CS512 Course Project: Graph-based recommendation system for Yelp dataset challenge - Fall 2016

Hanxiong Wang
Rutgers University
Netid:hw277
Email: hanxiong.wang@rutgers.edu

***Abstract*— Recommendation system can be used in many commercial areas, e.g. food market, travel market and moive industry. In this project, a graph-based algorithm for a recommendation system is developed combining with collaborative filtering and item-based methond . We compute the accuracy and efficiency of the algorithm. The results shows that...**

I. Project Description

A. Stage1 - The Requirement Gathering Stage.

- The general system description: We develop a graph-based recommendation system combining both collaborative filtering and item-based method. The aim of the recommendation system is to provide users a most accurate recommendation list according to user habits. This system could be used in various business and commercial websites, e.g., Amazon, Yelp, IMDb and TripAdvisor, which includes almost all the daily area. Advantages and disadvantages of the algorithm for implementation will be shown. We utilize a free datasets including both user and business information from Yelp Dataset Challenge: https://www.yelp.com/dataset challenge.
- The three types of users: people who seeks for recommendations in traveling, restaurant/food or hotel accommodation.
- The user's interaction modes: the system uses a graphic user interface (GUI) as interaction mode with the user, where the user can input his/her information/preferences possibly through graph-based, discrete-feature terms or even simple personal identity. The system will output a set of nodes (i.e. suggested cities, restaurants or hotels, etc.) as graph or text format.
- The real world scenarios:
  - Scenario 1 description: A user of our recommendation system is concerned what to eat for dinner. He/She wants to pick some new restaurant that satisfies his/her taste best while maintaining a low spending according to the budget restraint. Our system will detect users that share similar tastes based on location, food style, and rating on restaurant that similar user has previously been to, and then recommend several restaurants to our user that he/she would potentially like to try, which also takes budget into consideration.
  - System Data Input for Scenario1: Spending, preferred cooking style, location, minimum overall rating
  - Input Data Types for Scenario1: Nodes on the graph represent users. Edges indicate what restaurant those users have been to. Weights of edges represent an overall measurement of how well a user likes a restaurant based on the input variables.
  - System Data Output for Scenario1: Restaurant recommended by our system.
  - Output Data Types for Scenario1: Nodes which represent restaurant.
  - Scenario 2 description: A user of our system is in town on a business trip,and he only has limited time for lunch, and he wants to explore the best restaurant within a certain distance so that he can finish his lunch in time. And he does not care spending. Our system will finds out possible restaurants nearby, and recommend a list that most fit the users taste by searching restaurant raters with similar taste to our user.
  - System Data Input for Scenario2: Preferred cooking style, location, maximum distance, minimum overall rating.
  - Input Data Types for Scenario2: Nodes on the graph represent different users. Edges indicate what restaurant those users have been to. Weights of edges represent an overall measurement of how well a user likes a restaurant based on the input variables.
  - System Data Output for Scenario2: Restaurants recommendation by our system.
  - Output Data Types for Scenario2: A list of nodes which represent restaurant.
- Project Time line.
  - Stage 1: Oct. 14 to Oct. 25, 2016 We will be working on the requirement gathering stage, which includes

topic and data selection, model selection, input and output set up, as well as labor division.
- Stage 2: Oct. 26 to Nov. 10, 2016 Will be working on the deign of algorithm stage. Transformation on the project requirement into a system flow diagram will be done. We will carry out high level pseudo code of the overall system operation, and give a statement on run time and space complexity.
- Stage 3: Nov.11 to Nov.24, 2016 Will be working on the implementation stage. Programming language and environment will be settled down. Code, demo and sampling of finding will be done by Nov.25
- Stage 4: Nov.25 to Dec.9, 2016 Will be working on the implementation of user interface. Our system will output proper recommendation based on the input.
- Final stage: Dec.9-Dec.16, 2016 Final report composition and presentation materials will be done by the due date.

B. Stage2 - The Design Stage.

- Short Textual Project Description.
  Here, for conveniently understanding and evaluating our project, a recommendation system for users who are in demand of public services, like restaurants, pet shops, or grocery stores, is illustrated as an example. From our Yelp Dataset, as mentioned above, we extract three key factors, including user's ratings, the category of a business unit and its attributes (i.e. prices for different foods, alcohol provider or not, payment ways, Wi-Fi service, good for group food or not, good for kids or not, easy-appointment access, reservation time and wheelchair, etc.). The relative information between these entities (users and business unit) shows as graph edge weights in our algorithm, and will be calculated using users' ratings. When executing our recommendation system, an optimal recommendation list will be generated based on built-in grading function; and the recommendation list is composed of a set of users and business unit with some specific relationship. For examples, a list of restaurants are rated about the same within a certain set of users.

  Flow diagram textual description:

  1. Obtain raw dataset and collect the clean data satisfying the requirement for the further analysis using the procedure "BUILD_GRAP_LIST", where the user, business units (business company) information and corresponding review (e.g. restaurant ratings) are collected and finally form an adjacent list for these information.
  2. Build up the objective graph using three procedures: BUILD_FIRST_LAYER, BUILD_SECOND_LAYER and BUILD_THIRD_LAYER. Inside the objective graph, the edges weights between nodes in different definition layers (users and business units) are generated based on our specifically-designed algorithms. collaborative filtering; content based.
  3. Generate a recommendation list with the procedure "FORM_REC_LIST" with providing an accuracy/quality rate from the procedure "EVALUATE_ACCURACY".
  4. Evaluate different recommendation lists by modifying and improving the algorithm input parameters with corresponding accuracy/quality rates by repeating Step 1to 3.
  5. Select the best recommendation list with the optimal accuracy/quality rate.

  In our algorithm, the space complexity is individual_node_unit * $O(N$ = numbers of users and business units). Because the multiple pointers exist (i.e. a user can rate a lot of business units), individual_node_unit can be considered to use two ways: i) an adjacent list to preserve these points (PROS: minimize the space complexity; CONS: increase the algorithm complexity when building up the objective graph); i) an adjacent matrix (PROS: easy to access and build up an objective; CONS: have huge waste in space if the maximum rating number is very high and few users have provided multiple rating in different business units). We need to build up the objective graph and search the entire graph, based on different implementation methods, the optimal time complexity in our recommendation system is $O(E*\log(N)) \approx O(N*\log(N)$ (if multiple ratings are few), E is the total edge.

- Flow Diagram.

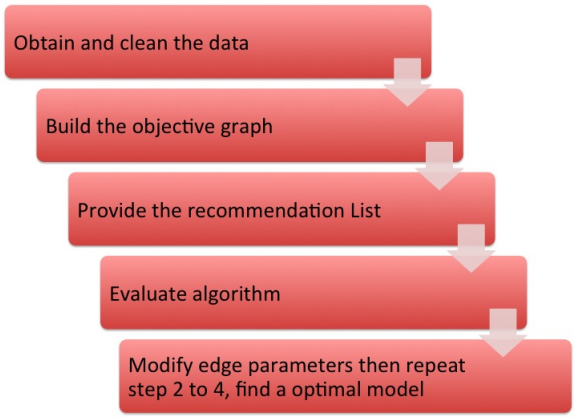

Fig. 1. Flow Diagram

- High Level Pseudo Code System Description.
- Algorithms and Data Structures.
  Our algorithm design bases on such a basic idea: users who have similar taste in business units tend to review a specific business units with close score. Such a set of users should be a potential consumer group for some business units in a particular category. For example, finding out the people who love French cuisine is very meaningful if we can also recommend some good French

Algorithm 1 Graph-based recommendation system

Clean and import the data
procedure BUILD_GRAP_LIST
user_info ← input user_info
business_info ← input business_info
review_info ← input review_info
G_list ← build a adjacent graph list by the above dataset

Build the objective graph
procedure BUILD_FIRST_LAYER(user node A, G_list)
find(first_ln) ← All nodes link to node A
for all $node \in first\_ln$ **do**
$node\_edge = RStar$
procedure BUILD_SECOND_LAYER(*first_ln, G_list*)
find(second_ln) ← All nodes link to the first layer
for all $node \in second\_ln$ **do**
$node\_edge = RStar + RAve()$
procedure BUILD_THIRD_LAYER(*first_ln*,*second_ln*,G_list)
find(third_ln) ← All nodes link to nodes in second layer
for all $node \in third\_ln$ **do**
if $node.cate \in set(first\_ln.cate)$ **then**
Aux(cate,attr) = Aux1(cate)+Aux(attr)
else
Aux(cate,attr) = 0
for all $node \in third\_ln$ **do**
$node\_edge = RStar + RAve() + Aux(cate, attr)$

Compute the longest path to each leaves and give the recommendation list
procedure FORM_REC_LIST(user node A, first_ln, second_ln, third_ln)
Rec_List ={}
for all $node \in third\_ln$ **do**
$node.lp()$ ← compute the longest path from node A by node.lp
Raw_Rec_List = sort(third_ln)
for all $element \in Raw\_Rec\_List$ **do**
if $element \in first\_ln$ **then**
do nothing
else
Rec_List.add(element)
return Rec_List

Provide the Accuracy Rate
procedure EVALUATE_ACCURACY count = 0
for $i \leftarrow 1$ to $length(first\_ln)$ **do**
if $Raw\_Rec\_List[i] \in first\_ln$ **then** count +=1
return Acc_rate = count / length(first_ln)

restaurant to them. Thus, the first thing we do here is to correlate the similar patterns of users' rating from different business units; then we can recommend the favorite business units of a user to other users who have similar taste.

The detailed contents and data structures of our algorithm pipeline are shown in "Algorithm 1". In the part I - "Clean and import the data", the raw data are provided and generated the specific data structure recording the ratings of each user for different restaurants: an adjacent list or an adjacent matrix for each user. In the part II - "Build the objective graph", the users' rating data are built up as an objective graph data structure with specifically-designed edge weights by connecting users and business units with similar ratings, etc. In the part III - "Compute the longest path to each leaves and give the recommendation list", we obtain list output using some algorithm similar to finding out the longest path in the directed acyclic graph (DAG). In the part IV - "Provide the Accuracy Rate", a double floating value is provided by in order accessing each unit of the recommendation list from the part III.

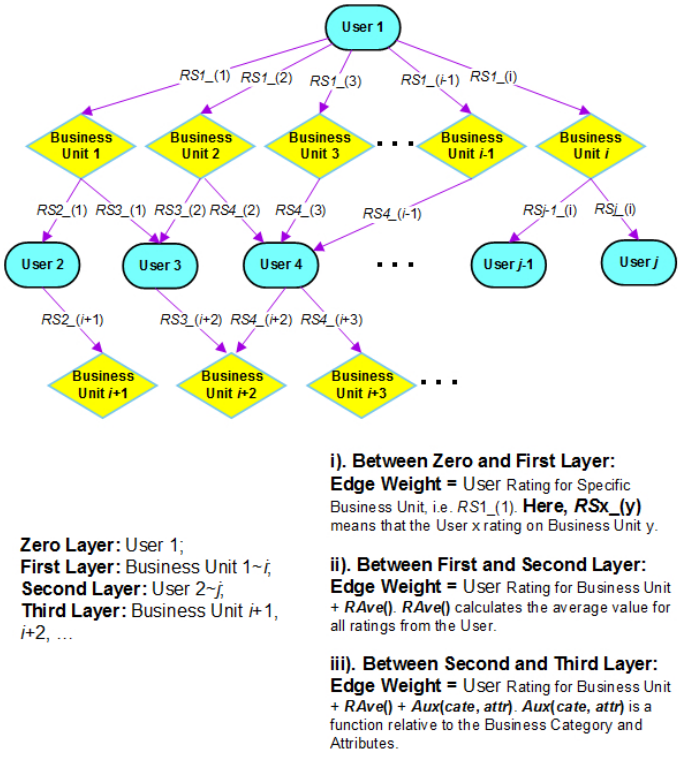

Fig. 2. The objective graph representation in Algorithm 1

- Flow Diagram Major Constraints.
  - Integrity Constraint.
    1. The input user need to have sufficient reviews on record. Our recommendation system provides recommendation list by finding similar users based on input user's reviews. And it does so by comparing the reviews on business units that serve same purpose.

The more reviews the input user has, the more accurate of measurement we have on its taste, and therefore, the better the recommendation can be.

C. Stage3 - The Implementation Stage.

- Sample small data snippet
  Our data sets are composed of three classes, which include review data, user data, and business data. As we have mentioned in the context above, our recommendation system focuses on Pennsylvania centered business. Eventually, we have 32134 as total number of users, 111542 of reviews, and 4086 of business units. The figure below displays an original sample data structure in each of the classes.

```
>>> data_user[1]
{'fans': 0, 'name': 'Stacey', 'elite': [], 'votes': {'cool': 1, 'useful'
: 5, 'funny': 0}, 'user_id': 'LF73iBr-ynbAi-swAQbGCQ', 'review_count': 7
, 'yelping_since': '2011-01', 'compliments': {}, 'average_stars': 4.43,
'friends': ['Io6At5jerTw_PpWj1hYXeA', 'kgr3sq5YHHDMBboawKZD1w', 'Bz-LTlf
j07PBNs48mZp0LA', '1PskDffSqJzjeDxNPZ4-hg', 'D2lUWZXJ7WJSDvY3lJUInw'], '
type': 'user'}
>>>
```
(a)

```
>>> data_business[1]
{'categories': ['Nightlife'], 'hours': {}, 'full_address': '202 McClure
St\nDravosburg, PA 15034', 'name': "Clancy's Pub", 'attributes': {'Price
Range': 1, 'Accepts Credit Cards': True, 'Outdoor Seating': False, 'Happ
y Hour': True, 'Good For Groups': True}, 'neighborhoods': [], 'latitude'
: 40.3505527, 'city': 'Dravosburg', 'stars': 3.0, 'business_id': 'UsFtqo
Bl7naz8AVUBZMjQQ', 'open': True, 'review_count': 5, 'longitude': -79.886
8138, 'state': 'PA', 'type': 'business'}
>>>
```
(b)

```
>>> data_review[1]
{'date': '2014-02-13', 'review_id': 'KPvLNJ21_4wbYNctrOwWdQ', 'stars': 5
, 'business_id': '5UmKMjUEUNdYWqANhGckJw', 'votes': {'cool': 0, 'useful'
: 0, 'funny': 0}, 'user_id': 'Iu6AxdBYGR4A0wspR9BYHA', 'type': 'review',
'text': "Excellent food. Superb customer service. I miss the mario machi
nes they used to have, but it's still a great place steeped in tradition
."}
```
(c)

Fig. 3. Sample data: (a). User Data; (b). Business Data; (c).Review Data

We consider a variable significant if it can directly reflex or indirectly help to interpret a user's preference on business units. All the significant variables are then taken into consideration in the recommendation process, with more or less weight.

For user data, average stars is the average review (level from 0 to 5) a user gave. User id is a unique id code assigned to each user. It is a key variable that relates the three classes of information. And review count, as the total number of review a user has given, will also be used, particularly in the evaluation step.

For review data, stars indicate how much the reviewer liked this business unit. As star increases, the reviewer likes more. Business id represent each business unit. Finally, user id in review data indicates from who this review was given.

As for the business data, categories and attributes will be used as characteristic description of a business units through our recommendation system. Business id is a unique code that represents a business store.

- Sample small output
  Given an eligible input user number, the recommendation system generates a recommendation list composed of all business units that has connection to the input user. Each element in the recommendation list has a total score measuring how similar this store is to something the user liked in the past. Our system chooses business units with greatest score by determining the longest path from the root node to each third layer nodes. Then it outputs the top ten on which input user has not reviewed before. The result will be an optimal solution based on our item plus user based collaborative filtering algorithm. Below is an example of data output for user 74.

```
Top ten recommend list:
Sausalido
Yoga Flow
Craftwork Kitchen
D's Six Pax & Dogz
E2
Red Robin Gourmet Burgers
Murray Avenue Grill
Evolve Wellness Spa
Curry on Murray
Meat & Potatoes


Spending time is  144.472801
```

Fig. 4. Recommendation list for user 74

As a result, our system is built with relatively hight accuracy and stability. We will show this in following section.

- Working code
  The code package, as a compressed file with the zip format, is attached in the report. Package includes two zip file named as: maincode.zip and data_preparation.zip.
- Representative of Sample Dataset and Data Size Analysis
  Due to the relatively huge volume of the "yelp academic dataset", it requires a long processing time for a normal desktop/laptop to execute the data mining task through the whole data set. Thus, only the data from the state of Pennsylvania (labeled as PA in the dataset) are extracted in execution as a sample/demo of our recommendation system. Based the design, our system mainly composes of two sub-procedures: i). generating of clean dataset with specific data structure; ii). Simulating a recommendation list of business units with a given user ID. Here, we provide a space analysis and a demo in the Non-graphical interface.
  There are three subsets in the PA data related to business dataset, users dataset and review dataset preserved as the specifically compressed format with sizes of about 5 MB, 19 MB and 111 MB, respectively. All of them are loaded in the RAM. In order to conveniently investigate the required RAM space for the objects during the execution, we utilize the functions in the packages of "cython" and "meliae" of Python. These data are separately loaded in the RAM and checked for the required space. The results are shown in Fig. 5. We can see that review dataset

requires the maximum RAM space. It is reasonable because the reviews might include some statements with connecting both users and business units. As seen, most of the space are dominated by the unicode strings data (52%, 47% and 64% for the business, user and review dataset, respectively). The total RAM is 555 MB when loading all dataset into RAM with 49 types of variables, which is shown in Fig. 6.

```
Total 306252 objects, 49 types, Total size = 36.3MiB (38018289 bytes)
 Index   Count   %      Size   % Cum     Max Kind
     0   36220  11  19958944  52  52   49432 dict
     1  239562  78  15700922  41  93     242 unicode
     2    8294   2    941848   2  96   36120 list
     3    4408   1    317603   0  97    4871 str
     4   12260   4    294240   0  97      24 float
     5      52   0    204096   0  98   12624 module
```

(a)

```
Total 1032494 objects, 52 types, Total size = 116.5MiB (122117170 bytes)
 Index   Count   %      Size   % Cum     Max Kind
     0  815565  78  57441570  47  47     108 unicode
     1   96674   9  54344240  44  91   49432 dict
     2   64392   6   8104192   6  98  824472 list
     3   32136   3    771264   0  98      24 float
     4   14848   1    356352   0  99      24 int
     5    4413   0    317924   0  99    4871 str
```

(b)

```
Total 2129499 objects, 51 types, Total size = 405.9MiB (425582438 bytes)
 Index   Count   %      Size   % Cum     Max Kind
     0 1896213  89 274919940  64  64   10048 unicode
     1  223356  10 148572576  34  99   49432 dict
     2     122   0    969352   0  99  954120 list
     3    4413   0    317862   0  99    4871 str
     4      52   0    204096   0  99   12624 module
     5     148   0    133792   0  99     904 type
```

(c)

Fig. 5. The require RAM analysis for the input dataset: (a). only business dataset loaded; (b). only user dataset loaded; (c). only review dataset loaded, respectively. The analysis results are from the package of "meliae" in Python.

```
Total 3447705 objects, 49 types, Total size = 554.8MiB (581724482 bytes)
 Index   Count   %      Size   % Cum     Max Kind
     0 2951340  85 348062432  59  59   10048 unicode
     1  355706  10 221986160  38  97   49432 dict
     2   72564   2   9160352   1  99  954120 list
     3   44394   1   1065456   0  99      24 float
     4   14902   0    357648   0  99      24 int
     5    4383   0    316470   0  99    4871 str
     6      52   0    204096   0  99   12624 module
     7     148   0    133792   0  99     904 type
     8     642   0     77040   0  99     120 function
     9     596   0     76288   0  99     128 code
    10     914   0     73120   0  99      80 wrapper_descriptor
    11     469   0     38992   0  99     288 tuple
    12     482   0     34704   0  99      72 builtin_function_or_method
    13     108   0     28640   0  99    2280 set
    14     343   0     24696   0  99      72 method_descriptor
    15     229   0     20152   0  99      88 weakref
    16      48   0     16512   0  99     344 WeakSet
    17      16   0     14464   0  99     904 ABCMeta
    18     175   0     12600   0  99      72 member_descriptor
    19     104   0      7488   0  99      72 getset_descriptor
```

Fig. 6. The require RAM analysis for loading the whole representative sample dataset collected from the package of "meliae" in Python.

- Sample findings and Demo Result
  As pointed out in the "Integrity Constraint" subsection, the performance of recommendation system depends on the dataset. In fact, this is one of the basic problem in any task relative to data mining system: valid data is often small. In our system, we can roughly estimate the size of valid data using two rat parameters: the reviews per business unit and the reviews per user. To some extent, we will hope these two ratios are high, which means that there are relationships between the observed users and business units. In our sample data, the reviews per business unit and the reviews per user are 27 and 3. The reviews per business can provides a reference what is a roughly average depth of a recommend graph/tree in our system; in the other hand, the reviews per user can provides a reference about how many new users involved in the next level. To analyze both them in whole dataset might be an extremely complicated task considering that each user has its specific recommendation graph network. Thus, to be simple, we only analyze the distribution of review count per user as shown in Fig. 7. The distribution is a smooth decreasing curve with the highest point in 1. This means that about 5000 of 32134 users have only a review. As we can see, the curve decrease very fast and the count of users is about 500 when user's review number is at 15. This indicates that only part of users kept writing down their reviews for different business units; this also means that these users have more relationships with others. Our system can provide better recommendation for these users by constructing their graph network with more features.

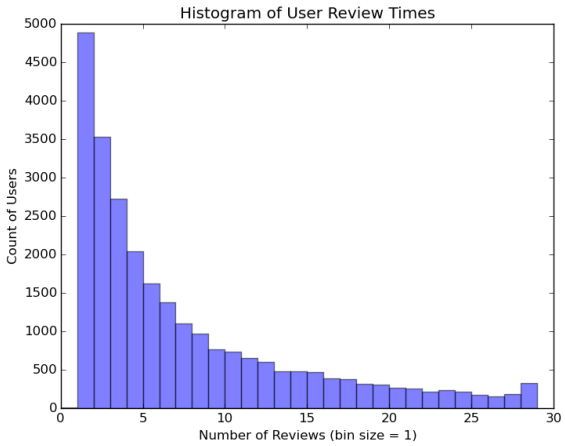

Fig. 7. Histogram of the reviews for different business units.

A demo in the Non-graphical interface is also displayed in Fig. 8. Here, our system simply accepts the user ID (an integer number) as the input rather than a users' name. Then, a network graph/tree is constructed for the specific input. By following this, our system will extract the existing attributes from the network to look for favorite recommendation with the setting parameters. Finally, a recommendation list including various business units is displayed. This list indicates this user might be interested at looking for the potential consumption from these business units. The performance and execution of our systems strongly depends on the number of existing

relationships between the input user and other users. More relationship will induce better performance with fast increasing running time requirement, mainly from the construction of a relationship network.

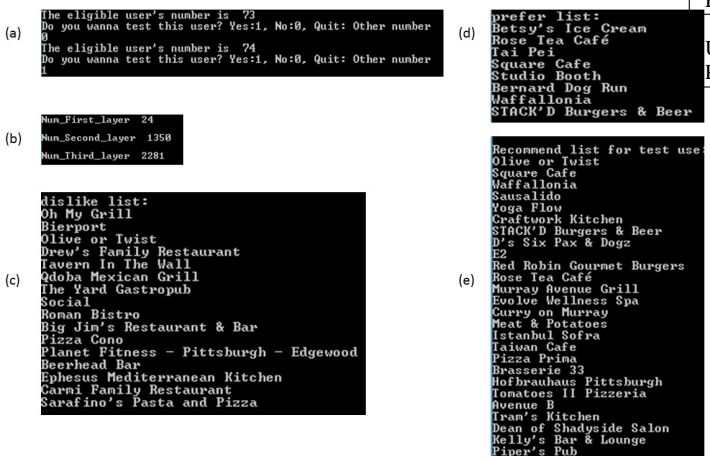

Fig. 8. A demo in the Non-graphical interface: (a) In cmd, using commend "python RecomSys.py" to run the system. A user ID is provided as input. You can execute our recommendation by clicking 1 or skip by clicking 0. Clicking 0 will generate another user ID. Just click any key rather than 0 or 1 to turn off the execution. (b) The number of nodes (Business Units or Users) in the first three layers are displayed. The first and third layer includes the business units as nodes and the second layer are the users as nodes, who have reviewed the business units in the first layer. (c) A disliked business unit list according to root user in the first layer is also provided. (d) A preferred business unit list according to root user in the first layer is represented. (e) Part of the recommendation list for the test use.(Not the final recommendation list, need remove the business unit that user has already reviewed.)

- Evaluation on the Algorithm Performance
  It is hard to know if a user is satisfied with our recommendation list, like a example given in Fig. 4, without any survey. Thus, we use the following method to evaluate our recommendation system roughly:
  Based on existing reviews, we can divide a user's review record into two sets: i). his/her well-liked business units and ii). disliked business units. To evaluate the performance of our recommendation system, we should investigate how well the recommendation list can match in such two set. We will take a pool of size a, which is the size of set of well-liked business units, from top of the long recommendation list. We look into the pool, and find out how many business units that the input user reviewed and liked are on list. If there are many such units being re-recommanded, it means the system works well on finding business unit that matched the user's taste. On the contrary, if the pool includes business units that the user reviewed and disliked, we consider it as a poor performance. Then we artifically enlarge the size of pool as 2 a, check the performance and stability of the algorithem in this stage.
  Because of the time limitation and a bit huge computation, we randomly take 100 eligible users to test the performance of our system and take the average

TABLE I
EVALUATION ON THE ALGORITHM PERFORMANCE.

	$\|\text{Size of Pool}\| = a$	$\|\text{Size of Pool}\| = 2a$
Correction Prediction Rate (CPR)	38.1%	50.2%
Uncorrection Prediction Rate (UPR)	5.4%	8.9%

performance. Table I shows the results illustrated by two parameters: Correct Prediction Rate (CPR) and Uncorrect Prediction Rate (UPR). The definitations on CPR and UPR are given in the equations 1 and 2.

$$\text{CPR} = \frac{L}{a}, \tag{1}$$

$$\text{UPR} = \frac{D}{b}. \tag{2}$$

where L is the number of business units in the pool correctly recommended by the prediction. This is, they are included in the list of the user's favorite business units. Relatively, D is the number of business units in the pool not matching the user' preference. a is the number of the user's well-liked business units and b is the number of the user's disliked business units in his reviews, respectively.

As shown in Table I, the CPR values are 38.1% and 50.2% comparing with the UPR values as 5.4% and 8.9%, respectively. Obviously, the higher the CPR value, the better the recommendation list. What about the UPR? Our system tries to minimize the UPR value. There are two reasons here. First, if both CPR and UPR are close, it will indicate that our system have weak ability to differentiate a user's favorite business units. In the other, our system also tends to find out user's potentially favorite business units, which have not yet been reviewed by the user. It can be seen as a pure prediction behavior. If the UPR is high, it possibly reveals that these potential business units do not match the user's preference. Based on these viewpoints, our recommendation system has exhibited a considerable and convincing performance. If we double the size of the pool, we observe an increase in both CPR and UPR. Table I shows that the CPR increases to 50.2% from 38.1%. We found that over half of the preferred business units are in the pool, while the un-preferred business units did not appear very often. UPR stays low as we increase the size of pool. This results shows that our recommendation system truly detects items that match user's preference. In another word, the disliked items are put in the bottom of the list, while the preferred items are on the top and to be recommended to the user. The result also proves a good stability. Again, the final recommendation list will be the top 10 business units from the long recommendation list that input user has

not yet reviewed as shown in Fig. 4.

D. Stage4 - User Interface.

We developed a User Interface using the package *Tkinter*. *Tkinter* is a standard Python interface to TK Gui toolkit, which allows us to generate an interface to guide users through our recommendation system. Figures below are the demonstration of the UI.

- The initial statement to activate the application with the corresponding initial UI screenshot.
  Run the Python script named *GUIpart* to start the UI. The initial nterface is shown below:

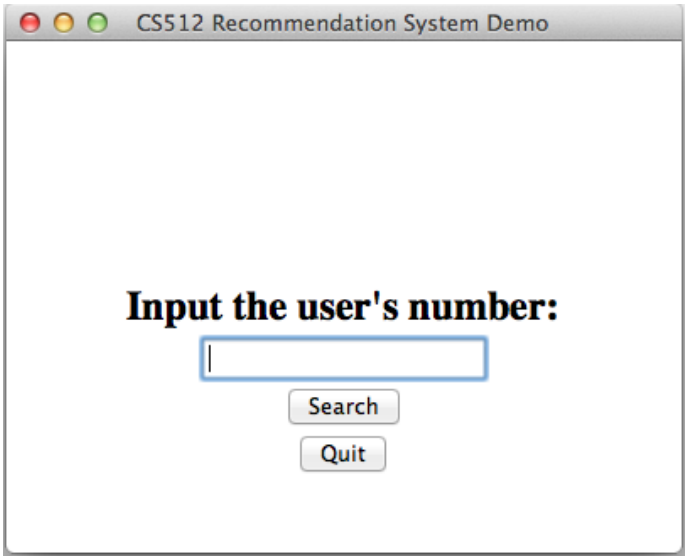

Fig. 9. An example of finding the recommendations for user 1

- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The error message: This is an invalid input, please input an integer between 0 to 32133.
  - The error message explanation (upon which violation it takes place): The error pops up when the input number is out of range, i.e. the input user is not in the database. So the system cannot find such user.
  - The error message example according to user(s) scenario(s): Input user number 1000000.
- The information messages or results that pop-up in response to user interface events.
  - The information message: shown in Fig. 11.
  - The recommendation list contains 10 business units that are most likely favorable by input user.

II. Project Highlights.

III. Appendix

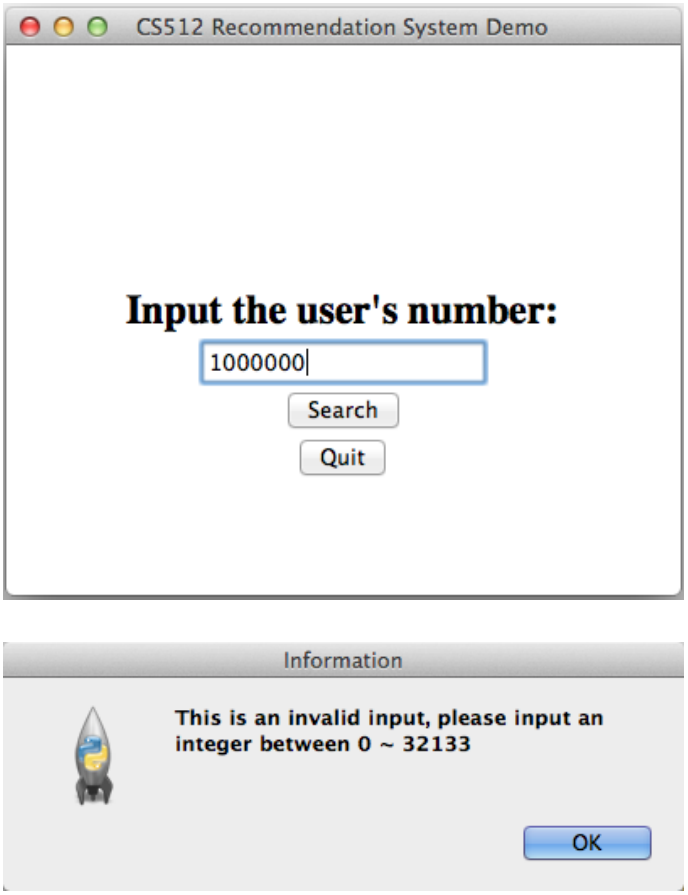

Fig. 10. Error message for inputting invalid user number 1000000

A. Details on edge weight formula in each layers

- First layer edge weight
  $node_edge = RStar$
  Recall the first layer of our graph is business units that have been reviewed by the input user. The weights on the first layer edges (edges points to first layer) are simply the review of a store by the input user, Users can give review of number from 0 to 5, indicating how much they like the store.
- Second layer edge weight
  $node_edge = RStar + RAve()$
  Recall the second layer of our graph are users who have reviewed a store in the first layer. The weights on the second layer edges are reviews of users in the second layer, adjusted by the average review score for all ratings the particular user previously gave. For example, an edge from layer 1 to 2 indicates how much the store in layer 1 is liked by a user given the user's average review.
- Third layer edge weight
  $node_edge = RStar + RAve() + Aux(cate, attr)$
  Recall the elements in the third layer are stores that users in layer two have reviewed. Each of the element in layer three will potentially be in the

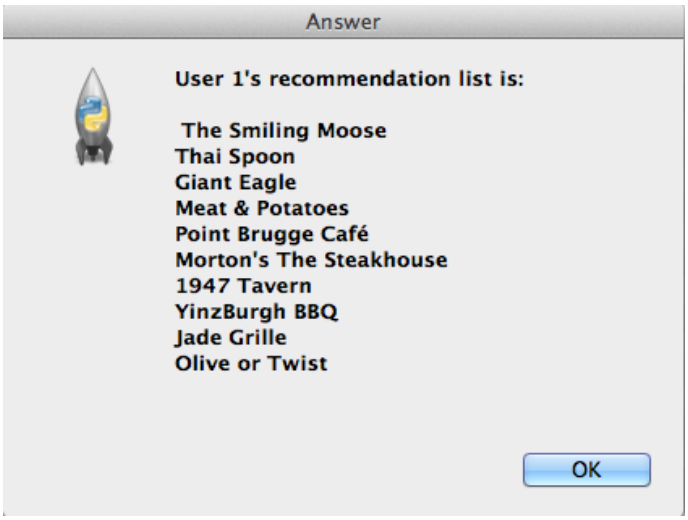

Fig. 11. UI demonstration of resulting recommendation for user 1

recommendation list. Here we weigh the third layer edges by adjusted reviews, same as second layer edges, and a auxiliary function which looks into category and attribute of business stores, and further detects similarities between users. For each of the stores in layer 3, we extract it's business category, denoted by A for convenience. Then we calculate the probability that category A appears in layer 1. Note that the probability represents level of demand in this category by the input user. Suppose the user has a very high demand in category A, we assign this category a higher weight. And on the other hand, if the need is 0, we conclude that the input user does not need recommendation on this kind of service, and we assign a tiny weight to stores in this category, so that it is unlikely to appear in the recommendation list. Attribute factor works in similar way as category.