**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**ADVANCED PROGRAM IN INFORMATION SYSTEM**

**NGUYỄN ĐỨC HUY – 14521166**

**HUỲNH TUẤN KIỆT – 14521168**

**SEMANTIC PATH BASED PERSONALIZED RECOMMENDATION SYSTEM (SEMREC) IN WEIGHTED HETEROGENEOUS INFORMATION NETWORK (WHIN)**

**BACHELOR OF ENGINEERING IN INFORMATION SYSTEMS**

**THESIS ADVISOR**

**ASSOC. PROF. DR. ĐỖ PHÚC**

**HO CHI MINH, 2018**

**ACKNOWLEDGEMENT**

We would like to express our gratitude to our advisor Associate Professor Dr. Đỗ Phúc who suggests, guides and inspires us in working with this thesis.

We want to express our thanks to the staffs in the faculty of Information Systems of the University of Information Technology and our instructor who emit the passion to us.

Finally, we would like to thank our families for providing us the best condition to study in Advance Education Program and finish this graduation thesis.

6 January, 2019

Nguyễn Đức Huy – Huỳnh Tuấn Kiệt

**ASSESSMENT (ADVISOR)**

**ASSESSMENT (COMMITTEE MEMBERS)**

**TABLE OF CONTENT**

****

[1.1. Overview 1](#_Toc534589878)

[1.2. Problem Statement 2](#_Toc534589879)

[1.3. Motivation 4](#_Toc534589880)

[1.4. Contributions 7](#_Toc534589881)

[1.5. Organization of Thesis 8](#_Toc534589882)

[Chapter 2: Fundamental theories 9](#_Toc534589883)

[2.1. Recommendation Filtering techniques 9](#_Toc534589884)

[2.1.1. Definition 9](#_Toc534589885)

[2.1.2. Two main kinds of Recommendation Systems 10](#_Toc534589886)

[2.2. Phases of Collaborative Filtering process 13](#_Toc534589887)

[2.2.1. Input 13](#_Toc534589888)

[2.2.2. Similarity Measures 25](#_Toc534589889)

[2.2.2.1. Cosine Vector Similarity 25](#_Toc534589890)

[2.2.2.3. Path-based Similarity (PathSim) 26](#_Toc534589897)

[2.3. Prediction generation 27](#_Toc534589898)

[2.4. Phases of Collaborative Filtering process 28](#_Toc534589899)

[2.5. Evaluation Metrics 33](#_Toc534589905)

[2.6. Related Works 36](#_Toc534589906)

[Chapter 3: Realization the Graph-based recommendation system 37](#_Toc534589907)

[3.1. Fundamentals of inputting dataset in graph 37](#_Toc534589909)

[3.2. Calculating the similarity measure by PathSim 45](#_Toc534589910)

[3.3. Prediction the rating scores 48](#_Toc534589911)

[Chapter 4: Experiments 52](#_Toc534589912)

[4.1. Datasets 52](#_Toc534589913)

[4.2. Environments and configures 52](#_Toc534589914)

[4.3. Experiment results 53](#_Toc534589915)

[Chapter 5: Conclusion and Future Development 63](#_Toc534589916)

[5.1. Conclusion 63](#_Toc534589918)

[5.2. Future Works 63](#_Toc534589919)

**LIST OF FIGURES**

****

[Figure 1.1.The recommendation speed of Google.com 4](#_Toc534038910)

[Figure 1.2.A Heterogeneous Information Network in most real systems which include a large number of interacting, multi typed components 6](#_Toc534038910)

[Figure 2.1.Amazon’s product recommendation 10](#_Toc534038910)

[Figure 2.2. Content-Based system example 11](#_Toc534038910)

[Figure 2.3. Graph sample 16](#_Toc534038910)

[Figure 2.4. An example of homogeneous information network and](#_Toc534038910)

[heterogeneous information network 17](#_Toc534038910)

[Figure 2.5. An example of weighted heterogeneous information network… 18](#_Toc534038910)

[Figure 2.6. An example of the relationship among nodes 19](#_Toc534038910)

[Figure 2.7. Conversion from HIN to Graph 22](#_Toc534038910)

[Figure 3.1. We see the phases to build a Graph – based RS 38](#_Toc534038910)

[Figure 3.2. ER diagram for the dataset sample 39](#_Toc534038910)

[Figure 3.3. Graph model from the dataset sample in Figure.3.1 40](#_Toc534038910)

[Figure 3.4. The Inputting Process 41](#_Toc534038910)

[Figure 3.5. Here is an overview of the Neo4j browser interface 43](#_Toc534038910)

[Figure 3.6. We see Neo4j's browser after executing the loading function 44](#_Toc534038910)

[Figure 3.7. We see an example for graph in Neo4j after executing the mapping](#_Toc534038910)

[function 45](#_Toc534038910)

[Figure 3.8. We see an example for User – Movie Graph](#_Toc534038910)

[with the rating scores contained on the links 46](#_Toc534038910)

[Figure 3.9. We see an example for calculating the PathSim similarity](#_Toc534038910)

[measure 48](#_Toc534038910)

[Figure 3.10. We see an example for calculating the PathSim similarity](#_Toc534038910)

[measure 48](#_Toc534038910)

[Figure 4.1. Accuracy in 100k MovieLens dataset 56](#_Toc534038910)

[Figure 4.2. Chart of The Time between 100K and 1M MovieLen dataset in Matrix 58](#_Toc534038910)

[Figure 4.3. Chart of the Time between 100K and 1M movielen dataset in Graph 59](#_Toc534038910)

[Figure 4.4. Amplitude of Time between 100k and 1M MovieLen datasets in Matrix and Graph 61](#_Toc534038910)

[Figure 4.5. Recommendation result of two system 62](#_Toc534038910)

**LIST OF TABLES**

****

[Table 2.1. The Utility Matrix 14](#_Toc534038910)

[Table 2.2. Meta path examples and their physical meanings on bibliographic data 22](#_Toc534038910)

[Table 2.3.a. Original utility matrix Y and mean user ratings 28](#_Toc534038910)

[Table 2.3.b. Normalized utility matrix 28](#_Toc534038910)

[Table 2.3.c. User similarity matrix S 29](#_Toc534038910)

[Table 2.3.d. Predict normalized ratings 29](#_Toc534038910)

[Table 2.3.e. Full Y 30](#_Toc534038910)

[Table 3.1. The results will be recommended for u4 51](#_Toc534038910)

[Table 4.1. The RMSE accuracy is tested on MovieLens 100k dataset 55](#_Toc534038910)

[Table 4.2. The Time between 100K and 1M MovieLen dataset in Matrix 57](#_Toc534038910)

[Table 4.3. Values the Time between 100K and 1M movielen dataset in Graph 59](#_Toc534038910)

**LIST OF FORMULAS**

****

[Formula 2.1. Cosine Similarity 26](#_Toc534038910)

[Formula 2.2. Meta-path similarity measure 27](#_Toc534038910)

[Formula 2.3. Predict Rating Formula.. 27](#_Toc534038910)

[Formula 2.4. Redefine Cosine similarity 31](#_Toc534038910)

[Formula 2.5. RMSE formula 34](#_Toc534038910)

[Formula 2.6. MAE formula 34](#_Toc534038910)

[Formula 2.7. MSE formula 35](#_Toc534038910)

**TABLE OF ABBREVIATIONS**

****

|  |  |
| --- | --- |
| **Abbreviation** | **Explanation** |
| **RS** | Recommendation System |
| **HIN** | Heterogeneous Information Networks |
| **WHIN** | Weighted Heterogeneous Information Networks |
| **CF** | Collaborative Filtering |
| **PathSim** | Path-based Similarity |
| **UBCF** | User-based collaborative filtering |
| **IBCF** | Item-based collaborative filtering |
| **MAE** | Mean Absolute Error |
| **MSE** | Mean Square Error |
| **RMSE** | Root Mean Square Error |

**ABSTRACT**

One of the collaborative recommendation methods, techniques based on nearest-neighbours are still researched the most popular during the later years, because they are simple, efficient and able to generate accurate and personalized recommendations Nowadays, heterogeneous information network (HIN) analysis has attracted a lot of attention, and many data mining tasks have been feated on HIN. Like an important data mining task, recommendation system (RS) involve in lots of object types (e.g., users, items, movies, and interest groups in movie recommendation) and the rich relationships among object types, which naturally create a HIN. The detailed information integration and rich semantic information of HIN will make it generate better recommendations. Apart from, Graph databases are a feasible replacement to Relational Database Systems. In social networking and recommendation engines with large-scale datasets, it has shown some certain effect. In this thesis, we prove the graph-based Collaborative Filtering RS can replace the traditional Collaborative Filtering RS (using matrix) based on time or memory and scalability. In these two methods the biggest difference is in Similarity, Cosine similarity is applied in the traditional Collaborative Filtering RS since its popularity and PathSim similarity is applied in the graph-based Collaborative Filtering RS depended on HIN theories. Based on that result, we make suggestions for future recommendation systems in e-commerce.

**Chapter 1: Introduction**

## **Overview**

In modern life, many e-commerce and retail companies are leveraging the potentiality of data and enhancing sales by implementing Recommendation Systems on their websites. In brief, the final purpose of these systems is to predict users’ interests and recommend items that quite likely are exciting for them. Data which are required for recommendation systems stem from explicit user ratings after watching a movie or reading a book from insinuated search engine queries and purchase histories, or from other knowledge about the users or items themselves. Sites; for instance, YouTube, Spotify or Netflix [1] use those data in other to suggest playlists or to make video recommendations, respectively. Facebook, Twitter, and other social networks use Collaborative Filtering technique to recommend new friends, groups, and other social connections (by checking the network of connections between a user and their friends). Twitter [2] uses lots of signals and in-memory computations for recommending to its users that they should "follow". Currently, Internet applications have turned to recommendation systems to aid users to navigate among the increasing number of available choices. Maybe Amazon is the most common example of such a system and helps customers to decide on which books, magazines, music, DVDs, videos, electronics, computers, software, accessories or shoes show to buy next.

Recommendation systems can be mainly categorized such as collaborative [9], content-based [6], or hybrid [7] [8] [9]. The collaborative recommendation is similar to word-of-mouth communication, in which the opinions of others are used to control the relevance of a recommendation. In the case that, a collaborative recommendation system uses the ratings provided by its users either to recommend an interesting item or to identify similar users. To illustrate, Amazon [4] uses this technique to recommend products highly rated by people who, in the past, rating videos in a similar way to the user. Content-based recommendation focuses on using the content of an item to assert its relevance. To put it differently, a music recommendation application that uses the content-based approach may use the kind of music to recommend other musical choices that share the same genre. The hybrid category is reserved for those systems that use both techniques when considering a recommendation. The most significant of the hybrid category is the music recommendation using the content-based approach and also extended to incorporate the collaborative method. In this situation, the recommendation would contain lots of music of the same genre that was also appreciated by like-minded users.

## **Problem Statement**

Recommendation systems are built as navigational tools and broadly deployed online. Actually, this means that a CF algorithm is executed and then trained with all the available ratings which the system has for the present content. Then, the algorithm can be queried to return recommendations for each user. This process is repeated in a cyclical way. So why? because the high latency usually affects to Collaborative Filtering algorithms - training an algorithm with the ratings of potentially millions of users is a very costly operation, often requiring the exponential growth of space and time, and can thus not be repeated at will. Hence, Recommendation Systems tend to implement iterative, regular updates. Users will not be consistently offered the latest computed recommendations and have to wait for updating that system for their latest ratings to be included in the Collaborative Filtering training phase. Since recommendations often obtain further ratings, CF algorithms are iteratively retrained so that they have learned from all the data (including any that may have been input since they were last trained). So, we want to study and propose a way to reduce space, time or memory and can handle well with big data (the datasets with a huge number of ratings) with the aim is to improve RS in the future. So how are these problems being solved? Well, one part of the solutions which we want to mention is the graph database. The graph database model concentrates on the relationships of the different nodes, or data-points. So, instead of looking at the value of the data-point (which is what the SQL database would do), the graph database is organizing and analyzing the messy data-points according to the relationships. But what is the importance of the node and relationship in a graph database? Why is it so effective in the way you analyze data. The answer is that it can define the interconnected data more precisely. In place of just understanding what is the value of specific data, you understand the value of the relationship between data. As we can see, an organization doesn’t just rely on data when it comes to decision-making. Assume that if you want to increase sales at a clothing shop, you don’t just need data on clothes that are being sold in order to raise sales. You need to understand the necessary information about customers such as gender, age, e.g. and how the relationships between customers and clothes are connected. If you figure out those relationships, you can increase sales easily.

## **Motivation**

**Figure 1.1. The recommendation speed of Google.com**

We can see in the above figure, this is the number of the recommended results is extremely more by Google in short time.

Neighborhood-based Collaborative Filtering methods [17] include in User-based and Item-based Collaborative Filtering which have several advantages related to their simplicity and intuitive approach. Because the approach manner of these methods is simple and intuitive, so they are easy to execute and debug. It is often easy to explain why a specific item is recommended [14], and the interpretability of item-based methods is particularly notable [5]. Besides, the recommendations are relatively stable with the addition of new items and users. Moreover, it gives recommendations that can be complements to the item the user was interacting with. This might be a stronger recommendation than what an item-based recommendation can provide as users might not be looking for direct substitutes to a movie they had just viewed or previously watched.

But User-based and Item-based Collaborative Filtering is a type of Memory-based Collaborative Filtering [13] that uses all user data in the database to create recommendations. Comparing the pairwise of every user in your dataset is not scalable Assume that the dataset contains millions of users, the computation process will be extremely time-consuming [3]. In addition, user-based collaborative filtering based on the choice history of the user to generate recommendations in future. The implications of this are that it assumes that a user’s taste remains more or less constant over time, which might not be true and makes it difficult to pre-compute user similarities offline.

The main disadvantage of these methods is their scalability. This might sometimes be too slow or space-intensive of hardware [14], as m is the number of tens of millions. Nevertheless, the online phase of neighborhood methods is quite efficient. The other main disadvantage of these methods is their limited coverage because of sparsity [16]. For instance, if none of Luke’s nearest neighbors have rated Avatar (film), it is not possible to provide a rating prediction of Avatar for Luke. On the other hand, we care only about the top-k films of Luke in most recommendation settings. If none of Luke’s nearest neighbors have rated Avatar, then it might be evidence that this film is not a good recommendation for Luke. Sparsity also creates challenges for robust similarity computation when the number of mutually rated items between two users is small.

Therefore, we really want to improve time (speed), memory and solve scalability problem of this recommendation system by Heterogeneous information network (HIN) [12] which was represented by graph database (Neo4j). Heterogeneous information networks can be constructed from many interconnected, large-scale datasets, ranging from social, education, scientific, engineering to business applications. In addition, they can be built almost in any fields, such as social networks (e.g., Facebook), e-commerce (e.g., Amazon and Lazada), online movie databases (IMDB), and various database applications.

An information network [10] [11] is defined as a directed graph G = (V, E) with an object type mapping function ϕ: V → A and a link type mapping function ψ: E → R. Each object v ∈ V belongs to one particular object type in the object type set A: ϕ (v) ∈ A, and each link e ∈ E belongs to a particular relation type in the relation type set R: ψ (e) ∈ R. each node represents an entity (e.g., user in a social network) and each link (e.g., friendship) a relationship between entities.

* Nodes (or vertices)/links may have attributes, labels, and weights.
* Links or edges or paths may carry rich semantic information.

**Figure 1.2. A Heterogeneous Information Network View of Recommendation**

In most real systems which consist of a large number of interacting, multi typed components as example of the above figure.

The advantage of this method is that the graph needs to be built only once. And then, it’s very easy to add items to it by just adding the nodes and edges to other items [18]. Also, the search algorithm in the graph-based recommendation system is the online feature and is fast in returning recommendations. Furthermore, the main advantage of graph-based methods is that two users do not need to have rated multiplies same items to be considered neighbors as long as many short paths exist between the two users. Therefore, this definition allows the construction of neighborhoods with the notion of indirect connectivity between nodes. Of course, if two users have rated many common items, then such a definition will also consider them close neighbors. Therefore, the graph-based approach provides a different way of defining neighborhoods, which can be useful in sparse settings.

## **Contributions**

In the following chapters, we address problems that revolve around the central theme of comparing two recommendation systems with different similarity measure ways. We divide the related issues in this into two groups:

* **Build two recommendation systems**. One is the system whose database is stored as a matrix and most of its calculations are represented by the matrix. But the other is stored by using graph database (Neo4j) with query language is Cypher(Chapter 3).
* **Evaluating the above two Recommendation Systems.** We compare two systems with the following criteria: accuracy in the recommendation results, time to each system can recommend for each user, the amount of memory used in the recommendation process (Chapter 4).

In summary, we make conclusions from all analyses and comparison to prove the graph-based Collaborative Filtering RS can replace the traditional Collaborative Filtering RS (using matrix) or if what we want to prove is not exactly, this is also a research direction so that the next people do not repeat this.

## **Organization of Thesis**

This thesis is organized as follows:

Chapter 1: Introduction

In this chapter, we give a short brief about our problem, motivation and our methodology to solve the problem.

Chapter 2: Fundamental theories

In chapter 2, we divide this part into 3 the smaller part:

* Recommendation Filtering techniques
* Phases of Collaborative Filtering process
* Evaluation Metrics
* Related Works

Chapter 3: Realization the Graph-based recommendation system

In chapter 3, we will present the analysis process to build a graph – based recommendation system:

* Fundamentals of inputting dataset to generate a Graph
* Calculating the similarity measure by PathSim
* Prediction the rating scores

Chapter 4: Experiments

In this chapter, we have two parts:

* Introduction to tools, conditions, and datasets in the experiment process
* We will show all the diagrams of the experiment process and evaluate to generate the reasonable results

Chapter 5: Conclusions and Future works

We summarize what we did and propose research directions for the future.

# **Chapter 2: Fundamental theories**

## **Recommendation Filtering techniques**

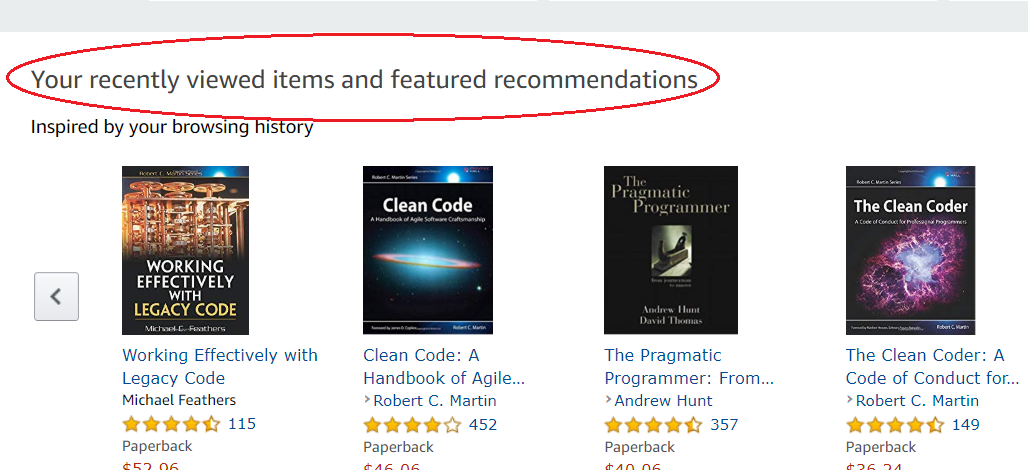
## **Definition**

Basically, to keep things simple, a recommendation system is able to provide suggestions (recommendations) to users, in multiple contexts such as when they are making a choice among a large catalog of items or whenever they want to receive suggestions. Identifies four key features:

* Aid support Decide: predicting a rating for a user for an item
* Aid support Compare: rank a list of items in a personalized way for a user
* Aid support Discover: provide a user with unknown items that will be appreciated
* Aid support Explore: give items similar to a given target item

Most of the applications of recommendation systems are on e-commerce websites. The site displays a list of recommended items to the end user [19].

For instance, in an ecommerce website (such as Lazada), to optimize customer’s purchasing, they are interested in what customers *liked* the product. By relying on their past data (this may the rating the user has voted on the product, the product's browsing time, the number of product clicks, etc.). The system will predict which users will like the product and give suggestions that are appropriate for them.



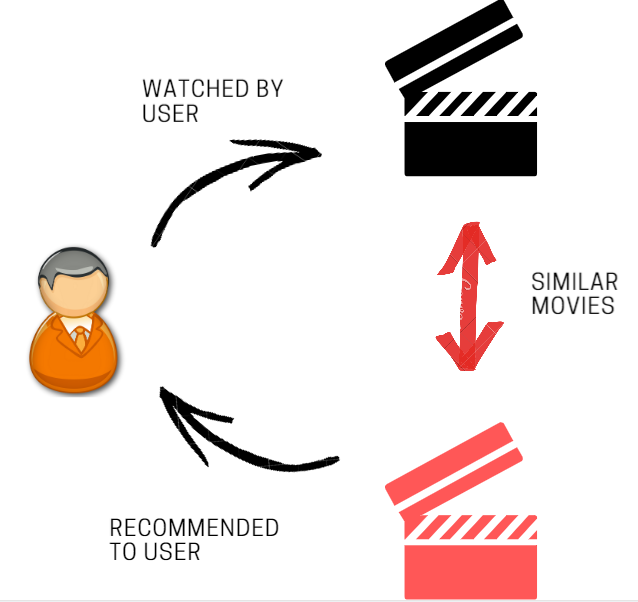
**Figure 2.1. Amazon’s product recommendation**

In Fig. 2.1 shows Amazon’s product recommendation relied on our recently viewed items and featured of the customer.

## **Two main kinds of Recommendation Systems**

Recommendation Systems are usually divided into two major groups:

***Content-Based*** systems focus on properties of items. The similarity of items is determined by measuring the similarity in their properties [20]. In Fig. 2.2 shows a user views a lot of horror movies, so suggests a movie in a database that shares the same horror characteristics to the user, such as the *The Nun* movie. However, there are items that do not have a specific group, and it is impossible to determine the group or characteristics of each item.

****

**Figure 2.2. Content-Based system example**

***Collaborative-Filtering (CF)***systems focus on the relationship between users and items. That is, in place of the item-profile vector for an item, we use its column in the utility matrix. Further, instead of contriving a profile vector for users, we represent them by their rows in the *utility matrix*. Users are similar if their vectors are close according to some distance measure such as Cosine similarity. Recommendation for a user u is then made by looking at the users that are the most similarity to u in this sense and recommending items that these users like. The process of identifying similar users and recommending what similar users like is called collaborative filtering [20]. *Collaborative-Filtering* was the first technique used by a recommendation system and is also considered to be the most popular and widely implemented. Examples of popular Web sites that make use of this technique are *Amazon*, *TiVo* and *Netflix*.

According to Adamavicius and Tuzhilin [22], CF algorithms can be grouped into two categories: memory-based and model-based.

1. *Memory-based methods*: Memory-based algorithms, also referred to as *neighborhood-based* algorithms, were among the earliest CF algorithms. These algorithms predict the ratings of user-item combinations based on their neighborhood. These neighborhoods can be defined in one of two ways:
   * ***User-based collaborative filtering (UBCF****):* these algorithms provide recommendations of items that were liked by similar users [22].
   * ***Item-based collaborative filtering*** *(****IBCF****)*: these algorithms provide recommendations of items similar to those that the user liked in the past [22].

The decision on which approach to use usually relies on the ratio of the number of users to the number of items. In those cases where the number of users is greater than the number of items, item-based approaches are more appropriate because they provide more accurate recommendations while being more computationally efficient. On the other hand, user-based approaches usually provide more original recommendations [21].

Collaborative Filtering method has more some major advantages than Content-based Filtering because it can execute in domains where there is not much content associated with items whose content is difficult for the system to analyze. Also, CF technique has the ability to recommend accidental results, that means it can recommend items that are related to the user even without the content being in the user’s profile [4]. Due to the success of CF techniques, so it was applied frequently in the recommendation systems. But the potential problems still exist quite many as follows:

**Cold-start**: This is a case when a recommendation system does not have enough information about a user or an item in order to make relevant predictions [5][4]. This is one of the main issues that reduce the efficiency of the recommendation systems. If all private information of new users or items will be empty since they have not rated any item; so, the system is not known to their taste.

**Data sparsity**: the problem is that when only a few of the total number of items available in a database are rated by users [4]. That is the common reason that happens to owe to lack enough information. This issue always tends to a sparsity of user-item matrix, inability to specify the nearest neighbors and finally, the accuracy of the recommendation systems will be very bad.

**Scalability**: This is a problem related to recommendation algorithms because all computation in the recommendation process will quickly raise when the number of users and items also increase [5]. That means the running time increases at most linearly with the size of the input. Therefore, it is really necessary to apply recommendation techniques expending scaling up in a successful way to deal with the big data (the huge number of datasets) [32].

In the thesis, we only focus on the Collaborative Filtering systems, especially the system whose database is stored as a matrix and most of its calculations are represented by the matrix. And the other is stored by using graph database (HIN).

## **Phases of Collaborative Filtering process**

## **Input**

In a recommendation-system application there are two classes of entities, which we shall refer to as users and items. Users have preferences for certain items, and these preferences must be teased out of the data. The data itself is represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item. Values come from an ordered set, e.g., integers 1–5 representing the number of stars that the user gave as a rating for that item [20].

**Example 2.1:** In Table 2.1 we see an example utility matrix, representing users’ ratings of movies on a 1–5 scale, with 5 the highest rating. All of ‘?’ sign with gray backgrounds represent the situation where the user has not rated the movie. Recommendation Systems need to automatically fill in these values.

**Table 2.1. The Utility Matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| Dead Pool | 5 | 5 | 1 | 1 | 1 | ? |
| Pacific Rim | 5 | ? | ? | 1 | ? | ? |
| Iron Man | ? | 4 | 1 | ? | ? | 1 |
| Love, Rosie | 1 | 1 | 4 | 4 | 4 | ? |
| La La Land | 1 | 2 | 5 | ? | ? | ? |

In this example, we have 6 users A, B, C, D, E, F and 5 movies. Blue cells indicate that a user has rated a song with a rating of 1 to 5. Cells with gray ‘?’ signs corresponding to cells without data. The work of a Recommendation Systems is to predict the value of these gray cells, thus giving the user a hint.

In this simple example, it is easy to see that there are two different categories of movies: three are action movies and the other two are romantic. From this data, we can also predict that A, B like action movies and C, D, E, F like romantic movies. Therefore, a good system should suggest *Pacific Rim* to B, *Iron Man* for A, *La La Land* for D, E, F. Assume that, there are only two types of film, when there is a new film, we just classify it into any genre and give recommend to each user.

Normally, there are a lot of users and items in the system, and each user usually rates a petite number of items, even those who do not rate items. Hence, the gray cells of the utility matrix in these problems is usually huge, and the number of cells filled is a little number.

Obviously, as more cells are filled, the accuracy of the system will be improved. So, systems always ask the user for their interest in the product and want the user to evaluate as many products as possible. Evaluation of products, accordingly, not only help other users know the quality of the product but also help the system know the user's preferences, thereby having a reasonable advertising policy.

After having input data, we will proceed to normalize this data to increase missing ratings as was mentioned above. Usually, normalization is used for mainly two purposes:

* Eliminating surplus data
* Ensuring data is logically stored

There are many methods to normalize data, with this article we will describe in detail this normalization in later.

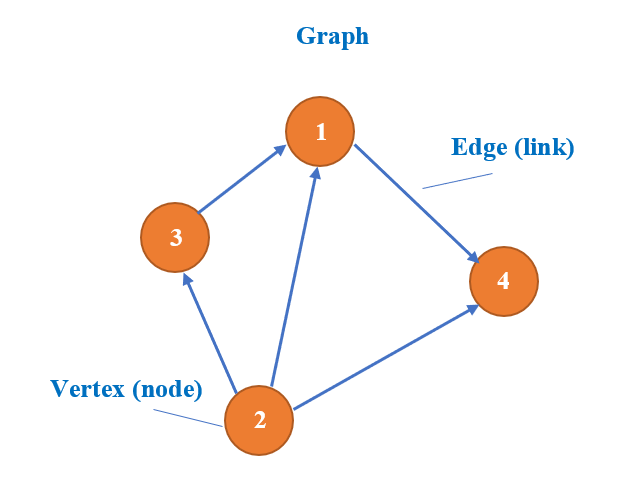
In next section, we introduce the key related concepts of HIN. We  
fist define the HIN and its network schema.

**Heterogeneous Information Network**

A heterogeneous information network [34] is a special type of information network with the data structure as a directed graph, which either contains multiple types of objects or multiple types of links. The most significant example of graph is social network, when we have a graph , where belongs to a particular relation :

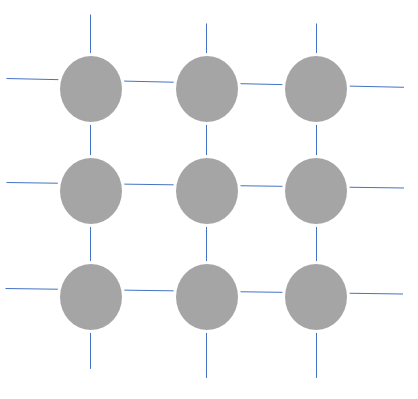
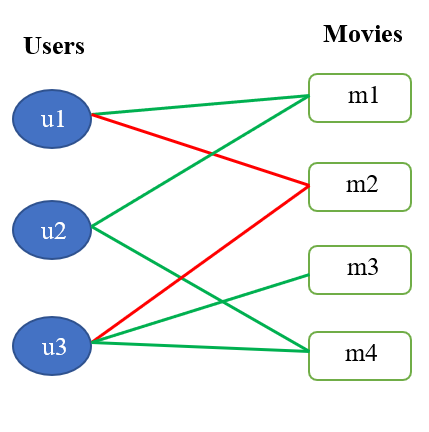
Where:

* V: nodes in the graph, which describe object.
* E: edges in the graph, which ties nodes, define node’s relation.

For most of the cases, and its inverse are not equal. But if the two types are the same, is symmetric. When the types of objects or the types of relations , the network is called **heterogeneous information network**; otherwise, it is a **homogeneous information network.**

#### **Figure 2.3. Graph sample**

As we can see, Figure 2.4. (a) is homogeneous information network because the type of objects or links is unique:



#### **Figure 2.4. An example of homogeneous information network and heterogeneous information network**

#### **(b)**

#### **(a)**

* Object type = 1
* Relation type = 1

And, Figure 2.4. (b) is Heterogeneous Information Network (HIN) because:

* Object type = 2
* Relation type > 1

**Weighted Heterogeneous Information Network**

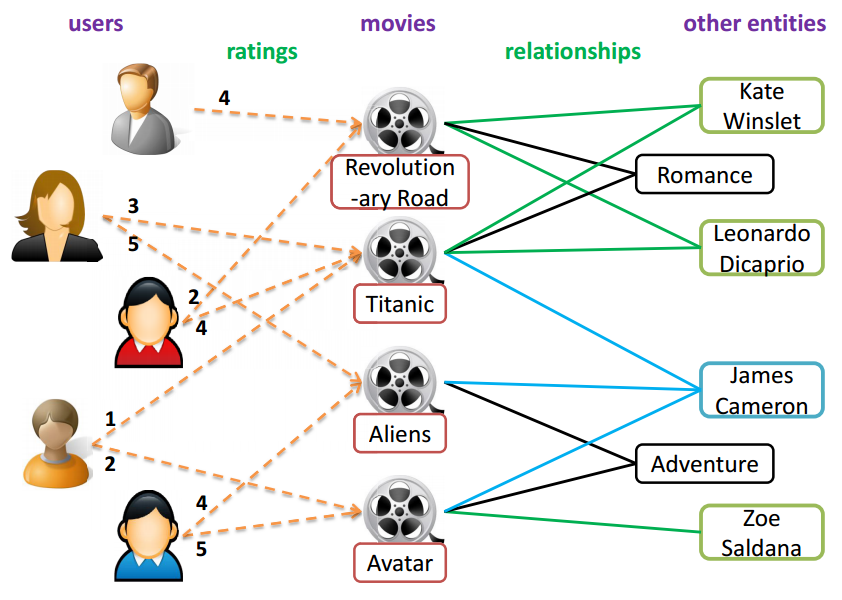
Similarly, we can understand in a simple way as WHIN is the expansion of HIN. In addition, WHIN contains the entity which HIN does not, that is: In each relation R of HIN, it contains a specific value which belongs to a specific type.

Assume that we have a set of attribute values on relations W, where each attribute value w ∈ W belongs to a specific attribute value type. In case, the network is called **unweighted heterogeneous information network**, when:

* The types of objects |A| > 1 (or the types of relations |R| > 1)
* The types of attribute values |W| = 0

And The network is called **weighted heterogeneous information network (WHIN)**, when:

* the types of objects |A| > 1 (or the types of relations |R| > 1)
* the types of attribute values |W| > 0

Conventional HIN is an unweighted HIN, where there are no attribute values on relations or we do not consider them. For a WHIN, there are attribute values on some relation types, and these attribute values may be discrete or continuous values.

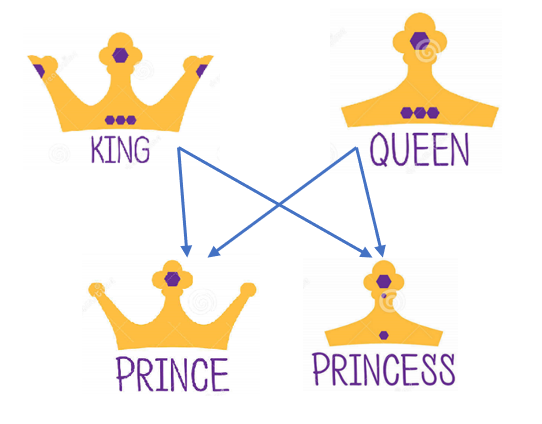
#### **Figure 2.5. An example of weighted heterogeneous information network**

We have this network contains one type of attribute value on the rating relation between users and movies, which take values from 1 to 5. Therefore, we will call this is a Weighted Heterogeneous Information Network

**Meta Path-based Similarity Framework**

In a heterogeneous network, two objects can be connected via different paths. For example, two authors can be connected via “author-paper-author” path, “author-paper venue-paper-author” path, and so on. Intuitively, the semantics underneath different paths imply different similarities. Formally, these paths are called meta paths.

We say a meta path is symmetric if the relation R defined by it is **symmetric**. A meta-path is symmetric if the relation R is symmetric. Suppose that if a relation exists from type A to type B, denoted as , and the inverse relation for. If the two types are the same, R is symmetric; otherwise, it is not symmetric. It means if it is symmetric, .

 The below are three examples about symmetric meta path:

#### **Figure 2.6. An example of the relationship among nodes**

Meta path 1: King PersonQueen

Meta path 2: King Royal Queen

Meta path 3: King Country Queen

Path Count between King and Queen is 2, because there are **2 path instances:**

King PrinceQueen

King PrincessQueen

With Path Count is the number of path instances between two nodes.

#### **Table 2.2. Meta path examples and their physical meanings on bibliographic data**

|  |  |
| --- | --- |
| Meta path | Physical meaning |
| Author-Paper-Author (APA) | Authors collaborate on the paper |
| Author-Paper-Venue-Paper-Author (APVPA) | Authors publish papers on the same venue |
| Author-Paper-Venue (APV) | Authors publish papers at a venue |

As examples shown in Tab. 2.2, authors can be connected via meta paths “Author-Paper-Author” path, “Author-Paper-Venue-Paper-Author” path, and so on. Moreover, Table 2.2 shows path instances and semantics of these meta paths. It is obvious that semantics underneath these paths are different. The path means authors collaborating on the same papers (i.e., co-author relation), while path means authors publishing papers on the same venue. The meta paths can also connect different types of objects. For example, the authors and venues can be connected with the path, which means authors publishing papers on venues. The rich semantics of meta path is an important characteristic of HIN. Based on different meta paths, objects have different connection relations with diverse path semantics, which may have an effect on many data mining tasks. For example, the similarity scores among authors evaluated based on different meta paths are different [14]. Under the path, the authors co-publishing papers will be more similar, while the authors publishing papers on the same venues will be more similar under the path. Another example is the importance evaluation of objects [21]. The importance of authors under path has a bias on the authors who write many papers having many authors, while the importance of authors under path emphasizes the authors who publish many papers on those productive conferences. As a unique characteristic and effective semantic capturing tool, meta path has been widely used in many data mining tasks in HIN, such as similarity measure [13], [14], clustering [15], and classification [16].

**Graph Databases**

The NoSQL acronym may be interpreted as a combination of two words: No and SQL. Authors wanted to emphasize that database is No RDBMS or No relational. Later, others tried to salvage the original term by a new acronym expansion to Not Only SQL. Whatever is the literal meaning, NoSQL is, today, a general term comprising databases and data stores that do not follow the RDBMS principles and usually relates to technologies allowing for processing large data sets and their distribution [26].

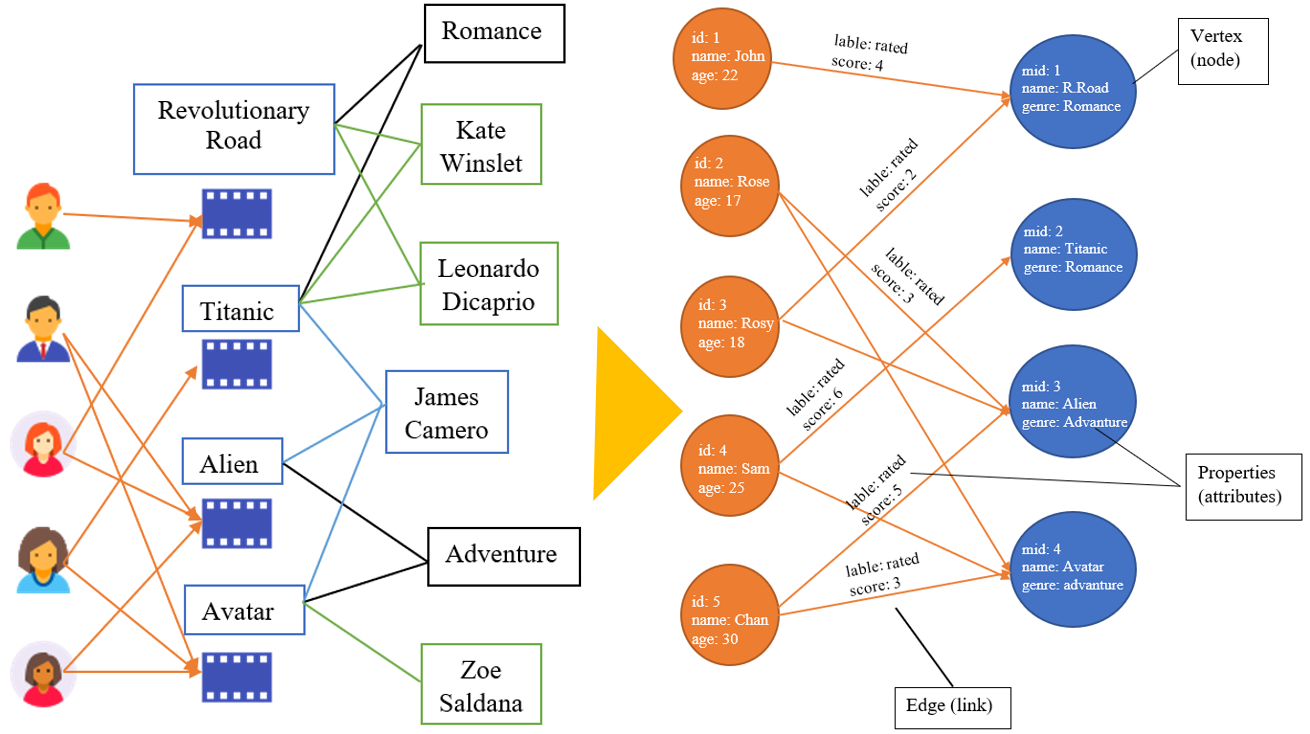
Graph databases is one of the four basic groups of NoSQL databases:

* Key-values Stores
* Document Databases
* Column Family Stores
* Graph Databases

Each of them unites a set of databases with similar behavior although some of them are on the edge and may fall into two groups.

Graph databases use a flexible graph model which can scale across multiple machines. Typical of this kind applications are social networking and recommendations, network and cloud management, geospatial, bioinformatics, and security and access control [27], [1]. In general, relational databases are not suitable for storing relationship data, especially because relationship queries in relational databases are complex, slow, and require many joins; they are therefore unusable for deep traversing. Graph databases differ the most form other NoSQL groups as they are based on different prerequisites. The representatives are Neo4j, Info-Grid, Infinite Graph, Apache Graph, DEX, and Orient-DB.

Heterogeneous Information Network is mostly presented by a graph G = (V, E, A), where:

* V: nodes in the graph, which represent object in HIN.
* E: edges in the graph, which represent the relationship between nodes.
* A: attributes which contains the values (private information) in that node.

#### **Figure 2.7. Conversion from HIN to Graph**

## **Similarity Measures**

## **Cosine Vector Similarity**

Cosine vector similarity is one of the popular metrics in statistics. Since it notionally considers only the angle of two vectors without the magnitude, it is an appropriate measurement with data missing preference information as it can count the number of times that term appears in the data [26].

In the following formula, the cosine vector similarity looks at the angle between two vectors (the target item *i* and the other item *j*) of ratings in n-dimensional item space.

#### **(2.1)**

=

Where:

* is the rating of the target item *i* by user *k*
* is the rating of the other item *j* by user *k*
* *n* is the total number of all rating users to item *i* and item *j*

When the angle between two vectors is near 0 degrees (they are in the same direction), *Cosine similarity* value, **sim(*i*,) is 1**, meaning very similar. When the angle between the two vectors is near 90 degrees, **sim(*i,*) is 0**, meaning irrelevant. When the angle between the two vectors is near 180 degrees (they are in the opposite direction), **sim(*i,*) is -1**, meaning very dissimilar. In case of information retrieval using CF, *sim(i,j)* ranges from 0 to 1.This is also reasonable, as the behavior of the two users is completely opposite when *the similarity* between the two vectors that is the lowest [26].



## **Path-based Similarity (PathSim)**

Except for the above similarity measure methods, we also have some popular methods for HIN. In this section, we first revisit neighborhood meta-path similarities and define distant meta-path similarities.

**Neighborhood meta-path similarity**. The neighborhood meta-path similarity indicates the pairwise proximity between entities linked by meta path(s) (entities are neighborhood entities to each other). Given two entities *i* and *j* connected by a meta-path *P*, if **M***P*(*i,j*) > 0, the neighborhood meta-path similarity is positive. Otherwise, the neighborhood meta-path similarity between *i* and *j* is 0. Both the PathSim and KnowSim [33] are neighborhood meta-path similarities.

**PathSim:** **A meta-path based similarity measure**. Given a symmetric meta-path *P*, PathSim between two entities *x* and *y* of the same entity type is:

#### **(2.2)**

*where is a path instance between x and y, is that between x and x, and is that between y and y.*

This shows that given a meta path , is defined in terms of two parts: their connectivity defined by the number of paths between them following ; and the balance of their visibility, where the visibility is defined as the number of path instances between themselves. Notice that we do count multiple occurrences of a path instance as the weight of the path instance.

## **Prediction generation**

Once CF computes the rating score between users or and items, it generates prediction of the target user’s interest as the most significant step in CF. The predictive rating for the target user *u* on the target item *i* is scaled by the weighted average of all neighbor items’ ratings given by the target user *u* according to the following common formula:

#### **(2.3)**

is the neighborhood of the most similar user u rated item i, is the rating of item i which was rated by user . *sim()* is the weighted similarity of the user and the neighbor users belong to , *n* is the total number of neighbor users.

## **Phases of Collaborative Filtering process**

To understand a system that suggests using the traditional CF method that we have mentioned in a section above. Let's look at the examples as follows:

**Table 2.3.a. Original utility matrix Y and mean user ratings [24]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | u0 | u1 | u2 | u3 | u4 | u5 | u6 |
| i0 | 5 | 5 | 2 | 0 | 1 | ? | ? |
| i1 | 4 | ? | ? | 0 | ? | 2 | ? |
| i2 | ? | 4 | 1 | ? | ? | 1 | 1 |
| i3 | 2 | 2 | 3 | 4 | 4 | ? | 4 |
| i4 | 2 | 0 | 4 | ? | ? | ? | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3.25 | 2.75 | 2.5 | 1.33 | 2.5 | 1.5 | 3.33 |

**Table 2.3.b Normalized utility matrix [24]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | u0 | u1 | u2 | u3 | u4 | u5 | u6 |
| i0 | 1.75 | 2.25 | -0.5 | -1.33 | -1.5 | 0 | 0 |
| i1 | 0.75 | 0 | 0 | -1.33 | 0 | 0.5 | 0 |
| i2 | 0 | 1.25 | -1.5 | 0 | 0 | -0.5 | -2.33 |
| i3 | -1.25 | -0.75 | 0.5 | 2.67 | 1.5 | 0 | 0.67 |
| i4 | -1.25 | -2.75 | 1.5 | 0 | 0 | 0 | 1.67 |

**Table 2.3.c. User similarity matrix S [24]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | u0 | u1 | u2 | u3 | u4 | u5 | u6 |
| u0 | 1 | 0.83 | -0.58 | -0.79 | -0.82 | 0.2 | -0.38 |
| u1 | 0.83 | 1 | -0.87 | -0.4 | -0.55 | -0.23 | -0.71 |
| u2 | -0.58 | -0.87 | 1 | 0.27 | 0.32 | 0.47 | 0.96 |
| u3 | -0.79 | -0.4 | 0.27 | 1 | 0.87 | -0.29 | 0.18 |
| u4 | -0.82 | -0.55 | 0.32 | 0.87 | 1 | 0 | 0.16 |
| u5 | 0.2 | -0.23 | 0.47 | -0.29 | 0 | 1 | 0.56 |
| u6 | -038 | -0.71 | 0.96 | 0.18 | 0.16 | 0.56 | 1 |

**Table 2.3.d. Predict normalized ratings [24]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | u0 | u1 | u2 | u3 | u4 | u5 | u6 |
| i0 | 1.75 | 2.25 | -0.5 | -1.33 | -1.5 | 0.18 | -0.63 |
| i1 | 0.75 | 0.48 | -0.17 | -1.33 | -1.33 | 0.5 | 0.05 |
| i2 | 0.91 | 1.25 | -1.5 | -1.84 | -1.78 | -0.5 | -2.33 |
| i3 | -1.25 | -0.75 | 0.5 | 2.67 | 1.5 | 0.S59 | 0.67 |
| i4 | -1.25 | -2.75 | 1.5 | 1.57 | 1.56 | 1.59 | 1.67 |

**Table 2.3.e. Full Y [24]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | u0 | u1 | u2 | u3 | u4 | u5 | u6 |
| i0 | 5 | 5 | 2 | 0 | 1 | 1.68 | 2.7 |
| i1 | 4 | 3.23 | 2.33 | 0 | 1.67 | 2 | 3.38 |
| i2 | 4.15 | 4 | 1 | -0.5 | 0.71 | 1 | 1 |
| i3 | 2 | 2 | 3 | 4 | 4 | 2.1 | 4 |
| i4 | 2 | 0 | 4 | 2.9 | 4.06 | 3.1 | 5 |

To obviously understand in the below sections, we will describe the process of recommending item to user.

The last row in Table 2.3.a is the average value for each user. The high value corresponds to the easy-going user and vice versa. Then, if we continue to subtract from each evaluation, take this value and replace the unknown value with zero, we will get *the normalized utility matrix* shown in Table 2.3.b. This normalization is important because:

* Subtracting the mean of each column results in positive and negative values ​​in each column. Positive values ​​correspond to the user's preference for the item, negative values ​​correspond to the user's dislike of the item. The value of 0 corresponds to the undetermined whether users like this item or not.
* Technically, the dimensions of *the utility matrix* are huge with millions of users and items, and if you save all of these values ​​in a matrix, high-capacity will not be enough. Observe that since the number of predictive evaluations is usually a small fraction of the utility matrix's size, it is better to save this matrix as a *sparse matrix*, saving only the non-zero values ​​and positions of them. So, better yet, marks '?' should be replaced with the '0' value, which does not specify whether the user likes the product or not. This not only optimizes memory, but *similarity matrix* calculations are also more efficient.

After normalizing the data above, some of *the similarity function* commonly used as mentioned above, in this article, we use Cosine because of its popularity.

The Cosine measure is redefined as show in equation 2.4, the cosine vector similarity looks at the angle between two vectors (the target user and the other user ).

#### **(2.4)**

=

Therein , is the vectors appropriate to users 1, 2 are normalized as above.

It suffers from the same bias as Pearson [34]. Moreover, in the case of collinear vectors, the cosine similarity is maximum. In a pure geometrical context, this result can make sense but in our case it does not. This problem as been well identified and described in (Breese et Kadie) [19].

*Similarity matrix* ***S*** is a symmetric matrix as is a positive function if user A is the same as user B, the opposite also is true. The blue cells on the diagonal are equal to 1 because that is the cosine of the angle between a vector and itself, cos(0) = 1. When calculating the following steps, we don’t care about these values. Continuing to observe the vectors in rows corresponding to , ,, we will observe some interesting things:

* is close to and (the likeness is positive) than the remnant. The similarity between and is understandable because they are both more concerned to than the remnant.
* is closer to than the rest of the users.
* is closer to than the rest of the users.

According to *the* *similarity matrix*, we can separate the users into two () and () groups. Due to, *the* ***S*** *matrix* decompose small, we can comfortably observe this, as the total users is larger, visually determining is not feasible. It is important to note that when the number of users is larger, *the* ***S*** *matrix* will also be huge and most likely insufficient memory to store. In those cases, we only demand to calculate and store the outcome of a row of *the similarity matrix*, corresponding to the similarity between that user and along with others user [24].

After created similarity matrix, in the next step we define rating predict of users to items.

Determining a user's level of interest in a item relied on neighbor users which is used in the K-nearest neighbors (KNN) method. When manipulating with the large-scale problem, we see that the KNN method is used substantially for its simplicity. Of course, we can’t use KNN directly, but we have to do more intermediate steps.

In Collaborative Filtering, *missing rating* is also based on information about *k neighbor users*. Of course, we only care about **users who have evaluated the item being reviewed**. *Predicted rating* is usually defined as a weighted average of normalized assessments. A noticeable point, in KNN, the weights are determined based on *the distance* between two points, and these distances are non-negative. In the CF, the weights are determined based on *the similarity* between the two users, which may be less than 0 as shown in Table 2.3.c.

Depending on prediction rating formula mentioned above, we can calculate predict normalized rating of on with a difference after normalized is in this case is average (after normalized) the rating of item which was rated by users belong to . Let’s see example below:

Example predict normalized rating of on with nearest neighbors (k = 2). The steps are:

1. Identifying users who rated : {, , }
2. Corresponding similarities: {0.83, -0.4, -0.23}

* Most similar users: (, ) = {, }

1. With normalized ratings {0.75, 0.5}
2. Predict rating: = 0.48

After predict rating, we have full matrix Y which shows in Table 2.3.e.

The system recommends items for each user can be identified in different ways. It is possible to arrange unrated items respectively large to small based on predicted ratings or select only those products that have positively normalized predicted ratings. This corresponds to the fact that this user more prefers.

As soon as comprehending the knowledge and how to build a traditional CF system. In the next chapter, we will discourse in detail the main problem of this thesis which is the CF system based on HIN is presented through Graph method.



## **Evaluation Metrics**

The efficiency of a recommendation algorithm can be appraised using different types of measurement which called be the accuracy of the algorithm. The type of metrics used relies on the type of filtering technique. The metric of prediction accuracy is essentially about the error of prediction. This metric is a common metric in evaluating various machine learning algorithms, such as regression or classification. This metric used in evaluating recommendation systems is mainly used to measure the ability to predict users’ behaviors. Prediction accuracy is the most important metric in the offline analysis of recommendation systems. This metric is commonly used in early papers of recommend systems researches to discuss the accuracy of different recommendation algorithms.

When calculating prediction accuracy, you need a set of the offline dataset that contains users’ s scores, such as users’ s ratings for a product or movie. The dataset is divided into the two smaller datasets, those are training and testing set. A users’ rating prediction model is trained on the training set, then the prediction of users’ rating is computed on the testing set. The error is the deviation between the predicted rating and the real rating. There are three metrics to measure the prediction accuracy: Mean Absolute Error (MAE), Mean Square Error (MSE) and Root Mean Square Error (RMSE), and the formulas are as follows:

**Root Mean Square Error (RMSE):**

#### **(2.5)**

**Mean Absolute Error (MAE):**

#### **(2.6)**

**Mean Square Error (MSE):**

#### **(2.7)**

Where:

* Q is the test set
* represents the user’s true ratings
* represents the prediction rating of the recommendation system

MAE is the simplest, but it does not take into account the direction of the error (positive error or negative error). MSE has a larger penalty on large errors and the squared error does not have an intuitive meaning. Therefore, RMSE is more widely used in computing the prediction accuracy of the recommendation system.

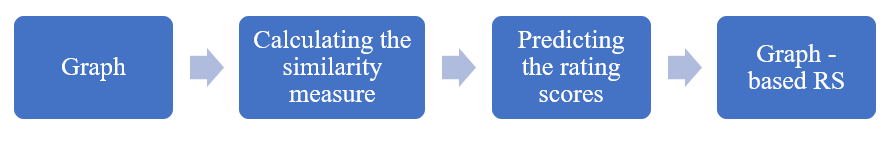
It is worth noting that while using the prediction accuracy to compare the recommendation algorithms, it is necessary to use the same dataset for different algorithms. Although the prediction accuracy is mainly concerned with predicting users’ rating, this metric often has guiding significance to the overall performance of recommendation system [6].

RMSE is suitable for the prediction task, because it measures inaccuracies on all ratings, either negative or positive. However, it is most suitable for situations where we do not differentiate between errors. For example, in the IMDB rating prediction, it may not be as important to properly predict the difference between 1 and 2 stars as between 2 and 3 stars. If the system predicts 2 instead of the real 1 rating, it is unlikely that the user will putative this as a recommendation. However, a predicted rating of 3 may seem like an encouragement to rent the movie, while a prediction of 2 is typically considered negative. It is arguable that the space of ratings is not truly uniform, and that it can be mapped to a uniform space to avoid such phenomena.

## **Related Works**

Recently there is a lot of comparison between filtering algorithms with another algorithm for recommendation systems based on Collaborative Filtering. *Emmanouil G. Vozalis and Konstantinos G. Margaritis* [27] reviewed of the experiments on two contrasting recommendation system's algorithms: classic Collaborative Filtering and Item-based Filtering. They discuss the results extracted from the experiments and test the validity of the claim that Item-based Filtering improves significantly on the performance of classic Collaborative Filtering. *Christian Desrosiers and George Karypis* [28] presented a comprehensive survey of neighborhood-based methods for the item recommendation. Especially, the main benefits of such methods, as well as their principal characteristics, are described. Furthermore, this document addresses the essential decisions that are required while implementing a neighborhood-based recommendation system and gives practical information on how to make such decisions. *Dilek Tapucu, Seda Kasap and Fatih Tekbacak* [29] analyzed CF algorithms and present results for combined user-based/item-based CF algorithms for different size of datasets. Their goal is to show combined solution results using Loglikelihood, Spearman, Tanimoto and Pearson algorithms. The contribution is to describe which user-based CF algorithms and user/item based combined CF algorithms perform better according to dataset, sparsity, execution time and k-neighborhood values. *Rasna R.Walia* [30] proposed the implementation of graph-based semi supervised learning methods and memory-based methods to the collaborative filtering scenario and compares these methods to baseline methods such as techniques based on weighted average. This work compares the predictive accuracy of these methods on the MovieLens dataset. *Taras Hnot, Tetiana Gladkikh* [31] compares performance of a few algorithms based on 1M MovieLens dataset.

# **Chapter 3: Realization the Graph-based recommendation system**

In this chapter, we focus on the process to build the Graph - based recommendation system by applying the PathSim similarity measure formula as we presented in Chapter 2. The below figure performs that process:

#### **Figure 3.1. We see the phases to build a Graph – based RS**

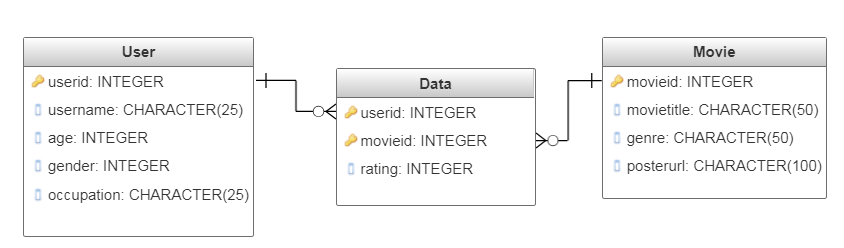
Hence, this chapter contains three parts:

* Fundamentals of inputting dataset to generate a Graph
* Calculating the similarity measure by PathSim
* Prediction the rating scores



## **Fundamentals of inputting dataset in graph**

In this section, we discuss the way to import dataset into graph. At first, we need to identify that data in all datasets were usually organized the tabular form (row - column) as relational databases. But the structure of a graph data model is laid out quite differently. Therefore, we must understand clearly how to translate the relational database structure to the graph.

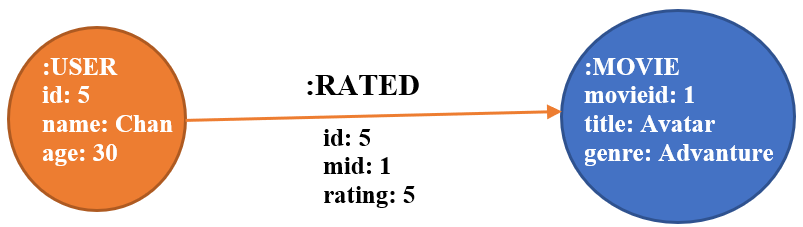
The following is an Entity-Relationship diagram of the dataset sample which includes in three table: user, movie, and data:

#### **Figure 3.2. ER diagram for the dataset sample.**

In above relational databases, we must reference primary key attributes via foreign key columns at different rows of the tables. Joins are computed by matching primary and foreign keys of all rows in the connected tables at query time.

All steps which we list the below used to derive a graph model into the specific environment is Neo4j Graph platform from a relational model:

* **Table to Node Label** – each entity table becomes a label on nodes.
* **Row to Node** – each row in a table becomes a node in the graph.
* **Column to Node Property** – columns (fields) become node properties.
* **Add Constraints/Indexes** – add unique constraints for primary keys, add indexes for frequent lookup attributes.
* **Foreign keys to Relationships** –foreign keys to relationships between tables.
* **Join tables to Relationships** – join tables will become relationships, columns on those tables become relationship properties

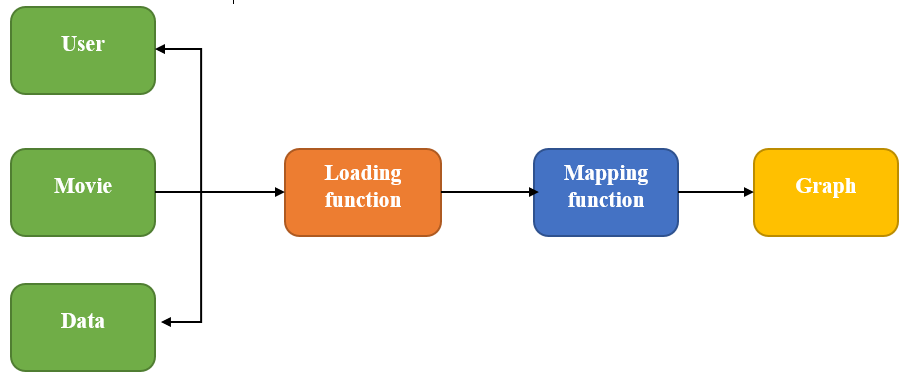
As illustrate, we applied the list above to generate a graph like the one shown below from Fig. 3.1:

#### **Figure 3.3. Graph model from the dataset sample in Figure.3.2.**

We have:

* Label of node :USER and :MOVIE represent each table name, and the label is :RATED which represents the Data minor (relationship) table
* Node: each node represents each user or each movie with its id index is unique. In this case, we perform two node equivalent with two rows (one row is the 5th row in USER table and one row is the 1st in MOVIE table)
* Property: as we can see, the left node contains three properties such as id, name, and age of a user. And the right node also contains three properties such as movieid, title, and genre of a movie
* Constraints/Indexes: adding constraints/indexes are quite similar with creating the primary key in relational database. The main purpose is to speed up the search process by identifying the specific location and we want to make sure we don’t store any duplicated user nodes.
* Foreign keys: the foreign keys of Data table represents the relationship between User and Movie table
* Join tables: we set the relationship of the two join tables is :RATED, it likes a label of this relationship

By using a graph, we do not worry about table joins and index lookups because graph is structured by individual entity and its relationships with other individual entities.

 After identifying the correlation between the tabular database structure and graph database structure, we keep on with the important section; that is, creating a fully-graph. In this case, this is each phase in the process to generate a graph in Neo4j, described in this figure:

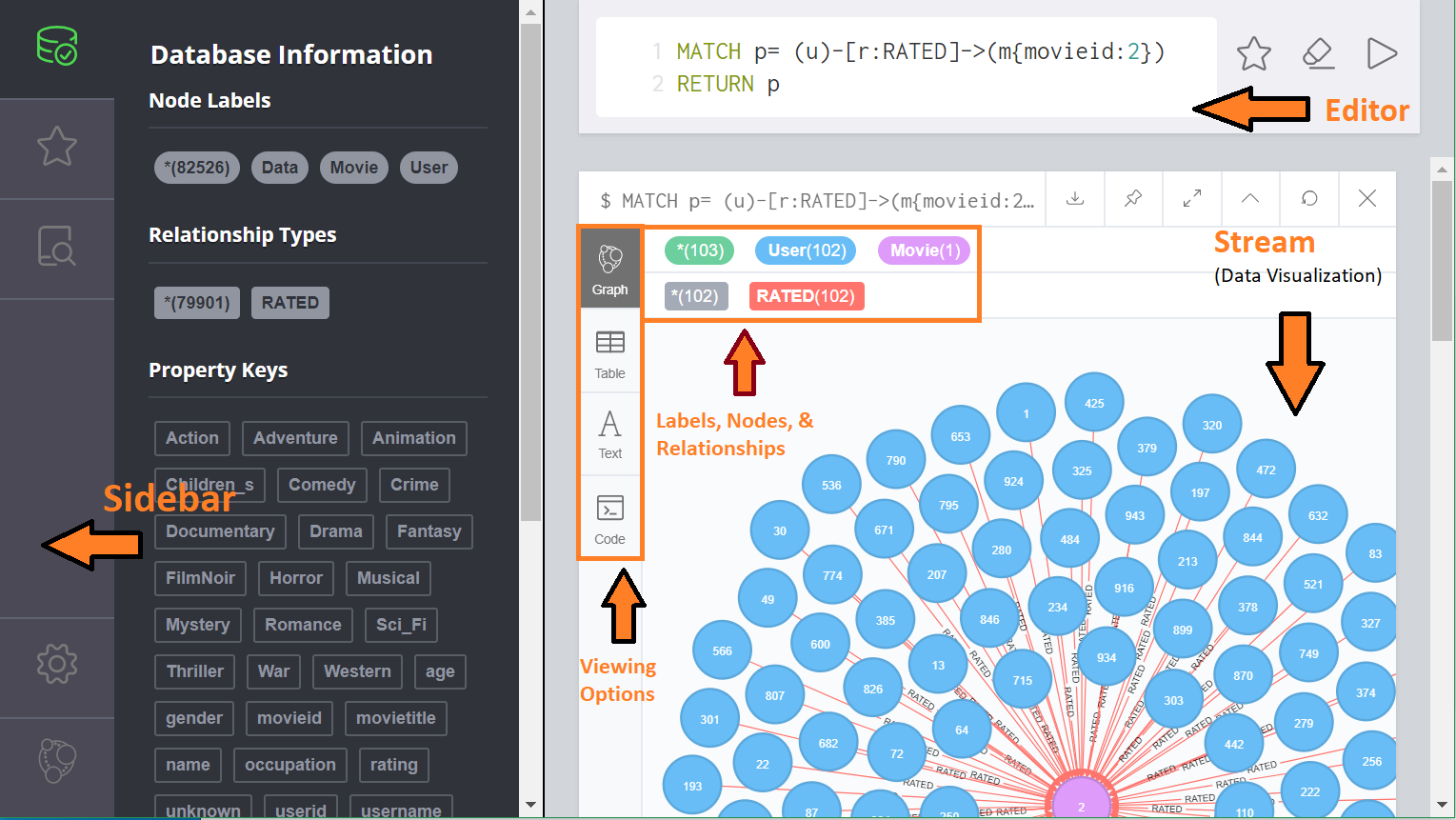
#### **Figure 3.4. The Inputting Process**

* In left hand side of the figure, the input of this process is the main three data file from a dataset.
* The Loading function uses these data as input in order to load all data from all files into Neo4j. After completing this function, we have all nodes in a graph. In this case, we have an incomplete graph due to the lack of edges (relationship among nodes). For traditional collaborative filtering which have been discussed in chapter 2 to find the answers.
* The Mapping function used to create all links among nodes. In this phase, the Data file is the most important because they are used to create the links among nodes in a graph and to add the values in those links (weighted).
* After all phases, we have a complete graph which represents the result of the mapping function includes in all nodes, edges, and all properties.

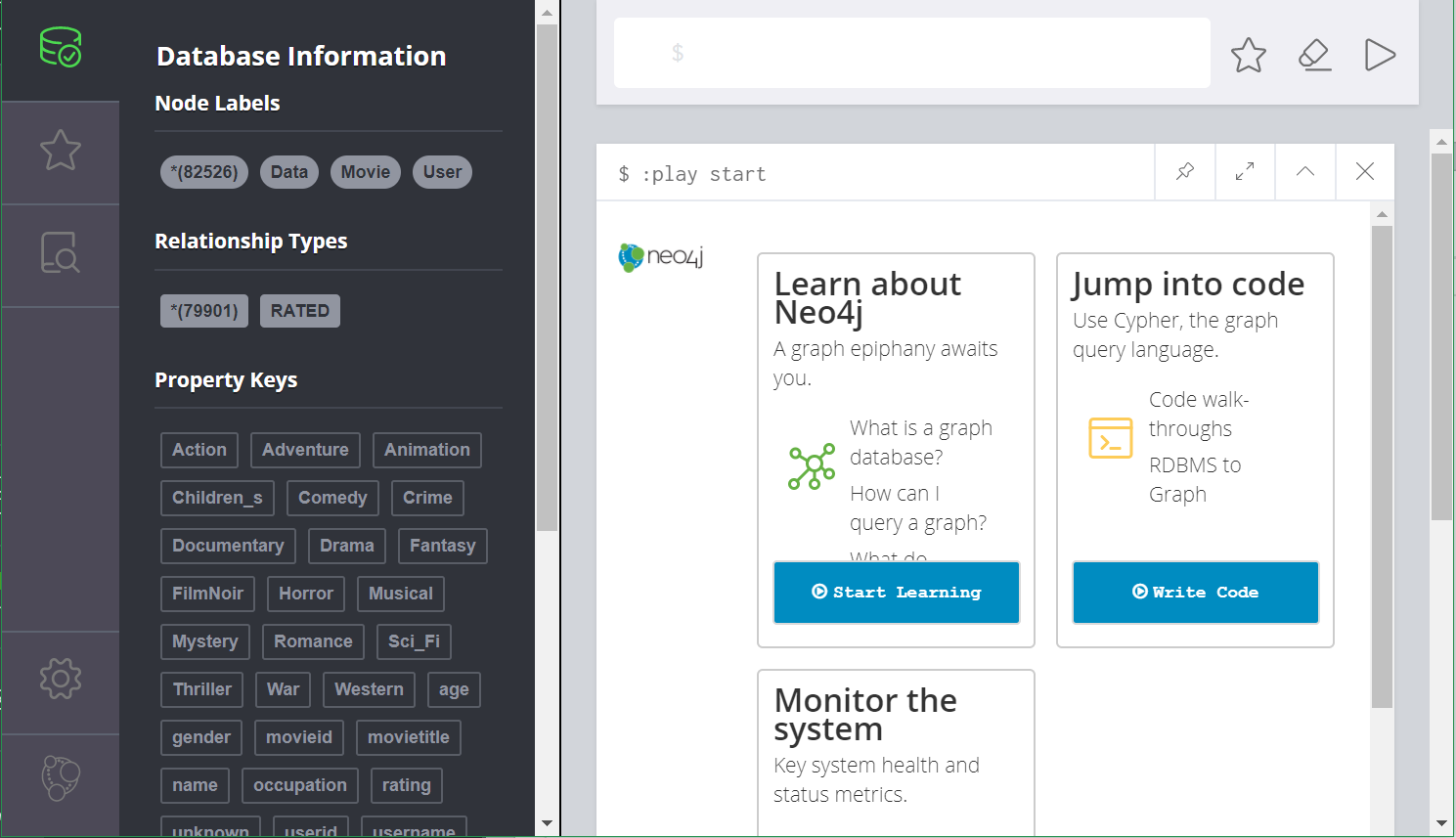
The compulsory requirement of input data in Neo4j must be the files have the extension .CSV [39], the reasons are listed as follows:

* supports loading / ingesting CSV data from an URI
* direct mapping of input data into complex graph/domain structure
* data conversion
* supports complex computations
* create or merge data, relationships and structure

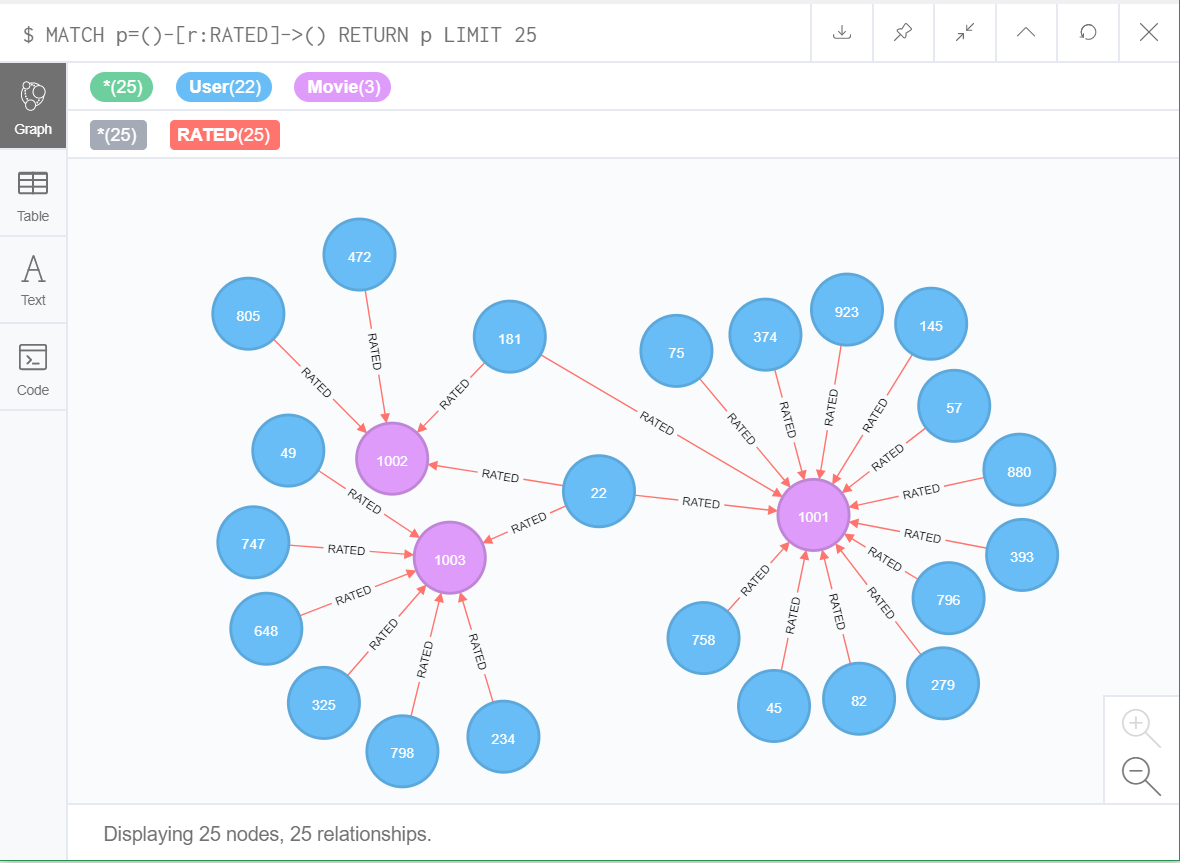
In above phases, we use Cypher - a [declarative](https://en.wikipedia.org/wiki/Declarative_programming) graph query language to deploy on these phases. The reason why we use Cypher is that its syntax is a relatively simple and the complicated database queries can easily be expressed through Cypher.

Firstly, we need to know graphical user interface (GUI) of the Neo4j browser:

#### **Figure 3.5. Here is an overview of the Neo4j browser interface**

 In the above figure, we can see all the data, which were saved into the graph, were displayed in left column Database Information:

#### **Figure 3.6. We see Neo4j's browser after executing the loading function**

* **Node Labels**: This is the area to display the label name of all nodes and the number of nodes which are in the present graph
* **Relationship Types**: This is also the area to display the label name of all relationships and the number of relationships (links) which are in the present graph
* **Property Keys**: This area displays all properties of all nodes in graph

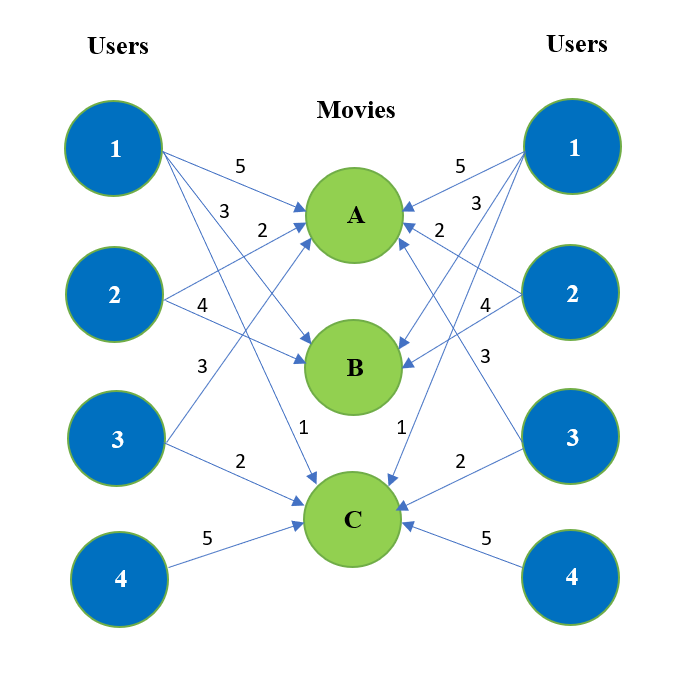
#### **Figure 3.7. We see an example for graph in Neo4j after executing the mapping function**

In the above figure, we can see a fully graph sample which contains 23 nodes of the User label, 3 nodes of the Movie label, and 25 relationships of the RATED label.

In order to make more clearly, you can refer to the Appendix section at the last thesis where contains all Cypher codes of the above entire phases.

## **Calculating the similarity measure by PathSim**

In this section, the next phase is one of the most phases in a recommendation system; that is, calculating the similarity measure. We only focus the similarity measure between users each other.

In the below sections, we will describe the process of recommending item to user.

**Figure 3.8. An example for User – Movie Graph sample with the rating scores contained on the links**

As we can see, the graph sample with the rating scores contained on the links, the relationship path between user and movie contains the rating score which is from 1 to 5.

**Input:**

* u: represent the user id
* m: represent the movie id
* rl: the rating score l at the relationship r between user and movie
* Calculate the similarity measure between the target user and the others
* Calculate the predicted rating score for the movies which the target user did not yet rate

**Output:**

Print the movies will be recommended for the target user

**Algorithm 1: PathSim – based recommendation algorithm**

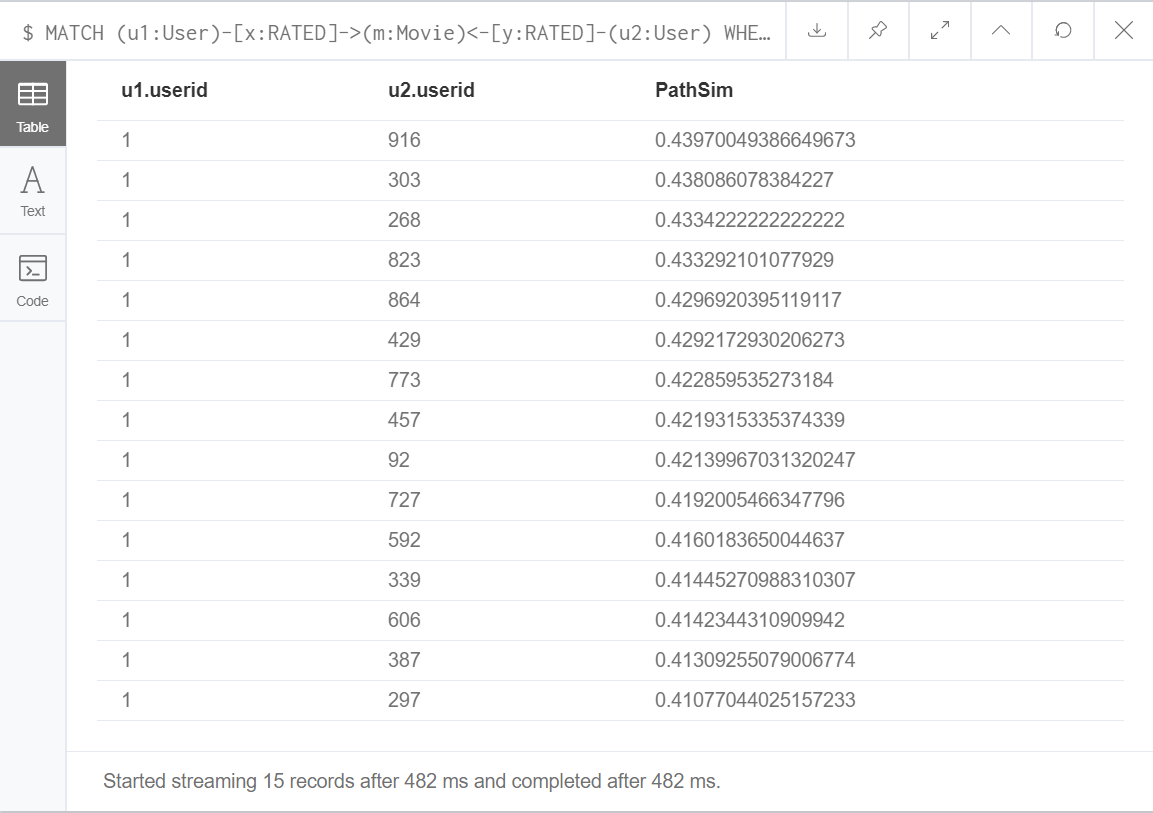
**The first step:**

* Calculate the similarity metric for each pair of users:

The following steps is to calculate the PathSim metric:

**The pseudo code of PathSim - based recommendation algorithm**

|  |
| --- |
| 1: : targetUser()  2: I: set of items  3: U: set of users  4: function PathSim():  5: foreach i in I:  6: if ( and rated the same item i with rating score )  7: numerator += [i] \* [i]  8: if ( rated for item i with rating score )  9: denominator += [i] \* [i]  10: if ( rated for item i with rating score )  11: denominator += [i] \* [i]  12: return (2\*numerator)/denominator |

 As illustrate, the following figure displays the results in Neo4j after executing the PathSim function:

**Figure 3.9. An example for calculating the PathSim similarity measure**

As we see, the target user is a user who has userid = 1. And, we calculate the PathSim similarity measure between each user and that target user. Besides, The PathSim results are sorted in descending order.

**The next steps:**

* For each target user u1, select the top k neighbors based on the PathSim metric
* Identify movies m rated by the top k neighbors that have not been rated by the target user
* Calculate predicted rating score each movie based on top k neighbors for the target user

## **Prediction the rating scores**

The following steps is to calculate the predicted rating score:

Suppose that:

* u1 is the target user
* M is a set of movies rated by top k neighbors that have not been rated by the target user
* H is a set of top k neighbors

**The pseudo code of Predicting scores function**

|  |
| --- |
| 1: Initialize Pred to zero  2: Initialize num to zero, the numerator of Pred  3: Initialize deno to one, the denominator of Pred  4: m: is a specific movie id in M  5: h: is the first neighbours in H  6: while h in H  7: if h had rx with m  8: Add the multiply rx and PathSim(u1, h) into num  9: Add PathSim(u1, h) into deno  10: The next neighbor h  11: Set Pred is the num divided by the deno |

**The final step:**

Print the top movies based on the predicted rating score.

**The specific example** from Figure 3.8:

We assume that is the target user.

The similarity metric between and the other users:

Similarly, we also have:

With:

* PI: Path Instances
* The rating score is the corresponding number of path instances between 2 nodes

Therefore, the neighbors of consist of , , and . But in this case, we will choose top k = 2. It’s up to you for choosing the k value, it depends on the number of neighbors who you want to get their rating scores to predict for the target user. So we only get and because their similarity metrics are closest with the target user .

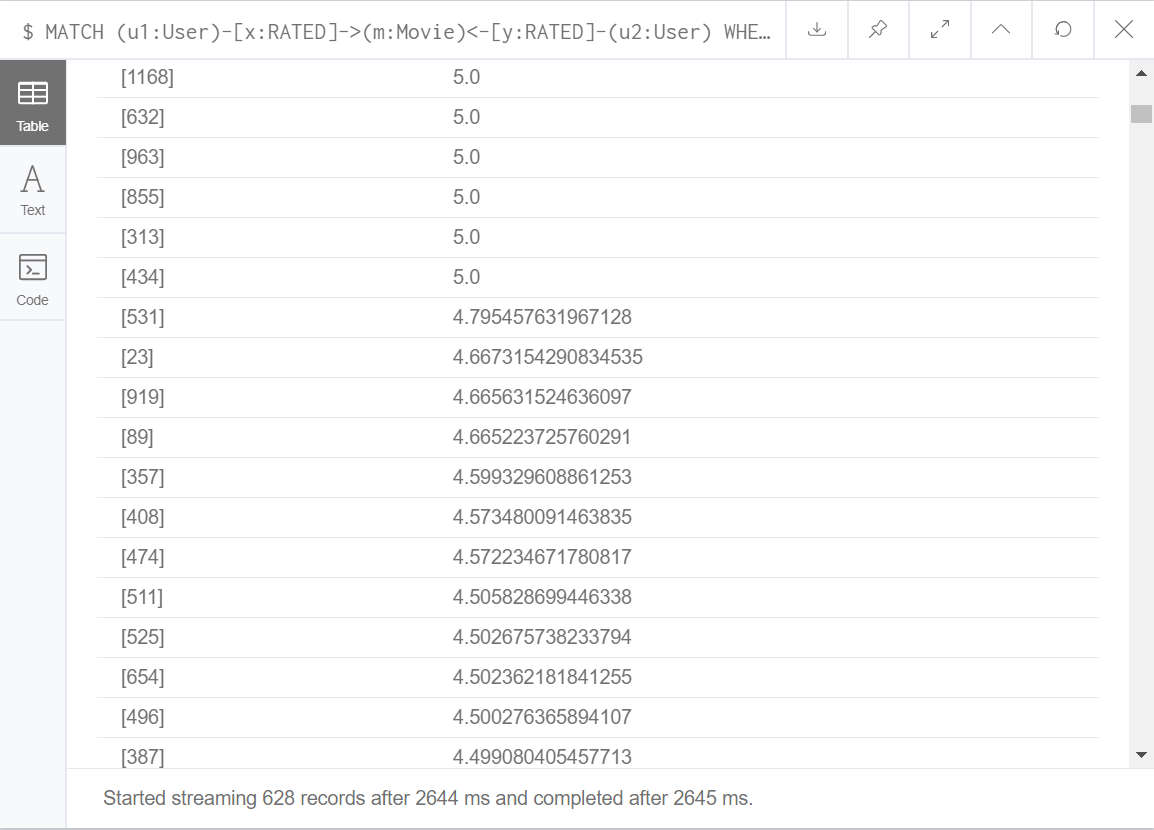
And movies rated by the neighbors and that have not been rated by are and . As a result, we must calculate the predicted rating score for and with the target user .

In conclusion, the recommended movies will be arranged base on the predicted rating score as follows:

**Table 3.1. The results will be recommended for u4**

|  |  |
| --- | --- |
| Movie | Predicted rating score |
| A |  |
| B |  |

To illustrate, the below figure displays the recommended results for the target user after completing the predicting phase in Neo4j:

****

**Figure 3.10. These are the final results which will be recommended for the target user**

## **Chapter 4: Experiments**

This part basically considers and analyzes the results of varying different parameters in a more sophisticated user-based CF algorithm. We illustrate our test results with different datasets which are supplied from MovieLens Research Website to evaluate variants of user-based recommendation algorithms between a Traditional Similarity Search and a Graph – based Similarity Search.

## **Datasets**

MovieLens has two available datasets:

* 100,000 ratings (1-5) for 1682 movies by 943 users (100K dataset which we used it in the first step of our experiments). Each user has rated at least 10 movies. Users and items are numbered consecutively from 1. The data is randomly ordered.
* 1 million ratings (1-5) for 3900 movies by 6040 users (1M dataset which is used also for a second step in our experiments).

## **Environments and configures**

We set up a computer (Intel Core i7-8550U, 8GB RAM, and 512 SSD) to support the storage and analysis of the experiments. And we use Python language to build the two recommendation systems with IDE is Visual Studio Code, and the database management system is SQLite. Additionally, we also use Neo4j, one of the most frequently used graph databases, to store the graph database and Cypher, Neo4j’s graph query language, allows users to store and retrieve data from the graph database. The Neo4j version which we use is Neo4j Desktop (free version).

The performance of Neo4j for a data search depends on the available memory to hold the entire graph database. If less memory is used than what the constructed graph database requires, a swap between the memory and hard disk should occur, but frequent swaps between memory and hard disk inevitably slow down the search speed. Thus, large memory is needed and should also be set up for Neo4j for full usage of the system memory. Thus, we don’t test the executed memory because we can configure the executed memory in Neo4j.

## **Experiment results**

The sensitivity of parameters as neighborhood size and the loading factor of the dataset should be determined for comparing algorithms and evaluation results. We use Root Mean Square Error (RMSE) as our choice of evaluation metric to evaluate the quality of our recommendation system and report prediction experiments because of its common usage and ease of implementation. RMSE evaluates the accuracy of a system by comparing the numerical recommendation scores against the actual user ratings for the user-item pairs in the test dataset.

The square root of this equation gives RMSE value. The lower the RMSE value is, the more accurately recommendation engine predicts user ratings.

In our experiments, we just based on User-Based CF for both methods. Cosine Similarity and PathSim Similarity algorithms are used and showed in the graphics of experimental evaluations.

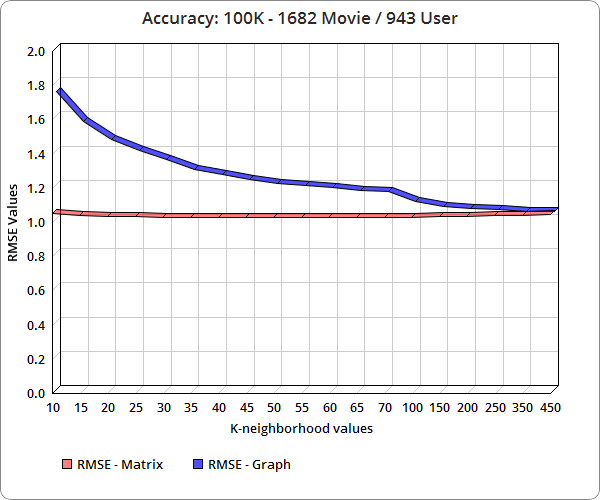
When the task is to generate a top-N recommendation, we need to find k most similar users or items (nearest neighbors) after computing the similarities, and then aggregate the neighbors to get the top-N most frequent items as the recommendation.

We discuss the results of Fig. 4.1 to Fig. 4.5 for user-based CF algorithms according to accuracy, memory, time and K-neighborhood parameters.

**Table 4.1. The RMSE accuracy is tested on MovieLens 100k dataset**

|  |  |  |
| --- | --- | --- |
| K-neighborhood values | RMSE-Matrix | RMSE-Graph |
| 10 | 1.04861122 | 1.742388 |
| 15 | 1.037894548 | 1.565377 |
| 20 | 1.032763404 | 1.460149 |
| 25 | 1.030238442 | 1.395637 |
| 30 | 1.028444872 | 1.340109 |
| 35 | 1.027688872 | 1.286397 |
| 40 | 1.027760522 | 1.255153 |
| 45 | 1.02776354 | 1.225625 |
| 50 | 1.027433409 | 1.204477 |
| 55 | 1.027637549 | 1.190593 |
| 60 | 1.028241791 | 1.177809 |
| 65 | 1.0286041 | 1.163393 |
| 70 | 1.028804371 | 1.152236 |
| 100 | 1.029211123 | 1.094498 |
| 150 | 1.029826424 | 1.065008 |
| 200 | 1.032013414 | 1.053908 |
| 250 | 1.036295234 | 1.047114 |
| 350 | 1.039103812 | 1.03857 |
| 450 | 1.041223454 | 1.036322 |

Let’s see result in graph below:



**Figure 4.1. Accuracy in 100k MovieLens dataset**

**Accuracy:** The accuracy of the traditional recommendation system is highest at the position k = 50 (the smallest value of RMSE is 1.027433409). If the out of that scope is the accuracy will be reduced. For the graph-based recommendation system, k is directly proportional to the accuracy. Therefore, the graph-based recommendation system will have the advantage of a quite high accuracy if that is a huge dataset (big data) and k gets the big value.

**K-neighborhood:** The size of the neighborhood has a considerable impact on the prediction quality. To determine the sensitivity of this parameter, we performed an experiment where we varied the number of neighbors to be used to find out the position k which the two recommendation systems will have the relatively highest accuracy.

The reason data storage by Matrix is to perform the users as vectors. Therefore, we can easily apply the Cosine formula to calculate the similarity between users.

* + If the angle between two vectors is 0 degree,
  + If the angle between two vectors is 90 degree,
  + If the angle between two vectors is 180 degree,

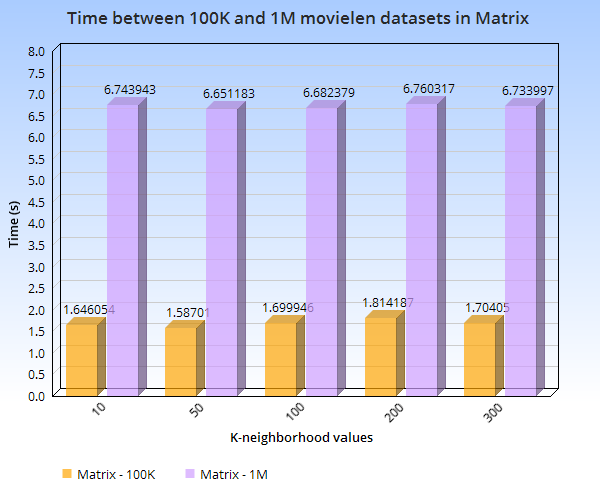
A value close to 1 indicates similarity, while a value less than 0 indicates just the opposite. Therefore, we can see:

With represents the users irrelevant or less relevant with the target user. Hence, we call these values is the negative effect. Because it will negatively affect the recommended results (the accuracy will be reduced).

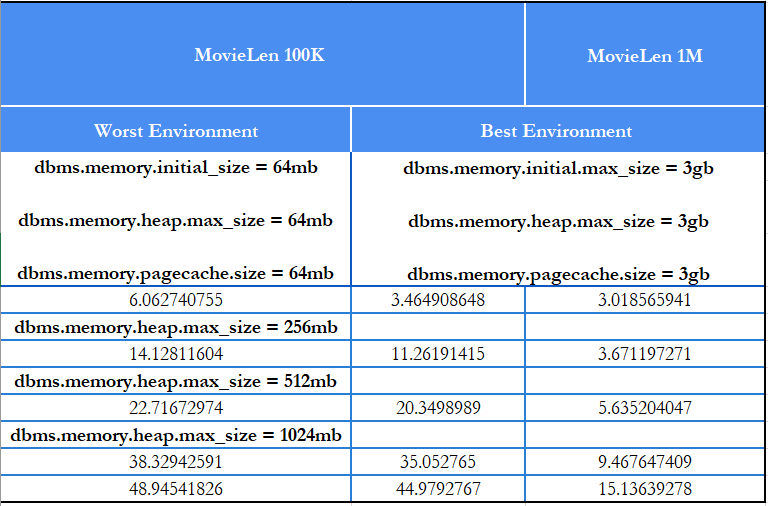
The next criteria are time testing:

**Table 4.2. The Time between 100K and 1M MovieLen dataset in Matrix**

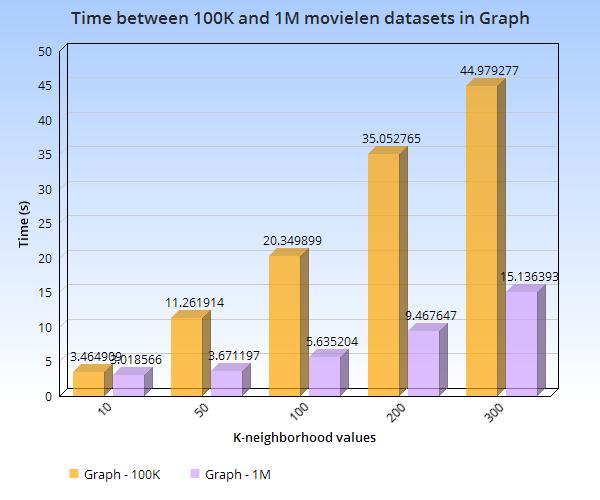
|  |  |  |
| --- | --- | --- |
| **K neighbor** | **Time** | |
| **MovieLen 100K** | **MovieLen 1M** |
| 10 | 1.646053839 | 6.743942642 |
| 50 | 1.587010145 | 6.65118261 |
| 100 | 1.699946295 | 6.682378769 |
| 200 | 1.814186573 | 6.760316515 |
| 300 | 1.704050207 | 6.733996997 |

****

**Figure 4.2. Chart of The Time between 100K and 1M MovieLen dataset in Matrix**

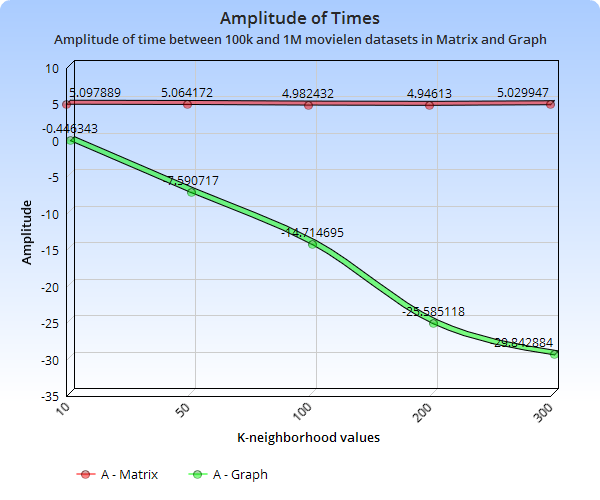
****

**Table 4.3. Values the Time between 100K and 1M movielen dataset in Graph**

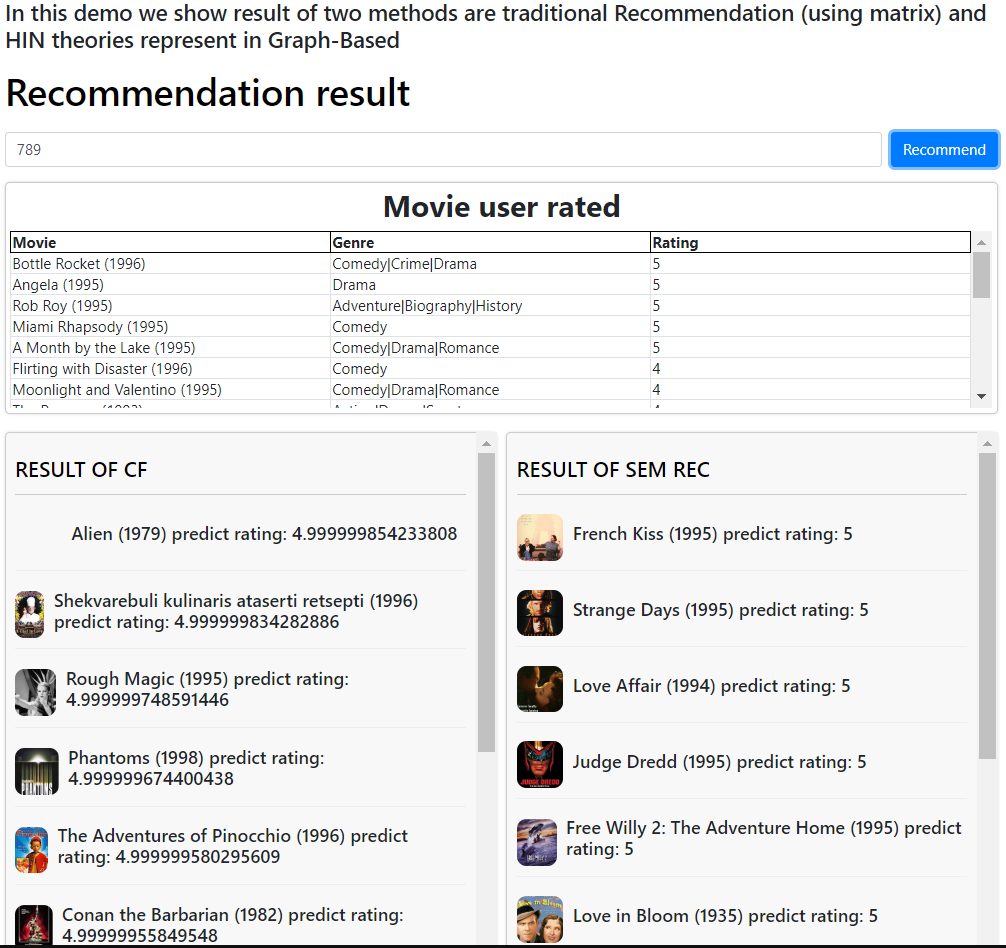


**Figure 4.3. Chart of the Time between 100K and 1M movielen dataset in Graph**

Figure below is Amplitude of time between 100K and 1M movielen datasets in Matrix and Graph.

****  
 As we can see, the different datasets are the differently recommended time. It usually depends on the number of ratings, the sparsity of the rating values, e.g. After the experiment on two above datasets, the recommended time of the graph-based RS is always faster than the traditional RS   
   
  
  
 After completing the above experiments, to observe the results of the two methods, Let's see the demo below:

**Figure 4.4. Amplitude of Time between 100k and 1M MovieLen datasets in Matrix and Graph**

****

**Figure 4.5. Recommendation result of two system**

Enter userid of a user who needs to be recommended movies

Movies were rated by users 789

## **Chapter 5: Conclusion and Future Development**



## **Conclusion**

Recommendation systems open new opportunities for retrieving personalized information on the Internet. It also helps to ease the problem of information overload which is a very common phenomenon with information retrieval systems and enables users to have access to products and services which are unavailable to users on the system.

This paper discussed the comparison of the two similarity search techniques and highlighted their strengths and challenges to step by step to improve their performances. Through the experiments, we can see that the Graph-based similarity search has the dominance about recommended time and consumes very little memory. But about the accuracy in prediction clearly is the graph-based recommendation system with single path is lower than the traditional recommendation system. But in this thesis, we only focus on single path (simple meta-path) of Weighted Heterogeneous Information System, that is the reason why the accuracy is affected.

In industry 4.0, Big Data is one of the outstanding technologies that all countries, including Vietnam, are aiming for. I predict that the traditional recommendation system will be replaced by the graph-based recommendation system in the future, because They can handle with big data superbly.

## **Future Works**

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. the experiments with real data are usually very time to consume, requiring even days to finish a once in running). Future work concerns a deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity.

This thesis has been mainly focused on the comparison between the traditional recommendation system and the graph-based recommendation system, and to clarify that whether or not the graph-based recommendation system can replace the past recommendation system? We must test the following ideas and generate the results from those:

We apply meta-path based similarities to the graph-based recommendation system. The different meta-paths or meta-graphs result in different similarities. Based on meta-path, several approaches have attempted to tackle the

1. Recommendation task based on HIN. In [39], meta-path based similarities are used as regularization terms in the matrix factorization framework. In [40], multiple meta-paths are used to learn user and item latent features, which are then used to recover similarity matrices combined by a weighted mechanism. In [29], users’ ratings to items are used to build a weighted HIN, based on which meta-path based methods are used to measure the similarities of users for recommendation. The combination of different meta-paths are explicit, using the similarities instead of latent features.
2. We can apply D-graph to build a distributed graph database which is a hard challenge. D-graph was designed keeping the problems of a fast-growing company in mind. It not only shards the data in a way where it can execute queries (joins, traversals, filters, sorts, paginations) without this universal view, it also minimizes the number of network calls and disk seeks required to execute these queries [35]. Both of these factors become prominent in datasets spanning hundreds of servers. Not only that, as you add more servers, data automatically gets shaded and moved to fill the new ones. D-graph automatically rebalances shards among servers to ensure that load is evenly distributed. While being distributed, D-graph provides lock-free ACID transactions with snapshot isolation, designed for low latency. Transactions make it a lot easier for application developers to think through database behavior, removing any data integrity issues [35].

In summary, the graph-based recommendation systems are being noticed by researchers from everywhere in the world and they will dominate with the advantages such as quickly recommended time and speed, less memory consumption, and scalability.

**REFERENCES**

[1] Netflix. Netflix recommendations: Beyond the 5 stars, 2012, [Online]. Available:

<http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

[2] Su Mon Kywe, Ee-Peng Lim, and Feida Zhu, "A survey of recommender systems in Twitter. (2012)." Social Informatics: 4th International Conference, SocInfo 2012, Lausanne, Switzerland, December 5-7, 2012: Proceedings. 7710, 420-433. Research Collection School of Information Systems.

[3] Poonam B.thorat, R. Goudar, and Sunita Barve, "Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System," International Journal of Computer Applications, 2015.

[4] F.O Isinkaye, Y.O. Folajimi, and B.A Ojokoh, "Recommendation systems: Principles, methods and evaluation", Egyptian Informatics Journal(2015) 16, 261-273.

[5] Atisha Sachan and Vineet Richariya, "A Survey on Recommender Systems based on Collaborative Filtering Technique", International Journal of Innovations in Engineering and Technology (IJIET), 2 April 2013.

[6] Zhongqi Lu, Zhicheng Dou, Jianxun Lian, and Xing Xie and Qiang Yang, "Content-Based Collaborative Filtering for News Topic Recommendation," Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence.

[7] Geetha G1, Safa, Fancy, and Saranya, "A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System," National Conference on Mathematical Techniques and its Applications (NCMTA 18), 2018.

[8] Leidy Esperanza MOLINA, and FERNÁNDEZ, "Recommendation System for Netflix," VRIJE UNIVERSITEIT AMSTERDAM, January 29, 2018.

[9] Reena Pagare, and Shalmali A. Patil, "Study of Collaborative Filtering Recommendation Algorithm - Scalability Issue," International Journal of Computer Applications (0975 - 8887), Volume 67 - No. 25, April 2013.

[10] Y. Sun, Y. Yu, and J. Han, “Ranking-based clustering of heterogeneous information networks with star network schema,” in KDD, 2009, pp. 797–806.

[11] Y. Sun and J. Han, “Mining heterogeneous information networks: a structural analysis approach,” SIGKDD Explorations, vol. 14, no. 2, pp. 20–28, 2012.

[12] Yizhou Sun, Xiang Ren, and Hongzhi Yin, "CONTEXT-RICH RECOMMENDATION: INTEGRATING LINKS, TEXT, AND SPATIOTEMPORAL DIMENSIONS", August 11, 2017.

[13] PETER BOSTRÖM, and MELKER FILIPSSON, "Comparison of User Based and Item Based Collaborative Filtering Recommendation Services," EXAMENSARBETE INOM TEKNIK, GRUNDNIVÅ, 15 HP STOCKHOLM, SVERIGE 2017.

[14] "Recommender systems", Computer Science Comprehensive Exercise, Carleton College in Northfield, MN, [Online]. Available:

<http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/memorybased.html>

[15] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S. Yu, and Xiao Yu,"Integrating Meta-Path Selection with User-Guided Object Clustering in Heterogeneous Information Networks," Volume 7 Issue 3, September 2013.

[16] YiBo Chen, ChanLe Wu, Ming Xie and Xiaojun Guo, "Solving the Sparsity Problem in Recommender Systems Using Association Retrieval," JOURNAL OF COMPUTERS, VOL. 6, NO. 9, SEPTEMBER 2011.

[17] Sivaramakrishnan N, Subramaniyaswamy V\*, Arunkumar S, Renugadevi A, and Ashikamai KK, "NEIGHBORHOOD-BASED APPROACH OF COLLABORATIVE FILTERING TECHNIQUES FOR BOOK RECOMMENDATION SYSTEM," International Journal of Pure and Applied Mathematics Volume 119 No. 12 2018.

[18] Kamille Nixon, "Sustainable Competitive Advantage: Creating Business Value through Data Relationships," The World’s Leading Graph Database, White Paper.

[19] Julien Gaillard, “Recommender Systems: Dynamic Adaptation and Argumentation”, Thesis, The University of Avignon, 2014.

[20] Jure Leskovec, Anand Rajaraman, Jeffrey D.Ullman, “Mining of Massive Datasets – Chapter 9. Recommendation System”, pp. 307 – 341, 2014.

[21] Dennis E. Bachmann, “Contextual Model-Based Collaborative Filtering for Recommender Systems”, Thesis, The University of Western Ontario, April 2017.

[22] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734–749, 2005.

[23] Yunkyoung Lee, “Recommendation systems using collaborative filtering”, Thesis, San Jose State University, 2015.

[24] “Neighbor-Based Collaborating Filtering”, [Online]. Available:

<https://machinelearningcoban.com/2017/05/24/collaborativefiltering/>

[25] Gong, Songjie, “A Collaborative Filtering Recommendation Algorithm

Based on User Clustering and Item Clustering”, 2010.

[26] Wikipedia (2015), “Cosine Similarity”. [Online]. Available:

<https://en.wikipedia.org/wiki/Cosine_similarity>

[27] Emmanouil G. Vozalis and Konstantinos G. Margaritis, “Recommender Systems: an Experimental Comparison of two Filtering Algorithms”, July 2008.

[28] Christian Desrosiers and George Karypis, “A comprehensive survey of neighborhood-based recommendation methods”, 2011.

[29] Dilek Tapucu, Seda Kasap and Fatih Tekbacak, “Performance Comparison of Combined Collaborative Filtering Algorithms for Recommender Systems”, 2012.

[30] Rasna R.Walia, “Collaborative Filtering: A Comparison of Graph-Based Semi-Supervised Learning Methods and Memory-Based Methods”, 2014.

[31] Taras Hnot, Tetiana Gladkikh (Oct 25, 2016.), “Recommender Systems Comparison: The Best Performing Algorithm”, [Online]. Available: <https://www.softserveinc.com/en-us/blogs/recommender-comparison-algorithm/>

[32] Greg Linden, Brent Smith, and Jeremy York, "Amazon.com Recommendations (Item-to-Item Collaborative Filtering)", January, February, 2003.

[33] Chenguang Wang, Yangqiu Song, Haoran Li, Ming Zhang and Jiawei Han "KnowSim: A Document Similarity Measure on Structured Heterogeneous Information Networks", School of EECS, Peking University, Department of Computer Science, University of Illinois at Urbana-Champaign, 2012.

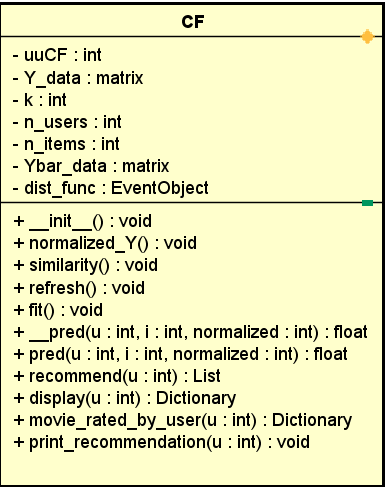
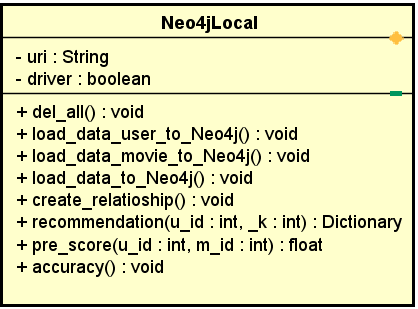
[34] Wikipedia, “Pearson correlation coefficient”, [Online]. Available: <https://en.wikipedia.org/wiki/Pearson_correlation_coefficient>

[35] “Releasing Dgraph 1.0: Production – Ready Graph Database”, [Online]. Available: <https://blog.dgraph.io/post/releasing-v1.0/>

**APPENDIXES**

**Recommendation System Implement**

In this implement, we have two main class which are CF and Neo4jLocal. CF class contain methods to recommend items for users based on traditional Collaborative

Filtering (using Cosine similarity) and Neo4jLocal class contain methods to recommend through HIN theories (PathSim similarity). The figure below indicated in depth of these classes:

Let begin with the CF class first. These table below explain the role of variables or fields, properties and methods:

**Variable of CF Class**

|  |  |
| --- | --- |
| Variable (Field) | Description |
| uuCF | Considering UUCF or IBCF (UBCF is using here). |
| Y\_data | Utility matrix. |
| Ybar\_data | Matrix copy of Y\_data. |
| dist\_funct | Similarity function (Cosine is using here). |
| k | Nearest neighbor. |
| n\_users | Number of users. |
| n\_items | Number of items (movies). |

**Method of CF Class**

|  |  |
| --- | --- |
| Method | Description |
| \_\_init\_\_() | Constructor. |
| normalized\_Y() | Normalize Utility Matrix. |
| similarity() | Similarity function. |
| refresh() | Normalize data and calculate similarity matrix again. |
| fit() | Call refresh() function. |
| \_\_pred() | Predict the rating of user u for item i (if you need the non). |
| pred() | Predict the rating of user u for item i (if you need the un). |
| recommend() | Determine all items should be recommended for user u. |
| print\_recommendation() | Print results of recommend() function. |
| display() | Return results of recommend() function and related information of user which get from database. |
| movie\_rated\_by\_user() | Return all movie rated by user u. |
| Beside all methods of class, we have 2 static methods to connect with database. | |
| create\_connection(db\_file) | Connect to database. |
| input\_data() | Get data from database. |

**Implementing code of CF class functions**

|  |
| --- |
| class CF(object):  def \_\_init\_\_(self, Y\_data, k, dist\_func = cosine\_similarity, uuCF = 1):  self.uuCF = uuCF  self.Y\_data = Y\_data if uuCF else Y\_data[:, [1, 0, 2]]  self.k = k  self.dist\_func = dist\_func  self.Ybar\_data = None  # number of users and items. Remember to add 1 since id starts from 0  self.n\_users = int(np.max(self.Y\_data[:, 0])) + 1  self.n\_items = int(np.max(self.Y\_data[:, 1])) + 1  def normalize\_Y(self):  users = self.Y\_data[:, 0]  self.Ybar\_data = self.Y\_data.copy()  self.mu = np.zeros((self.n\_users, ))  for n in range(self.n\_users):  ids = np.where(users == n)[0].astype(np.int32  ratings = self.Y\_data[ids, 2]  m = np.mean(ratings)  if np.mean(ratings):  m = 0  self.mu[n] = m  #normalize  self.Ybar\_data[ids, 2] = ratings - self.mu[n]  self.Ybar = sparse.coo\_matrix((self.Ybar\_data[:, 2], (self.Ybar\_data[:, 1], self.Ybar\_data[:, 0])), (self.n\_items, self.n\_users))  self.Ybar = self.Ybar.tocsr()  def similarity(self):  self.S = self.dist\_func(self.Ybar.T, self.Ybar.T)    def refresh(self):  """  Normalize data and calculate similarity matrix again  (after some few ratings added)  """  self.normalize\_Y()  self.similarity()    def fit(self):  self.refresh()    def \_\_pred(self, u, i, normalized = 1):  """  predict the rating of user u for item i (normalized)  if you need the non  """  # Step 1: find all users who rated i  ids = np.where(self.Y\_data[:, 1] == i)[0].astype(np.int32)  users\_rated\_i = (self.Y\_data[ids, 0]).astype(np.int32)  # Step 3: find similarity btw the current user and others  sim = self.S[u, users\_rated\_i] # dùng cosin\_similarity  # Step 4: find the k most similarity users  a = np.argsort(sim)[-self.k:]  # and the corresponding similarity levels  nearest\_s = sim[a]  # How did each of 'near' users rated item i  r = self.Ybar[i, users\_rated\_i[a]]  if normalized:  # caculate predict rating  # add a small number, for instance, 1e-8, to avoid dividing by 0  return (r\*nearest\_s)[0]/(np.abs(nearest\_s).sum() + 1e-8)  return (r\*nearest\_s)[0]/(np.abs(nearest\_s).sum() + 1e-8) + self.mu[u]  #pred theo u-uCF or i-iCF  def pred(self, u, i, normalized = 1):  """  predict the rating of user u for item i (normalized)  if you need the un  """  if self.uuCF: # (true == 1) u-u CF  return self.\_\_pred(u, i, normalized)  return self.\_\_pred(i, u, normalized)  def recommend\_descending\_by\_rating(self, u):  ids = np.where(self.Y\_data[:, 0] == u)[0]  items\_rated\_by\_u = self.Y\_data[ids, 1].tolist()  recommended\_items = []  for i in range(self.n\_items):  if i not in items\_rated\_by\_u:  rating = self.\_\_pred(u, i)  if rating > 0:  recommended\_items.append([i, rating  rs\_items = sorted(recommended\_items, key=lambda x : x[1], reverse = True)  return rs\_items[:10]  def display(self, uid):  conn = create\_connection("\\CF.db")  rated\_movie = []  with conn:  for movieid, rating in self.recommend\_descending\_by\_rating(uid):  tmp = pd.read\_sql\_query("select movietitle, posterurl from movie where movieid = {}".format(movieid), conn)  rated\_movie.append([tmp['movietitle'][0], rating, tmp['posterurl'][0]])  result = pd.DataFrame(rated\_movie, columns=['movietitle', 'p\_rating', 'posterurl'])  conn.close()  return result.T.to\_dict().values()  def movie\_rated\_by\_user(self, uid):  conn = create\_connection("\\CF.db")  with conn:  rated\_movie = pd.read\_sql\_query("""select m.movietitle, m.genre, u.rating  from movie as m, ua\_base as u  where u.movieid = m.movieid and u.userid = {}  order by u.rating desc """.format(uid), conn)  conn.close()  return rated\_movie.T.to\_dict().values()    def print\_recommendation\_chosen\_user\_descending\_by\_rating(self, userid):  print("Recommendation: ")  recommended\_items = self.recommend\_descending\_by\_rating(userid)  if self.uuCF:  print(' Recommend item(s): ')  print(' [movieid, rating]')  for rs\_items in recommended\_items:  print (rs\_items)  print(' for user ', userid)  def create\_connection(db\_file):  try:  path = os.path.dirname(\_\_file\_\_) + db\_file  conn = lite.connect(path)  return conn  except lite.Error as e:  print("Error %s:" %e.args[0])  return None    def input\_data():  conn = create\_connection("\\CF.db")  with conn:  ratings\_base = pd.read\_sql\_query("select \* from ua\_base", conn)  ratings\_test = pd.read\_sql\_query("select \* from ua\_test", conn)  rate\_train = ratings\_base.as\_matrix()  rate\_test = ratings\_test.as\_matrix()  return rate\_train, rate\_test |

In Neo4jLocal class, we can describe each element as fllows:

**Variable of Neo4jLocal Class**

|  |  |
| --- | --- |
| Variable | Description |
| uri | Port url. |
| driver | Connect to neo4j driver. |

**Method of Neo4jLocal Class**

|  |  |
| --- | --- |
| Method | Description |
| dell\_all() | Delete all data from database. |
| load\_data\_user\_to\_Neo4j() | Load user information to Neo4j. |
| load\_data\_movie\_to\_Neo4j() | Load movie information to Neo4j. |
| load\_data\_to\_Neo4j() | Load rating data to Neo4j. |
| create\_relationship() | Create relationship between user, movie and data. |
| pre\_score() | Predict the rating of user u for item i. |
| recommendation() | Determine all items should be recommended for user u. |
| accuracy() | Calculate accuracy. |

**Implementing code of Neo4jLocal class functions**

|  |
| --- |
| class Neo4jLocal(object):  def \_\_init\_\_(self, uri):  self.uri = uri  self.driver = GraphDatabase.driver(uri, auth = basic\_auth("neo4j", "1"))  def del\_all(self):  with self.driver.session() as session:  session.run("MATCH (n) DETACH DELETE n")  print("Has been deleted successfully")  session.close()  def load\_data\_user\_to\_Neo4j(self):  query = """  USING PERIODIC COMMIT  LOAD CSV WITH HEADERS FROM 'file:///user.csv' AS line FIELDTERMINATOR ','  MERGE (u:User {userid: toInteger(line.userid)})  ON CREATE SET  u.username = line.username,  u.age = toInteger(line.age),  u.gender = line.gender,  u.occupation = line.occupation  SET u.name = u.userid  """  with self.driver.session() as session:  session.run(query)  session.run("CREATE INDEX ON :User(userid)")  print("added user successfully")  session.close()  def load\_data\_movie\_to\_Neo4j(self):  query = """  USING PERIODIC COMMIT  LOAD CSV WITH HEADERS FROM 'file:///movie.csv' AS line FIELDTERMINATOR ','  MERGE (m:Movie {movieid: toInteger(line.movieid)})  ON CREATE SET  m.movietitle = line.movietitle,  m.genre = line.genre,  m.posterurl = line.posterurl  SET m.name = line.movieid  """  with self.driver.session() as session:  session.run(query)  session.run("CREATE INDEX ON :Movie(movieid)")  print("Added movie successfully")  session.close()  def load\_data\_to\_Neo4j(self):  query = """  USING PERIODIC COMMIT  LOAD CSV WITH HEADERS FROM 'file:///ua\_base.csv' AS line FIELDTERMINATOR ','  CREATE (d:Data { userid: toInteger(line.userid), movieid: toInteger(line.movieid), rating: toInteger(line.rating)})  SET d.name = line.userid  """  with self.driver.session() as session:  session.run(query)  session.run("CREATE INDEX ON :Data(userid,movieid)")  print("Added data successfully")  session.close()  def create\_relatioship(self):  query = """  MATCH (u:User),(m:Movie),(d:Data)  WHERE d.userid = u.userid AND d.movieid = m.movieid  MERGE (u)-[r:RATED{rating: d.rating}]->(m)  """  with self.driver.session() as session:  session.run(query)  print("Created relationship successfully")  session.close()  def recommendation(self, u\_id, \_k = 15):  query = """  MATCH (u1:User)-[x:RATED]->(m:Movie)<-[y:RATED]-(u2:User)  WHERE u1 <> u2 AND u1.userid = {uid}  WITH u1, u2, COUNT(m) AS numbermovies, SUM(x.rating \* y.rating) AS xyDotProduct  MATCH (u1:User)-[x:RATED]->(m:Movie)  WITH u1, u2, numbermovies, xyDotProduct,  REDUCE(xDot = 0.0, a IN COLLECT(x.rating) | xDot + a^2) AS xLength  MATCH (u2:User)-[y:RATED]->(m:Movie)  WITH u1, u2, numbermovies, xyDotProduct, xLength,  REDUCE(yDot = 0.0, b IN COLLECT(y.rating) | yDot + b^2) AS yLength  WHERE numbermovies > 0  WITH u1, u2, 2\*xyDotProduct / (xLength + yLength) AS pathsim  ORDER BY pathsim DESC  WITH u1, COLLECT([u2.userid, pathsim])[0..{k}] AS neighbours  UNWIND neighbours AS neighbour  WITH u1, neighbour  MATCH (h:User)-[r:RATED]->(m:Movie)  WHERE h.userid IN neighbour AND NOT EXISTS ((u1:User)-[:RATED]->(m:Movie))  WITH DISTINCT COLLECT([m.movieid, m.movietitle, m.posterurl]) AS recom, h // add more object if want to get  UNWIND recom AS recoms  WITH h, recoms  MATCH (p2:User)-[a:RATED]->(m:Movie)<-[b:RATED]-(h:User)  WHERE p2.userid = {uid}  WITH h, p2, recoms ,COUNT(m) AS numbermovies, SUM(a.rating \* b.rating) AS abDotProduct  MATCH (h:User)-[b:RATED]->(m:Movie)  WITH h, p2, recoms, numbermovies, abDotProduct,  REDUCE(bDot = 0.0, c IN COLLECT(b.rating) | bDot + c^2) AS bLength  MATCH (p2:User)-[a:RATED]->(m:Movie)  WITH h, p2, recoms, numbermovies, abDotProduct, bLength,  REDUCE(aDot = 0.0, d IN COLLECT(a.rating) | aDot + d^2) AS aLength  WHERE numbermovies > 0  WITH h, p2, recoms, 2\*abDotProduct / (bLength + aLength) AS newpathsim  MATCH(h)-[z:RATED]->(j:Movie)  WHERE j.movieid in recoms  RETURN j.movietitle as movietitle, j.posterurl as posterurl, SUM(z.rating\*newpathsim)/SUM(newpathsim) as score\_  ORDER BY score\_ DESC  LIMIT 10  """  with self.driver.session() as session:  result = session.run(query, uid = u\_id, k = \_k)  session.close()  record = []  for r in result:  record.append([r['movietitle'], r['posterurl'], r['score\_']])  result = result = pd.DataFrame(record, columns=['movietitle', 'posterurl', 'p\_rating'])  return result.T.to\_dict().values()  def pre\_score(self, u\_id, m\_id):  query = """  MATCH (u1:User)-[x:RATED]->(m:Movie)<-[y:RATED]-(u2:User)  WHERE u1 <> u2 AND u1.userid = {uid}  WITH u1, u2, COUNT(m) AS numbermovies, SUM(x.rating \* y.rating) AS xyDotProduct  MATCH (u1:User)-[x:RATED]->(m:Movie)  WITH u1, u2, numbermovies, xyDotProduct,  REDUCE(xDot = 0.0, a IN COLLECT(x.rating) | xDot + a^2) AS xLength  MATCH (u2:User)-[y:RATED]->(m:Movie)  WITH u1, u2, numbermovies, xyDotProduct, xLength,  REDUCE(yDot = 0.0, b IN COLLECT(y.rating) | yDot + b^2) AS yLength  WHERE numbermovies > 0  WITH u1, u2, 2\*xyDotProduct / (xLength + yLength) AS pathsim  ORDER BY pathsim DESC  WITH u1, COLLECT([u2.userid, pathsim])[0..20] AS neighbours  UNWIND neighbours AS neighbour  WITH u1, neighbour  MATCH (h:User)-[r:RATED]->(m:Movie)  WHERE h.userid IN neighbour AND NOT EXISTS ((u1:User)-[:RATED]->(m:Movie)) AND m.movieid = {mid}  WITH h  MATCH (p2:User)-[a:RATED]->(m:Movie)<-[b:RATED]-(h:User)  WHERE p2.userid = {uid}  WITH h, p2 ,COUNT(m) AS numbermovies, SUM(a.rating \* b.rating) AS abDotProduct  MATCH (h:User)-[b:RATED]->(m:Movie)  WITH h, p2, numbermovies, abDotProduct,  REDUCE(bDot = 0.0, c IN COLLECT(b.rating) | bDot + c^2) AS bLength  MATCH (p2:User)-[a:RATED]->(m:Movie)  WITH h, p2, numbermovies, abDotProduct, bLength,  REDUCE(aDot = 0.0, d IN COLLECT(a.rating) | aDot + d^2) AS aLength  WHERE numbermovies > 0  WITH h, p2, 2\*abDotProduct / (bLength + aLength) AS newpathsim  MATCH(h)-[z:RATED]->(j:Movie)  WHERE j.movieid = {mid}  RETURN SUM(z.rating\*newpathsim)/(SUM(newpathsim) + 1e-8) as score\_  """  result = 0  with self.driver.session() as session:  for record in session.run(query, uid = int(u\_id), mid = int(m\_id)):  result = record["score\_"]  session.close()  return result  def accuracy(self):  r\_cols = ['userid', 'movieid', 'rating']  ratings\_test = pd.read\_csv('ua\_test.csv', sep=',', header=0, names=r\_cols, encoding='latin-1')  rate\_test = ratings\_test.as\_matrix()  n\_tests = rate\_test.shape[0]    SE = 0  AE = 0  for n in range(n\_tests):  pred = self.pre\_score(rate\_test[n, 0], rate\_test[n, 1])  SE += (pred - rate\_test[n, 2])\*\*2  AE += np.abs(pred - rate\_test[n, 2])  RMSE = np.sqrt(SE/n\_tests)  MAE = AE/n\_tests  print ("\n RMSE = %f" % (RMSE))  print ("\n MAE = %f" % (MAE)) |