EAI Endorsed Transactions
on Industrial Networks and Intelligent Systems Research Article EAI.EU

Improving Customer Behaviour Prediction with the Item2Item model in Recommender Systems

T.T.S. Nguyen[1,*], P.M.T. Do[1] and T.T. Nguyen[2]

[1]School of Computer Science & Engineering, International University, VNU-HCMC, Ho Chi Minh City, Vietnam
nttsang@hcmiu.edu.vn, dophamminhthu2403@gmail.com

[2]University of Buckingham, UK
tuan.nguyen@buckingham.ac.uk

Abstract

Recommender Systems are the most well-known applications in E-commerce sites. However, the trade-off between run-time and the accuracy in making recommendations is a big challenge. This work combines several traditional techniques to reduce the limitation of each single technique and exploits the Item2Item model to improve the prediction accuracy. As a case study, this paper focuses on user behaviour prediction in restaurant recommender systems and uses a public dataset including restaurant information and user sessions. Within this dataset, user behaviour can be discovered for the collaborative filtering, and restaurant information is extracted for the content-based filtering. The idea of the pre-trained word embedding in Natural Language Processing is utilized in the item-based collaborative filtering to find the similarity between restaurants based on user sessions. Experimental results have shown that the combination of these techniques makes valuable recommendations.

Keywords: recommender systems, sequence mining, item2item.

Received on 31 August 2018, accepted on 12 December 2018, published on 19 December 2018

doi: 10.4108/eai.19-12-2018.156079

1. Introduction

It is undoubted that a person can make his decisions from when getting enough information. To make right decisions from enormous information, however, is not a trivial task. A Recommendation System or Recommender System is introduced to handle this problem. It is a type of information retrieval systems which are used to predict items of interest for a specific user. Recommendation Systems are categorized into different types, e.g. Content-based filtering, Collaborative filtering and combination of these techniques. Content-based filtering algorithms base on similar items and item descriptions. They, then, try to recommend items similar to what a user has purchased or has been interested in. Therefore, user profiles are important in these techniques. Whereas, collaborative filtering algorithms use the same tastes between user-to-user to suggest items.

Collaborative filtering technique has two types, user-based collaborative filtering and item-based collaborative filtering [1]. User-based collaborative filtering algorithms try to find the nearest neighbours of a target user by computing similarity between users, and then give suggestions based on these neighbours' interests. In another hand, item-based collaborative filtering algorithms take into account user-user relations through item-item relations in order to recommend items close to the target user's interests.

The purpose of this paper is to build a hybrid restaurant recommendation technique which is the combination of content-based filtering, user-based collaborative filtering and item-based collaborative filtering.

Firstly, the cosine similarity in the content-based filtering process is used to calculate distance between

*Corresponding author. Email: nttsang@hcmiu.edu.vn

restaurant profiles to find the nearest neighbours for each restaurant visited by an active user. The most similar restaurants are selected for recommendation. Secondly, in the user-based filtering process, user-to-user similarity is represented by Euclidean distance. Users who are the most similar to the target user are selected. To reduce the number of distance computations, users are organised into groups by a clustering algorithm. We, then, only need to evaluate distances between users in the same group. Thirdly, in item-based collaborative filtering process, the Word2Vec model, a pre-trained word embedding, in the natural language processing [2] is utilised to learn relationships between restaurants in user sessions and then recommend interesting restaurants. The final suggestion is the combination of results from each single technique.

The paper is presented as follows. Section 2 briefly introduces related work; Section 3 discusses methodology; Section 4 summaries Experimental results, and finally conclusions in Section 5.

2. Related work

As known, two main approaches to Recommender Systems are content-based and collaborative filtering. Both need a sufficient amount of data for recommendations but different types of data. The content-based filtering needs item descriptions and user profile, whereas the collaborative filtering needs transaction patterns or user transaction history.

The advantage of **content-based recommender systems** is that it can start to recommend as soon as there are item descriptions. In other words, new items can be suggested before being rated by users. When users interact with the system, user profile will be collected as input data. Providing more inputs, such as item descriptions and user profile, item recommendation becomes more and more accurate. However, this recommendation does not depend on the similarity between users, though this information may be useful.

In contrast, the recommendation engine of the **collaborative filtering** is based on the real-life activity, sequences of user behaviour or user sessions, so the more people interact with system, the better recommendations can be made. The strong points of this technique are no item description needed and it can capture changes of user interests over time. However, this type of system cannot give good recommendations to new users because it needs user's rating, transaction history, or user history. Moreover, new items cannot be recommended by collaborative filtering techniques neither if these items have not appeared in any transaction histories. This problem is also known as the cold-star problem which can be solved by Content-based filtering techniques.

Hybrid Recommender Systems are the mix of the above recommendation techniques to gain better performance. This recommendation engine inherits advantages of each single technique together with reducing their limitations. Netflix is a good example of Hybrid Recommendation Systems. Its recommendation system is based on the watching habits of similar users and similar films which users have already rated. In this study, we focus on clustering techniques for the content-based filtering, and sequence mining techniques for the collaborative filtering.

2.1. Recommendation based on clustering

K-Means clustering algorithm is one of the most used algorithms. By clustering and giving an active item, we can quickly find the most relevant items in the same cluster. K-Means, however, has limitations like computational speed, sensitive to outliers and inefficiency for large datasets. Mini-Batch K-Means [3], a modified version of K-Means, is faster than K-Means and commonly used for large datasets. It can solve the low computational speed of K-Means by using mini-batches to reduce the computation time. The main idea of Mini-Batch K-Means is to randomly extract subsets from the whole dataset. For each iteration, a new random sample from dataset is used to update the cluster until convergence.

In the both clustering algorithms, we need to determine the "right" number of clusters in a dataset. Figuring out what the right number of clusters often depends on the distribution's shape and scale in the dataset, as well as the clustering resolution required by the user [4]. There are several simple ways to determine the number of clusters:

- A simple method to determine the number of clusters is about $\sqrt{\frac{n}{2}}$, where n is the number of points in a dataset. In expectation, each cluster has $\sqrt{2n}$ points.
- Elbow method: Technically, given a number k > 0 (k is the number of clusters), we calculate the sum of within-cluster variance of each cluster, var(k). And then the curve of var is plotted with respect to k. The first (or most significant) turning point of the curve suggests the "right" number.

2.2. Recommendation based on sequence mining

User behaviours or the sequences of user interactions play an important role in making recommendations. Many techniques are being applied to analyse sequences of user behaviours for discovering frequent patterns, such as Frequent Pattern Mining with Apriori algorithm, Association Rules, Frequent Pattern tree, Markov Chain Modeling, and Hidden Markov Model [4]. More advanced, based on the idea of Word Embedding Model, particularly, word2Vec model in Natural Language Processing, Neural Item Embedding called item2Vec has been used to analyse item-item relations to produce item-to-item similarities [5].

2.2.1. Word Embedding in Natural Language Processing

2.2.1.1. Introduction

The main challenge in Natural Language Processing (NLP) is how to make computers understand natural human language since they can only understand number and binary. According to Wikipedia, word embedding is "the collective name for a set of language modelling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers". Word embedding is the type of mapping that allows words with similar meaning having similar representations. Word2Vec is one of the popular word-embedding models used to represent a word as a vector.

2.2.1.2. Word2Vec

Word2Vec was developed by a researcher team led by Tomas Mikilov in 2013 at Google [6]. The idea of word2Vec is to transfer one word from a one-hot vector to a new vector with a defined vector size. Word2Vec is a feed-forward and fully connected neural network. In the word2Vec model, the input is a text corpus, and its output is the set of feature vectors for each word in the corpus. It first constructs a vocabulary from the training text data and then learns the vector representation of words. Most applications of word2Vec use the cosine similarity to quantify the closeness of words. Two important models inside word2Vec are Continuous Bag of Words (CBOW) and Skip-gram. In Skip-gram, the model tries to predict the neighbours or context of a word. The input is a target word and the output are words surrounding the target.

CBOW model tries to predict the word given its neighbours or context. It predicts the word "by summing all the context word vectors together to represent the word" [7]. So, it is similar to skip-gram except swapping the input and output.

Therefore, the big difference between two models is how feature vectors are generated and the Skip-gram model is sensitive to positions of context words. According to Mikolov, the Skip-gram works well with small training data and presents good representations for rare words, while CBOW performs better in large training data [7].

2.2.2. From word2Vec to item2Vec

The result of word2Vec models is the set of feature vectors. These vectors can be used to determine similarity between words. Based on this idea, item2Vec was introduced by Oren Barkan and Noam Koenigstein [5] in order to determine the similarity between items for recommendation. It is argued that there is an equivalence between words in NLP and items in user behaviour sequence mining, so word2Vec becomes item2Vec [8].

Let $U = \{u_1, u_2, u_3, \ldots, u_n\}$ be a set of users and $I = \{i_1, i_2, i_3, \ldots, i_m\}$ be a set of items, where n, m denotes the number of users and items, respectively. For each user u, transaction history of user u is given by:

$T^u := T^u_1, T^u_2, \ldots, T^u_t$, where $T^u_t \subseteq I$

The transaction history of all users is denoted as $T = \{T^1, T^2, \ldots, T^u\}$, as the input of the item2Vec model.

3. Methodology

3.1. Overview framework

This subsection describes the framework of the proposed system as Figure 1. It is a hybrid restaurant recommendation given restaurant features and user behaviours of restaurants visited.

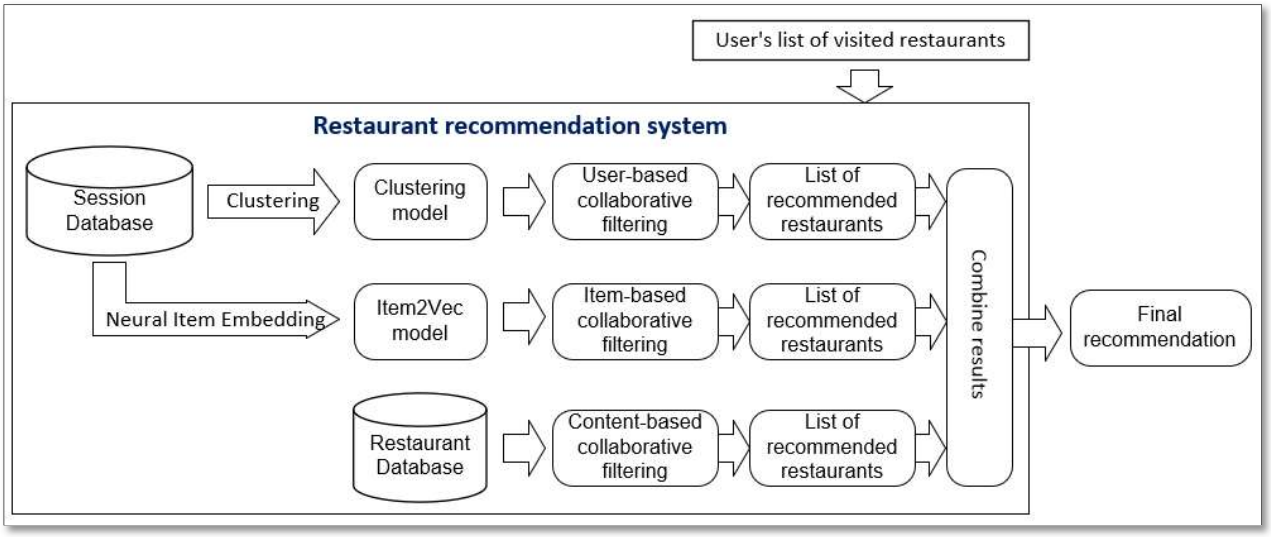

Figure 1. The proposed framework of the restaurant recommendation system

Each record in the Restaurant Database stores the features of one restaurant. The Session Database presents restaurant interactions as user sessions. Three data mining techniques: (1) content-based, (2) user-based collaborative

and (3) item-based collaborative filtering, are conducted for making hybrid restaurant recommendations.

In the content-based filtering, the cosine similarity is used to calculate the similarity between features of different restaurants, and n most similar restaurants with restaurants that a user has already visited are selected. The Session Database is used to build two models: clustering model and item2Vec model. The user-based collaborative filtering finds users having the same interests. In that, the clustering is performed to organize users into groups, k most similar users are chosen for recommendation given a target user. The user similarity is computed based on the list restaurants that users have already visited. The restaurant similarity is calculated using the Item2Vec model which is constructed from the list of visited restaurant sequences. In the item-based collaborative filtering, from the list of visited restaurants, the system finds restaurants usually visited together. The three above results are combined to obtain a final recommendation for the active user.

3.2. DATA PREPROCESSING

As a case study, the Entrée Chicago Restaurant dataset [9] is used in the proposed system. There are two main files in the dataset, one is restaurants file and the other is sessions file. The raw data needs to be pre-processed, i.e. reformatted and clean, for later mining. Table 1 describes partial data of restaurant features after formatting, including restaurant ID, restaurant name, feature codes. The meanings of feature codes are explained in the dataset description. There are totally 256 features, e.g. Excellent Food, Excellent Service, etc. Table 2 describes the partial session data after formatting, including date, IP, entry point, restaurant rating. Actually, this data describes how users interact Chicago restaurant recommendation systems, and rate restaurants. In this dataset, rating a restaurant means having some behaviour, e.g. moving from one restaurant to another, searching for a restaurant cheaper or nicer, etc. Therefore, behaviours moving from one restaurant to another are extracted and considered as users' behaviours of restaurant visits.

Table 1. Restaurant data after formatting

Restaurant ID	Restaurant Name	Features
0	Moti Mahal	214, 035, 149, 021, 117, 075, 204, 051, 163
1	Village	026, 249, 174, 004, 132, 249, 198, 191, 192, 125, 075, 205, 054, 165

After formatting, the data is clean, that is, irrelevant, insufficient or unnecessary data instances are removed.

Table 2. Session data after formatting

Date	IP	Entry Point	Restaurant Rating
29/Mar/1999 :06:32:41	152.163. 207.79	0	330L,540L,99L,490L,500
29/Mar/1999 :09:40:38	204.221. 190.230	0	369L,316

3.3. Content-based filtering engine

The main idea of the content-based filtering is based on the similarity between restaurant features. The process of its engine is shown in Figure 2.

When a new sequence comes, a list of recently visited restaurants of a user, the system calculates the similarity between the recently visited restaurants with all restaurants to choose k most similar restaurants followed these steps:

- Convert restaurant features into vectors: The length of vector is 256 represented for 256 features. If a restaurant has the i^{th} feature, 1 is assigned to the i^{th} position of the vector, otherwise 0 assigned.

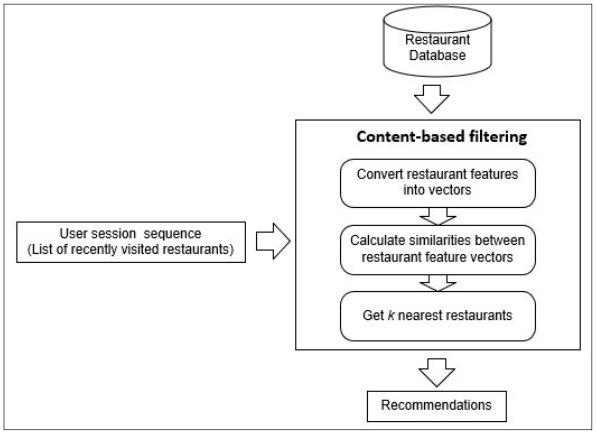

Figure 2. Content-based filtering process

- Calculate similarities between restaurant feature vectors: For each $restaurant_i$ stored in the database, calculate the total similarity between $restaurant_i$ with each restaurant in the recently visited restaurant list. The Cosine similarity is used to calculate distance between two feature vectors. The greater the similarity between two vectors is, the more similar two restaurants are.
- Get k nearest neighbours and recommend: sort the list of the calculated similarity scores in descending order and choose k first restaurants for recommendation.

3.4. User-based collaborative filtering engine

In the user-based collaborative filtering, recommendations are generated based on the restaurant visit history of nearest users to the target user. So, the main step is to find *k* most similar users by calculating the user-to-user similarity. When the number of users is small, it is easy to compute user similarities. But in real applications, when a large number of users interact with the system, this is, then, a bad solution because it takes too much time for similarity calculations. To solve this, a clustering technique is firstly used to organize users/customers into groups and similarity calculations are made within a group. Figure 3 describes steps of the user-based collaborative filtering process.

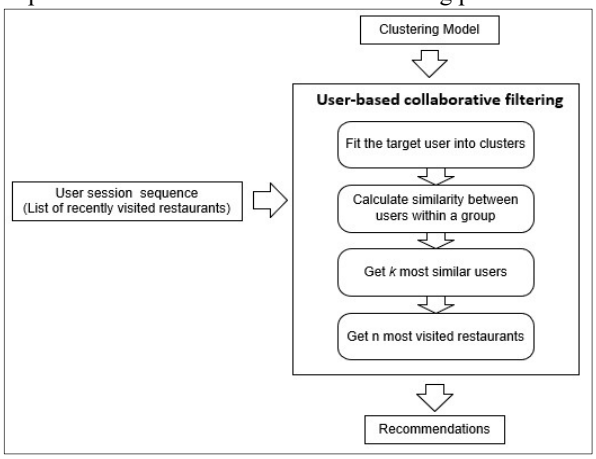

Figure 3. User-based collaborative filtering process

3.4.1. Customer clustering

A customer clustering is the process that clusters users into groups based on common characteristics. The process of the customer clustering is showed in Figure 4.

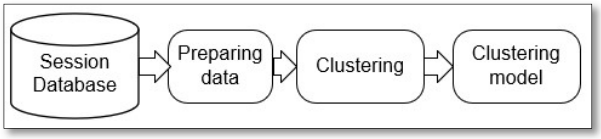

Figure 4. Customer clustering Process

Preparing data

Each transaction in the Session database represents restaurants visited by one user, which is explored from the user session data. Before clustering, each transaction needs to be transformed into one vector. The length of vectors is 676 represented for 676 restaurants in Chicago. Each element value in a vector is the number of times that a user visited a respective restaurant.

For example, we have a transaction [452, 186, 186, 186, 186, 513, 249, 1, 362, 155, 334, 134], in which each number stands for a restaurant ID. This transaction is converted to a vector with 676 elements. Restaurant ID starts from 0 and the vector index also starts from 0. In the transaction, the user visited the restaurant with ID 186 in four times then the 187th element of the vector has value 4. This manner is applied for all other elements in the vector.

Clustering

The prepared data are supplied into a clustering algorithm to organize users into groups.

3.4.2 Fitting the target user into clusters

To make suggestions for a target user, we need to know which user group that the target user belongs to. After fitting the user into clusters, a group of users similar to the target user is generated.

3.4.3. Calculating similarity between users within a group

If recommendations are given based on users in one large cluster, there still have some users whose similarities with the target user are low. It leads to degrade the recommendation accuracy. In this case, we should choose some most similar users for recommendations.

Therefore, Euclidean measure is used to calculate the similarity between users, i.e. the distance between vectors represented for users. The lower the value is, the more similar the users are.

3.4.4. Recommending based on the most similar users

Based on the *k* most similar users at the above step, the system can provide a list of all restaurants that these users have already visited along with the number of visits. Then the top-n most visited restaurants are selected.

3.5. Item-based collaborative filtering engine

The main idea of the item-based collaborative filtering process is constructed on the similarity between items generated from sequences of user behaviours. It means that if restaurants X and Y are visited together in most transactions, it can conclude that restaurant X and Y are similar. So, if a user visited a restaurant X, then the system will recommend a restaurant Y to this user, and vice versa.

In this study, item2Vec is used to recognize similar items given a sequence of user behaviours. Based on the way items appear together in user sessions, the similarities between items are found. Hence, the item2Vec model is built before collaborative filtering. The details of building the item2Vec model is presented in Sub-section 3.5.1.

The process of the item-based collaborative filtering engine is shown in Figure 5.

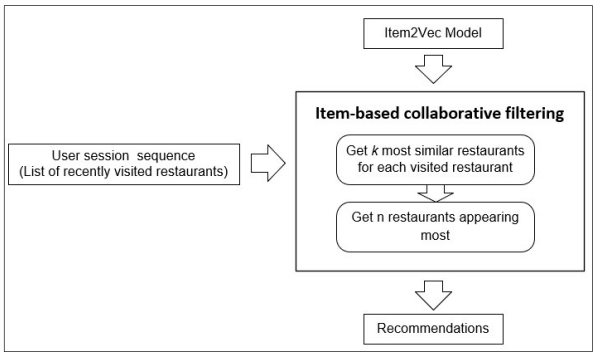

Figure 5. Item-based collaborative filtering process

3.5.1. Neural item embedding in item-based collaborative filtering

The Item2Vec model is used for item-based collaborative filtering that produces embedding for items in a latent space [5]. The process of building an item2Vec model is showed in Figure 6.

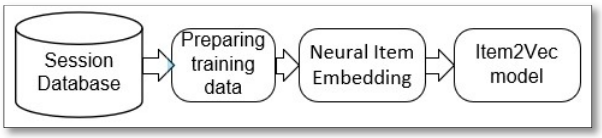

Figure 6. Process of building an Item2Vec model

Preparing training data

In the user session data, each transaction is the list of visited restaurant IDs. Each list is a sequence of items, i.e., restaurants, is formatted as shown in Figure 7.

```
[['564', '627', '380'],
 ['383', '186'],
 ['635', '407', '407'],
 ['369', '111', '133', '632', '336', '672', '201'],
 ['441', '630', '400', '317', '624'],
 ['264', '654']]
```

Figure 7. Examples of item sequences

Building the item2Vec model

The prepared data is input into the neural item embedding process to build the item2Vec model.

3.5.2. Get *k* most similar restaurants for each visited restaurant in a user transaction sequence

To give suggestions for a user, a user transaction sequence of recently visited restaurants is taken into account. For each visited restaurant in the sequence, similar restaurants are fetched and ranked in a descending order of the similarity by applying the item2Vec model. Then top-n items with highest similarity are selected and stored in lists.

3.5.3. Get *n* restaurants appearing most for recommendation

In Sub-section 3.5.2, the system finds the top-n most similar restaurants corresponding to each restaurant in the active user transaction sequence. A list of most similar restaurants is combined and sorted in the descending order of the number of occurrences in the list, and first *n* restaurants after sorting are selected for recommendations.

3.6. Combining the recommended results

The main purpose of this process is to combine the results recommended from the above collaborative filtering engines. This helps to reduce the weakness of each single collaborative filtering technique. First, the restaurants most recommended by the three engines are selected. Next, the remaining restaurants suggested by item-based collaborative filtering are listed. Then, the ones suggested by user-based collaborative filtering and by content-based filtering are appended to the final recommendation list.

For example, given a list of restaurants visited by a user, the content-based collaborative filtering (CF) engine recommends restaurants: A, D, C, E; the user-based CF engine recommends restaurants: B, A, D, Q; and the item-based CF engine recommends restaurants: B, T, H, D. It is noticed that the restaurants in the lists generated by the three single engines are sorted in descending order of score. The three lists are combined as follows:

- Restaurant D is recommended by the three engines, so we select D first.
- Restaurants A, B are recommended by the two single engines, so we select A, B next. The priority is not defined yet.
- Restaurant C, E, Q, T, H are recommended by every single technique. So, restaurants recommended by the item-based one are prioritised, namely T, H, are appended into the final recommendation list. After that, the restaurant recommended by the user-based one, i.e., Q, is selected. Finally, restaurants C and E recommended by the content-based one are selected.

Thus, the final recommendation list is D, A, B, T, H, Q, C, E.

4. Experimental results

The main purpose of building a recommender system is to increase the revenue. It leads to the important of accuracy in making recommendations.

As mentioned, the Entrée Chicago Recommendation dataset is used in experiments. This data contains a record of user interactions with the Entrée Chicago restaurant recommendation system from September 1996 to April 1999. The data has been pre-processed as described in Section 3.

4.1. Evaluation measures

According to Zhou [10], the accuracy of item recommendations can be measured by precision and satisfaction. In all experiments, the two measures are used to evaluate the performance of the proposed recommender system.

Let S = $a_{1,}a_{2\ldots.}a_k a_{k+1}\ldots a_n$ be the user transaction sequence. Each sequence is the list of visited restaurants of one user. For each prefix sequence $S_{prefix} = a_1 a_2 \ldots . a_k$ ($k \leqslant (sequence'slength) - 1$), a recommendation rule is generated $RR = e_1, e_2, \ldots, e_M$ using the recommendation engine, and a correct recommendation and/or satisfied recommendation is determined based on following conditions:

- If $a_{k+1} \in RR$, RR is correct
- If $\exists a_i \in RR$, (k+1 ≤ i ≤ n), RR is satisfied.

R= { $RR_1, RR_2, \ldots, RR_N$} be a set of recommendation rules and |R| = N is total number of recommendations.

- Precision

Let $R_c$be the subset of R that consists of all correct recommendation rules.

$$\text{Precision} = \frac{|R_C|}{N}$$

- Satisfaction

Let $R_s$be the subset of R that consists of all satisfied recommendation rules.

$$\text{Satisfaction} = \frac{|R_S|}{N}$$

```
Input: The testing dataset
Output: Precision and Satisfaction
Process:
Let Rc = 0, Rs = 0 and R = 0;
for each sequence in the testing data
   Set Si = a0a1… an;
   for k from 1 to n
      recently_visited_restaurants = a0a1… ak;
      next_visited = ak+1ak+2… an;
      RR = a recommendation rule generated based on
        recently_visited_restaurants;
   //Check Precision and Satisfaction
      if ak+1 ∈ RR then Rc increased by 1;
      if ∃ai (k+1 ≤ i ≤ n) ∈ RR then Rs increased by 1;
      R increased by 1;
   end for
end for
Precision = Rc / R;
Satisfaction = Rs / R;
return Precision and Satisfaction;
```

Figure 8. Process of evaluating recommendation performance

The process of evaluating the recommendation performance is depicted in Figure 8.

4.2. Experiments and results

There are four experiment cases carried out for validating the built recommendation engines.

- Case 1 is for the content-based filtering engine. Recommendations are based on similarity between features of different restaurants.
- Case 2 is for the user-based collaborative filtering engine. Recommendations are based on users with same interests.
- Case 3 is for the item-based collaborative filtering engine. Recommendations are based on restaurants that are usually visited together.
- Case 4 is for the hybrid recommendation engine. It is the combination of the three techniques.

The experiments are conducted on an Intel Core i3 processor with a CPU clock rate of 1.8 GHz, 4GB of main memory, running on an Ubuntu 16.04 LTS. The algorithms are implemented in Python. The results of each experiment case are evaluated using 10-fold cross validation.

4.2.1. Experiments on Content-based filtering technique

This experiment follows the procedure mentioned in Figure 8, in which the recommendation engine is the content-based filtering engine.

The important parameter in the content-based filtering technique is the number of the most similar restaurants recommended to an active user (denoted by *n*). The main purpose of this experiment is to find the most suitable parameter *n*. As a result, precisions and satisfactions are measured with different parameters *n*, as shown in Figure 9 and 10, respectively.

As we can see from the graph in Figure 9 and 10, the higher the value *n* is, the higher the precision and satisfaction are. It can be easy to explain, when more and more similar restaurants are selected, the higher probability that has a restaurant in the recommendation list matching with the user's choice is. However, a long suggestion list is not often used in recommender systems. In this study, the default value *n* in the content-based filtering is 20.

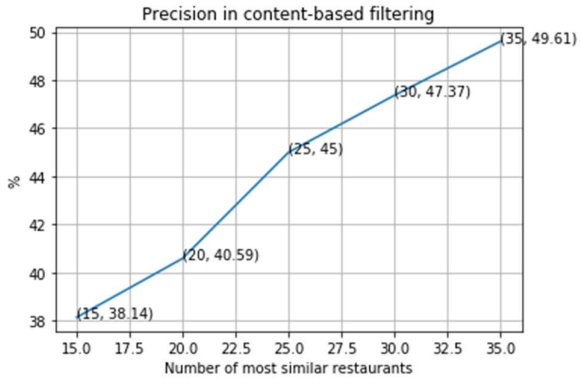

Figure 9. Precision in the content-based filtering model with different parameters *n*

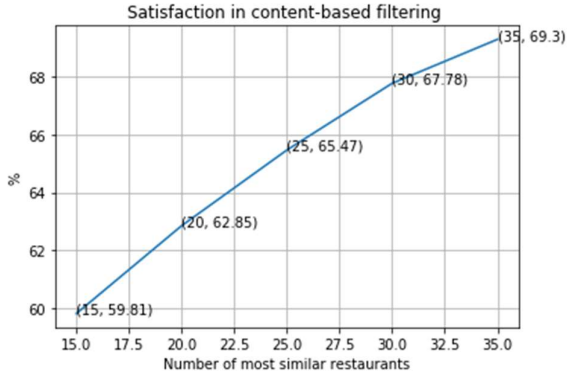

Figure 10. Satisfaction in the content-based filtering model with different parameters *n*

For *n*=20, top-20 restaurants are recommended for each request of an active user, evaluation results are as follows:

- Total number of recommendations: 183,253
- Total number of correct recommendations: 74,385 → precision: 40.59%
- Total number of satisfied recommendations: 115,176 → satisfaction: 62.85%

After obtaining the precision and satisfaction for all recommendations, the precision and satisfaction for each user are considered and split into six levels:

- 0%
- From greater than 0% to less than 25%
- From 25% to less than 50%
- From 50% to less than 75%
- From 75% to less than 100%
- 100%

Session sequences having the length equal 1 are ignored, so the total number of users is 48,417. With the parameter n = 20, the number of users having precision and satisfaction in six levels is presented column (1) and (2), respectively, in Table 3.

Table 3. Precision and satisfaction of users in the content-based filtering engine

Accuracy level	(1)	(2)
0%	19,227	18,118
(0%, 25%)	2,335	895
[25%, 50%)	7,880	3,691
[50%, 75%)	10,301	8,433
[75%, 100%)	2,087	4,996
100%	6,587	12,284

From the table above, the recommender system gives suggestions with 100% precision to 13.60% of users and 100% satisfaction to 25.37% of users. And the percentages of users who are given recommendations of the 0% accuracy level are quite high, that is, 39.71% users get low precision and 37.42% users get low satisfaction. It can give recommendations with precision greater than 50% to 39.19% of users, and with the satisfaction greater than 50% to 53.11% of users.

4.2.2 User-based collaborative filtering

There are two main parts in the user-based collaborative filtering engine: clustering users and making recommendations.

Clustering method:

First, we make the comparison between K-Means and Mini-Batch K-Means. Mini-Batch K-Means algorithm runs faster and especially good for a large dataset. For example:

- With the dataset containing 450 users, the clustering algorithms divide users into 15 groups.
  - K-Means: total CPU times is 2.03(s).
  - Mini-Batch K-Means: total CPU times 1.85 (s).
- With the dataset containing 40536 users and number of clusters is 142.
  - K-Means: total CPU times is 7min 58s.
  - Mini-Batch K-Means: total CPU times 14.5 (s).

As can be seen from two examples, Mini-Batch K-Means runs faster than K-Means. When the dataset is smaller (450 users), the time difference is not too much: 2.03(s) of K-Means compared with 1.85(s) of Mini-Batch K-Means. However, when increasing the number of users in that dataset, the time difference is bigger: 7 min 58s of K-Means compared with 14.5(s) of Mini-Batch K-Means. So that is why Mini-Batch K-Means clustering algorithm is better for massive datasets than K-Means clustering algorithm.

In this study, the Mini-Batch K-Means clustering algorithm is used to cluster users into groups. There are two main parameters in this algorithm:

- batch-size: it controls the number of randomly selected observations in each batch. In this experiment, the number of batch_size chosen is 100.

- Number of clusters: defines the number of groups.

Choosing the number of clusters suggested in [4] is $\sqrt{\frac{n}{2}}$ (n is number of users). In this experiment, the number of users in the training data is 45,605, so the number of clusters is 151. We tried some values around that value. The number of most similar users is fixed at 20 for recommendations. The precision and satisfaction corresponding to different number of clusters are shown in Figure 11 and Figure 12.

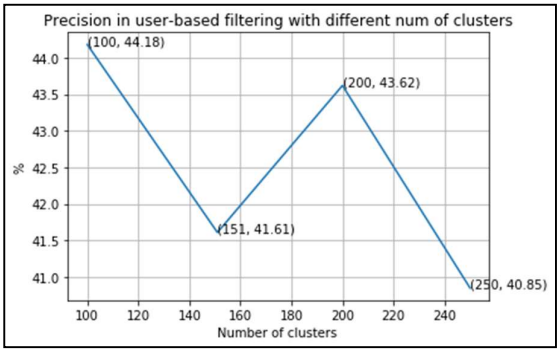

Figure 11. Precision in the user-based collaborative filtering model with different number of clusters

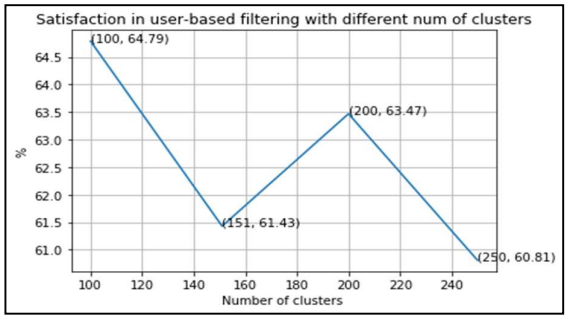

Figure 12. Satisfaction in the user-based collaborative filtering model with different number of clusters

As we can see in Figure 11 and Figure 12, when users are organized into 100 groups, we can achieve the best accuracy. However, the time of giving one suggestion is larger, around 3.5 seconds in comparison with 1.8 seconds (when number of clusters is 200). Because the difference of accuracy in the two cases is small, the number of clusters chosen is 200. This is the trade-off between the accuracy and the running time.

The precision and satisfaction of the user-based collaborative filtering technique corresponding to parameter *n* (number of most similar users) are shown in Figure 13 and Figure 14.

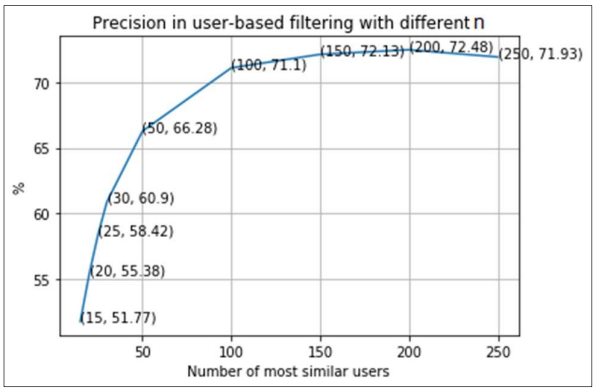

Figure 13. Precision in the user-based collaborative filtering model with different parameter *n*

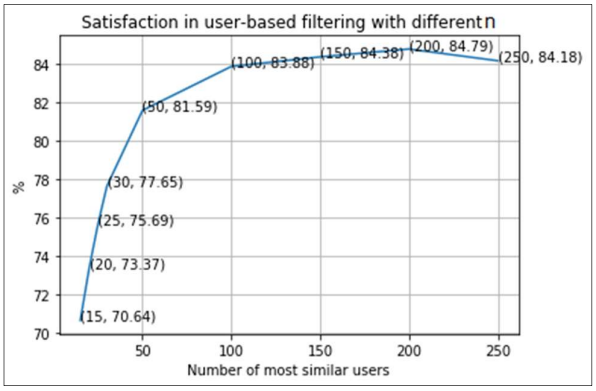

Figure 14. Satisfaction in the user-based collaborative filtering model with different parameter *n*

As can be seen from Figure 13 and Figure 14, $n = 200$ gives the highest precision and satisfaction.

With the number of suggestions (*n*) given in each recommendation rule is 20:

- Total number of recommendations: 183,253
- Total correct recommendations: 132,824 → precision: 72.48%
- Total satisfied recommendations: 155,374 → satisfaction: 84.79%

Similar as Section 4.2.1, we consider the accuracy for each user.

From the Table 4, the recommender system gives recommendations with 100% precision to 42.8 % of users and 100% satisfaction to 60.77% of users. And the percentage of users who are given recommendations with accuracy 0% is lower (17.61% users get low precision and 17.07% users get low satisfaction). The engine can give recommendations with precision greater than 50% to 75.89% of users, and with satisfaction greater than 50% to 80.25% of users.

Table 4. Precision and satisfaction of users in the user-based collaborative filtering engine

Accuracy level	(1)	(2)
0%	8,526	8,266
(0%, 25%)	380	130
[25%, 50%)	2,775	1,168
[50%, 75%)	9,790	5,810
[75%, 100%)	6,233	3,619
100%	20,723	29,424

4.2.3 Item-based collaborative filtering

In the item-based collaborative filtering, there are two parts, building the item2Vec model and recommending most similar restaurants based on the results of item2Vec model. The combination of following parameters is considered: the training algorithm, vector size (vec-size), min_count, number of epochs and number of most similar restaurants (*n*). We proceed to change one parameter while other parameters are fixed, so that best parameters can be selected. The number of final suggested restaurants given to an active user is 20.

- vec-size: the size of feature vectors defined in the item2Vec model. Each vector represents an item in the latent space which is modelled from the user session sequences and each item is a visited restaurant.
- min_count: all items have frequency less than this value are ignored. In other words, it is used to make a set of items, equivalent to the vocabulary in Word2Vec model. To put all restaurants into this set, min_count should be 1. The number of epochs is set to 100.

First, training algorithm is Skip-gram. Vec-size takes one of values in the list [100, 200, 300, 400] and *n* takes values in the list [10, 15, 20, 25, 30]. Precision and satisfaction of each case are showed in Figure 15.

When Skip-gram is used as the training algorithm, min_count and number of epochs are set to 1 and 100, respectively, the set of parameters that gives highest precisions and satisfactions in recommendations is {vec-size: 300, *n*: 20}. The highest precision is 63.24%. The highest satisfaction is 80.48%.

Next, when training CBOW algorithm, the precision and satisfaction of each case changing parameters is showed in Figure 16.

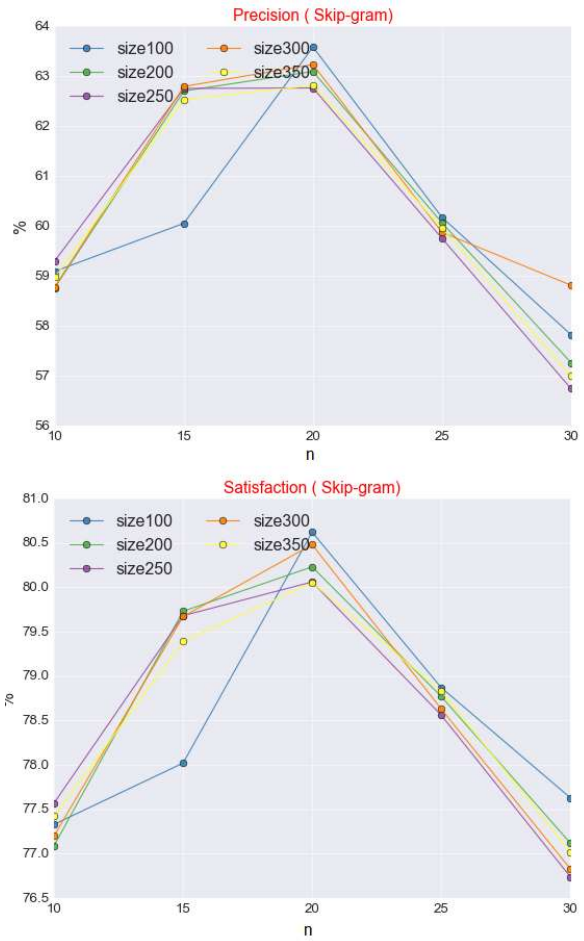

Figure15. Precision and Satisfaction with different vector sizes and *n* using Skip-gram

When CBOW is used as the training algorithm, min_count and number of epochs are set to 1 and 100, respectively. The set of parameters that give highest precisions and satisfactions in recommendations is {vec-size: 300, n: 20}. The highest precision is 70.32%. The highest satisfaction is 84.89%. Therefore, CBOW is chosen as the training algorithm. Next are the experiments for choosing number of epochs. At this time, the training algorithm is CBOW, min_count is 1, vec-size is 300 and *n* is 20. The precision and satisfaction for each case in experiments is showed in Figure 17.

When CBOW is used as the training algorithm, the set of parameters that give highest precision and satisfaction is {vec-size: 300, min_count: 1, number of epochs: 1000, *n*: 20}. The highest precision is 71.72%. The highest satisfaction is 86.58 %. In conclusion, the set of parameters for the item-based collaborative filtering engine are:

- Training algorithm: CBOW
- Dimensionality of feature vector: 300
- Min_count: 1
- Training epoch: 1000
- *n*: 20.

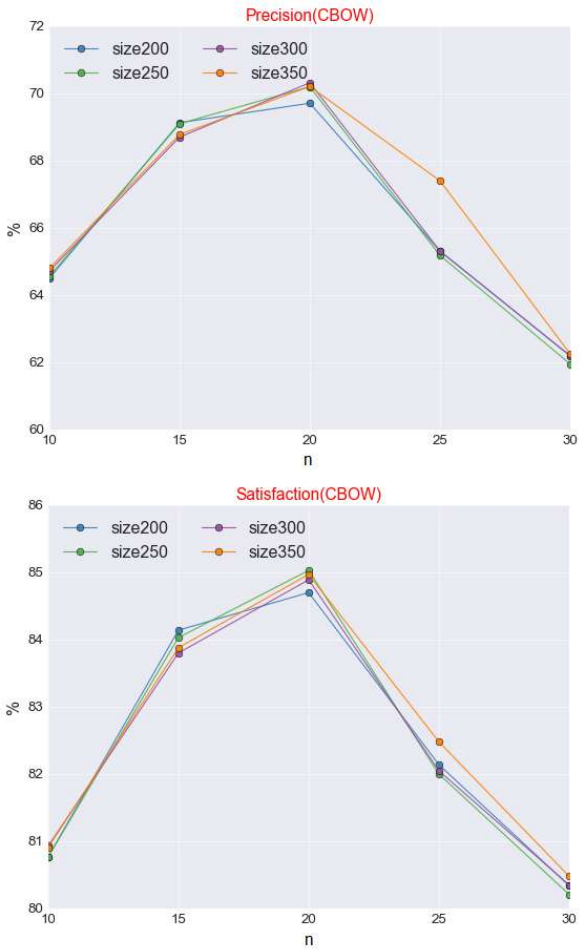

Figure 16. Precision and Satisfaction with different vector sizes and *n* using CBOW

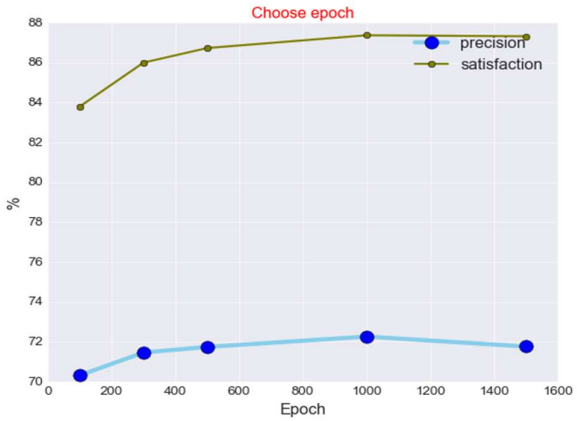

Figure 17. Choosing number of epochs in building the item2Vec model

With the number of suggestions (*n*) given in each recommendation rule is 20:

- Total number of recommendations: 183,253
- Total correct recommendations: 131,424 → precision: 71.72%
- Total satisfied recommendations: 158,660 → satisfaction: 86.58%

Similar as Section 4.2.1, we consider the accuracy for each user.

Table 5. Precision and satisfaction of users in the item-based collaborative filtering engine

Accuracy level	(1)	(2)
0%	6,252	5,944
(0%, 25%)	482	127
[25%, 50%)	3,005	996
[50%, 75%)	9,705	4,799
[75%, 100%)	5,563	3,458
100%	23,410	33,093

From the table above, the recommender system gives recommendations with 100% precision to 48.35% of users and 100% satisfaction to 68.35% of users. And the percentage of users who are given recommendations with accuracy 0% is quite small (12.91% users get low precision and 12.28% user get low satisfaction). It can give recommendations with the precision greater than 50% to 79.89% of users and with the satisfaction greater than 50% to 85.40% of users.

4.2.4 Hybrid recommendation engine

Hybrid recommendation engine is the combination of three recommendation engines which are based on content, user and item. Top-20 suggested restaurants in each single recommendation engine are merged and sorted in the descending order of occurrences, then the first 20 restaurants in the sorted list are suggested to the target user.

With the number of suggestions (*n*) given in each recommendation rule is 20:

- Total number of recommendations: 183,253
- Total correct recommendations: 143,151 → precision: 78.12%
- Total satisfied recommendations: 163,108 → satisfaction: 89.0%

Similar as Section 4.2.1, we consider the accuracy for each user, as shown in Table 6. The recommender system gives suggestions with 100% precision to 53.42% of users and 100% satisfaction to 70.81% of users. And the percentage of users who are given recommendations with accuracy 0% is lowest (12.09 % users getting low precision and 11.76% users getting low satisfaction. It can give recommendations with precision greater than 50% to 83.25%of users and with satisfaction greater than 50% to 86.44% of users.

Table 6. Precision and satisfaction of users in the hybrid recommendation engine

Accuracy level	(1)	(2)
0%	5,856	5,692
(0%, 25%)	224	79
[25%, 50%)	2,029	796
[50%, 75%)	8,524	4,548
[75%, 100%)	5,922	3,018
100%	25,862	34,284

4.3. EVALUATION

In this section, we make the comparison between the built recommendation engines. There are four recommendation techniques to compare: content-based filtering technique (C1), user-based filtering technique (C2), item-based filtering technique (C3) and hybrid recommendation technique (C4), as presented in Table 7. The comparison is based on these criteria:

- Precision: the number of correct recommendations over the number of generated recommendations.
- Satisfaction: the number of satisfied recommendations over the number of generated suggestions.
- % users that get precision at least 50%: the percentage of users that are given recommendations with precision greater than or equal 50%. Remember that, we ignore users who just visited only one restaurant.
- % users that get satisfaction at least 50%: the percentage of users that are given recommendations with satisfaction greater than or equal 50%.
- % users that get precision 100%: the percentage of users that are given recommendations with precision equal 100%.
- % user that get satisfaction 100%: the percentage of users that are given recommendations with satisfaction equal 100%.
- Run-time: the average time that a recommendation engine gives one recommendation. Time is measured in seconds.

Table 7. Comparison between the recommendation engines

	C1	C2	C3	C4
Precision	40.59%	72.48%	71.72%	**78.12%**
Satisfaction	62.85%	84.79%	86.58%	**89.0%**
% users that $precision \geq 50\%$	39.19%	75.87%	79.89%	**83.25%**
% users that $satisfaction \geq 50\%$	53.11%	80.25%	85.40%	**86.44%**
% users that all recommendations are correct	13.60%	42.80%	48.35%	**53.42%**
% users that all recommendation are satisfied	25.37%	60.77%	68.35%	**70.81%**
Average run-time	0.471	1.603	**0.005**	2.052

- Case 1 (C1) – Content-based filtering engine: Only using k nearest neighbours for prediction. The dataset contains 676 restaurants. It means that the algorithm needs to calculate similarities of one restaurant with the remaining 675 restaurants. This case has the lowest precision and satisfaction.
- Case 2 (C2) – User-based collaborative filtering engine: Users are organized into groups and recommendations are made based on n most similar users in the same group. The performance is better than Case 1.
- Case 3 (C3) – Item-based collaborative filtering engine: The item2Vec model is used to analyse the similarities between restaurants. The strongest point of this engine in comparison with other techniques is that the time of making a recommendation is fastest.
- Case 4 (C4) – Hybrid recommendation engine: The results of three recommendation engines are mixed together to make final recommendations. Except that the time making one recommendation is lowest, this solution gives the best accuracy. The strongest point of this technique is that it can solve the cold-start problem, meaning that when a new restaurant is added, it can still be recommended to users.

In conclusion, the content-based filtering engine gives us the worst results (i.e. lowest precision and satisfaction). Next is the user-based and item-based collaborative filtering engines. The hybrid recommendation engine gives the best accuracy. In four cases, the item-based collaborative filtering engine can make a recommendation fastest, i.e. 94 times faster than the content-based filtering engine; 320 times faster than the user-based collaborative filtering engine; more than 400 times faster than the hybrid recommendation engine. For the used dataset, if we focus on the running time, the item-based collaborative filtering is the best solution. However, if the accuracy is the first priority, the hybrid recommendation approach is the best solution. It needs to have the trade-off between running time and accuracy.

5. Conclusions

A Recommender System plays an important role in many on-line services and websites. Recommender Systems can be categorized into two main types: content-based recommender systems and collaborative filtering recommender systems. While content-based recommender

systems are based on the item characteristics, collaborative filtering recommender systems are based on the user behaviour. Each technique has its own advantages and limitations, so hybrid recommender systems are the combination of these types in order to reduce the limitations of each single recommendation technique.

In this study, some issues were addressed. In the content-based recommendation engine, the similarity between restaurant features is calculated by the cosine similarity and top-n "nearest" restaurants are recommended to user. In the user-based collaborative filtering engine, the Mini-Batch K-Means organises users into groups and recommendations for the target user are based on the top-n most similar users within a group. In the item-based collaborative filtering engine, item2Vec - a neural embedding algorithm - is used to analyse the "relationship" between restaurants given the visited restaurant sequences. Finally, the experimental results have figured out that the hybrid recommender system is the best architecture for the restaurant recommender system with the highest accuracy.

Acknowledgements.

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number: 06/2018/TN.

References

[1] Charu C. Aggarwal (2016) *Recommender Systems*, First Edition, Springer.

[2] Mikolov, T., et al. (2013) Distributed Representations of Words and Phrases and their Compositionality, In *Proceedings of NIPS*, pp. 3111-3119, arXiv:1310.4546.

[3] D. Sculley (2010) Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, ACM, pp. 1177–1178.

[4] Jiawei Han, Micheline Kamber, and Jian Pei (2011) *Data Mining: Concepts and Techniques*, Third Edition, Morgan Kaufmann.

[5] Oren Barkan and Noam Koenigstein (2016) Item2vec: Neural item embedding for collaborative filtering, *IEEE Workshop on MLSP*.

[6] Mikolov, T., et al. (2013) Distributed Representations of Words and Phrases and their Compositionality, In *Proceedings of NIPS*, pp. 3111-3119, arXiv:1310.4546.

[7] Lifeng Jin and William Schuler (2015) A Comparison of Word Similarity Performance using Explanatory and Non-explanatory Texts, In *Proceedings of the North American Association for Computational Linguistics* (NAACL'15) Boulder, Colorado.

[8] Yilma Bereket Abera (2017) *Recommendation based on Sequence: Item2Vec*, DOI: 10.13140/RG.2.2.25358.97601.

[9] Dua, D. and Karra Taniskidou, E. (2017) *UCI Machine Learning Repository* [http://archive.ics.uci.edu/ml/datasets/Entree+Chicago+Recommendation+Data], Irvine, CA: University of California, School of Information and Computer Science.

[10] Zhou, B. (2004) *Intelligent Web Usage Mining*, Nanyang Technological University.