MACAU UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Computer Science and Engineering

Faculty of Innovation Engineering

Senior Thesis for the Degree of Bachelor of Science

Title: A Knowledge Graph Enhanced Matrix Factorization Movie Recommendation System

Student Name : Li Yiran

Student No. : 1909853D-I011-0156

Supervisor : Wu Yang

May, 2023

澳門科技大學

創新工程學院

計算機科學與工程學院

理學學士學位畢業論文

論文題目：基於知識圖譜的矩陣分解電影推薦系統

姓　　名：李弈然

學　　號：1909853D-I011-0156

指導老師：吳泱

2023 年 5 月

Abstract

There are many kinds of personalized recommendations on today's OTT (Over-the-Top) platforms, such as Netflix and YouTube, which can better retain users and increase their time on the platform by randomly recommending content they may like based on their usual viewing preferences [1]. Recommender systems (RSs) are intelligent applications that assist users in their information search tasks by recommending items (products, information, etc.) that best fit their needs and preferences [2]. In order to improve the personalized recommendation function of RSs, this paper proposes a new personalized movie recommendation system that combines knowledge graph and matrix factorization by setting the implicit factor and generating tripartite graphs, and finally using SVD++ factorization to improve the recommendation effect and reduce the computational complexity [8]. The lab results show that this system improves the recommendation accuracy and real time performance.

Keywords: Knowledge Graph; Matrix Factorization; SVD++; Machine Learning

摘要

如今，在 Netflix 和 YouTube 這類 OTT（Over-the-Top）平台上有許多種個性化推薦，它可以根據用戶平時的觀看偏好隨機推薦他們可能喜歡的內容，從而更好地留住用戶，增加他們在平台上的停留時間[1]。而推薦系統（RSs）是一種智能應用，通過推薦最適合用戶需求和偏好的項目，協助用戶完成信息搜索任務[2]。但是目前遇到的問題有系統推薦耗時長，推薦結果不準確等，為了改良 RSs 的個性化推薦功能，本文提出了一種新型的個性化電影推薦系統，結合了知識圖譜和矩陣分解，通過設定隱式因子，並以次生成三分圖，最後使用 SVD++分解來提高推薦效果並降低計算複雜度[8]。實驗結果表明，本系統在推薦實時性方面有很大提升。

關鍵詞：知識圖譜；矩陣分解；SVD++；機器學習

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

With the increasing abundance of content in various applications, people are exposed to a large amount of content every day, which makes it very difficult to filter the information that is useful or interesting to them. For example, in today's Over-the-Top (OTT) platforms such as Netflix and YouTube, users can approach almost countless media content every day, which indicates the importance of personalized recommendations. Personalized recommendations can suggest random content that users may like based on their usual viewing preferences, leading to better retention of users and increasing their time spent on the platform[1].

Recommender systems (RSs) are intelligent applications which assist users in their information-seeking tasks, by suggesting the items (products, information, etc.) that best suit their needs and preference[2]. These systems analyze the users' taste and mood at that moment. Based on the result, they create a recommendation that particularly suits each user. Such recommendation systems can be used in many areas, such as recommending books, movies, music, and even in e-shopping, which is frequently used these days. The development of RSs stems from a rather simple observation: individuals often rely on advice provided by others to make everyday decisions. For instance, when choosing books to read, it is common to search for recommendations from peers first; employers rely on references before hiring someone; and when choosing clothes to buy on e-commerce platforms, people will read reviews and photos left by people who have already bought the clothes to determine whether the clothes are suitable for them[3].

All these examples are based on the fact that people will only consult the opinions of those who are somewhat similar to themselves, which brings us to one of the currently two most common types of RSs: collaborative filtering (CF). The technical basis of CF is the U-I matrix, based on which other information about the user, such as other information about the user's association with the project, is usually combined with the matrix while factorizing it to form a more complete recommendation scenario that yields more accurate results.

Another common approach is content-based filtering (CB), in which the algorithm attempts to recommend items similar to those that the user has liked in the past or is currently viewing. The content-based filtering approach is best suited for situations where known item data exists but no user data exists; it treats recommendations as a specific classification problem and learns the user's likes and dislikes based on the item's characteristics.

Therefore, in order to better improve the effectiveness of personalized recommendation, an optimized model combining CB and CF will be proposed in this thesis. After organizing the collected data into a knowledge graph, it will be transformed into a tripartite graph and further derived into two matrices with smaller dimensions describing user preferences and item classification, respectively, and then analyzed by the CF algorithm.

Chapter 2 Related Work

Research shows that during 1998 to 2014, more than 200 researcher articles about recommender systems were published, which reveals the fervent research and application of different recommender systems[4]. Among all the research, each recommender system has its own advantages.

However, during the research, it is found that more than half of the recommendation methods used CB (55%), and only 18% applied CF[4]. Meanwhile, this review also shows some shortcomings of the current study, but it is still difficult to choose a better method between these two recommendation methods. This is because sometimes content-based filtering performs better than collaborative filtering, but sometimes it performs worse, which leads to unclear results.

For CB, it can improve the relevance of the recommendation, in this at the same time can also accumulate the user's behavior data, forming a matrix to facilitate further analysis. However, CB can only produce relevance effect, and the result is not necessarily what users really like. CF, on the other hand, is a direct mining of users' preferences, with higher recommendation accuracy than CB and better results than CB, because on the basis of a certain database, it can analyze and predict users' previous behaviors by itself, and in the process, this model can also be trained spontaneously to achieve better results. At the same time, unlike CB, CF is the result of the analysis of users' preferences and makes use of the group wisdom of similar people, while CB simply recommends things to users that have a certain degree of similarity. But again, the disadvantage is obvious, if there is no sufficient amount of data to support, it is impossible to perform the matrix operation, thus not getting accurate results, which may lead to the final result being much different from the expected result.

2.1 Content-Based Filtering

CB has a recommendation system that recommends items similar to items they have liked before to a given user. Compared to CF, the principle of CB is a bit simpler and does not require a user behavior matrix, and thus eliminates the process of constant training of the recommendation model. In CB, a graph similar to a knowledge graph is generally used to assist in recommendation, where the user and the item are different nodes, and the value of their connected edges indicate the relationship between the nodes.

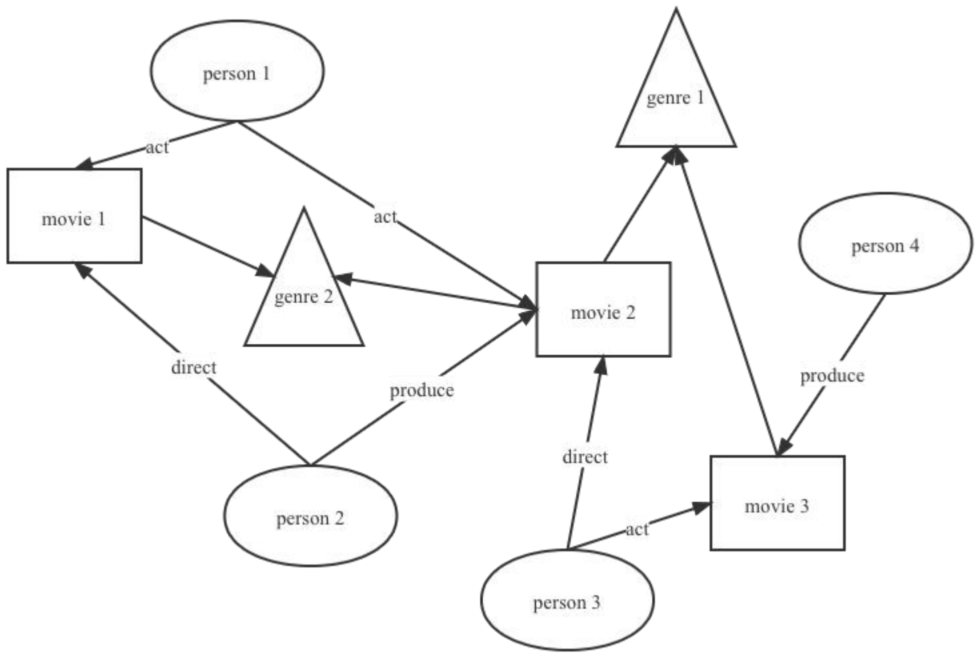


Figure 2-1 Knowledge Graph

2.1.1 *Basic Method*

In content-based recommendation methods, the utility $u(c,s)$ of item s for user c is estimated based on the known utility $u(c,s_i)$ of user c for item s_i that is similar to item s. One of the best-known measures for specifying keyword weights in information retrieval is the term frequency/inverse document frequency (TF-IDF) measure, which is defined as follows. Assume that N is the total number of documents that can be recommended to the user, and that keyword k_j appears in n_i

of them. In addition, assume that $f_{i,j}$ is the number of times keyword k_i appears in document d_j. Then, $TF_{i,j}$, the word frequency of keyword k_i in document d_j is defined as Equation (2-1).

$$TF_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}} \tag{2-1}$$

The measurement of the inverse document frequency (IDF_i) is usually used in combination with the simple term frequency ($TF_{i,j}$). The inverse document frequency of keyword k_i is usually defined as Equation (2-2).

$$IDF_i = \log \frac{N}{n_i} \tag{2-2}$$

Then, the TF-IDF weight of keyword k_i in document d_j is defined as Equation (2-3) and the content of document d_j is defined as Equation (2-4).

$$w_{i,j} = TF_{i,j} \times IDF_i \tag{2-3}$$

$$Content\,(d_j) = (w_{1,j}, \cdots, w_{i,j}) \tag{2-4}$$

Then let $ContentBasedProfile(c)$ serve as a profile for user c, containing the tastes and preferences of that user. That is, $u(c,s)$ can be defined as Equation (2-5)[9].

$$\begin{aligned} &u(c,s) \\ &= score\,(ContentBasedProfile(c), Content(s)) \end{aligned} \tag{2-5}$$

2.1.2 Optimized Methods

For more convenience, Zan Huang et al. collected certain information about customers and books in their study, and then modeled them based on this information, using the similarity weights obtained in the first stage to construct a two-layer graph, consisting of a book layer and a customer layer.

Each node in the book layer represents a book, and the weight of each edge between any two books represents the content similarity between them. Each node in the customer layer represents a customer, and the weight of the edge between customer nodes is the similarity between two customers. In addition to the inner links, there are also connected edges between these two layers. These links are based on the purchase history of all customers, where purchases are represented by edges between nodes in the book layer and nodes in the customer layer[10].

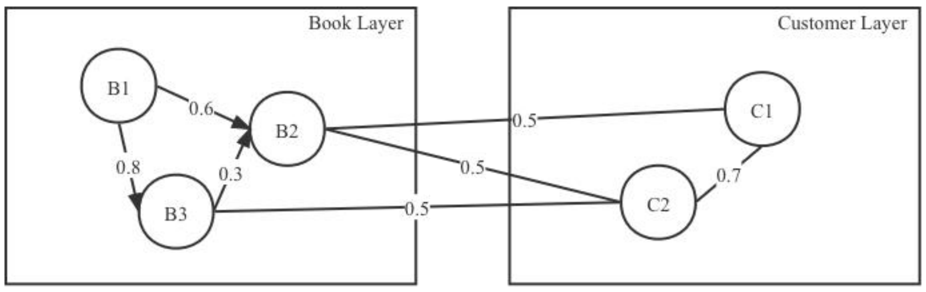
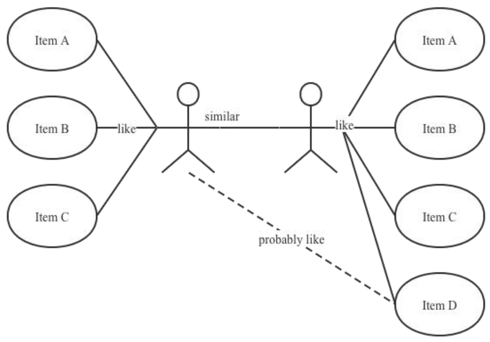


Figure 2-2 Two-Layer Graph Model of Books, Customers and Purchases

Based on this, Eduard Fritscher et al. further refined the recommendation algorithm by means of a graph, which traverses the graph structure starting from the initial node. In this case, the initial nodes represent the user's interests, which may be a movie or a movie genre that the user likes. The algorithm tries to find the node in the graph that is closest to all initial nodes and then returns it as a recommendation. To achieve the final goal, they designed and implemented four separate algorithms to perform the comparison[11].

2.2 Collaborative Filtering

It is further divided into user-based collaborative filtering and item-based collaborative filtering, which are, respectively, recommending items that several similar people may like by their liking of different items, and predicting several people's liking of the same item by their liking of another similar item.



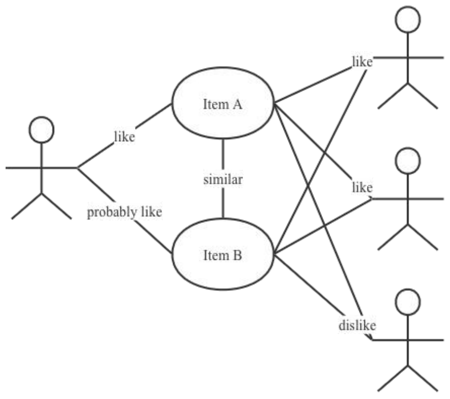


Figure 2-3 User-Based Collaborative Filtering Figure 2-4 Item-Based Collaborative Filtering

2.2.1 *Basic Method*

In this, the user's rating of an item is constructed as the following user behavior matrix, where an element of the matrix represents a user's rating of an item with increasing values starting from 0. The row vector represents the vector of all items rated by a user, and the column vector represents the vector of all items rated by a user. With the row vectors and column vectors, we can calculate the similarity between the users and the items:

$$\begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ R_{21} & R_{22} & \cdots & R_{2m} \\ \cdots & \cdots & R_{ij} & \cdots \\ R_{n1} & R_{n2} & \cdots & R_{nm} \end{pmatrix},$$

where the value of R_{ij} indicates the preference of user i for item j. If $R_{ij} > 0$, the user's preference can be expressed directly by rating, or by using a binary value indicating whether the user clicked on, or viewed, the item. Yet the known preferences of users for items are usually very limited, which makes the matrix R usually sparse. This is where it is possible to use $R_{ij} = ?$ to represent the case where user i's preference for item j is unknown.

For the user-based approach, R_{kj} can be used as the degree of liking of object j by user k, and the derived $sim(i,k)$ is the similarity between user u and k, from which the possible liking of object j by user i can be calculated as Equation (2-6), where Z_i is the set of K neighboring users of user i, C is a normalizing

$$\hat{R}_{ij} = \frac{1}{C} \sum_{k \in Z_i} sim(i,k) R_{kj} \qquad (2\text{-}6)$$

constant, and then the highly rated result is recommended to the user.

The item-based approach is basically similar to the one above, where R_{ik} is used as user i's liking of object k, and the resulting $sim(j,k)$ is the similarity between object j and k, from which the possible liking of object j by user i can be calculated as Equation (2-7) and then the highly rated result is recommended to the user[5].

$$\hat{R}_{ij} = \frac{1}{C} \sum_{k \in Z_i} sim(j,k) R_{ik}, \qquad (2\text{-}7)$$

2.2.2 *Matrix Factorization*

To solve the problem that sparse matrices cannot be recommended directly, matrix factorization is usually applied. It simulates various real-life situations and provides great flexibility for subsequent recommendations. Another advantage of matrix factorization is that it allows the merging of additional information. When explicit feedback (direct ratings, etc.) is not available, RSs can use implicit feedback to further infer user preferences, e.g., by indirectly reflecting opinions by observing user behavior, including purchase history, and browsing history. They usually indicate the presence or absence of events and are therefore usually represented by densely populated matrices[6].

After the original algorithm proposed by Simon Funk in his blog post[7], an algorithm called SVD has been developed to make better use of explicit and invisible feedbacks. Given m users, n items form the scoring matrix $R' \in R^{m\times n}$, then R can be decomposed by SVD into the form like Equation (2-8), where $U \in R^{k\times m}$ and $V \in R^{k\times n}$ are both orthogonal matrices and $S \in R^{k\times k}$ is a diagonal matrix. Each non-zero element of S is a singular value.

$$R' = U^T SV, \tag{2-8}$$

To reduce the dimensionality of R, the largest lf singular values in S are chosen to form a new diagonal matrix S_{lf}, and the corresponding rows and columns of U and V are taken out and they are represented as U_{lf} and V_{lf}. In this way, the reduced dimensional rating matrix is as Equation (2-9), where R'_{lf} is the predicted value of the user's rating of the item[8].

$$R'_{lf} = U^T{}_{lf} S_{lf} V_{lf}, \tag{2-9}$$

In addition to this, there are two other common methods. Given an equation $Ax = b$, where $A = [a_{ij}]$ is an $n \times n$ matrix, $x = \{x_i\}$ and $b = \{b_i\}$ are both vectors of size n. When A is symmetric, the Cholesky factorization can be used to obtain the result shown in Equation (2-10), where L is the lower triangular matrix.

$$A = LL^T, \tag{2-10}$$

This factorization method is called the triangular factorization method[12].

Alternatively, using the LU factorization, we obtain Equation (2-11), where L is the lower triangular matrix and U is the upper triangular matrix[14].

$$A = LU, \tag{2-11}$$

The other one is called QR factorization method, and the basic principle is given a matrix $M \in R^{m\times n}$ with $m > n$ first, then consider the QR factorizations of the form like Equation (2-12), where $Q \in R^{m\times m}$ is orthogonal, $A_k \in R^{k\times k}$ is upper triangular with non-negative diagonal elements, $B_k \in R^{k\times(n-k)}$, $C_k \in R^{(m-k)\times(n-k)}$, and $O_k \in R^{(m-k)\times k}$, which is zero matrix, and $\Pi \in R^{n\times n}$ is a permutation matrix chosen to reveal linear dependence among the columns of M. Usually, k is chosen to be the smallest integer $1 \le k \le n$ for which $\|C_k\|_2$ is sufficiently small[13].

$$M\Pi = QR \equiv Q\begin{pmatrix} A_k & B_k \\ O_k & C_k \end{pmatrix}, \tag{2-12}$$

Once the factorization is obtained, the difference between the actual score value and the predicted value is obtained, and the loss is calculated. In order to make the difference between the actual score and the predicted value as small as possible, this loss function should be minimized, which is further solved by applying gradient descent in machine learning. In this way, a trained model can be obtained, and the sparse matrix can be supplemented with predictions based on this model, thus improving the problem that the matrix is too sparse to make accurate recommendations.

Chapter 3 Design Idea

Considering the respective advantages of CF and CB, using them together can achieve better results. First, the dataset is organized into a knowledge graph, and then the graph structure is extracted by further structuring according to the category of each vertex. With the help of this structured graph, the huge matrix in the CF process can be better decomposed to carry out further subsequent steps of CF.

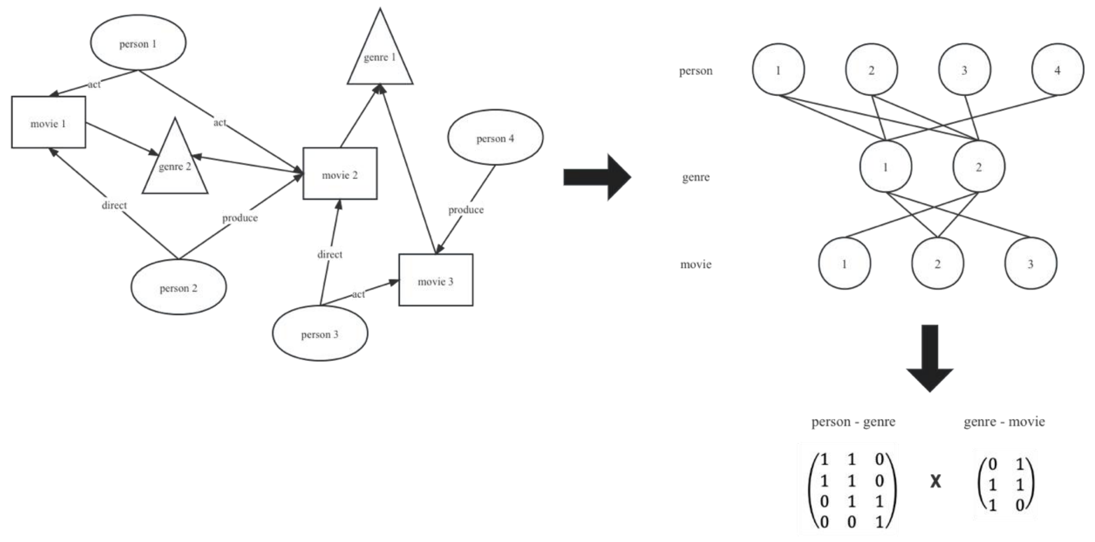


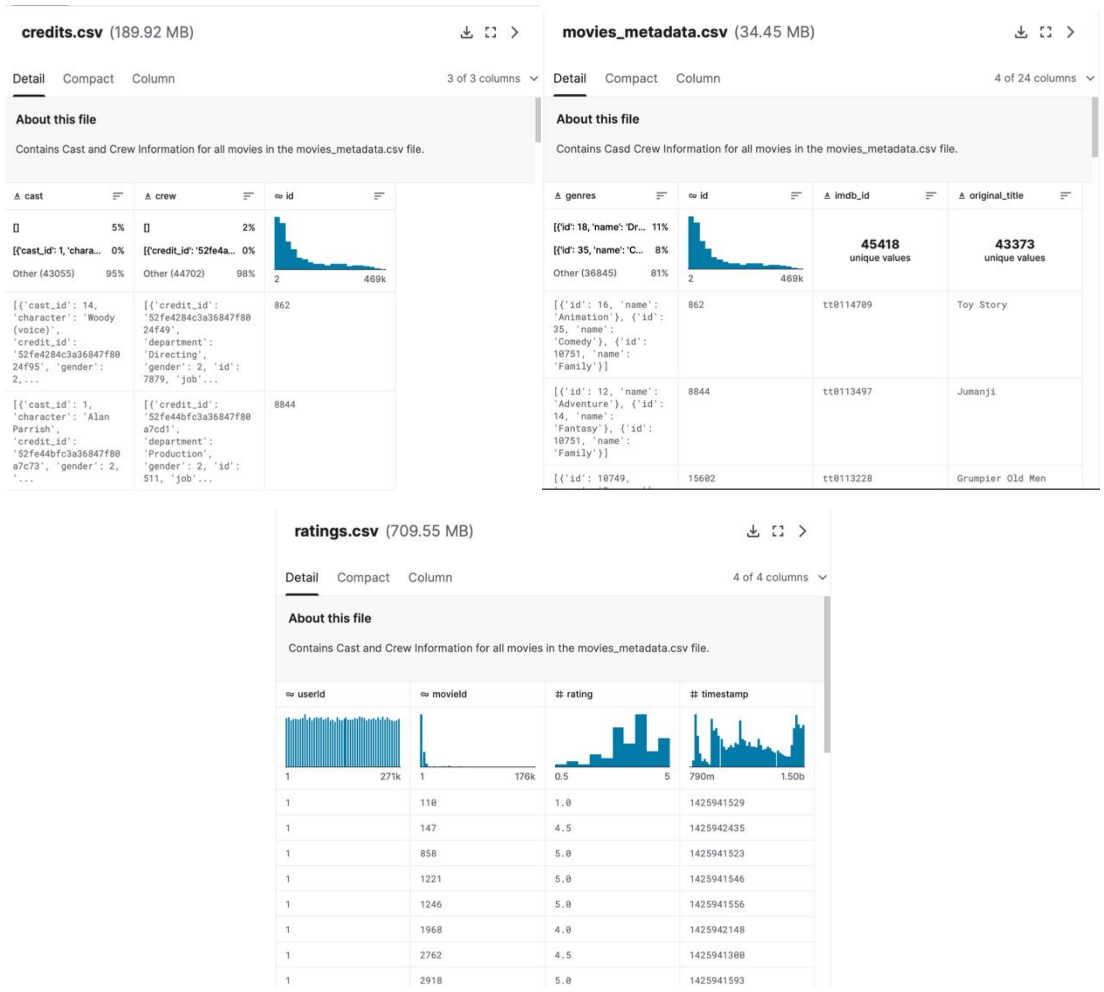
Figure 3-1 General Flow of The Model

3.1 Data Collection

In order to get more accurate recommendation results, I first needed a real and accurate movie dataset with a certain size, and after searching, a dataset named The Movies Dataset is chosen from Kaggle website, which was uploaded by Rounak Banik five years ago. In this dataset, there are six files which contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

Also, it has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

Due to the excessive information in this dataset, which most of them has little relation to the user's personality recommendation, only part of the data from the three files cresits.csv, movies_meatdata.csv and ratings.csv will be used in order to reduce the time lost in processing the data.



credits.csv (189.92 MB)

3 of 3 columns

About this file

Contains Cast and Crew Information for all movies in the movies_metadata.csv file.

cast	crew	id
[] 5% [{'cast_id': 1, 'chara... 0% Other (43055) 95%	[] 2% [{'credit_id': '52fe4a... 0% Other (44702) 98%	2 469k
[{'cast_id': 14, 'character': 'Woody (voice)', 'credit_id': '52fe4284c3a36847f8024f95', 'gender': 2,...	[{'credit_id': '52fe4284c3a36847f8024f49', 'department': 'Directing', 'gender': 2, 'id': 7879, 'job'...	862
[{'cast_id': 1, 'character': 'Alan Parrish', 'credit_id': '52fe44bfc3a36847f80a7c73', 'gender': 2, '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'department': 'Production', 'gender': 2, 'id': 511, 'job'...	8844

movies_metadata.csv (34.45 MB)

4 of 24 columns

About this file

Contains Casd Crew Information for all movies in the movies_metadata.csv file.

genres	id	imdb_id	original_title
[{'id': 18, 'name': 'Dr... 11% [{'id': 35, 'name': 'C... 8% Other (36845) 81%	2 469k	45418 unique values	43373 unique values
[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'}, {'id': 10751, 'name': 'Family'}]	862	tt0114709	Toy Story
[{'id': 12, 'name': 'Adventure'}, {'id': 14, 'name': 'Fantasy'}, {'id': 10751, 'name': 'Family'}]	8844	tt0113497	Jumanji
[{'id': 10749,	15602	tt0113228	Grumpier Old Men

ratings.csv (709.55 MB)

4 of 4 columns

About this file

Contains Cast and Crew Information for all movies in the movies_metadata.csv file.

userId	movieId	rating	timestamp
1 271k	1 176k	0.5 5	790m 1.50b
1	110	1.0	1425941529
1	147	4.5	1425942435
1	858	5.0	1425941523
1	1221	5.0	1425941546
1	1246	5.0	1425941556
1	1968	4.0	1425942148
1	2762	4.5	1425941300
1	2918	5.0	1425941593

Figure 3-2 Dataset from Kaggle

3.2 Generating knowledge graph

Due to the limited number of node types in the existing knowledge graph, they are often unable to meet the specific requirements of this experiment. To address this issue, I decided to create a new knowledge graph by leveraging Neo4j and the dataset mentioned above. This new knowledge graph will offer a more robust and detailed understanding of the complex relationships between various entities and concepts in the dataset. Additionally, it will enable us to take full advantage of the available data and generate more nuanced insights and predictions.

To develop this knowledge graph, I will first select a few columns in the dataset that can be used as the basis for the graph structure. Once imported, Neo4j's powerful graph database tools will then be used to populate the graph with nodes, edges and weights. This will involve identifying and extracting relevant data from the dataset and structuring it to facilitate integration into the knowledge graph.

3.2.1 Introduction of Neo4j

Neo4j is an open-source graph database management system that has gained immense popularity for its ability to efficiently store, manage, and query large amounts of interconnected data. Its unique graph database model is specifically designed to handle complex relationships between data points, outperforming traditional relational databases in terms of natural and intuitive querying. With Neo4j, users can easily analyze and traverse large and complex datasets with intricate relationships, making it an ideal choice for diverse applications such as social networks, e-commerce platforms, fraud detection, recommendation engines, and more. The system supports several query languages, including the specialized Cypher query language for graph queries, and multiple programming languages like Python, Java, and JavaScript, making it incredibly flexible and effortless to integrate within existing systems.

Its scalability is also a key feature, as it is built to handle massive amounts of data and thousands of users simultaneously, making it particularly suitable for complex enterprises and organizations that deal with big data. Additionally, Neo4j offers various tools and plugins that facilitate integration with popular data management and analytics platforms, such as Hadoop, Spark, and Tableau.

In summary, Neo4j is a top choice for individuals and organizations that need to handle interconnected and complex data sets due to its ability to model, store, and query vast amounts of data with ease and efficiency.

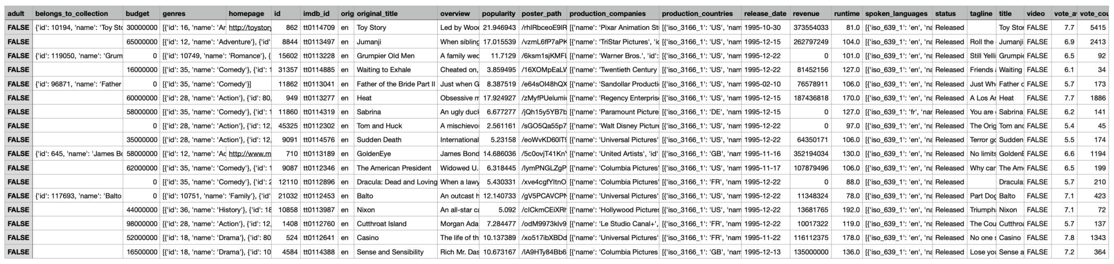
3.2.2 Data Preprocessing

Before importing the data into the knowledge graph, it is necessary to pre-process the data because of the large number of unwanted columns in the original dataset. In addition, each column in the dataset may also contain redundant information. Therefore, pre-processing of the dataset is crucial to ensure that only relevant information is included in the knowledge graph.

The preprocessing phase involves several steps in order to transform the data into a format suitable for importing into the knowledge graph. First, data cleaning is performed to remove any irrelevant or duplicate data. This involves identifying any missing values, outliers and inconsistencies and removing or replacing them as appropriate. Second, perform data conversion to normalize the data and ensure that it is recognized by Neo4j upon import. This may involve conversion of data types, normalization of data separators, etc. Finally, feature engineering is performed to extract useful features from the data set. This involves selecting relevant features, assessing their importance, and creating new features that may be more informative.

So, the pre-processing phase is a key step in the knowledge graph creation process, as it ensures that only relevant information is included, resulting in a more effective and efficient knowledge graph.

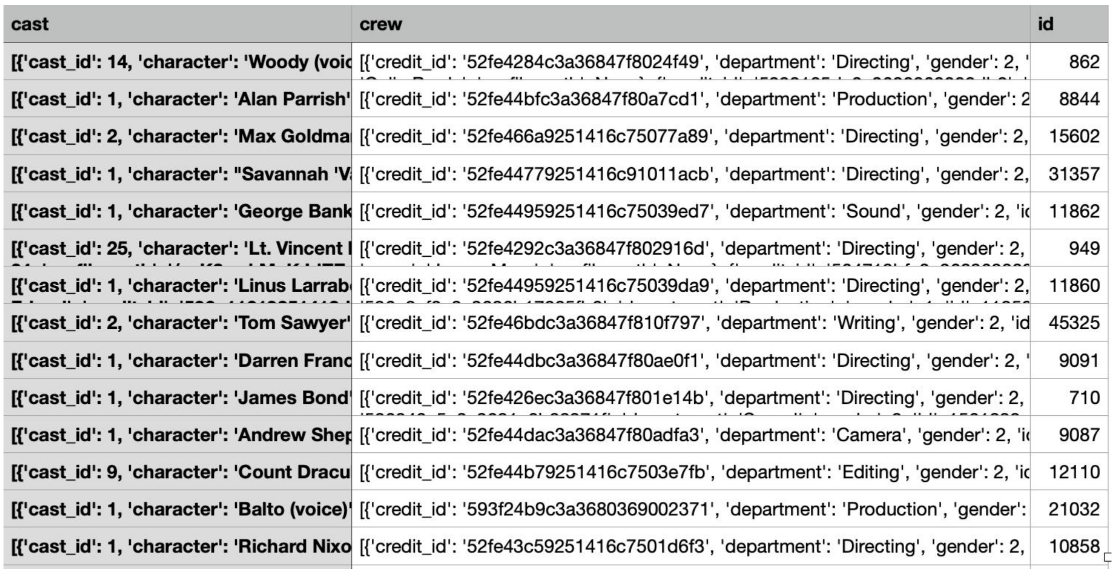
In movies_meatdata.csv, the id, original_title, and genres columns are finally reserved for getting the genres to which the movies belong and matching a unique id to the movie and genre respectively for subsequent processing.



adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	orig	original_title	overview	popularity	poster_path	production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	title	video	vote_a	vote_cou
FALSE	{'id': 10194, 'name': 'Toy Sto	30000000	[{'id': 16, 'name': 'Ar	http://toystory	862	tt0114709	en	Toy Story	Led by Wooc	21.946943	/rhIRbceoE9lR	[{'name': 'Pixar Animation St	[{'iso_3166_1': 'US', 'nam	1995-10-30	373554033	81.0	[{'iso_639_1': 'en', 'na	Released		Toy Stor	FALSE	7.7	5415
FALSE		65000000	[{'id': 12, 'name': 'Adventure'}, {'id'		8844	tt0113497	en	Jumanji	When sibling	17.015539	/vzmL6fP7aPK	[{'name': 'TriStar Pictures', 'ic	[{'iso_3166_1': 'US', 'nam	1995-12-15	262797249	104.0	[{'iso_639_1': 'en', 'na	Released	Roll the	Jumanji	FALSE	6.9	2413
FALSE	{'id': 119050, 'name': 'Grum	0	[{'id': 10749, 'name': 'Romance'}, {		15602	tt0113228	en	Grumpier Old Men	A family wec	11.7129	/6ksm1sjKMFL	[{'name': 'Warner Bros.', 'id':	[{'iso_3166_1': 'US', 'nam	1995-12-22	0	101.0	[{'iso_639_1': 'en', 'na	Released	Still Yelli	Grumpie	FALSE	6.5	92
FALSE		16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 1		31357	tt0114885	en	Waiting to Exhale	Cheated on,	3.859495	/16XOMpEaLW	[{'name': 'Twentieth Century	[{'iso_3166_1': 'US', 'nam	1995-12-22	81452156	127.0	[{'iso_639_1': 'en', 'na	Released	Friends a	Waiting	FALSE	6.1	34
FALSE	{'id': 96871, 'name': 'Father	0	[{'id': 35, 'name': 'Comedy'}]		11862	tt0113041	en	Father of the Bride Part II	Just when G	8.387519	/e64sOI48hQX	[{'name': 'Sandollar Producti	[{'iso_3166_1': 'US', 'nam	1995-02-10	76578911	106.0	[{'iso_639_1': 'en', 'na	Released	Just Wh	Father c	FALSE	5.7	173
FALSE		60000000	[{'id': 28, 'name': 'Action'}, {'id': 80,		949	tt0113277	en	Heat	Obsessive m	17.924927	/zMyfPUelumi	[{'name': 'Regency Enterpris	[{'iso_3166_1': 'US', 'nam	1995-12-15	187436818	170.0	[{'iso_639_1': 'en', 'na	Released	A Los Ar	Heat	FALSE	7.7	1886
FALSE		58000000	[{'id': 35, 'name': 'Comedy'}, {'id': 1		11860	tt0114319	en	Sabrina	An ugly duck	6.677277	/jQh15y5YB7b	[{'name': 'Paramount Picture	[{'iso_3166_1': 'DE', 'nam	1995-12-15	0	127.0	[{'iso_639_1': 'fr', 'nar	Released	You are	Sabrina	FALSE	6.2	141
FALSE		0	[{'id': 28, 'name': 'Action'}, {'id': 12,		45325	tt0112302	en	Tom and Huck	A mischievo	2.561161	/sGO5Qa55p7	[{'name': 'Walt Disney Picture	[{'iso_3166_1': 'US', 'nam	1995-12-22	0	97.0	[{'iso_639_1': 'en', 'na	Released	The Orig	Tom anc	FALSE	5.4	45
FALSE		35000000	[{'id': 28, 'name': 'Action'}, {'id': 12,		9091	tt0114576	en	Sudden Death	International	5.23158	/eoWvKD60lT	[{'name': 'Universal Pictures'	[{'iso_3166_1': 'US', 'nam	1995-12-22	64350171	106.0	[{'iso_639_1': 'en', 'na	Released	Terror gc	Sudden	FALSE	5.5	174
FALSE	{'id': 645, 'name': 'James Bc	58000000	[{'id': 12, 'name': 'Ac	http://www.m	710	tt0113189	en	GoldenEye	James Bond	14.686036	/5c0ovjT41Kn	[{'name': 'United Artists', 'id'	[{'iso_3166_1': 'GB', 'nam	1995-11-16	352194034	130.0	[{'iso_639_1': 'en', 'na	Released	No limits	GoldenE	FALSE	6.6	1194
FALSE		62000000	[{'id': 35, 'name': 'Comedy'}, {'id': 1		9087	tt0112346	en	The American President	Widowed U.	6.318445	/lymPNGLZgP	[{'name': 'Columbia Pictures'	[{'iso_3166_1': 'US', 'nam	1995-11-17	107879496	106.0	[{'iso_639_1': 'en', 'na	Released	Why car	The Am	FALSE	6.5	199
FALSE		0	[{'id': 35, 'name': 'Comedy'}, {'id': 2		12110	tt0112896	en	Dracula: Dead and Loving	When a lawy	5.430331	/xve4cgfYItnO	[{'name': 'Columbia Pictures'	[{'iso_3166_1': 'FR', 'nam	1995-12-22	0	88.0	[{'iso_639_1': 'en', 'na	Released		Dracula:	FALSE	5.7	210
FALSE	{'id': 117693, 'name': 'Balto	0	[{'id': 10751, 'name': 'Family'}, {'id'		21032	tt0112453	en	Balto	An outcast h	12.140733	/gV5PCAVCPN	[{'name': 'Universal Pictures'	[{'iso_3166_1': 'US', 'nam	1995-12-22	11348324	78.0	[{'iso_639_1': 'en', 'na	Released	Part Dog	Balto	FALSE	7.1	423
FALSE		44000000	[{'id': 36, 'name': 'History'}, {'id': 18		10858	tt0113987	en	Nixon	An all-star ca	5.092	/cICkmCEiXRh	[{'name': 'Hollywood Pictures	[{'iso_3166_1': 'US', 'nam	1995-12-22	13681765	192.0	[{'iso_639_1': 'en', 'na	Released	Triumph	Nixon	FALSE	7.1	72
FALSE		98000000	[{'id': 28, 'name': 'Action'}, {'id': 12,		1408	tt0112760	en	Cutthroat Island	Morgan Ada	7.284477	/odM9973klv9	[{'name': 'Le Studio Canal+',	[{'iso_3166_1': 'FR', 'nam	1995-12-22	10017322	119.0	[{'iso_639_1': 'en', 'na	Released	The Cou	Cutthroa	FALSE	5.7	137
FALSE		52000000	[{'id': 18, 'name': 'Drama'}, {'id': 80		524	tt0112641	en	Casino	The life of th	10.137389	/xo517ibXBDd	[{'name': 'Universal Pictures'	[{'iso_3166_1': 'FR', 'nam	1995-11-22	116112375	178.0	[{'iso_639_1': 'en', 'na	Released	No one s	Casino	FALSE	7.8	1343
FALSE		16500000	[{'id': 18, 'name': 'Drama'}, {'id': 10		4584	tt0114388	en	Sense and Sensibility	Rich Mr. Das	10.673167	/lA9HTy84Bb6	[{'name': 'Columbia Pictures	[{'iso_3166_1': 'GB', 'nam	1995-12-13	135000000	136.0	[{'iso_639_1': 'en', 'na	Released	Lose you	Sense a	FALSE	7.2	364

Figure 3-3 Part of movies_meatdata.csv

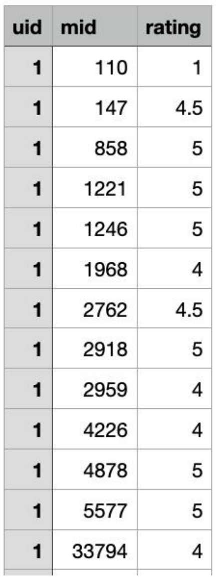
There are a total of three columns in credits.csv, which are used to obtain the corresponding cast information of each movie, making the content of the knowledge graph richer and more comprehensive.



cast	crew	id
[{'cast_id': 14, 'character': 'Woody (voic	[{'credit_id': '52fe4284c3a36847f8024f49', 'department': 'Directing', 'gender': 2, '	862
[{'cast_id': 1, 'character': 'Alan Parrish'	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'department': 'Production', 'gender': 2	8844
[{'cast_id': 2, 'character': 'Max Goldma	[{'credit_id': '52fe466a9251416c75077a89', 'department': 'Directing', 'gender': 2,	15602
[{'cast_id': 1, 'character': "Savannah 'V	[{'credit_id': '52fe44779251416c91011acb', 'department': 'Directing', 'gender': 2,	31357
[{'cast_id': 1, 'character': 'George Bank	[{'credit_id': '52fe44959251416c75039ed7', 'department': 'Sound', 'gender': 2, 'ic	11862
[{'cast_id': 25, 'character': 'Lt. Vincent	[{'credit_id': '52fe4292c3a36847f802916d', 'department': 'Directing', 'gender': 2,	949
[{'cast_id': 1, 'character': 'Linus Larrab	[{'credit_id': '52fe44959251416c75039da9', 'department': 'Directing', 'gender': 2,	11860
[{'cast_id': 2, 'character': 'Tom Sawyer'	[{'credit_id': '52fe46bdc3a36847f810f797', 'department': 'Writing', 'gender': 2, 'id	45325
[{'cast_id': 1, 'character': 'Darren Franc	[{'credit_id': '52fe44dbc3a36847f80ae0f1', 'department': 'Directing', 'gender': 2, '	9091
[{'cast_id': 1, 'character': 'James Bond'	[{'credit_id': '52fe426ec3a36847f801e14b', 'department': 'Directing', 'gender': 2,	710
[{'cast_id': 1, 'character': 'Andrew Shep	[{'credit_id': '52fe44dac3a36847f80adfa3', 'department': 'Camera', 'gender': 2, 'ic	9087
[{'cast_id': 9, 'character': 'Count Dracu	[{'credit_id': '52fe44b79251416c7503e7fb', 'department': 'Editing', 'gender': 2, 'ic	12110
[{'cast_id': 1, 'character': 'Balto (voice)'	[{'credit_id': '593f24b9c3a3680369002371', 'department': 'Production', 'gender':	21032
[{'cast_id': 1, 'character': 'Richard Nixo	[{'credit_id': '52fe43c59251416c7501d6f3', 'department': 'Directing', 'gender': 2,	10858

Figure 3-4 Part of credits.csv

Finally, there are altogether three columns in ratings.csv, which are used to get the rating information of each user for different movies, and are used to get the user's preferences for subsequent personalized recommendations.



uid	mid	rating
1	110	1
1	147	4.5
1	858	5
1	1221	5
1	1246	5
1	1968	4
1	2762	4.5
1	2918	5
1	2959	4
1	4226	4
1	4878	5
1	5577	5
1	33794	4

Figure 3-5 Part of ratings.csv

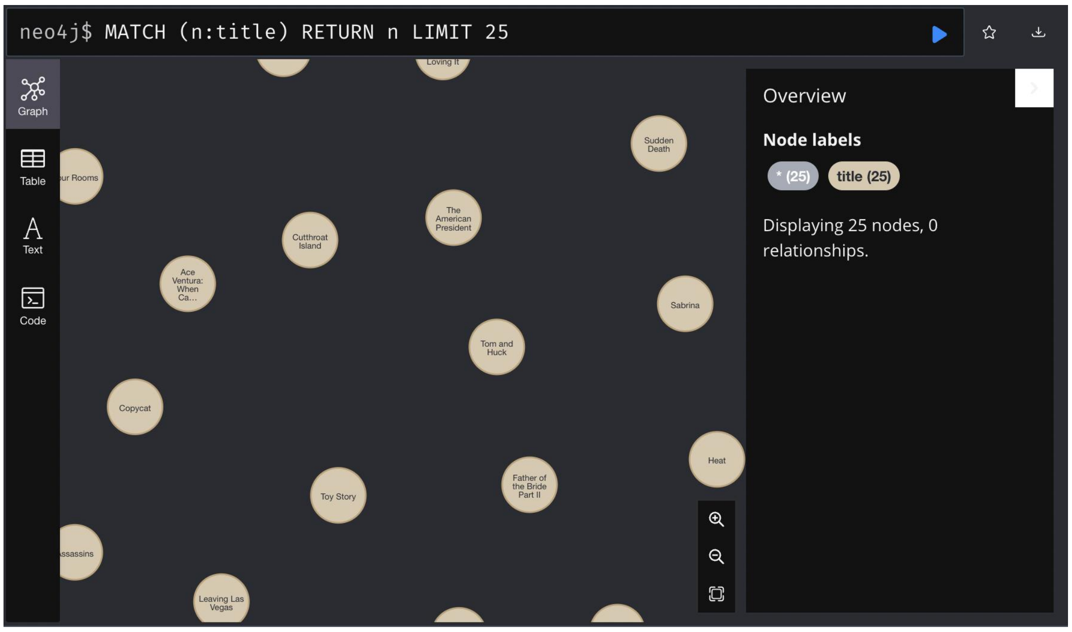
3.2.3 Data Importing

To create a comprehensive knowledge graph using this dataset, it is necessary to import the various data categories into Neo4j. This step is crucial as it lays the foundation for the subsequent stages of graph construction and analysis. By importing each data category into the system, we can establish the connections and relationships between the data points. Once the data is imported and organized, we can begin to apply graph algorithms and queries to explore the graph and generate useful visualizations. Overall, the process of importing the individual data categories into Neo4j is a critical component of knowledge graph construction and serves as a starting point for deriving insights from complex and interconnected datasets.

Before creating the various complex relationships, the nodes need to be first imported, which will use the following code:

```
LOAD CSV WITH HEADERS  FROM "file:///title.csv" AS line
MERGE (z:title{name:line.name})
```

Table 3-1 Import Movie Information



```
neo4j$ MATCH (n:title) RETURN n LIMIT 25
```

Overview

Node labels

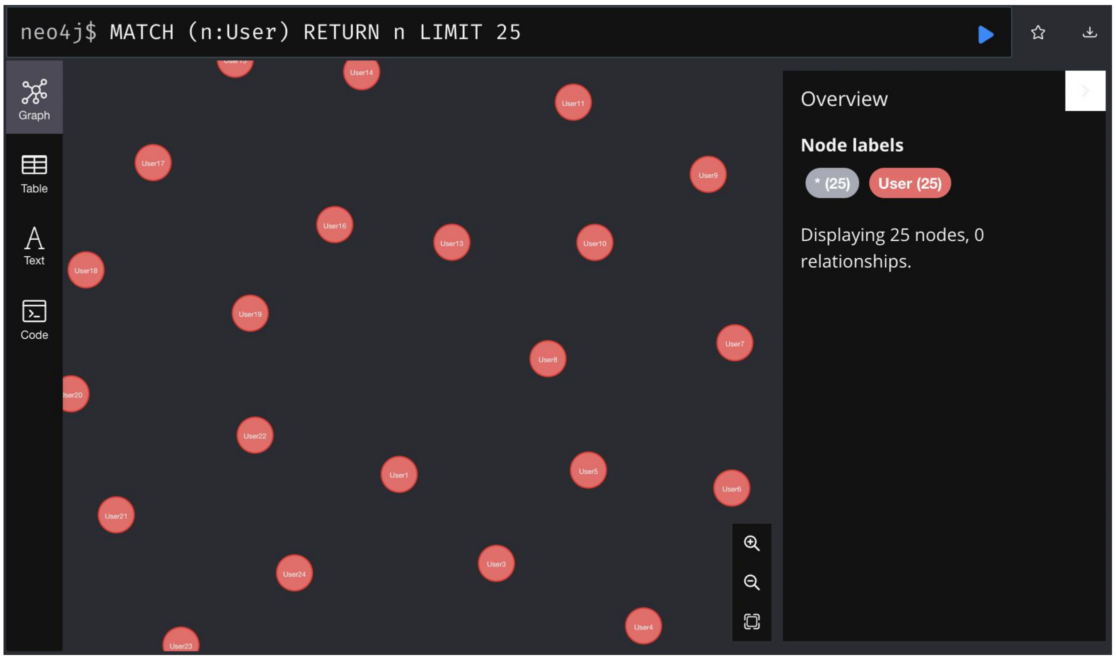
* (25) title (25)

Displaying 25 nodes, 0 relationships.

Figure 3-6 Part of the Nodes of Movies

```
LOAD CSV WITH HEADERS  FROM "file:///user.csv" AS line
MERGE (z:User{name:line.name})
```

Table 3-2 Import User Information



```
neo4j$ MATCH (n:User) RETURN n LIMIT 25
```

Overview

Node labels

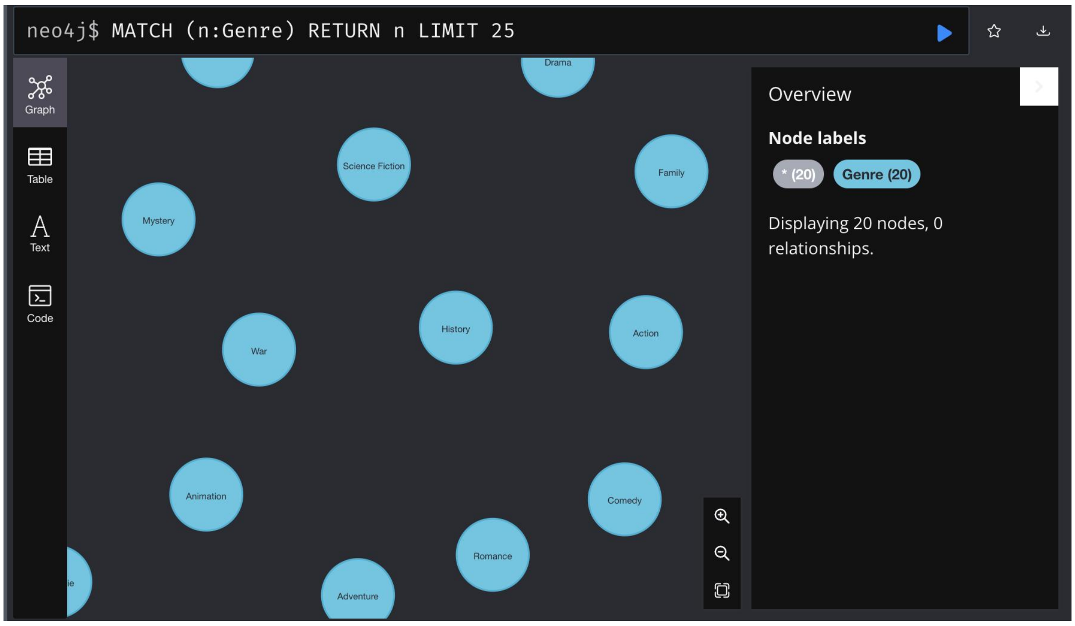
* (25) User (25)

Displaying 25 nodes, 0 relationships.

Figure 3-7 Part of the Nodes of Users

```
LOAD CSV WITH HEADERS  FROM "file:///genre.csv" AS line
MERGE (z:Genre{name:line.name})
```

Table 3-3 Import Genre Information



```
neo4j$ MATCH (n:Genre) RETURN n LIMIT 25
```

Overview

Node labels

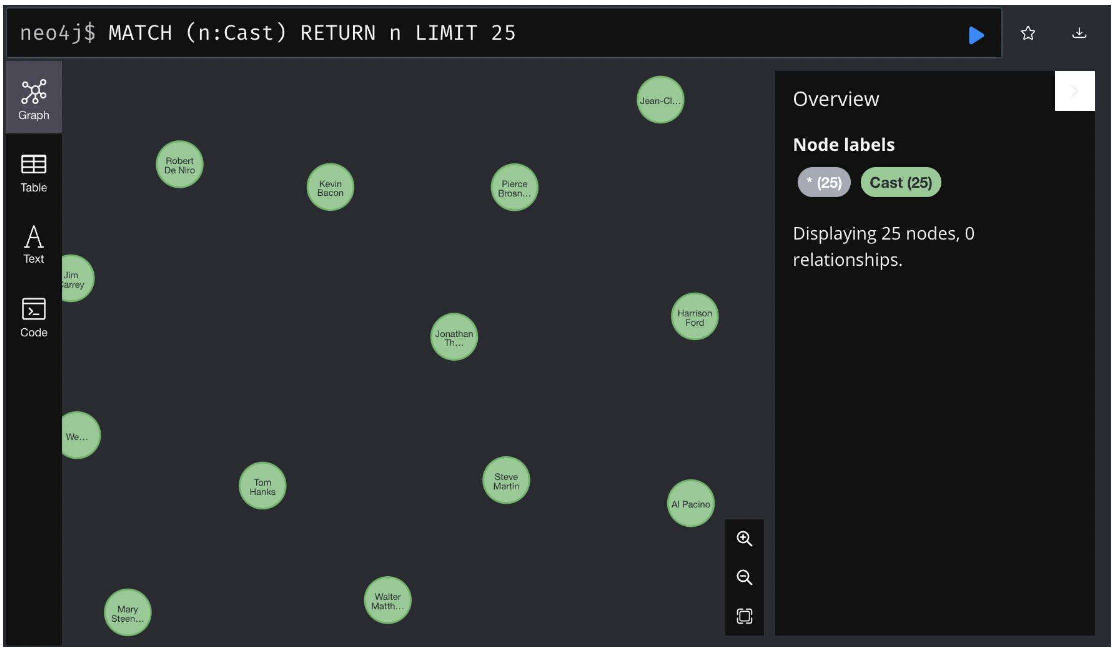
* (20) Genre (20)

Displaying 20 nodes, 0 relationships.

Figure 3-8 Part of the Nodes of Genres

```
LOAD CSV WITH HEADERS  FROM "file:///cast.csv" AS line
MERGE (z:Cast{name:line.name})
```

Table 3-4 Import Cast Information



```
neo4j$ MATCH (n:Cast) RETURN n LIMIT 25
```

Overview

Node labels

* (25) Cast (25)

Displaying 25 nodes, 0 relationships.

Figure 3-9 Part of the Nodes of Casts

The aforementioned pieces of code serve to import the requisite nodes, namely movies, users, genres, and casts, into this graph database management system. Each of these nodes is represented by a distinct color in Neo4j, which facilitates easy identification and visualization of their respective relationships within the knowledge graph. To transform the disparate nodes into a cohesive graph, it is necessary to establish connections between them through the creation of relationships that capture their interdependencies. By doing so, the isolated data points are amalgamated into a comprehensive knowledge graph that reveals the underlying structure of the dataset.

Thus, the relationship information in the dataset can be imported into the graph by running the following code:

```
LOAD CSV WITH HEADERS FROM "file:///is.csv" AS line
MATCH (from:title{title:line.from}),(to:Genre{Genre:line.to})
MERGE (from)-[r:is{relation:row.relation}]->(to)

LOAD CSV WITH HEADERS FROM "file:///rated.csv" AS line
MATCH (from:User{User:line.from}),(to:title{title:line.to})
MERGE (from)-[r:rated{relation:row.relation}]->(to)
SET r.weight = toInteger(line.rate)

LOAD CSV WITH HEADERS FROM "file:///acted_in.csv" AS line
MATCH (from:Cast{Cast:line.from}),(to:title{title:line.to})
MERGE (from)-[r:acted_in{relation:row.relation}]->(to)
```

Table 3-5 Import the Relationships

The above code successfully interconnects the four nodes User, title, Genre and Cast, where ***'title'*** is ***'Genre'***, ***'Cast'*** acted in ***'title'***, ***'User'*** rated ***'title'***.

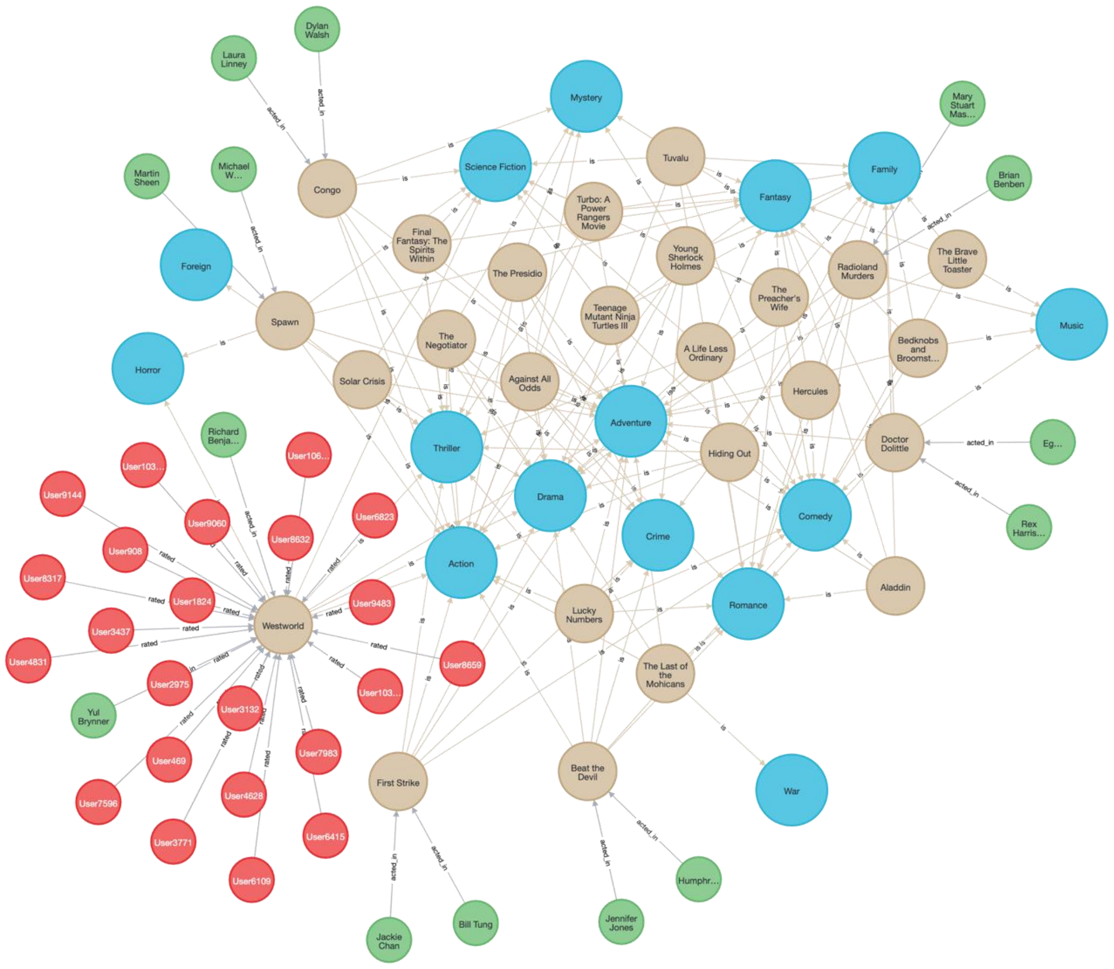
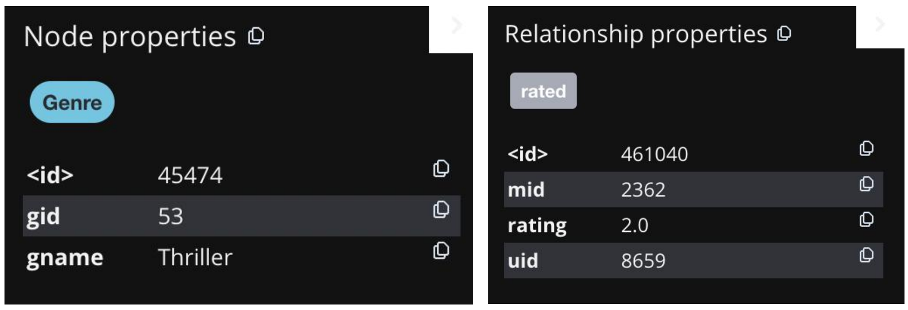


Figure 3-10 Part of the Knowledge Graph

Note that since the relationship between User and Genre is a rating relationship, it is important to add the corresponding weight for each edge after adding the connected edges, so as to record the user's rating of the movie and facilitate the subsequent calculation. Once imported, each node and relationship will be given a unique id, which can be checked at any time in the properties window along with other information imported before.



Node properties

Genre

<id>	45474
gid	53
gname	Thriller

Relationship properties

rated

<id>	461040
mid	2362
rating	2.0
uid	8659

Figure 3-11 Properties Window

Upon completing the data import process described above, a comprehensive knowledge graph is generated, thereby establishing a platform for further analysis and inquiry. By leveraging the advanced graph data model, which facilitates the representation of complex interdependencies between entities, the generated knowledge graphs allow to extract the desired data information for the next step of analysis.

3.3 Graph Analytics

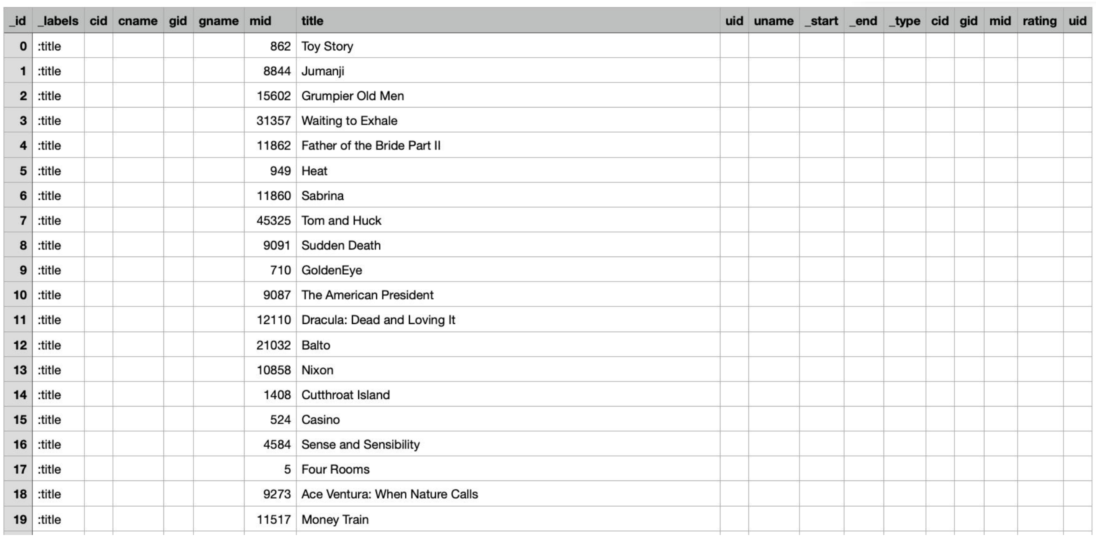
3.3.1 Data Export

To facilitate the processing of the data, the information in the knowledge graph needs to be exported to a csv file first, and then further computed using Python. To achieve this step, the following code needs to be run in Neo4j first:

```
CALL apoc.export.csv.all("export.csv", {})
```

Table 3-6 Export the Knowledge Graph

It can export all the information in the entire knowledge graph together to a specified csv file as shown below:



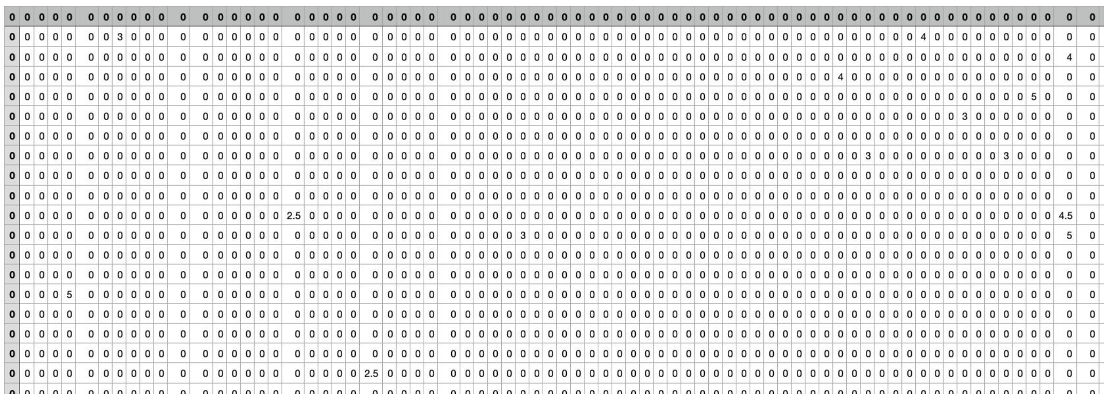
_id	_labels	cid	cname	gid	gname	mid	title	uid	uname	_start	_end	_type	cid	gid	mid	rating	uid
0	:title					862	Toy Story										
1	:title					8844	Jumanji										
2	:title					15602	Grumpier Old Men										
3	:title					31357	Waiting to Exhale										
4	:title					11862	Father of the Bride Part II										
5	:title					949	Heat										
6	:title					11860	Sabrina										
7	:title					45325	Tom and Huck										
8	:title					9091	Sudden Death										
9	:title					710	GoldenEye										
10	:title					9087	The American President										
11	:title					12110	Dracula: Dead and Loving It										
12	:title					21032	Balto										
13	:title					10858	Nixon										
14	:title					1408	Cutthroat Island										
15	:title					524	Casino										
16	:title					4584	Sense and Sensibility										
17	:title					5	Four Rooms										
18	:title					9273	Ace Ventura: When Nature Calls										
19	:title					11517	Money Train										

Figure 3-12 Part of Export.csv

In preparation for matrix factorization, a sparse matrix of user-movie ratings must be generated from the existing graphs. First, it was necessary to export the relationship between users and movies, movies and genres to two separate csv files by Neo4j. The two files can then each be converted to matrix form by running the following code:

```
import pandas as pd

data = pd.read_csv('./movie_rating.csv')

df = data.pivot(index='user', columns='movie', values='rate')
df = df.reset_index()
df.fillna(0, inplace=True)

outputpath = './movie.csv'
df.to_csv(outputpath, index=False, header=True)
```

Table 3-7 Generate the Rating Matrix

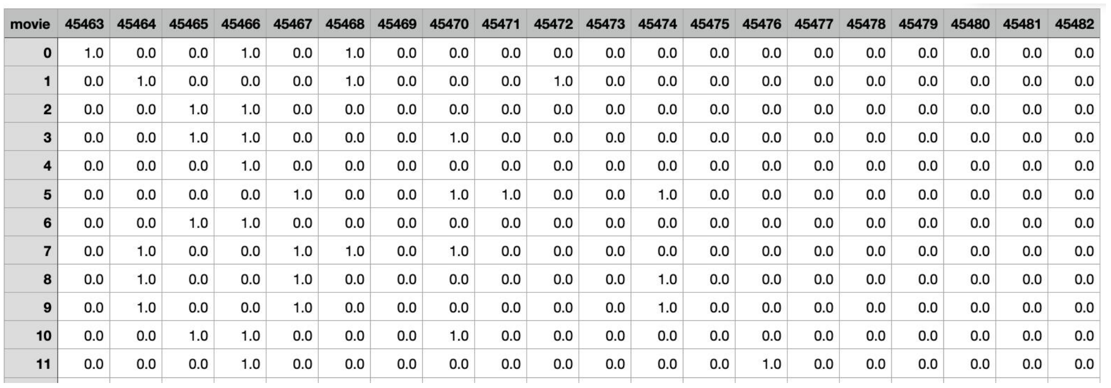


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4.5	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-13 Part of movie.csv

```
import pandas as pd

data = pd.read_csv('./movie_genre.csv')

df = data.pivot(index='movie', columns='genre', values='val')
df = df.reset_index()
df.fillna(0, inplace=True)

outputpath = './MovieGenre.csv'
df.to_csv(outputpath, index=False, header=True)
```

Table 3- 8 Generate the Adjacency Matrix



movie	45463	45464	45465	45466	45467	45468	45469	45470	45471	45472	45473	45474	45475	45476	45477	45478	45479	45480	45481	45482
0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 3-14 Part of MovieGenre.csv

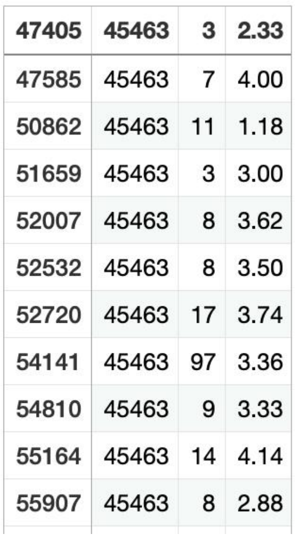
3.3.2 Data Analysis and Updates

To construct the tripartite graph, the genre category is selected as an intermediary between users and movies. However, at present, only the user ratings of movies and the concatenation of movies with genre information are available, and these data require further processing to derive the ratings of each user for different genres.

```
import re, json, os
from collections import OrderedDict
import json, csv
import difflib

class QueryNo4j:
  def get_relation(self, filename_csv):
    fw = open("rela-route.csv", "w", encoding="utf-8", newline='')
    writer1 = csv.writer(fw)
    fw2 = open("rela-new.csv", "w", encoding='utf-8', newline='')
    writer = csv.writer(fw2)

    hash_genre = {}
    hash_Cast = {}
    hash_rela_new = {}
    hash_rela = {}
    hash_rela_rating = {}

    csv.field_size_limit(500 * 1024 * 1024)
    f = open(filename_csv, "r", encoding='utf-8')
    fr = csv.reader(_.replace('\x99', '') for _ in f)
    header_list = []
    all_num = 0
    for item in fr:
      all_num += 1
      if all_num == 1:
        header_list = item
        continue
      hash_data = OrderedDict(zip(header_list, item))
      hash_data = json.loads(json.dumps(hash_data))
      if hash_data["_labels"] == ":Genre":
        hash_genre[hash_data["_id"]] = []
      if hash_data["_labels"] == ":Cast":
        hash_Cast[hash_data["_id"]] = []
      if hash_data["_start"] != "":
        if hash_data["_start"] in hash_Cast or hash_data["_end"] in 
hash_Cast: continue
      str_start = hash_data["_start"]
      str_end = hash_data["_end"]
      if str_end not in hash_rela:
        hash_rela[str_end] = []
      hash_rela[str_end].append(str_start)
      if hash_data["rating"] != "":
        new_re = str(str_start) + "->" + str(str_end)
        hash_rela_rating[new_re] = hash_data["rating"]

  for item in hash_genre.keys():
    if item in hash_rela:
      for item1 in hash_rela[item]:
        if item1 in hash_rela:
          for item2 in hash_rela[item1]:
            new_re = str(item2) + "->" + str(item)
            if new_re not in hash_rela_new:
              hash_rela_new[new_re] = [0, []]
              user_title = str(item2) + "->" + str(item1)
              hash_rela_new[new_re][0] += 1
              if user_title in hash_rela_rating:
                hash_rela_new[new_re][1].append(float(hash_rela_rat
ing[user_title]))
              else:
                print("=======" + new_re)
              fw0.write(str(item2) + "->" + str(item1) + "->" + str(item)
+ "\n")
              writer1.writerow([item2, item1, item])

for key, val in hash_rela_new.items():
  arr = key.split("->")
  rating = '%.2f' % (sum(val[1]) / val[0])
  fw1.write(key + "<#>" + str(val[0]) + "<#>" + str(rating) + "\n")
  arr.append(val[0])
  arr.append(rating)
  writer.writerow(arr)

if __name__ == '__main__':
  _qN = QueryNo4j()

  filename_csv = r"./export.csv"
  _qN.get_relation(filename_csv)
  exit()
```

Table 3-9 Finding Paths and Generating New Edges

By running this code, a new file can be obtained: rela-new.csv, which is used to store the queried user-to-genre paths and take the average value of user ratings for movies in all paths as the weight of this new path. The file contains four columns in total, namely user node id, genre node id, number of paths, and the new weight.



47405	45463	3	2.33
47585	45463	7	4.00
50862	45463	11	1.18
51659	45463	3	3.00
52007	45463	8	3.62
52532	45463	8	3.50
52720	45463	17	3.74
54141	45463	97	3.36
54810	45463	9	3.33
55164	45463	14	4.14
55907	45463	8	2.88

Figure 3-15 Part of rela-new.csv

With this new connectivity file, a new edge with weights between the user and genre nodes can be created by importing into Neo4j, again using the Cypher statement:

```
LOAD CSV FROM "file:///rela-new.csv" AS line
MATCH (n1:User) WHERE id(n1)=toInteger(line[0])
MATCH (n2:Genre)  WHERE id(n2)=toInteger(line[1])

MERGE(n1)-[r:relaNew{rating:line[3]}]->(n2)
```

Table 3-10 Import the New Relationship

In this way, a more complete knowledge graph can be obtained, based on which the Cast node and rated and acted_in relationships are deleted to obtain the tripartite graph of User-Genre-Movie.

```
MATCH (x:Cast) - [r:acted_in] -> (y:title)  DELETE r
MATCH (x:Cast)  DELETE x

MATCH (x:User) - [r:rated] -> (y:title)  DELETE r
```

Table 3-11 Delete the Useless Nodes

After this operation, there are only two types of relationship information left in the graph, respectively between the user and genre and between genre and the movie.

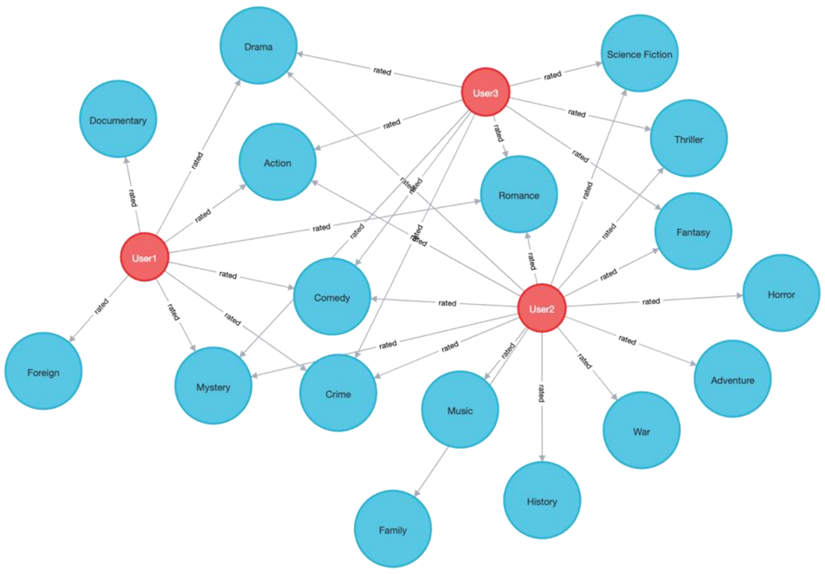


Figure 3-16 Part of the Relationship Between Users and Genres

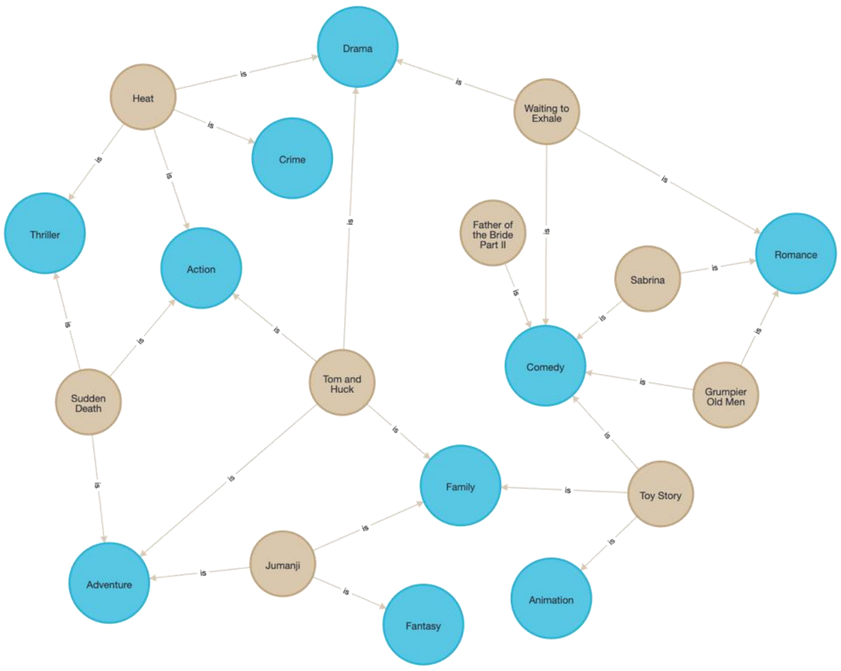


Figure 3-17 Part of the Relationship Between Movies and Genres

In order to make the graph more standardized, the previously imported relaNew relationship needs to be renamed to rated, and then the terminally required tripartite graph is generated.

```
MATCH(n)-[r:relaNew]->(m) CREATE(n)-[r2:rated]->(m)
SET r2=r WITH r
DELETE r
```

Table 3-12 Rename the Relationship

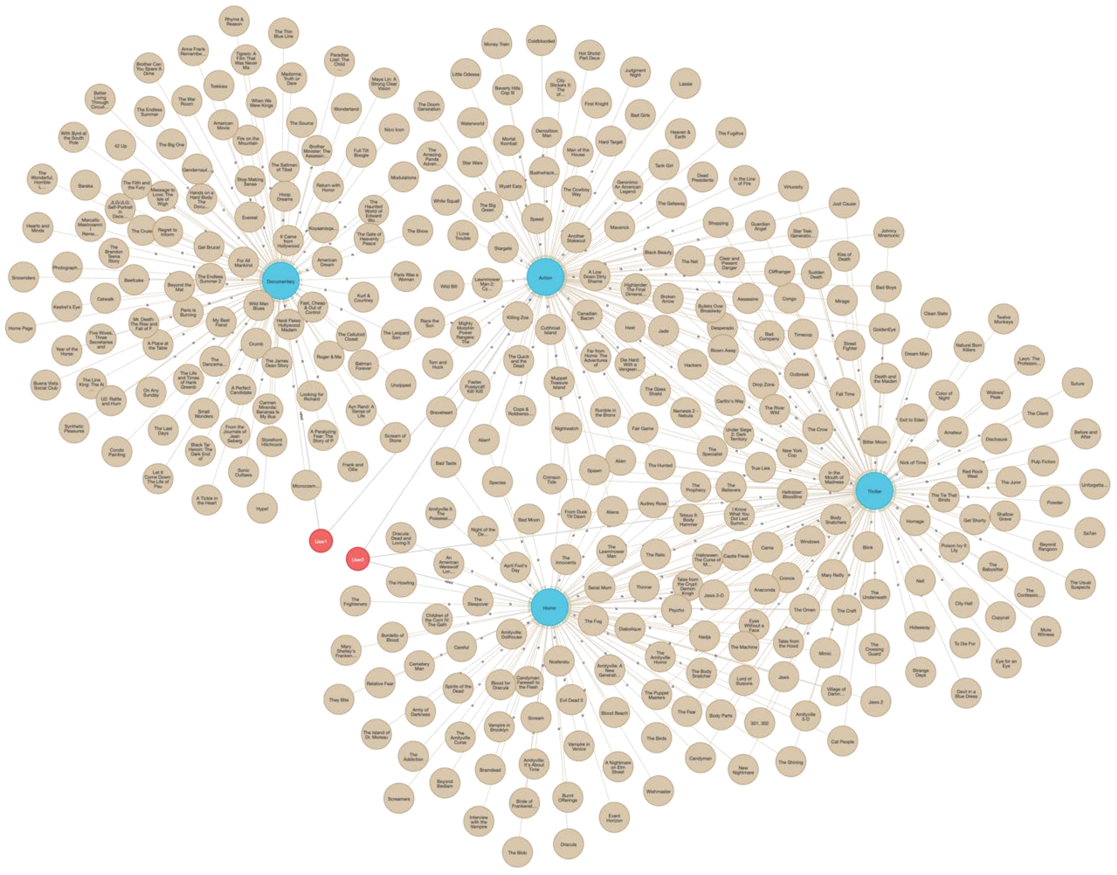


Figure 3-18 Part of the Final Version of the Tripartite Graph

CHAPTER 4 Lab Result

In this section, all my algorithms and code are based on python 3.10 and my development environment is macOS. the IDE I use is PyCharm.

4.1 SVD++ Factorization

Since the Genre node is chosen as the intermediary between users and movies when generating the tripartite graph, the traditional matrix factorization is not very applicable here, so I choose the SVD++ factorization based on SVD factorization, in which the relationship between User and Genre is used as the implicit vector for matrix factorization and prediction during the process of factorization through operations.

The factorization process of SVD++ is basically the same as that of SVD, the biggest difference is that the implicit vector is taken into account together in the final prediction function to generate a more personalized recommendation result.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \bullet (p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j) \qquad (4\text{-}1)$$

In the Equation (4-1), μ is the global average rating, b_u is the user bias, which is set to 0 here, and b_i is the item bias. q_i is the item vector, p_u is the user's vector, N_u is the set of objects that user u has rated, $|N_u|^{-\frac{1}{2}}$ is a normalization factor to ensure that more weight is assigned to the implicit feedback when there are more user rating behaviors, and y_j is the vector of objects corresponding to the implicit feedback.[8]

And here is the code to achieve this process:

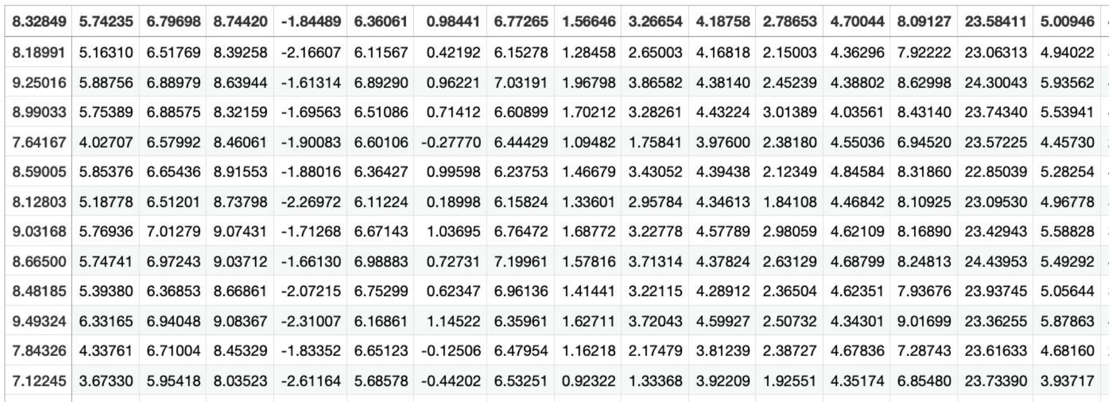
```
import numpy as np
from scipy.sparse.linalg import svds
import pandas as pd
import time


start = time.perf_counter()
R = pd.read_csv('movie.csv').to_numpy()
X = pd.read_csv('UserGenreR.csv').to_numpy()

num_users, num_items = R.shape
num_factors = 20
lambda_value = 0.1

global_mean = np.mean(R[np.where(R != 0)])

U, S, Vt = svds(R - global_mean, k=num_factors)
S = np.diag(S)

user_bias = np.zeros(num_users)

item_means = np.nanmean(np.where(R != 0, R, np.nan), axis=0)
item_bias = item_means - global_mean

num_rated = np.zeros(num_users)
num_rated = num_rated.astype(int)
movie_genre = pd.read_csv('MovieGenres.csv', index_col=0).to_nump
y()


y = np.zeros((0, num_factors))
for i in range(num_users-1):
  for j in range(num_items-1):
    if R[i, j] != 0:
      new_row = X[i, :] * movie_genre[j, :]
      y = np.vstack([y, new_row])
      num_rated[i] += 1

P = np.dot(U, np.sqrt(S))
Q = np.dot(np.sqrt(S), Vt)

num_iterations = 100
learning_rate = 0.01
for i in range(num_iterations):
  for u in range(num_users):
    for j in range(num_items):
      if R[u, j] > 0:
       err = R[u, j] - global_mean - user_bias[u] - item_bias[j] -
np.dot(P[u, :], Q[:, j])
       P[u, :] += learning_rate * (err * Q[:, j] - lambda_value * P[u, :])
       Q[:, j] += learning_rate * (err * P[u, :] - lambda_value * Q[:,
j])

R_pred = np.zeros((num_users, num_items))
y_j = np.zeros(num_factors)
y_j = np.sum(y, axis=0) / (500 * np.sqrt(num_items))

for i in range(num_users-1):
  for j in range(num_items-1):
    R_pred[i, j] = global_mean + user_bias[i] + item_bias[j] +
np.dot(Q[:, j], P[i, :] + y_j)


print(R_pred)
end = time.perf_counter()
print('time cost: ', end - start)

np.savetxt('predictions.csv', R_pred, delimiter=',', fmt='%.5f')
```

Table 4-1 SVD++ Code

Herewith, in order to reduce the wasted time in the operation process, the two matrixes are manually reduced in size before the factorization, and only 7000 users' rating information and 5086 movies' classification information are retained, and the number of genres is not reduced here as only 20 kinds are listed.

In addition, since the genres was set as an implicit factor in this experiment, and the user's rating of genre was used as implicit preference information, it was found by calculation that the $|N_u|^{-\frac{1}{2}}$ was too large to get the correct prediction information, so the normalization factor $|N_u|^{-\frac{1}{2}}$ was adjusted to $\frac{1}{\sqrt{500\times|N_u|}}$ in the prediction function to get a closer prediction value.

And the predicted results are shown below:



8.32849	5.74235	6.79698	8.74420	-1.84489	6.36061	0.98441	6.77265	1.56646	3.26654	4.18758	2.78653	4.70044	8.09127	23.58411	5.00946
8.18991	5.16310	6.51769	8.39258	-2.16607	6.11567	0.42192	6.15278	1.28458	2.65003	4.16818	2.15003	4.36296	7.92222	23.06313	4.94022
9.25016	5.88756	6.88979	8.63944	-1.61314	6.89290	0.96221	7.03191	1.96798	3.86582	4.38140	2.45239	4.38802	8.62998	24.30043	5.93562
8.99033	5.75389	6.88575	8.32159	-1.69563	6.51086	0.71412	6.60899	1.70212	3.28261	4.43224	3.01389	4.03561	8.43140	23.74340	5.53941
7.64167	4.02707	6.57992	8.46061	-1.90083	6.60106	-0.27770	6.44429	1.09482	1.75841	3.97600	2.38180	4.55036	6.94520	23.57225	4.45730
8.59005	5.85376	6.65436	8.91553	-1.88016	6.36427	0.99598	6.23753	1.46679	3.43052	4.39438	2.12349	4.84584	8.31860	22.85039	5.28254
8.12803	5.18778	6.51201	8.73798	-2.26972	6.11224	0.18998	6.15824	1.33601	2.95784	4.34613	1.84108	4.46842	8.10925	23.09530	4.96778
9.03168	5.76936	7.01279	9.07431	-1.71268	6.67143	1.03695	6.76472	1.68772	3.22778	4.57789	2.98059	4.62109	8.16890	23.42943	5.58828
8.66500	5.74741	6.97243	9.03712	-1.66130	6.98883	0.72731	7.19961	1.57816	3.71314	4.37824	2.63129	4.68799	8.24813	24.43953	5.49292
8.48185	5.39380	6.36853	8.66861	-2.07215	6.75299	0.62347	6.96136	1.41441	3.22115	4.28912	2.36504	4.62351	7.93676	23.93745	5.05644
9.49324	6.33165	6.94048	9.08367	-2.31007	6.16861	1.14522	6.35961	1.62711	3.72043	4.59927	2.50732	4.34301	9.01699	23.36255	5.87863
7.84326	4.33761	6.71004	8.45329	-1.83352	6.65123	-0.12506	6.47954	1.16218	2.17479	3.81239	2.38727	4.67836	7.28743	23.61633	4.68160
7.12245	3.67330	5.95418	8.03523	-2.61164	5.68578	-0.44202	6.53251	0.92322	1.33368	3.92209	1.92551	4.35174	6.85480	23.73390	3.93717

Figure 4-2 The Predicted Matrix

4.2 Basic Matrix Factorization

To verify whether the present model is more efficient, I factorized and predicted the user's rating matrix of the movie again using the basic matrix factorization method and compared the results.

```
from math import pow
import numpy as np
import pandas as pd
import time


start = time.perf_counter()

def MF(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
  n, m = R.shape
  result = []
  for step in range(steps):
    for i in range(n):
      for j in range(m):
        eij = R[i][j] - np.dot(P[i, :], Q[:, j])
        for k in range(K):
          if R[i][j] != 0:
            P[i][k] += alpha * (2 * eij * Q[k][j] - beta * P[i][k])
            Q[k][j] += alpha * (2 * eij * P[i][k] - beta * Q[k][j])
    eR = np.dot(P, Q)
    e = 0
    for i in range(n):
      for j in range(m):
        if R[i][j] != 0:
          e += pow(R[i][j] - np.dot(P[i, :], Q[:, j]), 2)
            for k in range(K):
              e += (beta / 2) * (pow(P[i][k], 2) + pow(Q[k][j], 2))
     result.append(e)

     if e < 0.001:
       break
   return P, Q, result

if __name__ == "__main__":
   R = pd.read_csv('movie.csv').to_numpy()
   N, M = R.shape
   K = 3
   P = np.random.rand(N, K)
   Q = np.random.rand(K, M)
   nP, nQ, result = MF(R, P, Q, K)

   print(R)
   R_MF = np.dot(nP, nQ)
   print(R_MF)
   end = time.perf_counter()
   print('time cost', end - start)

   np.savetxt('predictions2.csv', R_MF, delimiter=',', fmt='%.5f')
```

Table 4-2 Basic MF Code

4.3 Comparison of the Two Methods

Since the original rating matrix is too sparse, and no suitable effective method for factorizing sparse matrix has been cited in this model, which only focuses on the optimization effect of implicit factors on matrix factorization, the accuracy optimization level of the prediction matrix is not compared here yet.

As for the timing optimization, the time.perf_counter() function added in the code is used to calculate the time required for the whole program to run and to determine whether the prediction speed after applying the new model is faster than that using the basic matrix factorization method.

After testing, the SVD++ code in Table 4-1 takes 1543.8s to run, while the MF code in Table 4-2 takes over 4 hours to run, although it has been further reduced in size to a matrix of 6249 times 3055. It can be clearly seen that the prediction time is greatly reduced in the application of the new model containing implicit factors.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

The research in this paper aims to address the challenges in personalized recommendation systems, such as large amount of data and high computational complexity. By combining knowledge graph and matrix factorization, and modeling with tripartite graphs and implicit factors, it successfully reduces the computational complexity. Overall, the personalized movie recommendation system proposed in this paper has wide application prospects and can provide users with more personalized and satisfactory recommendation services.

5.2 Future Work

Although the research in this paper has achieved some results, there are still many shortcomings that need to continue to be studied in depth, so future research directions include but are not limited to the following:

First, since a suitable method for factorizing the sparse matrix has not yet been introduced in this paper, resulting in the lack of a multifaceted evaluation of the degree of optimization of this model based on detailed data, I believe that a method for factorizing the sparse matrix needs to be found and added to the SVD++ factorization in order to achieve optimization of its prediction accuracy before other further studies can be conducted.

Besides, we can consider introducing technologies from other fields into personalized recommendation systems, such as natural language processing and sentiment analysis, to provide more reference values to better analyze user needs and sentiment preferences, so as to improve recommendation quality and user experience and make recommendations more humanized.

Chapter 6 Reference

[1] Lee, C., Han, D., Han, K., & Yi, M. (2022). Improving Graph-Based Movie Recommender System Using Cinematic Experience. *Applied Sciences*, *12*(3), 1493. https://doi.org/10.3390/app12031493

[2] Mahmood, T., & Ricci, F. (2009). Improving recommender systems with adaptive conversational strategies. *Proceedings of the 20th ACM Conference on Hypertext and Hypermedia - HT '09*, 73. https://doi.org/10.1145/1557914.1557930

[3] Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 1–35). Springer US. https://doi.org/10.1007/978-0-387-85820-3_1

[4] Beel, J., Gipp, B., Langer, S., & Breitinger, C. (2016). Research-paper recommender systems: A literature survey. *International Journal on Digital Libraries*, *17*(4), 305–338. https://doi.org/10.1007/s00799-015-0156-0

[5] Shi, Y., Larson, M., & Hanjalic, A. (2014). Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges. *ACM Computing Surveys*, *47*(1), 1–45. https://doi.org/10.1145/2556270

[6] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, *42*(8), 30–37. https://doi.org/10.1109/MC.2009.263

[7] Funk, Simon. "Netflix Update: Try This at Home".

[8] Yancheng Jia, Changhua Zhang, Qinghua Lu, & Peng Wang. (2014). Users' brands preference based on SVD++ in recommender systems. *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, 1175–1178. https://doi.org/10.1109/WARTIA.2014.6976489

[9] Adomavicius, G., & Tuzhilin, A. (2005). 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*, *17*(6), 734–749. https://doi.org/10.1109/TKDE.2005.99

[10] Huang, Z., Chung, W., Ong, T.-H., & Chen, H. (n.d.). *A Graph-based Recommender System for Digital Library*. 9.

[11] Demovic, L., Fritscher, E., Kriz, J., Kuzmik, O., Proksa, O., Vandlikova, D., Zelenik, D., & Bielikova, M. (2013). Movie Recommendation Based on Graph Traversal Algorithms. *2013 24th International Workshop on Database and Expert Systems Applications*, 152–156. https://doi.org/10.1109/DEXA.2013.24

[12] Hu, S., & Yang, W. (2019). Add–sub pivoting triangular factorization for symmetric matrix. *Journal of Computational and Applied Mathematics*, *355*, 116–127. https://doi.org/10.1016/j.cam.2019.01.006

[13] Gu, M., & Eisenstat, S. C. (1996). Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization. *SIAM Journal on Scientific Computing*, *17*(4), 848–869. https://doi.org/10.1137/0917055

[14] Strang, G. (2016). Introduction to Linear Algebra. In *Introduction to Linear Algebra* (5th edition, p. 97). Wellesley-Cambridge Press.

Chapter 7 Resume

1. Resume

Name: Li Yiran

Gender: Female

Email: vliyr@outlook.com

Education:

2016 ~ 2019, High School, Qingdao No.9 High School.

2019 ~ 2023, Bachelor of Science, Macau University of Science and Technology.

Awards:

11/2020, First Prize of the 8th Cultural Exhibition Competition of Macau University of Science and Technology.

Work experience:

22/11/2021 ~ 31/12/2021, Data Analysis, Google.

04/07/2022 ~ 31/08/2022, Assistant Project Manager, Haier Group.

2. Publications

[1] Improved GAN model for Image Animation, which will be published in *IEECT 2022*

Chapter 8 Acknowledgements

First of all, I would like to thank my supervisor, Prof. Wu Yang, who gave me a lot of guidance and support throughout my final year project and helped me to clarify the direction and goal of the whole project. Secondly, I would also like to thank the other two supervisors of the same defense team, Prof. Lu Xiaoping and Prof. Lan Ting, who gave me a lot of valuable suggestions during the mid-term and final defense, which helped me to further improve my project.

In addition, I would also like to thank my classmates and friends, who helped me to find solutions to problems that I could not solve by myself, and who were willing to accompany me when I was stressed and felt very anxious. In my last year at M.U.S.T., I am very happy to be able to support each other and progress hand in hand with all of you, and successfully finish my last project during university. This precious memory will always be remembered in my heart.