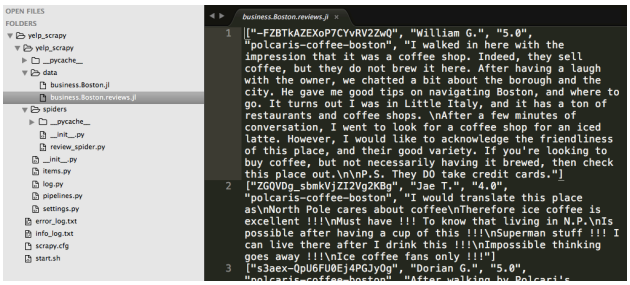


Then, in terminal, cd to the root folder of yelp_scarpy and run “sh start.sh”.



Note: Yelp might ban your IP.

The results are saved in “yelp_scrapy/data/business.Boston.reviews.jl”.



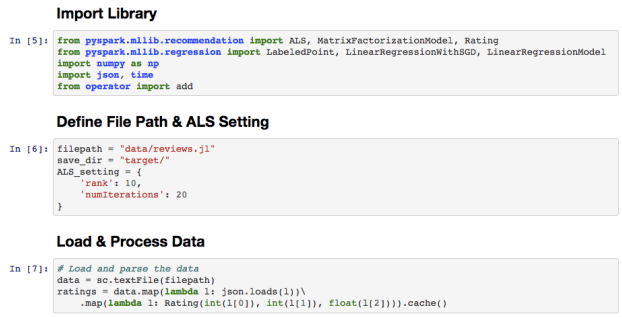
Each line is a record of review including user_id, user_name, rating and review content.

3. Train Model

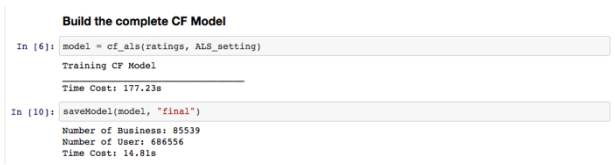
Path: spark/recommendation.ipynb

Open the file in Jupyter Notebook and run the scripts in order.

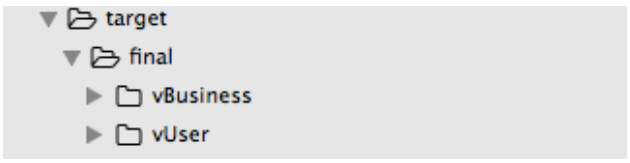
The configure the ALS_setting to change the training parameters.



Train & save the model.



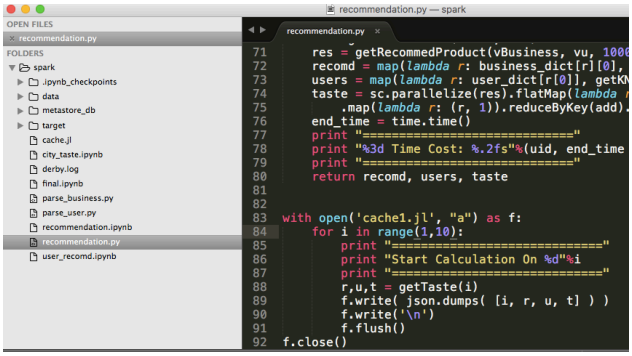
The saved User & Business Feature Matrix.



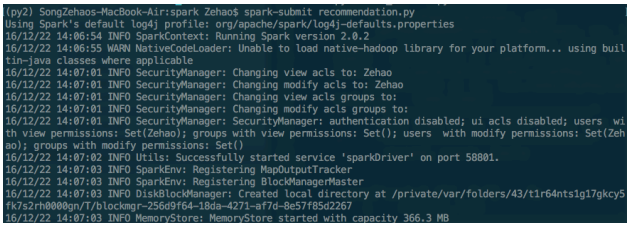
4. Recommendation

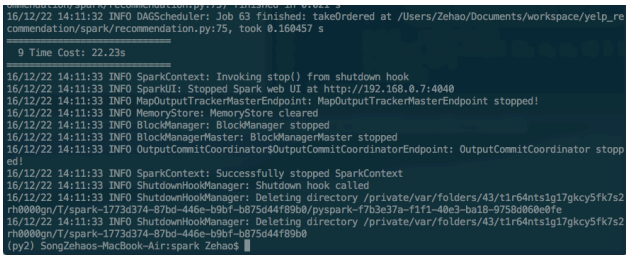
Path: spark/recommendation.py

First, modify the range of users to be recommended and the save file path.



Second, run the scripts “spark-submit recommendation.py”.





The results are saved into “cache1.jl”.

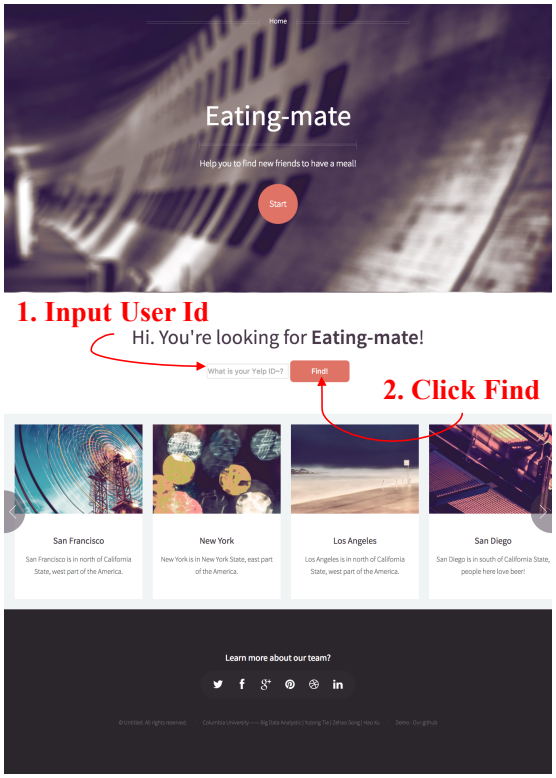
Each line is a recommendation for a particular user, including user_id, list of recommended restaurants, list of eating mates and taste analysis.

5. Web App

Path: web/web.py

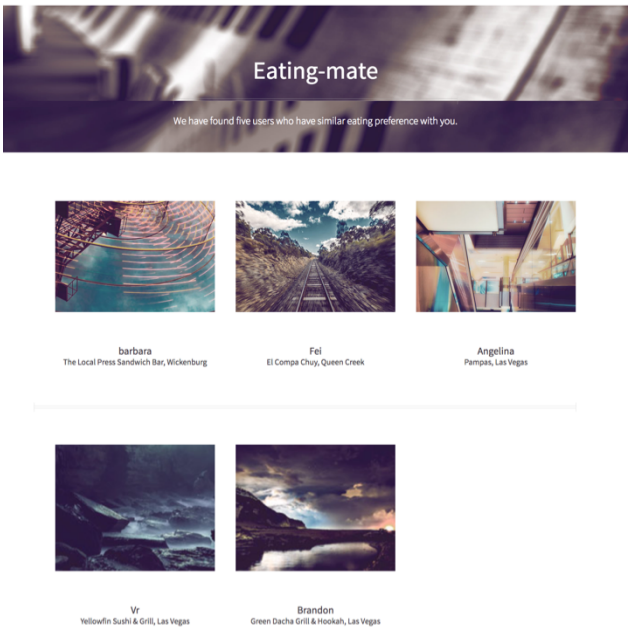
Start the Flask Server by “python web.py”. Note, python 2 is required.

The main page of the web. Input the user id to be recommended. Then, click the “Find” button. The page will be redirected to the result page.



The analysis of the user's taste. The result is shown in word cloud.

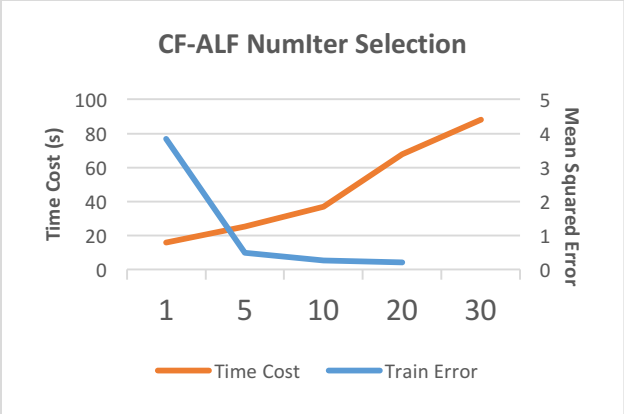
Recommended eating mates and restaurants.



VI. Experiment Results

1. Collaborative Filtering (CF)

As the rank grows, the train error decreases and the time cost increases. Although the long training time is acceptable since we only need to train one time, we should choose a relatively small rank. Because the number of reviews provided by new users must larger than the rank, otherwise the Linear Regression will fail. So. **Rank 20** is chosen.

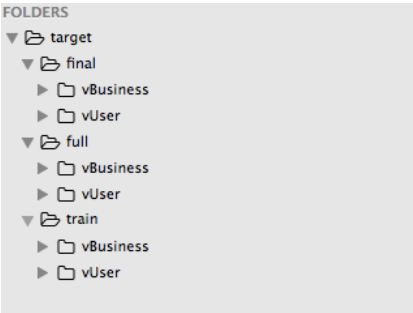


Number of iteration of ALS is the second parameter to be tuned. The larger the iterNum, the lower the train error & the higher the time cost. Since we only need to train once, we prefer to chose a relatively high numIter. Thus, $numIteration = 20$ is chosen.

Builde CF Model based on training data

```
In [8]: trainModel = cf_als(train_ratings, ALS_setting)

Training CF Model
_____________________________________
Time Cost: 67.94s

In [9]: cf_als_eval(trainModel, train_ratings)

Evaluating CF Model
_____________________________________
Mean Squared Error = 0.202851587955
Time Cost: 117.99s

In [35]: saveModel(trainModel, "full")

Number of Business: 85538
Number of User: 685557
Time Cost: 14.18s
```

The user and business feature matrixes are saved as pickle file for future usage.



2. Linear Regression (LR)

```
In [20]: vu, err = getPredictedUserFeature(9, test_ratings, vBusiness)
         vu

Out[20]: array([ 0.85343739, -1.01137154, -1.61861862, -0.8111291 , -1.14654225,
                -0.84237122, -0.03597422,  0.3464966 , -0.56901162, -0.25857191])
```

3. K-NN & Recommend

Get the eating mates of a particular user (vu)

```
In [22]: getKNN(vUser, vu, 1000)

Out[22]: [(111900, 0.57082987474015556),
          (470260, 0.5805114246678903),
          (95447, 0.6244813045550115),
          (299306, 0.62680960603461477),
          (169270, 0.63099199680208717),
          (358501, 0.63812444296578785),
          (172936, 0.64374111614503959),
          (546037, 0.64735737398127591),
          (1070, 0.65324326022194978),
          (475477, 0.65700562683601915),
```

Eating mate user id, similarity & recommended shared reataurants.

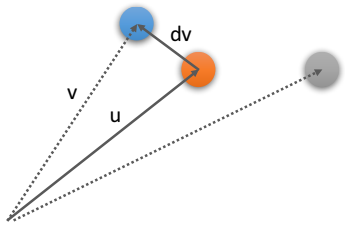
```
In [17]: evalEatingMateRecomd(9, test_ratings, trainModel)

111900 79% [70360, 76034, 54231, 82374, 75712, 57194]
470260 78% [70360, 78534, 83881, 76034, 83340]
 95447 77% [70360, 82374, 54231, 76034, 26317]
299306 77% [70360, 83340, 82374, 75712, 54231]
169270 77% [70360, 76034, 54231, 83340]
358501 76% [70360, 82374, 54231, 83340, 57194, 78534, 26317]
172936 76% [70360, 26317, 78534, 57194, 83340, 82374, 54231]
546037 76% [70360, 57194, 83340, 78534, 76034, 26317, 83881]
  1070 76% [54231, 78534, 70360, 76034, 82374]
475477 76% [70360, 57194, 76034, 26317]
Time Cost: 35.28s
MSE of User Vector: 1.11
```

Evaluation:

The evaluation of recommendation system is hard. In real world, companies could use A/B test to compare different recommendation algorithm. Obviously, we don't have such circumstance.

Let's see whether the recommended eating mates is the real friend of the original user.

Concept of Similarity



$$similarity = \max\left\{0, \quad 1 - \frac{\|dv\|^2}{\|v\|^2}\right\}$$

From the previous result, we could see that the recommended eating mates has a similarity larger than 70%.

If we check the real friends of the users, we could plot the histogram of their similarity between the base user.

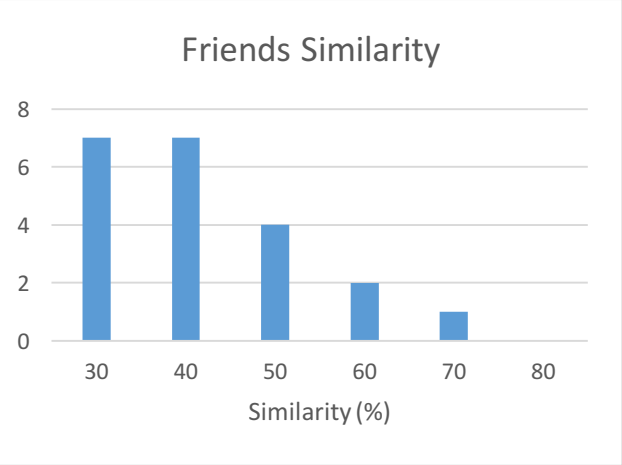
Count the similarity between user and his friends

```
vu, err = getPredictedUserFeature(9, test_ratings, vBusiness)
sims = []
for fid in friends:
    if fid <= 1000: continue
    vf = getUserFeature(vUser, fid)
    dv = np.linalg.norm(vu-vf)
    sims.append( int(100 * getSimilarity(dv, vu)) )
print sims
```

```
[32, 39, 58, 14, 56, 17, 13, 20, 0, 0, 5, 0, 28, 49, 19, 0, 47, 38, 0, 0, 29, 0, 0, 0, 11, 61, 0,
 44, 0, 3, 11, 24, 11, 0, 0, 0, 0, 0, 8, 25, 5, 0, 0, 28, 20, 0, 18, 0, 0, 0, 36, 0, 0, 36, 0, 0,
 0, 0, 16, 16, 17, 38, 0, 0, 48, 4, 0, 34, 0, 0, 0, 0, 1, 0, 0, 0]
```

```
hist = np.histogram(sims, np.arange(0, 110, 10))
hist
```

```
(array([44, 11,  7,  7,  4,  2,  1,  0,  0,  0]),
 array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100]))
```



It seems that the similarity between friends are not as high as the recommended eating mates. It's reasonable since in the model we only consider the ratings of businesses. But this result also indicates our naïve CF recommendation system is not good enough. We should take into account more other features of users and businesses, like location and price. A simple solution is to recommend a potential restaurants pool of large size (eg. 1000) and then use other algorithm to filter the results. Constrained by the time, we didn't implement this idea.

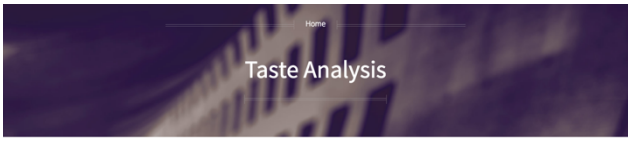
4. Taste Analysis

```
In [85]: recomd, users, taste = getTaste(9)
         Time Cost: 13.86s

In [86]: json.dumps(taste)

Out[86]: '[["Pizza", 117], ["Mexican", 114], ["Sandwiches", 108], ["American (Traditional)", 78], ["Italia
         n", 77], ["Fast Food", 74], ["American (New)", 69], ["Chinese", 65], ["Japanese", 51], ["Burgers",
          45], ["Cafes", 45], ["Mediterranean", 38], ["Sushi Bars", 35], ["Delis", 35], ["Thai", 31], ["Sea
         food", 30], ["Asian Fusion", 27], ["Coffee & Tea", 27], ["Steakhouses", 23], ["Barbeque", 23], ["C
         hicken Wings", 22], ["Middle Eastern", 20], ["Salad", 19], ["Bakeries", 18], ["Indian", 18], ["Gre
         ek", 17], ["Vegan", 17], ["French", 17], ["Vegetarian", 17], ["Latin American", 16], ["Hot Dogs",
          16], ["Juice Bars & Smoothies", 15], ["Sports Bars", 14], ["Hawaiian", 14], ["Vietnamese", 14],
          ["Food Trucks", 14], ["Gluten-Free", 13], ["Desserts", 13], ["Specialty Food", 13], ["Pubs", 10],
          ["Ice Cream & Frozen Yogurt", 10], ["Cocktail Bars", 10], ["Korean", 9], ["Wine Bars", 9], ["Ethn
         ic Food", 8], ["Grocery", 8], ["Soul Food", 8], ["Caribbean", 7], ["Soup", 7], ["Tex-Mex", 7]]'
```

After getting the list of categories frequency in the following format,

```
[
    ["Category Name", 30]
]
```

Generate the word cloud in front end via WordCloud2.



VII. CONCLUSION

Conclusion.

In this paper, we covered the entire workflow of our project, dive deep into the algorithms we chose to use, the dataset we collected in our own hand, the process of hyper parameter selection and evaluation of algorithms.

To sum up, we have built an eating mate recommendation system successfully. The system only needs some piece of information about user's reviews, then it would generate a feature vector of that user, find the nearest neighbor and finally output the eating mate that is recommended. We also deployed a site that supports eating mate recommendations. The result of recommendation is a combination of eating mate and a list of restaurant that could satisfy both users. According to several test results, the recommendation is reasonable and future customer feedback is to be collected.

Contributions.

Major works for each team member are listed as follow:

Hao Xu (hx2208):

1). Implement a script to get business ids from Yelp with Yelp search API.

2). Preprocess the data collected by Scrapy and map each user and business into our preferred ids.

3). Final Presentation (Slides & Present).

4). Final documentation.

Zehao Song (zs2324):

1). Implement python crawler via Scrapy to collect data from Yelp by using business ids collected by Yelp API.

2). Project proposal presentation in front of the class.

3). Design and develop algorithms for recommendations and run hyper parameter experiments and evaluate the result.

4). Deploy a server that supports the recommendation.

5). Final documentation.

6). Video Demo

Yutong Tie (yt2549):

1). Design PowerPoint for project proposal presentation.

2). Develop a front end display that supports eating mate recommendation as well as word cloud display.

Future works.

Due to the time limit, our project is not as perfect as we designed at the very beginning, therefore, several future works are needed to enhance the performance of our project:

1). More data

Currently, due to a limitation of the dataset we collected, we only support making recommendations for Las Vegas, San Francisco, Pittsburgh, Charlotte. In the future, we want to extend this project into the whole United States areas, which requires to collect review data all over the nation.

2). Scalability

While the size of dataset is increasing, we need to guarantee the performance of our algorithm, which should not be effected too much by the size of dataset and the running time of the whole process should not take too long, therefore, when the dataset is large, we should have method to deal with it, such as sampling, etc.

3). Dashboard

As we dive deep into the project, we found some interesting features other than eating mate could be derived from the dataset we collected. For example, city tastes. We could have a dashboard to display some interesting features, which could make our website more attractive.

4). Feedbacks

As mentioned in the final presentation, it is hard for us to evaluate the recommendation result. What we did in our experiment is to try to select the model that makes less training error. Therefore, we need some real data, which reflects how well our recommendation performs. We could collect feedbacks from users that whether they are satisfied about the recommendation and comments like how could we make improvements.

VIII. ACKNOWLEDGMENT

We would like to thank Prof. Lin for this fantastic semester, the informative lectures and relax atmosphere.

We would like to thank each Teaching Assistant for their helps during office hours.

We would like to every team member for the productive ideas and supportive for each other.

IX. APPENDIX

X. REFERENCES

[1] T. McNichol, "Word on the streed," NYSE magazine, 2012. Retrieved from http://www.nysemagazine.com/statics/NYSE_fall_2012.pdf.http://www.nysemagazine.com/statics/NYSE_Fall_20

[2] Yelp. (2016). *About us.* Retrieved from https://www.yelp.com/about.

[3] Facebook. Avaliable: https://www.facebook.com.

[4] N. Vignesh, “How does Facebook’s friend recommendation system work”. Retrieved from https://www.quora.com/How-does-Facebooks-friend-recommendation-system-work

[5] Liknedin. Avaliable: https://www.linkedin.com

[6] Amazon. Avaliable: https://www.amazon.com

[7] Spark Collaborative Filtering - RDD-based API. Avaliable: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html