PROJECT PROPOSAL

# A Graph Database Powered Recommendation System

SUBMITTED BY:

**Nupur Pathak**
**Sree Divya Cheerla**
**Vani Bhat**

INSTRUCTOR:

**Dr. Ming-Hwa Wang**

## Table of Contents

## Table of Figures

## Abstract

In this modern era, with the advent of big data, huge amounts of structured, semi-structured and unstructured data are getting generated from various sources especially social networks, e-commerce platforms etc. Storing highly interconnected data from various platforms and getting insights from massive amounts of data is quite a big challenge. NoSQL graph databases like Neo4j are gaining popularity which can provide an effective/efficient solution to storing and querying huge interconnected data. The presence of huge data online poses a potential problem to the users making their choices difficult. In this case, recommender systems (recommendation engines) are very helpful which solves this problem by mining the large volume of dynamically generated interconnected data to provide the users with personalized suggestions.

In this paper, we explore how Neo4J database can be used to store and retrieve the unstructured data in the form of graphs using Cypher query language, how Neo4j's Graph Data Science Library (GDS) can be used to build and implement recommender systems using various graph algorithms to recommend movies to the users which they would prefer to watch next.

The outcomes derived from this paper spotlights the enormous potential for movie recommendations using the graph algorithms of a graph database platform Neo4j.

# 2. Introduction

## 2.1 Objective

The main objective of this paper is to leverage graph database platform Neo4J and its Graph Data science library to build and implement personalized product recommender systems for a dataset of movies and movie ratings using different Graph Algorithms.

Underlying this objective, is the study of

A. Neo4J graph database platform to store interconnected data in the form of graphs.

B. Neo4J graph database platform to query the graphs using Cypher Language.

C. Content based and Collaborative filtering-based techniques to implement a recommender system to recommend movies using Neo4Js node similarity and KNN algorithm respectively.

## 2.2 What is the problem

As the need for providing personalized services to the customers is growing, businesses are increasingly adopting recommendation systems. This will help to improve product discovery on the platforms, increase user engagement, and increase up-sell and cross-sell of their products.

These recommendation systems combine the large volume of data from different sources that reside in silos and generate insights to provide timely and accurate recommendations. The system helps in identifying the correlations between users, items and user/item to provide the recommendations. Most of the current recommendation systems factor only the item-item correlations, while the user preferences and behavior are not taken into account. A database system that can handle large volumes of data with multiple relationships (connected data) is required to provide fast and efficient real-time recommendations.

## 2.3 Why this is a project related to this class

The course is ‘Database Systems’ where we have theoretically imbibed knowledge on 4 different NoSQL in which Graph database is one of them. This project is going to be the practical implementation of knowledge gained in the course ‘Database Systems for analytics’ to solve a real-world problem caused due to the advent of unstructured big data. The centre of attention of this project is using Neo4j which is a graph database platform and its inbuilt Graph Data Science library (GDS) to build and implement a recommender system to recommend the next watchable movie to the user. This project is going to help us learn Neo4j graph database in detail starting from installation to using its inbuilt methods for implementing recommendation systems to address the real-world user and business problem caused by overload of information.

Moreover, working on this project gears us to place ourselves in the competitive industry where many of the companies like Netflix, Amazon, Facebook, Google use NoSQL graph databases to provide recommendations to the user’s which is the substantial part of company’s revenue.

## 2.4 Why other approach is no good

With growing business, the volume gets bigger and bigger everyday but connected data grows exponentially. As the number and depth of relationships between the data increases, storing the data in traditional databases will become complicated and difficult and hence impact its performance. While saving huge volumes of data has been made easier using NoSQL databases, most NoSQL databases store the data as aggregates which are disconnected. They lack relationships which makes it inefficient to use for connected data and graphs which play a role in developing any recommendation system.

A Graph database handles fine-grained networks of information. It consists of two elements: a *node* and a *relationship*. Each node represents an entity (a person, place, thing, category or other piece of data) while each relationship represents how two nodes are connected. Here the connected data has more importance than the individual data points and relationships take highest priority and hence make it ideal for building recommendation systems.

### 2.5 Why you think your approach is better

Among all the other graph database technologies, when it comes to graph storage and processing the connected data, Neo4J leads as it uses Native graph processing model (Index - free adjacency) to store the data in which the connected nodes physically point to each other in the database as it uses graph structures that have nodes, edges and properties to define and store the data.

Since it doesn't require predefined schema unlike the relational databases where we store the data in rows and columns, Neo4J is more flexible when the data changes gradually making it ideal for recommendation systems where user preferences change with time.

### 2.6 Area or scope of investigation

The scope of this project is limited to the usage of Neo4j graph database. Within Neo4j the scope would be loading the data, graph construction, querying the graph using cipher language and usage of graph algorithms like node similarity, and KNN from the graph data science library (GDS). The scope of the recommender systems using Neo4j is limited to recommend movies only from the prefetched data using content-based filtering, collaborative-based filtering (user-based filtering, item-based filtering).

Cloud deployment from Neo4j, Real time recommendation of movies from the websites, YouTube video recommendations are considered out of scope.

Overall, in this paper we primarily focus on using existing movies dataset to recommend movies to the users using various built in graph algorithms in Neo4j database.

## 3. Theoretical bases and Literature review

### 3.1 Definition of the problem

Recommendation systems combine the large volume of data from different sources that reside in silos and generate insights to provide timely and accurate recommendations. A database system that can handle large volumes of data with multiple relationships (inter connected data) is required to provide fast and efficient real-time recommendations.

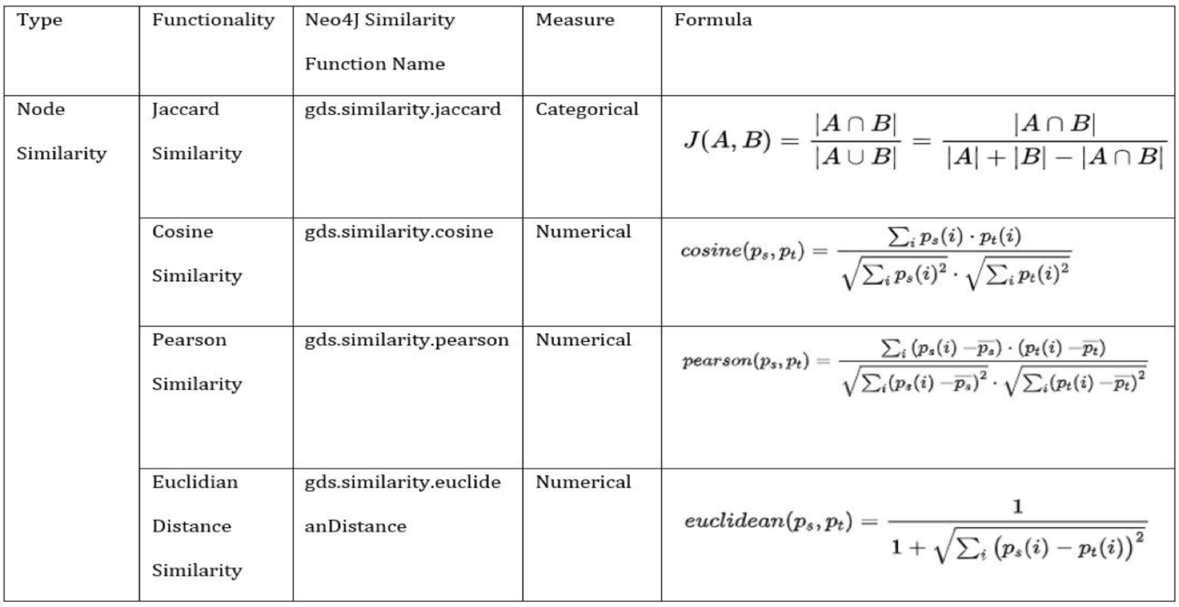
In order to implement the recommendation system using a graph database, Neo4J will be used to create the graph structure. For the calculation of the correlations, the Neo4J tool functionality will be leveraged.

Neo4J provides various functions to calculate the similarity between the nodes. There are two approaches to find the similarity between two nodes based on the type of measure used for computing the similarity.

Let us calculate similarity between two arrays $p_s$, $p_t$ of numbers. The types of measures are (9.2.e):

- Categorical measures: In categorical measures, the arrays are treated as sets (A, B) and similarity is calculated based on the intersection between the two sets.

- Numerical measures: In numerical measures, the similarity between the arrays is calculated based on the closeness of the numbers in the arrays.



| Type | Functionality | Neo4J Similarity Function Name | Measure | Formula |
|---|---|---|---|---|
| Node Similarity | Jaccard Similarity | gds.similarity.jaccard | Categorical | $J(A,B) = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert} = \frac{\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert - \lvert A \cap B \rvert}$ |
| | Cosine Similarity | gds.similarity.cosine | Numerical | $cosine(p_s, p_t) = \frac{\sum_i p_s(i) \cdot p_t(i)}{\sqrt{\sum_i p_s(i)^2} \cdot \sqrt{\sum_i p_t(i)^2}}$ |
| | Pearson Similarity | gds.similarity.pearson | Numerical | $pearson(p_s, p_t) = \frac{\sum_i (p_s(i) - \overline{p_s}) \cdot (p_t(i) - \overline{p_t})}{\sqrt{\sum_i (p_s(i) - \overline{p_s})^2} \cdot \sqrt{\sum_i (p_t(i) - \overline{p_t})^2}}$ |
| | Euclidian Distance Similarity | gds.similarity.euclideanDistance | Numerical | $euclidean(p_s, p_t) = \frac{1}{1 + \sqrt{\sum_i (p_s(i) - p_t(i))^2}}$ |

Fig 1: Mathematical equations for Node similarity algorithm

## 3.2 Theoretical background of the problem

We referred to multiple research papers from 2017 - 2022 which gave us an idea on different ways researchers have implemented recommendation systems.

We will be using following recommendation algorithms based on the type of recommendation systems [9.2.d]:

### 3.2.1 Content-based filtering

In content-based filtering, the data derived from the user's actions or item features is used to provide recommendations by using keywords and attributes of each object and making recommendations that match them. The advantages of using content-based recommendation systems are that they are easy to create as they don't require data from other users to create recommendations. As the algorithm mostly relies on matching the characteristics or attributes of a database object with the user's profile, the recommendations are highly relevant. These types of recommendations are easier to scale to a large number of users as each recommendation is specific to the user

While the downside is that since all recommendations are user specific, they lack novelty and diversity. Even though it is easy to scale to a large number of users, if a new item / object is added or removed, we need to redefine the attributes which is time consuming. Since the recommendations are mainly focused on content, if the attributes are incorrect or inconsistent, it might result in wrong results.

### 3.2.2 Collaborative filtering

#### 3.2.2.1 User-based collaborative filtering

In user-based collaborative filtering, recommendations are provided based on identifying similar users to the target user. There can be two methods followed in this [9.2.f]:

Method 1:

- Step1: Finding the K-nearest neighbors (KNN) to the target user a using a similarity function w

$$\textit{Similarity}(a,i) = w(a,i),\ i \in K$$

- Step2: Predicting the rating that user a will give to all items the k neighbors have consumed but a has not. We Look for the item j with the best predicted rating

Method 2:

- Calculate similarity among the items
    - Cosine-Based Similarity
    - Correlation-Based Similarity
    - Adjusted Cosine Similarity
    - Jaccard Similarity

This method provides accurate recommendations and is easy to implement. The major drawback is the cold start problem when a new node is added.

#### 3.2.2.2 Item-based collaborative filtering

In item-based collaborative filtering, recommendations are provided based on identifying similar items consumed by the target user [9.2.f]. Item-based collaborative filtering has following steps:

- Calculate similarity among the items using one of the below options.
    - Cosine-Based Similarity
    - Correlation-Based Similarity
    - Adjusted Cosine Similarity
    - Jaccard Similarity

## 3.3 Related research to solve the problem (your own words or quoted phases with credits)

In ‘Similarity Based Collaborative Filtering Model for Movie Recommendation Systems’ research paper, user-based and item-based collaborative filtering is used to compute the target user’s ratings based on similar user ratings. Similar users are identified based on the similarity of other users with the active user. They have also experimented and analyzed the performances of similarity metrics: Cosine similarity, Euclidean distance similarity, Jaccard similarity and Pearson similarity.

## 3.4 Advantage/disadvantage of those research

**Advantages:** The research is experimenting and comparing the performances of various similarity metrics for collaborative-based filtering technique like Jaccard, Cosine, Pearson and Euclidean distance and suggesting which is better

**Disadvantages:** The paper focuses only on collaborative-based filtering technique and not content-based filtering technique. So, in a way we are missing out on the personalized recommendations.

## 3.5 Our solution to solve this problem

The first thing that needs to be addressed is a suitable platform which can handle interconnected data. So, there cannot be any other platform better than Neo4j. The approach that we would be adopting for the recommendation system is using both content based and collaborative based filtering models.

**Content based filtering model approach options**

- Jaccard similarity
- Cosine similarity
- Pearson similarity
- Euclidean distance

**Collaborative based filtering approach**

- **KNN**
    1. Identify K nearest neighbors using similarity metrics (**Cosine**/Pearson/Euclidean Distance)
    2. Recommend items based on the items consumed by K neighbors

## 3.6 Where our solution different from others

In 'Similarity Based Collaborative Filtering Model for Movie Recommendation Systems' research paper, they have implemented a similarity based collaborative model for movie recommendation using Python programming language and Jupyter notebook.

Our solution differs in a way it is implemented. The recommendation system would be built on the Neo4j database by using inbuilt functions in Neo4J GDS (Graph Data science library). We would be implementing content-based filtering, collaborative based filtering using Node similarity & KNN graph algorithm respectively.

## 3.7 why our solution is better

We are implementing our solution on Neo4j where graphs can be constructed and queried with ease comparatively. Also, the algorithms are built in.

By using content-based filtering technique we are providing personalized recommendations for the users where the model captures interests of only that particular user.

## 4. Hypothesis

### 4.1 Single/Multiple hypothesis

Our main hypothesis for this project is to build recommendation systems that are scalable and robust using techniques like Content-based filtering, Collaborative filtering (user-user and user-item) available on Graph database platforms (Neo4j) by apply various graph algorithms like Node Similarity, K Nearest Neighbor(KNN) on the top of the graph model built in Neo4J.

### 4.2 Positive (only for proof correctness) hypothesis

By analyzing the genre of the movies watched previously by the user, his neighbors and also the user ratings of each movie, we hypothesize to generate personalized recommendations for each user and test if we are able to provide the recommendation specific to that user based on his movie genre preferences and the ratings he provided or if they are generic to all the users.

## 5. Methodology

### 5.1 How to generate/collect input data

Our project focuses on the movie recommendation for each user. The dataset we use is movie dataset which is available on Kaggle. It includes information about the user, movies watched by each user, the genre of the movies and the individual user ratings through the years 1995 - 2015. The data is collected from 138,493 users selected at random and has about 20,000,263 ratings across 10,866 movies with each user rating at least 20 movies each. The data is available in 2 .csv files - movies, ratings.

**Dataset info:**

**Dataset 1: IMDB *Movies dataset*** - contains unique Movie IDs, their titles and the genre of each movie. It has ~10866 records

**Dataset 2: User Movie Ratings dataset -** contains unique User IDs, Movie IDs and the ratings given by each user on a scale of 1-5 scale along with timestamps.

## 5.2 How to solve the problem

1. **Load dataset into Neo4j** - The movie dataset which has 3 .csv files is to be loaded into the Neo4J Desktop using Cypher Query Language.
2. **Graph construction** - It is the process, where by defining the nodes and edges from the dataset, we construct a well-connected graph of nodes and relationships with properties and labels in the form of Cypher queries.
3. **Graph Algorithms** - Once the graph model is constructed, the next step is to apply various techniques such as Content-based filtering and Collaborative filtering models on the top of the graph model by using graph algorithms like Node similarity and K Nearest Neighbors that are built-in for Neo4J and analyze the movie dataset to generate recommendations for each user on which movie he should watch next based on his preferred movie genre and the user ratings.

## 5.3 Algorithm Design

The algorithmic design mainly focuses on application of graph algorithms on the graph structures. Neo4j provides various built in graph algorithms which represent different problem classes. Few of the graph algorithms are as below.

- *Node similarity algorithm*
- *KNN*

These can be modelled on the graph structures. We make use of above algorithms for techniques used in this paper which are as below.

**Content Based Filtering using Node similarity algorithm:**

This algorithm allows us to search and recommend contents that are similar to what that user has watched before. It does not take into account the other users' contents. In this use case of movie recommendation, we will recommend the next movie to be watched based on the details of what he has already watched. Using our movie data, one way we could define similarity is movies that have common genres. Of course, there are many more traits in addition to just genre that we can consider to compute similarity, such as actors and directors. We can use weighted sum to score the recommendations based on the number of actors, genres and directors they have in common to boost the score.

The algorithm is based on Jaccard similarity metrics. This is the robust way to quantify similarity. It plays an important role in generating personalized recommendations that allows us to quantify how similar 2 items are. The algorithm supports Directed, Undirected, homogenous, Heterogenous and Weighted graphs.

The Jaccard index is a number between 0 and 1 which shows similarity between 2 sets.

- If J = 1, then 2 sets are identical
- If J = 0, then 2 sets do not have a common element.

Higher the Jaccard index, it is more likely that user will enjoy the recommended movie shows a typical design/ flow can be seen below.

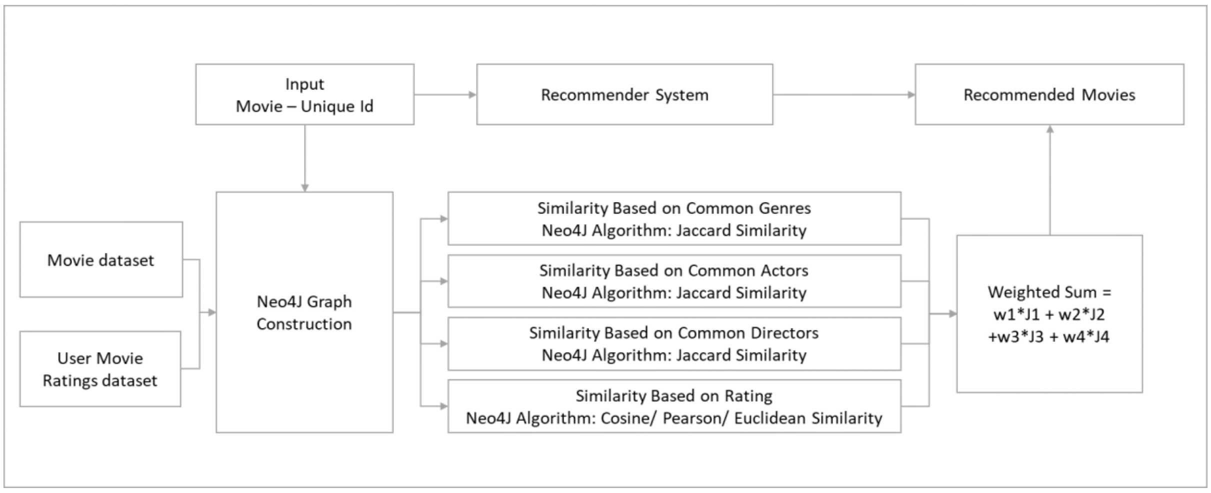

Input
Movie – Unique Id
Recommender System
Recommended Movies
Movie dataset
User Movie Ratings dataset
Neo4J Graph Construction
Similarity Based on Common Genres
Neo4J Algorithm: Jaccard Similarity
Similarity Based on Common Actors
Neo4J Algorithm: Jaccard Similarity
Similarity Based on Common Directors
Neo4J Algorithm: Jaccard Similarity
Similarity Based on Rating
Neo4J Algorithm: Cosine/ Pearson/ Euclidean Similarity
Weighted Sum = w1*J1 + w2*J2 +w3*J3 + w4*J4

Fig 2: Content Based Filtering using Node similarity algorithm flowchart

**Collaborative Based Filtering using Node similarity and KNN algorithm:**

**User based collaborative filtering -** An algorithm that considers users' interaction with products, with the assumption that other users will behave in a similar way. It takes into consideration similarity between users. With respect to movie dataset, it uses the preferences, ratings and actions of other users in the network to find movies to recommend.

**Step1:** Calculate the similarity between the users using Cosine/Pearson coefficient. The value of this index fluctuates between -1 and 1. If the returned value is -1 it means that the two users are totally different, while if the return value is 1 the two users are close to each other. To compare two users by the Cosine Similarity, we build two vectors that “represent” the users.

**Step2:** To recommend top k users whose tastes are similar to the user. We use KNN (K nearest neighbour) algorithm built in the GDS library of Neo4j on top of the cosine coefficient algorithm.

A typical design/ flow can be seen below.

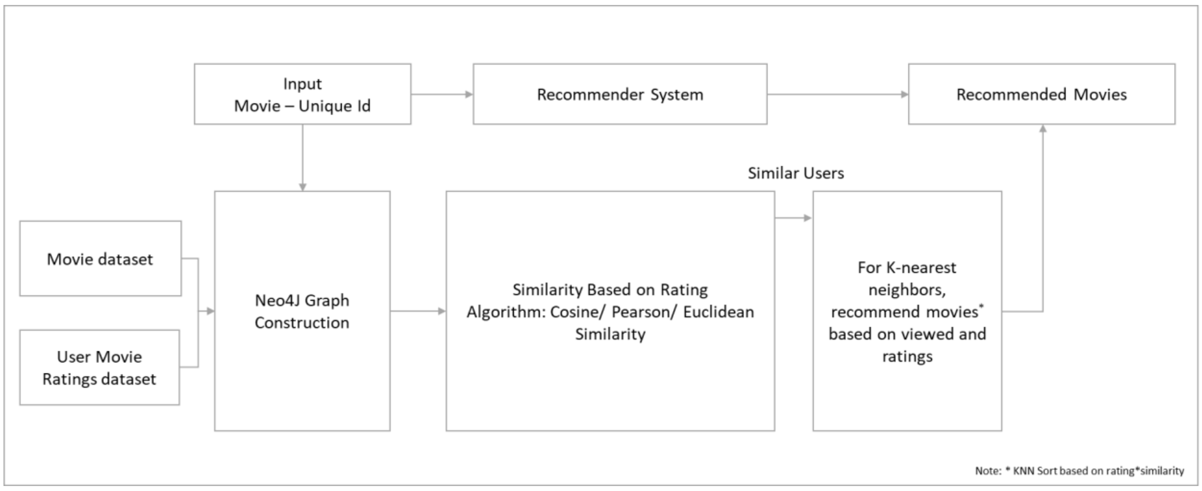

Input
Movie – Unique Id
Recommender System
Recommended Movies
Similar Users
Movie dataset
User Movie Ratings dataset
Neo4J Graph Construction
Similarity Based on Rating Algorithm: Cosine/ Pearson/ Euclidean Similarity
For K-nearest neighbors, recommend movies* based on viewed and ratings
Note: * KNN Sort based on rating*similarity

Fig 3: Collaborative Based Filtering using Node similarity and KNN algorithm

## 5.4 Language used

We build and analyze the graph model in Neo4J, a graph database using Cypher Query Language (CQL). Cypher is a graph query language which is very similar to SQL and can be used for efficient data querying from the graph.

## 5.5 Tools used

Neo4J Database

a. **Neo4j Desktop**

   It is the local instance of Neo4J with access to all the capabilities and features of Neo4j Enterprise Edition

b. **Graph Data Science (GDS) library**

   GDS library includes many graph algorithms used for path finding, centrality, similarity and Node embeddings. The algorithms are divided into categories each representing different problem classes.

   1. **Production-quality**

- Indicates that the algorithm has been tested with regards to stability and scalability.
- Algorithms in this tier are prefixed with gds.<algorithm>.

2. **Beta**
    - Indicates that the algorithm is a candidate for the production-quality tier.
    - Algorithms in this tier are prefixed with gds.beta.<algorithm>.
3. **Alpha**
    - Indicates that the algorithm is experimental and might be changed or removed at any time.
    - Algorithms in this tier are prefixed with gds.alpha.<algorithm>.

c. **Node Similarity**

Based on the neighborhoods and their properties, each similarity algorithm computes the similarity of pairs of nodes in a different way.

Node Similarity compares a set of nodes and consider them similar by checking if they share the same neighbors and computes pairwise similarities using Jaccard Similarity Score, or the Overlap coefficient

**d. K Nearest Neighbors**

The K-Nearest Neighbors algorithm compares the properties of each node and defines k nodes as its nearest neighbors if their properties are similar and then computes the distance between all node pairs and creates new relationships between each node and its k nearest neighbors.

### 5.6 How to generate output

Once we construct the graph models on the dataset, the next step is to apply the graph algorithms like Node similarity and K nearest Neighbors on the graph models.

Each algorithm operates in 4 execution modes - Stream, Stats, Mutate and Write.

- **Stream -** In this mode, for each relationship between 2 nodes a similarity score is calculated allowing the user to inspect the performance in Cypher.
- **Stats -** In this mode, the algorithm returns the summary of the result in a single row which is useful for evaluating the performance of the algorithm by inspecting computeMillis return item. This execution mode does not have any side effects.
- **Mutate -** This mode is the extension of stats mode but with an important side effect. It updates the named graph if there is any new relationship property which contains the similarity score for that relationship. This mode is especially used when multiple algorithms are used in conjunction.
- **Write -** The write mode creates a relationship for each pair of nodes as a property to the Neo4J database. The writeRelationshipType configuration parameter, we can see the type of relationship specified.

Once the nodes, edges and their properties are defined using the graph algorithms, by using techniques like Content-based filtering and Collaborative filtering to develop recommendation systems to provide timely and effective recommendations.

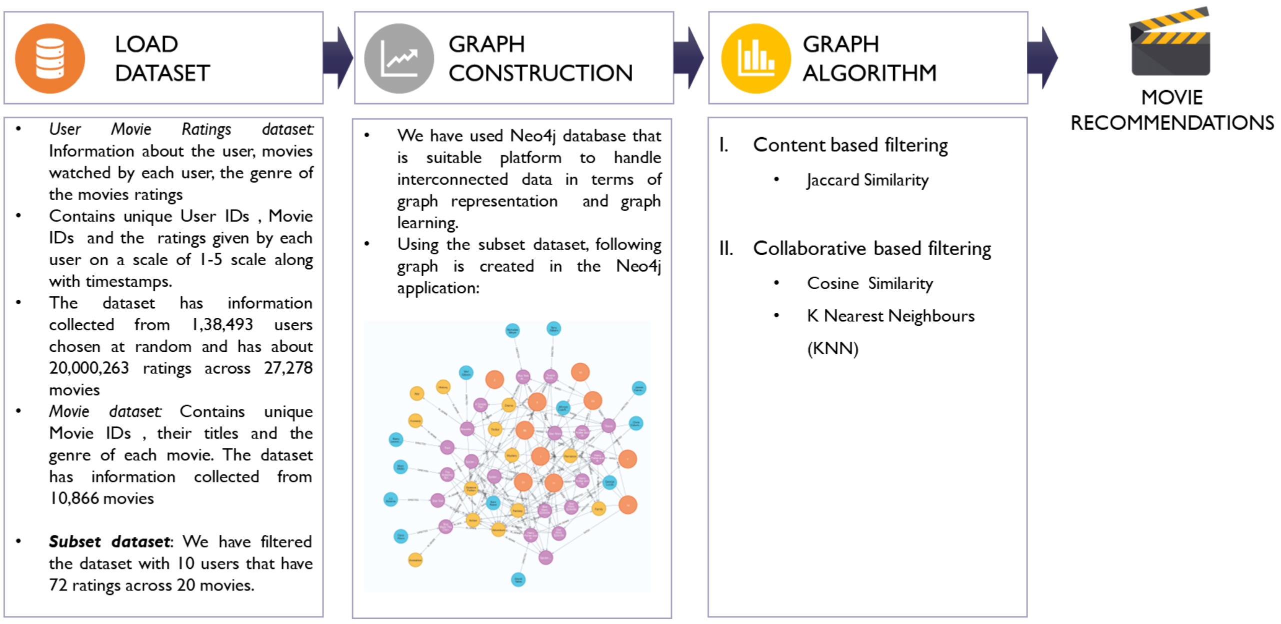
## 6. Implementation

### 6.1 Code (refer programming requirements)

Refer to the appendix section and submitted README file for code and necessary information.

## 6.2 Design document and flowchart

We have designed the flowchart for movie recommendation as mentioned below.



- *User Movie Ratings dataset:* Information about the user, movies watched by each user, the genre of the movies ratings
- Contains unique User IDs , Movie IDs and the ratings given by each user on a scale of 1-5 scale along with timestamps.
- The dataset has information collected from 1,38,493 users chosen at random and has about 20,000,263 ratings across 27,278 movies
- *Movie dataset:* Contains unique Movie IDs , their titles and the genre of each movie. The dataset has information collected from 10,866 movies
- ***Subset dataset***: We have filtered the dataset with 10 users that have 72 ratings across 20 movies.

- We have used Neo4j database that is suitable platform to handle interconnected data in terms of graph representation and graph learning.
- Using the subset dataset, following graph is created in the Neo4j application:

I. Content based filtering
  - Jaccard Similarity

II. Collaborative based filtering
  - Cosine Similarity
  - K Nearest Neighbours (KNN)

Fig 4: Flowchart for movie recommendation

### Graph Construction:

Below are the nodes and edges of the constructed graph on our data.

- Nodes created: User, Movie, Director, Genre
- Edges: RATED, DIRECTED, IN_GENRE
- User has rated the movie which is indicated by the edge 'RATED'
- Movie belonging to a genre is represented with the edge 'IN_GENRE'
- The director who directed the movie is represented by the relationship 'DIRECTED'

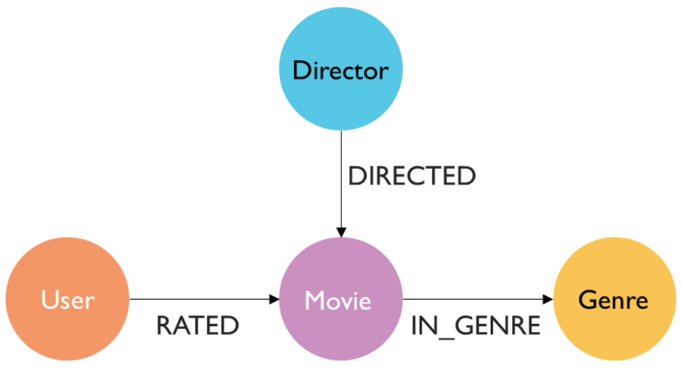


Fig 5: Nodes and Edges for the graph construction

Below is the heterogeneous directed graph of the subset data with 10 users.

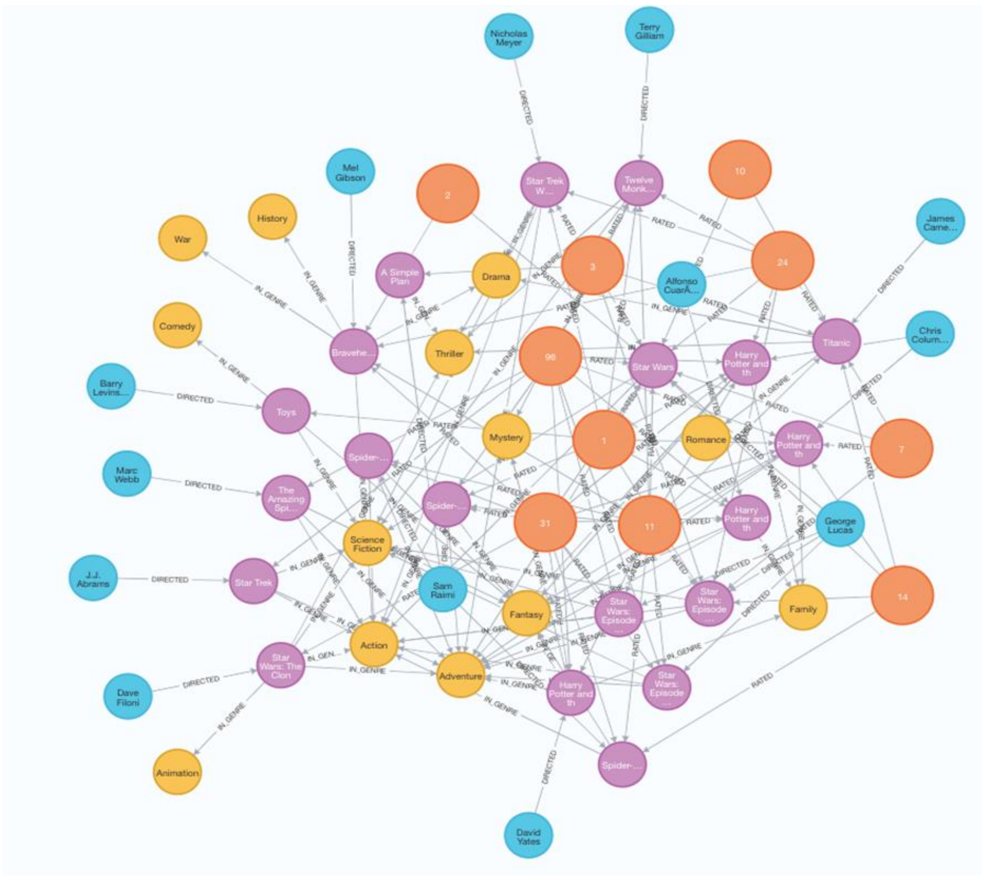
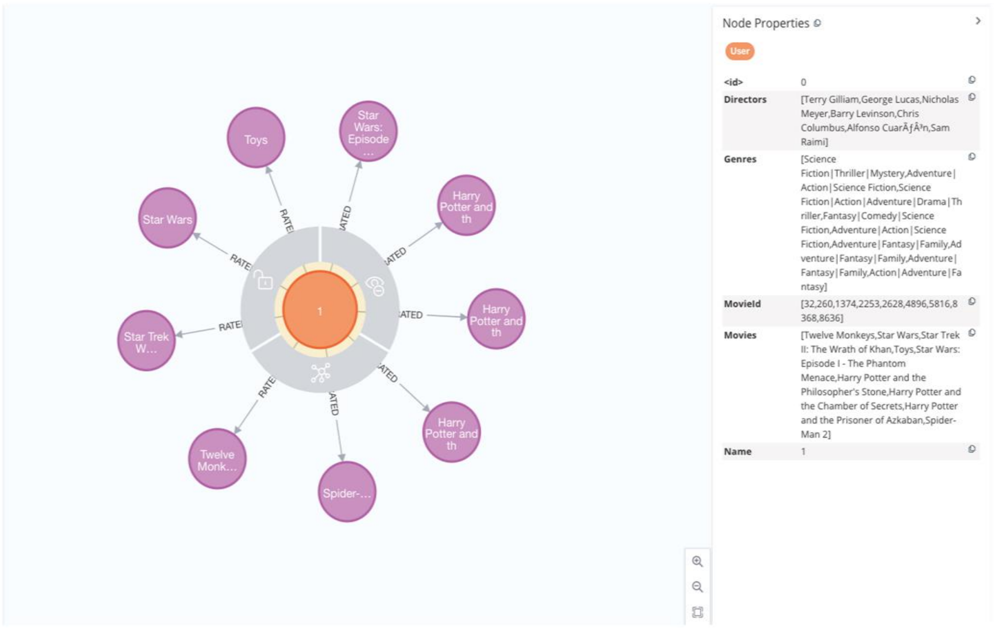


Fig 6: Heterogeneous directed graph for movie dataset

Below represents the node properties of a sample user



Node Properties

User

| | |
|---|---|
| <id> | 0 |
| Directors | [Terry Gilliam,George Lucas,Nicholas Meyer,Barry Levinson,Chris Columbus,Alfonso CuarÃƒÂ³n,Sam Raimi] |
| Genres | [Science Fiction\|Thriller\|Mystery,Adventure\|Action\|Science Fiction,Science Fiction\|Action\|Adventure\|Drama\|Thriller,Fantasy\|Comedy\|Science Fiction,Adventure\|Action\|Science Fiction,Adventure\|Fantasy\|Family,Adventure\|Fantasy\|Family,Adventure\|Fantasy\|Family,Action\|Adventure\|Fantasy] |
| MovieId | [32,260,1374,2253,2628,4896,5816,8368,8636] |
| Movies | [Twelve Monkeys,Star Wars,Star Trek II: The Wrath of Khan,Toys,Star Wars: Episode I - The Phantom Menace,Harry Potter and the Philosopher's Stone,Harry Potter and the Chamber of Secrets,Harry Potter and the Prisoner of Azkaban,Spider-Man 2] |
| Name | 1 |

Fig 7: Node properties for sample user

The implementation is done by defining the user1 to be the target user. Below is the graph representation for target user, 'User1' and its association with other nodes. It shows the movies rated/watched by User1, their genres and movie directors.

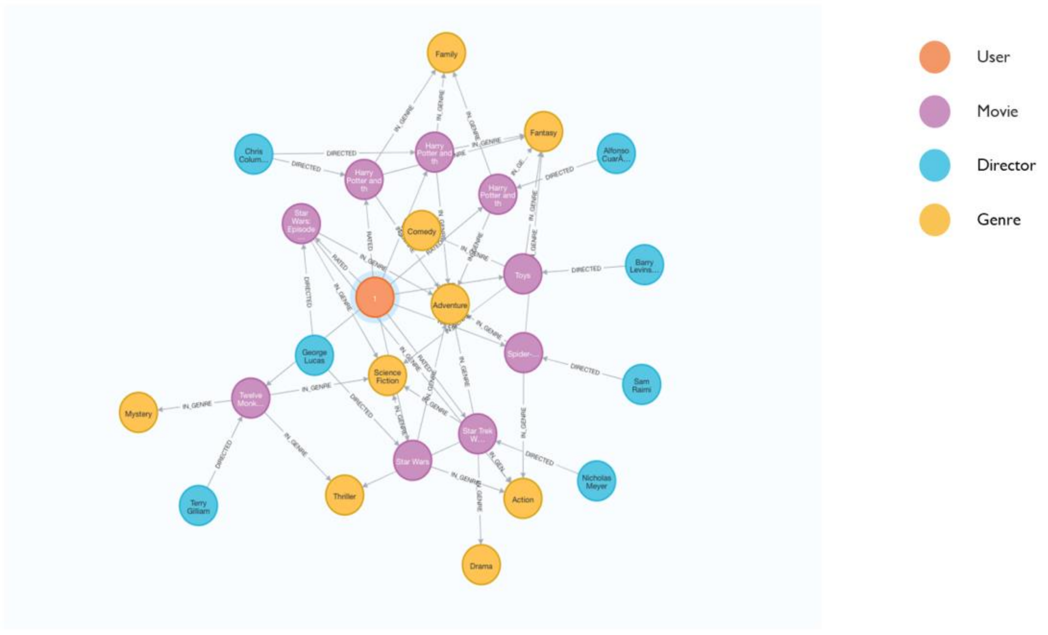
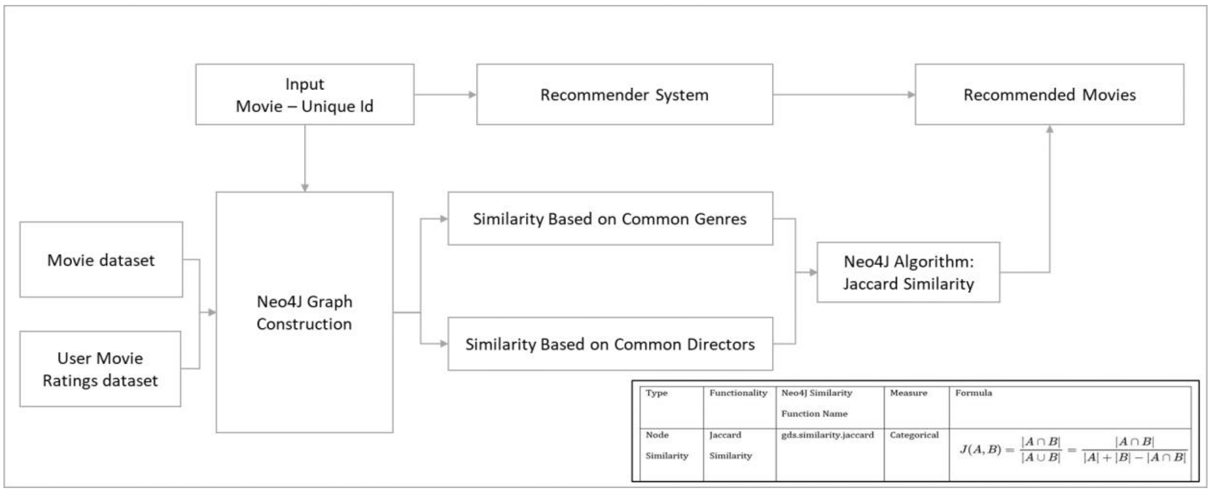


Fig 8: Heterogeneous directed graph for "User 1"

## Graph Algorithm

### I. Content-Based Filtering:

This algorithm allows us to search and recommend contents that are similar to what that user has done before.

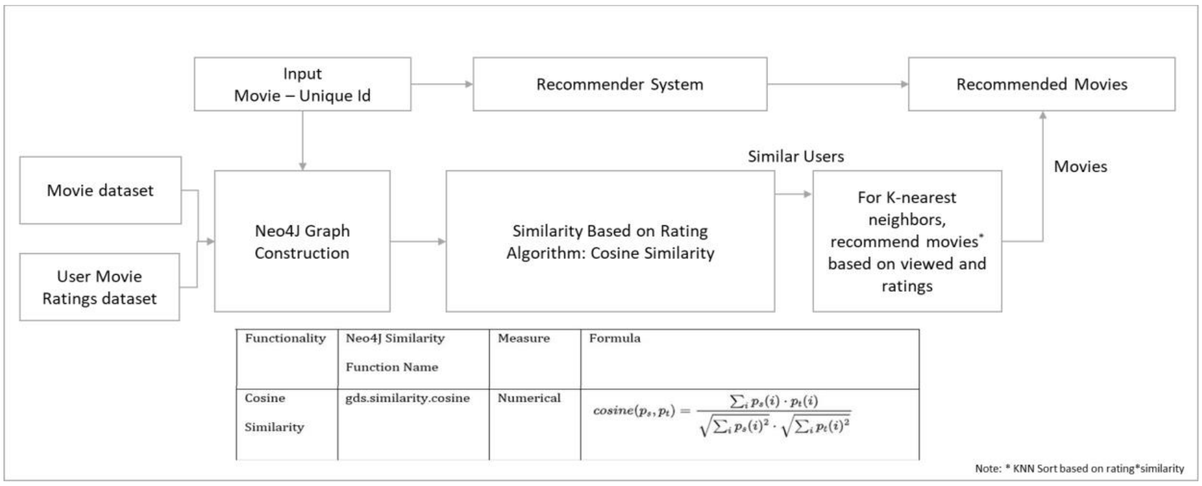

Input
Movie – Unique Id
Recommender System
Recommended Movies
Movie dataset
User Movie Ratings dataset
Neo4J Graph Construction
Similarity Based on Common Genres
Similarity Based on Common Directors
Neo4J Algorithm: Jaccard Similarity

| Type | Functionality | Neo4J Similarity Function Name | Measure | Formula |
|---|---|---|---|---|
| Node Similarity | Jaccard Similarity | gds.similarity.jaccard | Categorical | $J(A,B) = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert} = \frac{\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert - \lvert A \cap B \rvert}$ |

Fig 9: Content based filtering flowchart for target user

### II. User-based Collaborative Filtering:

An algorithm that considers users' interaction with products, with the assumption that other users will behave in a similar way. It takes into consideration similarity between users.


Input
Movie – Unique Id
Recommender System
Recommended Movies
Movie dataset
User Movie Ratings dataset
Neo4J Graph Construction
Similarity Based on Rating Algorithm: Cosine Similarity
Similar Users
For K-nearest neighbors, recommend movies* based on viewed and ratings
Movies

| Functionality | Neo4J Similarity Function Name | Measure | Formula |
|---|---|---|---|
| Cosine Similarity | gds.similarity.cosine | Numerical | $cosine(p_s, p_t) = \frac{\sum_i p_s(i) \cdot p_t(i)}{\sqrt{\sum_i p_s(i)^2} \cdot \sqrt{\sum_i p_t(i)^2}}$ |

Note: * KNN Sort based on rating*similarity

Fig 10: Collaborative filtering flowchart for target user

## 7. Data analysis and Discussion

### 7.1 Output generation

**Content Based Filtering:**

Recommendation system built using content-based filtering, recommends movies based on user profile i.e., on the choice of movies the user has watched previously. Analysing the content of movies, the user has watched previously will recommend similar movies. It does not consider other user preferences.

**Use Case 1: Recommendation based on User's genre preferences**

By considering the genres of the movies he watched previously, we have recommended movies that are of the same genres.

**Use Case 2: Weighted Recommendation based on User's genre or director preferences**

Next, we have recommended movies using the Weighted Algorithm method. Based on the genres and director of the movies they have in common; we have calculated a weighted sum by giving higher arbitrary value to the feature of high priority(genre) and lesser arbitrary value to feature of low priority (director). The movies with the highest weighted sum as score are recommended for that user.

**Use Case 3: Jaccard Similarity Recommendation based on User's genre or director preferences**
Finally for this project, we have used Jaccard similarity to get more personalized recommendations by measuring how similar two movies are and recommended the Top N movies which have the highest similarity metric.

**Jaccard Index:**

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The Jaccard index is a number between 0 and 1. The value indicates how similar the two movies are. If two movies do not have any similarity, then the Jaccard index is 0. The Jaccard is calculated by dividing the size of the intersection of two movies by the union of the two movies.

Here we have considered the Genre and Director of each movie and measured the Jaccard index for every 2 movies and recommended those movies which have the highest Jaccard Index value.

**Collaborative Filtering:**
Recommendation system built using collaborative filtering, recommends movies based on similarities between the users i.e., based on the preferences of the users that seem to have a similar taste in movies. For each target user, we will identify the most similar users to them and preferences of these similar users are then used to generate recommendations for the target user.

**Use Case 1: User-based Recommendation based on other similar user's movie preferences**

For this project, we have first compared 2 users (target user and similar user) that have watched similar movies and have given the other movies which similar user has watched as recommendations to target user.

**Use Case 2: Item- based Recommendation based on users average rating and genre preferences/**
Next, we have by calculating the average rating by target user, by comparing 2 users (target user and similar user) that have watched similar movies, have given the other movies which similar user has watched which have rating higher than the average rating given by the target user as recommendations to target user

**Use Case 3: User -based Cosine Similarity Recommendation based on users' ratings**
Finally, we have used Cosine similarity to get more personalized recommendations by measuring how similar two movies are and recommended the Top N movies which have the highest similarity metric.

**Cosine Similarity Index:**

$$similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from **0–1.**
Here, we have compared two movies based on their genres and director to measure how similar they are. However, in addition we have also considered the rating of the movie the users have rated and the weight of rating is also considered for calculation and recommended those movies which have the highest Cosine similarity value.

**Use Case 4: User-based KNN Recommendation based on similar user's movie preferences**

**KNN (K- Nearest Neighbor) algorithm:**
The K-Nearest Neighbors algorithm computes a distance between each node pairs in the graph and creates new relationships between each node and its k nearest neighbors. The K-Nearest Neighbors algorithm then compares properties of each node. The k nodes with most similar properties are defined as its k-nearest neighbors.

Here, by using the K nearest neighbor algorithm, we have found clusters of similar users based on common genres and directors. Then, by finding the nearest similar user to the target user, we have recommended the other movies the nearest neighbor has watched to the target user.

Below are the implementation steps for KNN.

- **Stream** - the algorithm returns the similarity score for each relationship.
- **Stats** - the algorithm returns a single row containing a summary of the algorithm result.
- **Mutate** - updates the named graph with a new relationship property containing the similarity score for that relationship.

- **Write** - for each pair of nodes it creates a relationship with its similarity score as a property to the Neo4j database. Each new relationship stores the similarity score between the two nodes it represents.

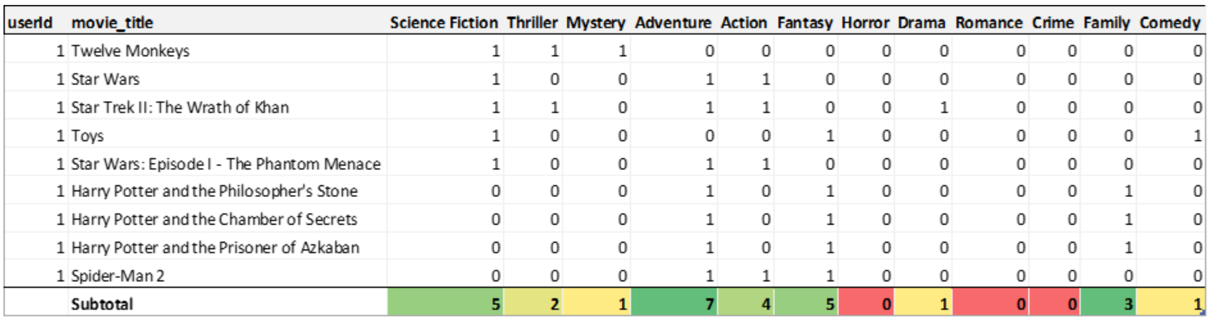
## 7.2 Output analysis

Let's take **"User 1"** as the target user and perform various algorithms to recommend movies he can watch next.

### Content Based Filtering:

### Use Case 1: Recommendation based on User's genre preferences

This is a simple content-based filtering recommendation without using any formulas/functions. We are recommending movies similar to that genre the user has already watched except the movies that he has already watched.

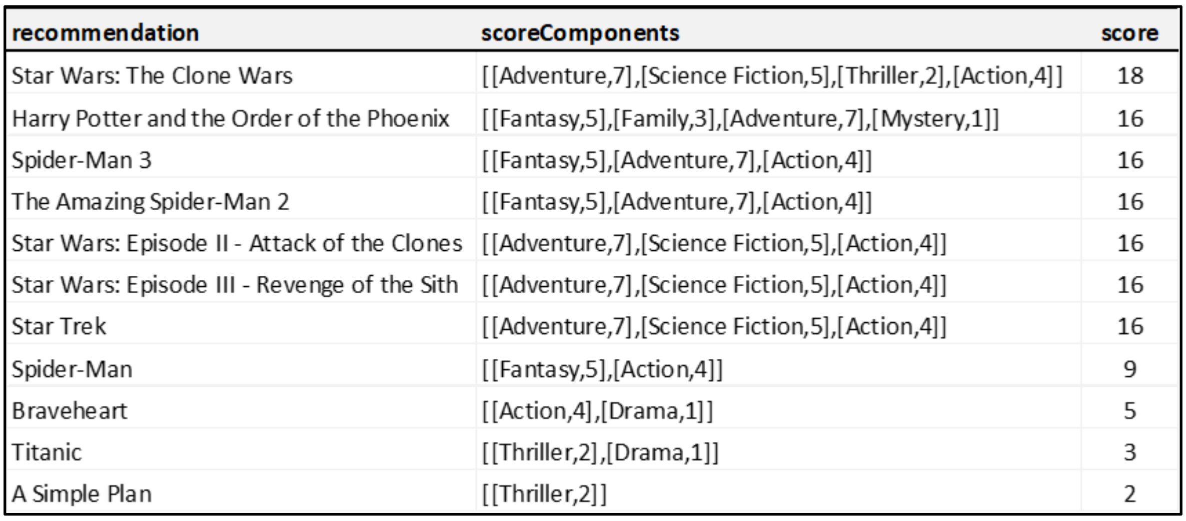


| userId | movie_title | Science Fiction | Thriller | Mystery | Adventure | Action | Fantasy | Horror | Drama | Romance | Crime | Family | Comedy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Twelve Monkeys | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | Star Wars | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | Star Trek II: The Wrath of Khan | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | Toys | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | Star Wars: Episode I - The Phantom Menace | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | Harry Potter and the Philosopher's Stone | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | Harry Potter and the Chamber of Secrets | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | Harry Potter and the Prisoner of Azkaban | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | Spider-Man 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **Subtotal** | **5** | **2** | **1** | **7** | **4** | **5** | **0** | **1** | **0** | **0** | **3** | **1** |

We can see from the above image that "User 1" has watched movies of 9 different genres - Adventure, Science Fiction, Fantasy, Action, Family, Thriller, Mystery, and Drama (in decreasing order of preference). The recommendations are based on the weightage of the genre preference of the user.

Form the Neo4j output image below,
The numeric value in the score components is the number of movies "User 1" has watched that belong to that specific genre. The score in the output is the total number of movies "User 1" has watched which belong to those genres.

Neo4j Output:



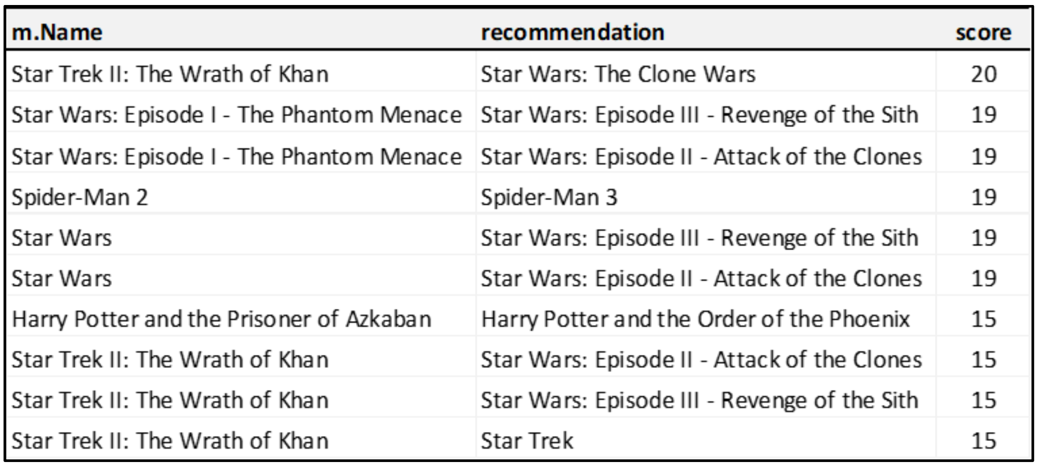
| recommendation | scoreComponents | score |
|---|---|---|
| Star Wars: The Clone Wars | [[Adventure,7],[Science Fiction,5],[Thriller,2],[Action,4]] | 18 |
| Harry Potter and the Order of the Phoenix | [[Fantasy,5],[Family,3],[Adventure,7],[Mystery,1]] | 16 |
| Spider-Man 3 | [[Fantasy,5],[Adventure,7],[Action,4]] | 16 |
| The Amazing Spider-Man 2 | [[Fantasy,5],[Adventure,7],[Action,4]] | 16 |
| Star Wars: Episode II - Attack of the Clones | [[Adventure,7],[Science Fiction,5],[Action,4]] | 16 |
| Star Wars: Episode III - Revenge of the Sith | [[Adventure,7],[Science Fiction,5],[Action,4]] | 16 |
| Star Trek | [[Adventure,7],[Science Fiction,5],[Action,4]] | 16 |
| Spider-Man | [[Fantasy,5],[Action,4]] | 9 |
| Braveheart | [[Action,4],[Drama,1]] | 5 |
| Titanic | [[Thriller,2],[Drama,1]] | 3 |
| A Simple Plan | [[Thriller,2]] | 2 |

Fig 11: Neo4j output for use case 1 using content-based filtering

**Use Case 2: Weighted Recommendation based on User's genre or director preferences**

Here, by comparing 2 movies that have either genre or director in common, we have calculated a weighted sum by giving higher arbitrary value (5) to the feature of high priority(genre) and lesser arbitrary value (4) to feature of low priority (director) and calculating the score. The movies with the highest weighted sum as score are recommended for that user.

Neo4j Output:



| m.Name | recommendation | score |
|---|---|---|
| Star Trek II: The Wrath of Khan | Star Wars: The Clone Wars | 20 |
| Star Wars: Episode I - The Phantom Menace | Star Wars: Episode III - Revenge of the Sith | 19 |
| Star Wars: Episode I - The Phantom Menace | Star Wars: Episode II - Attack of the Clones | 19 |
| Spider-Man 2 | Spider-Man 3 | 19 |
| Star Wars | Star Wars: Episode III - Revenge of the Sith | 19 |
| Star Wars | Star Wars: Episode II - Attack of the Clones | 19 |
| Harry Potter and the Prisoner of Azkaban | Harry Potter and the Order of the Phoenix | 15 |
| Star Trek II: The Wrath of Khan | Star Wars: Episode II - Attack of the Clones | 15 |
| Star Trek II: The Wrath of Khan | Star Wars: Episode III - Revenge of the Sith | 15 |
| Star Trek II: The Wrath of Khan | Star Trek | 15 |

Fig 12: Neo4j output for use case 2 using content-based filtering

### Use Case 3: Jaccard Similarity Recommendation based on User's genre or director preferences

Here, we have considered the Genre and Director of each movie and measured the Jaccard index for every 2 movies and recommended those movies which have the highest Jaccard Index value.
From the Neo4j output image below, we can see the list of recommendations for "User 1". The column "m.Name" has the list of movies "User 1" has watched. The column "s1" has the genres each movie belongs to. The column "other.Name" has other movies to compare with movies in the "m.Name" column. The column "s2" has the genres of each movie in the "other.Name" column. In the "intersection" column, we have the number of genres in common for each movie in "m.Name " and "other.Name " columns. The "Jaccard" column has the Jaccard similarity index value calculated for those movies.

Neo4j Output:

| m.Name | s1 | other.Name | s2 | intersection | jaccard |
|---|---|---|---|---|---|
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Wars: Episode III - Revenge of the Sith | [Action,Science Fiction,George Lucas,Adventure] | 4 | 1 |
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Wars: Episode II - Attack of the Clones | [Action,George Lucas,Adventure,Science Fiction] | 4 | 1 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | Spider-Man 3 | [Adventure,Action,Fantasy,Sam Raimi] | 4 | 1 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Wars: Episode III - Revenge of the Sith | [Action,Science Fiction,George Lucas,Adventure] | 4 | 1 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Wars: Episode II - Attack of the Clones | [Action,George Lucas,Adventure,Science Fiction] | 4 | 1 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | Spider-Man | [Action,Fantasy,Sam Raimi] | 3 | 0.75 |
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Trek | [J.J. Abrams,Action,Science Fiction,Adventure] | 3 | 0.6 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | The Amazing Spider-Man 2 | [Action,Marc Webb,Adventure,Fantasy] | 3 | 0.6 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Trek | [J.J. Abrams,Action,Science Fiction,Adventure] | 3 | 0.6 |
| Harry Potter and the Prisoner of Azkaban | [Alfonso CuarÃƒÂ³n,Fantasy,Family,Adventure] | Harry Potter and the Order of the Phoenix | [Fantasy,Mystery,Adventure,Family,David Yates] | 3 | 0.5 |

Fig 13: Neo4j output for use case 3 using content-based filtering

## Collaborative Filtering:

### Use Case 1: Recommendation based on other similar user's movie preferences

Here, by comparing the "User 1" with every other user who has watched similar movies, we have given the movies which other similar users have watched except those except those "User 1" has already watched as recommendations.
From the Neo4j output below, we can see the list of recommended movies for "User 1".

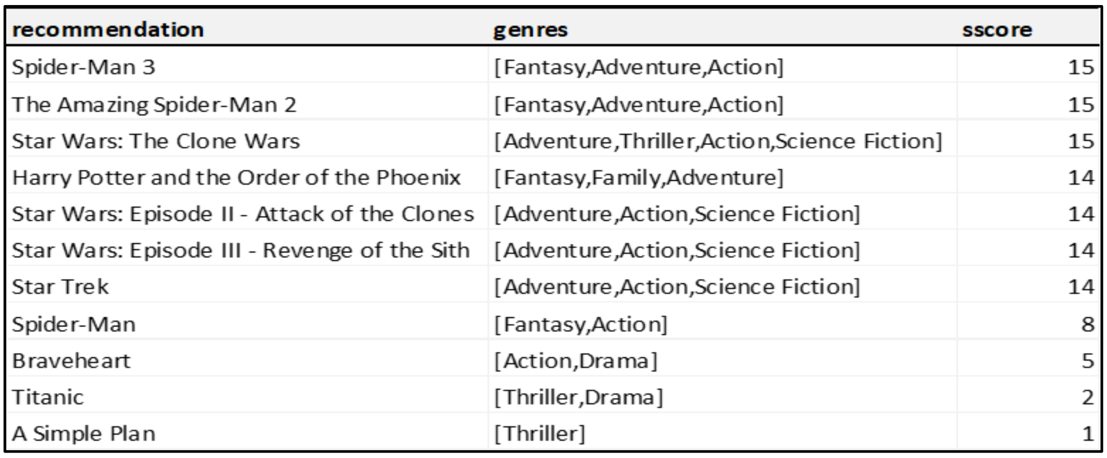
Neo4j Output:

| rec.Name |
|---|
| Star Wars: Episode III - Revenge of the Sith |
| Harry Potter and the Order of the Phoenix |
| Spider-Man 3 |
| Titanic |
| Spider-Man |
| Star Wars: Episode II - Attack of the Clones |
| Braveheart |
| Star Trek |
| The Amazing Spider-Man 2 |
| Star Wars: The Clone Wars |
| A Simple Plan |

Fig 14: Neo4j output for use case 1 using collaborative filtering

**Use Case 2: Recommendation based on users average rating and genre preferences**
Here, first we calculate the average rating given by "User 1" for all the movies he has watched. Now by comparing "User 1" with every other user who has watched similar movies, we are giving movies that the other similar users have watched with ratings higher than the average rating given by "User 1" as recommendations to "User 1".
The numeric value in the score is the total number of movies "User 1" has watched that belong to those specific genres.

Neo4j Output:

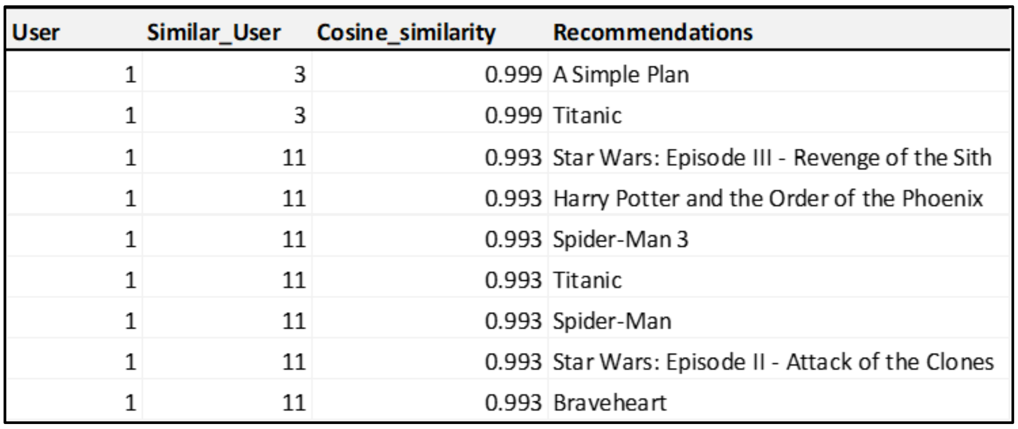


| recommendation | genres | sscore |
|---|---|---|
| Spider-Man 3 | [Fantasy,Adventure,Action] | 15 |
| The Amazing Spider-Man 2 | [Fantasy,Adventure,Action] | 15 |
| Star Wars: The Clone Wars | [Adventure,Thriller,Action,Science Fiction] | 15 |
| Harry Potter and the Order of the Phoenix | [Fantasy,Family,Adventure] | 14 |
| Star Wars: Episode II - Attack of the Clones | [Adventure,Action,Science Fiction] | 14 |
| Star Wars: Episode III - Revenge of the Sith | [Adventure,Action,Science Fiction] | 14 |
| Star Trek | [Adventure,Action,Science Fiction] | 14 |
| Spider-Man | [Fantasy,Action] | 8 |
| Braveheart | [Action,Drama] | 5 |
| Titanic | [Thriller,Drama] | 2 |
| A Simple Plan | [Thriller] | 1 |

Fig 15: Neo4j output for use case 2 using collaborative filtering

**Use Case 3: Cosine Similarity Recommendation based on users' ratings**
Finally, here we have used Cosine similarity to get more personalized recommendations. We have taken the list of movies "User 1" has watched and compared them with all other movie other users have watched based on their genres and director to measure how similar and recommended the Top N movies which have the highest cosine similarity metric.

Neo4j Output:

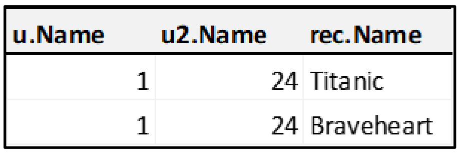


| User | Similar_User | Cosine_similarity | Recommendations |
|---|---|---|---|
| 1 | 3 | 0.999 | A Simple Plan |
| 1 | 3 | 0.999 | Titanic |
| 1 | 11 | 0.993 | Star Wars: Episode III - Revenge of the Sith |
| 1 | 11 | 0.993 | Harry Potter and the Order of the Phoenix |
| 1 | 11 | 0.993 | Spider-Man 3 |
| 1 | 11 | 0.993 | Titanic |
| 1 | 11 | 0.993 | Spider-Man |
| 1 | 11 | 0.993 | Star Wars: Episode II - Attack of the Clones |
| 1 | 11 | 0.993 | Braveheart |

Fig 16: Neo4j output for use case 3 using collaborative filtering

**Use Case 4: KNN Recommendation based on similar user's movie preferences**
**KNN (K- Nearest Neighbor) algorithm:** By using K- nearest neighbor algorithm, we have found clusters of similar users to "User 1" based on common genres and directors. Then, by finding the nearest similar user to the "User 1", we have recommended the other movies the nearest neighbor has watched as recommendations.

Neo4j Output:



| u.Name | u2.Name | rec.Name |
|---|---|---|
| 1 | 24 | Titanic |
| 1 | 24 | Braveheart |

Fig 17: Neo4j output for use case 4 using collaborative filtering

## 7.3 Compare output against hypothesis

The hypothesis of our project is to build a recommendation system that recommends movies to the users based on content based filtering and collaborative based filtering technique by applying various graph algorithms like Node Similarity and K Nearest Neighbour (KNN) on the top of the graph model built in Neo4J.

We have successfully built a movie recommendation system that recommends movies to users based on two filtering techniques.

- **Content based filtering**

  We have used the node similarity algorithm – Jaccard Similarity to find the similarity score ranging from 0 to 1 and recommend based on the order of the highest similarity score.

- **Collaborative based filtering**
  - We have used the node similarity algorithm – Cosine Similarity to find the similarity score ranging from 0 to 1 and recommend based on the order of the highest similarity score.
  - We have used the Graph Algorithm K Nearest Neighbors(KNN) to recommend the top movies to the user based on the other user's preferences

## 7.4 Discussion

Building a real-time recommendation system using Graph database is highly efficient as it outperforms relational and other NoSQL data stores when it comes to connecting large volumes of data (and connected data in general) and to gain insight into user choices. Graph databases are now used as core technology platforms for internet giants like Google, Facebook and LinkedIn as instead of building in house data stores from scratch and off-the-shelf graph databases, Neo4j makes it easy and available to any business wanting to make the most of real-time recommendation engines.

We have used Neo4j as it is very efficient to understand user's movie preferences by considering multiple factors like genres, directors and user ratings etc., and provide a perfect tool for movie recommendations. The data model generated from using Neo4j is organized and more understandable than one with a traditional relational database or other NoSQL database as it is very efficient to model and manage large movie data with associated edges in a simple and intuitive way.

We have used a subset of data with 10 users for our analysis as it makes it easy to build the graph model and verify the results of recommendations of content based and collaborative filtering techniques.

We had initially thought of building a recommendation system using GNN- Graph Neural Networks as well, however due to the complexity involved to create node embeddings and decode them using algorithms in limited time we have reduced the scope to Content and Collaborative filtering.

# 8. Conclusions and Recommendations

## 8.1 Summary and Conclusions

We have successfully implemented content based and collaborative based filtering techniques within Neo4j using node similarity and KNN graph algorithms.

Each of the models has its own advantages and disadvantages. A Content-Based filtering model does not need other users' data, since the recommendations are specific to a particular user. This makes it easier to scale down the same to a large number of users. A similar cannot be done for Collaborative Filtering Methods. Collaborative filtering methods use other user's preference and hence gives exposure to various other movies.

We believe content-based filtering outperforms collaborative based filtering technique as content-based focuses on the history of the same user and hence is personalised.

## 8.2 Recommendations for future studies

- There are always limitations to the current state of recommendation systems that can be researched and addressed.

- The methods face First Rater Problem (cold start problem) which is difficulty of making recommendations when users are new and also sparsity of data meaning too less data to understand the preferences of users which remains a great challenge. This needs to be studied further to analyse the measures that can be taken to address this within Neo4j.

- The movie recommendation system can also be developed using Graph Neural Network (GNN) within Neo4j where Graph Sage algorithm can be used to develop the node embeddings. Further study and analysis need to be done on the choice of the algorithm to decode the embeddings to provide the necessary recommendations.

# 9. Bibliography

## 9.1 Research Papers:

a. Similarity Based Collaborative Filtering Model for Movie Recommendation Systems
b. Design and Implementation of Movie Recommender System Based on Graph Database
c. Graph Neural Networks in Recommender Systems: A Survey
d. A Survey on Session-based Recommender Systems
e. Heterogeneous Global Graph Neural Networks for Personalized Session-based Recommendation

2. Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions
3. Exploiting Group Information for Personalized Recommendation with Graph Neural Networks
4. An Efficient and Effective Framework for Session-based Social Recommendation

## 9.2 Articles/ Web Resources:

a. https://guides.neo4j.com/sandbox/recommendations
b. https://neo4j.com/docs/graph-data-science/current/algorithms/similarity-functions/
c. https://47billion.com/blog/recommendation-system-using-graph-database/
d. https://www.kaggle.com/code/saurav9786/recommender-system-using-amazon-reviews/notebook
e. https://neo4j.com/docs/graph-data-science/current/algorithms/similarity-functions/
f. https://neo4j.com/docs/graph-data-science/current/machine-learning/node-embeddings/graph-sage/
g. https://neo4j.com/docs/graph-data-science/current/algorithms/knn/
h. https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f

# 10. Appendices

## 10.1 Program source code with documentation

- Use the below code to import data and create above defined nodes and relationships

```
LOAD CSV WITH HEADERS FROM 'FILE:/Data_05232022.csv' as row with row where
row.userId is not null
            MERGE(u:User{Name:row.userId})
            MERGE(d:Director{Name:row.director})
            MERGE(m:Movie{Name:row.movie_title})
            MERGE (u) -[:RATED{rate:row.rating}]->(m)
            MERGE (d)-[:DIRECTED]->(m)
            FOREACH (Genre IN split(row.genres, '|') |
            MERGE (g:Genre {Name: Genre})
            MERGE (m)-[:IN_GENRE]->(g)
)
WITH u, collect(row.movie_title) as mov_agg, collect(row.genres) as gen_agg,
collect(distinct row.director) as dir_agg,
collect(toInteger(row.movieId)) as movieid_agg
SET u.Movies = mov_agg, u.Genres = gen_agg, u.Directors = dir_agg,
u.MovieId = movieid_agg
```

- View the graph that you created:

```
MATCH (n) RETURN (n)
```

I. Content based filtering
Provide recommendation based on the history of the use without considering other users.

- **Use-Case-1:** Recommendation based on user's genre preference.
This is a simple content-based filtering recommendation without using any formulas/functions. We are recommending movies similar to those genre the user has already watched except the movies that he has already watched.

```
MATCH (u:User {Name: "1"})-[r:RATED]->(m:Movie),
(m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-(rec:Movie)
WHERE NOT EXISTS( (u)-[:RATED]->(rec) )
WITH rec, [g.Name, COUNT(*)] AS scores
RETURN rec.Name AS recommendation,
COLLECT(scores) AS scoreComponents,
REDUCE (s=0,x in COLLECT(scores) | s+x[1]) AS score
ORDER BY score DESC
```

- **Use-Case-2:** Weighted recommendation based on user's genre or director preference.
Recommendation based on the movies genres and directors the users has already watched. We have used the weighted technique where more weight has been assigned to genre

```
MATCH (u:User {Name: "1"})-[r:RATED]->(m:Movie)-[:IN_GENRE]-(t)-
[:IN_GENRE]-(rec:Movie)
WHERE NOT EXISTS( (u)-[:RATED]->(rec) )
WITH m, rec, COUNT(*) AS gs
OPTIONAL MATCH (u:User {Name: "1"})-[r:RATED]->(m:Movie)-[d:DIRECTED]-
(t)-[:DIRECTED]-(rec:Movie)
WITH m, rec, gs, COUNT(d) AS ds
RETURN m.Name, rec.Name AS recommendation, (5*gs)+(4*ds) AS score
ORDER BY score DESC
```

- **Use-Case-3:** Jaccard similarity recommendation based on user's genre or director preference

Recommendation based on the movies genres/directors the users has already watched. We are using Jaccard based similarity index to provide recommendation.

```
MATCH (u:User {Name: "1"})-[r:RATED]->(m:Movie)-[:IN_GENRE|DIRECTED]-
(t)-[:IN_GENRE|DIRECTED]-(other:Movie)
WHERE NOT EXISTS( (u)-[:RATED]->(other))
WITH m, other, COUNT(t) AS intersection, COLLECT(t.Name) AS i
MATCH (m)-[:IN_GENRE|DIRECTED]-(mt)
WITH m, other, intersection,i, COLLECT(mt.Name) AS s1
MATCH (other)-[:IN_GENRE|DIRECTED]-(ot)
WITH m,other,intersection,i, s1, COLLECT(ot.Name) AS s2
WITH m,other,intersection,s1,s2
WITH m,other,intersection,s1+[x IN s2 WHERE NOT x IN s1] AS union, s1, s2
RETURN m.Name,
other.Name,intersection,s1,s2,((1.0*intersection)/SIZE(union)) AS jaccard
ORDER BY jaccard DESC
```

II. Collaborative filtering
Provide recommendation to a user based on other similar users.

- **Use-Case-1:** User based collaborative filtering: Recommendation based on other similar users.
This is a simple collaborative filtering recommendation without using any formulas/functions. We are recommending movies to a user based on other similar users

```
MATCH (u:User {Name: "1"})-[:RATED]->(:Movie)<-[:RATED]-(o:User)
MATCH (o)-[:RATED]->(rec:Movie)
WHERE NOT EXISTS( (u)-[:RATED]->(rec))
RETURN DISTINCT rec.Name
```

- **Use-Case-2:** Item based collaborative filtering: Recommending movies based on users average ratings and genre preference.
For a particular user, what genres have a higher-than-average rating? Use this to score similar movies

```
MATCH (u:User {Name: "1"})-[r:RATED]->(m:Movie)
WITH u, avg(toInteger(r.rate) )AS mean
MATCH (u)-[r:RATED]->(m:Movie)-[:IN_GENRE]->(g:Genre)
WHERE (toInteger(r.rate) > mean)
WITH u, g, COUNT(*) AS score
MATCH (g)<-[:IN_GENRE]-(rec:Movie)
WHERE NOT EXISTS((u)-[:RATED]->(rec))
RETURN rec.Name AS recommendation,COLLECT(DISTINCT g.Name) AS
genres, SUM(score) AS sscore
ORDER BY sscore DESC
```

- **Use-Case-3:** User based collaborative filtering: Cosine Similarity recommendation based on user ratings
  Recommending movies based on most similar user preferences to user 1 on user ratings by using cosine similarity function.

```
MATCH (p1:User{Name:'1'})-[x:RATED]->(movie)<-[x2:RATED]-(p2:User)
WHERE p2 <> p1
WITH p1, p2, collect(toFloat(x.rate)) AS p1Ratings, collect(toFloat(x2.rate)) AS
p2Ratings
WHERE size(p1Ratings) > 1
MATCH (p2)-[:RATED]->(rec:Movie)
WHERE NOT EXISTS ((p1)-[:RATED]->(rec))
RETURN p1.Name AS User,p2.Name AS Similar_User,
round(gds.similarity.cosine(p1Ratings, p2Ratings),3) AS Cosine_similarity,
rec.Name AS Recommendations
ORDER BY Cosine_similarity DESC
```

- **Use-Case-4:** User based collaborative filtering: KNN recommendation based on similar user's movie preferences.
  We are recommending movies to the user based on other similar user's movie preferences. We are using KNN model to generate recommendations.

```
CALL gds.graph.project(
  'myGraph',
  {
    Person: {
      label: 'User',
      properties: ['MovieId']
    }
  },
  '*'
);

CALL gds.knn.write.estimate('myGraph', {
 nodeProperties: ['MovieId'],
 writeRelationshipType: 'SIMILAR',
 writeProperty: 'score',
 topK: 1
})
YIELD nodeCount, bytesMin, bytesMax, requiredMemory

CALL gds.knn.stream('myGraph', {
  topK: 1,
  nodeProperties: ['MovieId'],
  // The following parameters are set to produce a deterministic result
  randomSeed: 1337,
  concurrency: 1,
  sampleRate: 1.0,
  deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).Name AS Person1,
gds.util.asNode(node2).Name AS Person2, similarity
ORDER BY similarity DESCENDING, Person1, Person2

CALL gds.knn.stats('myGraph', {topK: 1, concurrency: 1, randomSeed: 42,
nodeProperties: ['MovieId']})
YIELD nodesCompared, similarityPairs


CALL gds.knn.mutate('myGraph', {
  mutateRelationshipType: 'SIMILAR',
  mutateProperty: 'score',
  topK: 1,
  randomSeed: 42,
  concurrency: 1,
  nodeProperties: ['MovieId']
})
YIELD nodesCompared, relationshipsWritten


CALL gds.knn.write('myGraph', {
  writeRelationshipType: 'SIMILAR',
  writeProperty: 'score',
  topK: 1,
  randomSeed: 42,
  concurrency: 1,
  nodeProperties: ['MovieId']
})
YIELD nodesCompared, relationshipsWritten


MATCH (u)-[:SIMILAR]->(u2)
RETURN u,u2

MATCH (u:User{Name:'1'})-[:SIMILAR]->(u2)
RETURN u2.Movies

MATCH (u:User{Name:'1'})-[:SIMILAR]->(u2)
MATCH (u2)-[:RATED]->(rec:Movie)
WHERE NOT EXISTS((u)-[:RATED]->(rec))
RETURN rec
```

## 10.2 Input/Output listing

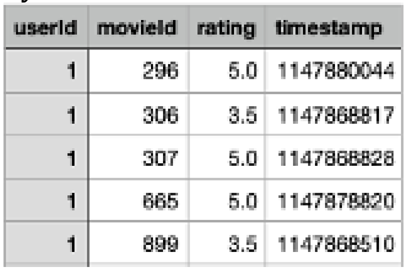
### Input:

*Movie dataset* – contains unique Movie IDs, their titles and the genre of each movie

| movieId | movie_title | cast | director | genres |
|---|---|---|---|---|
| **117529** | Jurassic World | Chris Pratt\|Bryce Dallas Howard\|Irrfan Khan\|Vincent D'Onofrio\|Nick Robinson | Colin Trevorrow | Action\|Adventure\|Science Fiction\|Thriller |
| **122882** | Mad Max: Fury Road | Tom Hardy\|Charlize Theron\|Hugh Keays-Byrne\|Nicholas Hoult\|Josh Helman | George Miller | Action\|Adventure\|Science Fiction\|Thriller |
| **130490** | Insurgent | Shailene Woodley\|Theo James\|Kate Winslet\|Ansel Elgort\|Miles Teller | Robert Schwentke | Adventure\|Science Fiction\|Thriller |
| **130634** | Furious 7 | Vin Diesel\|Paul Walker\|Jason Statham\|Michelle Rodriguez\|Dwayne Johnson | James Wan | Action\|Crime\|Thriller |
| **139385** | The Revenant | Leonardo DiCaprio\|Tom Hardy\|Will Poulter\|Domhnall Gleeson\|Paul Anderson | Alejandro GonzÃ¡lez IÃ±Ã¡rritu | Western\|Drama\|Adventure\|Thriller |
| **120799** | Terminator Genisys | Arnold Schwarzenegger\|Jason Clarke\|Emilia Clarke\|Jai Courtney\|J.K. Simmons | Alan Taylor | Science Fiction\|Action\|Thriller\|Adventure |

Fig 18: Dataset 1

*User Movie Ratings dataset* - contains unique User IDs, Movie IDs and the ratings given by each user on a scale of 1-5 scale along with timestamps.



| userId | movieId | rating | timestamp |
|---|---|---|---|
| 1 | 296 | 5.0 | 1147880044 |
| 1 | 306 | 3.5 | 1147868817 |
| 1 | 307 | 5.0 | 1147868828 |
| 1 | 665 | 5.0 | 1147878820 |
| 1 | 899 | 3.5 | 1147868510 |

Fig 19: Dataset 2

We have joined the 2 datasets on the movieId.

| movieId | movie_title | cast | director | genres |
|---|---|---|---|---|
| **117529** | Jurassic World | Chris Pratt\|Bryce Dallas Howard\|Irrfan Khan\|Vincent D'Onofrio\|Nick Robinson | Colin Trevorrow | Action\|Adventure\|Science Fiction\|Thriller |
| **122882** | Mad Max: Fury Road | Tom Hardy\|Charlize Theron\|Hugh Keays-Byrne\|Nicholas Hoult\|Josh Helman | George Miller | Action\|Adventure\|Science Fiction\|Thriller |
| **130490** | Insurgent | Shailene Woodley\|Theo James\|Kate Winslet\|Ansel Elgort\|Miles Teller | Robert Schwentke | Adventure\|Science Fiction\|Thriller |
| **130634** | Furious 7 | Vin Diesel\|Paul Walker\|Jason Statham\|Michelle Rodriguez\|Dwayne Johnson | James Wan | Action\|Crime\|Thriller |

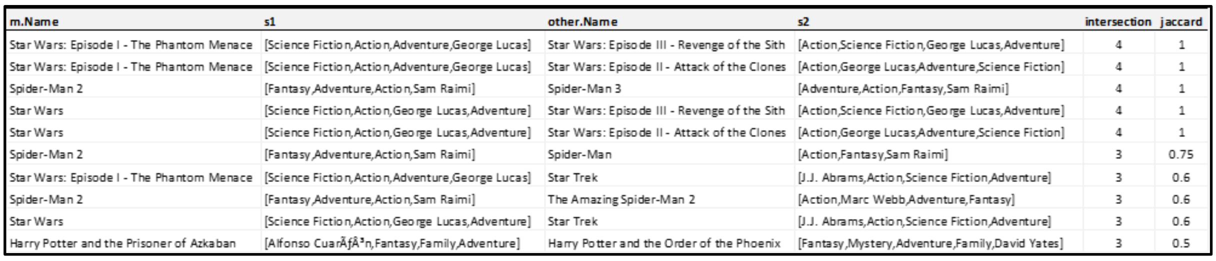
| | | | | |
|---|---|---|---|---|
| **139385** | The Revenant | Leonardo DiCaprio\|Tom Hardy\|Will Poulter\|Domhnall Gleeson\|Paul Anderson | Alejandro GonzÃ¡lez IÃ±Ã¡rritu | Western\|Drama\|Adventure\|Thriller |

Fig 20: Final dataset

Output Listing:

Below is the sample output listing for content-based recommendation based on User's genre or director preferences.

All the use case-based outputs can be seen on output analysis section



| m.Name | s1 | other.Name | s2 | intersection | jaccard |
|---|---|---|---|---|---|
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Wars: Episode III - Revenge of the Sith | [Action,Science Fiction,George Lucas,Adventure] | 4 | 1 |
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Wars: Episode II - Attack of the Clones | [Action,George Lucas,Adventure,Science Fiction] | 4 | 1 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | Spider-Man 3 | [Adventure,Action,Fantasy,Sam Raimi] | 4 | 1 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Wars: Episode III - Revenge of the Sith | [Action,Science Fiction,George Lucas,Adventure] | 4 | 1 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Wars: Episode II - Attack of the Clones | [Action,George Lucas,Adventure,Science Fiction] | 4 | 1 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | Spider-Man | [Action,Fantasy,Sam Raimi] | 3 | 0.75 |
| Star Wars: Episode I - The Phantom Menace | [Science Fiction,Action,Adventure,George Lucas] | Star Trek | [J.J. Abrams,Action,Science Fiction,Adventure] | 3 | 0.6 |
| Spider-Man 2 | [Fantasy,Adventure,Action,Sam Raimi] | The Amazing Spider-Man 2 | [Action,Marc Webb,Adventure,Fantasy] | 3 | 0.6 |
| Star Wars | [Science Fiction,Action,George Lucas,Adventure] | Star Trek | [J.J. Abrams,Action,Science Fiction,Adventure] | 3 | 0.6 |
| Harry Potter and the Prisoner of Azkaban | [Alfonso CuarÃƒÂ³n,Fantasy,Family,Adventure] | Harry Potter and the Order of the Phoenix | [Fantasy,Mystery,Adventure,Family,David Yates] | 3 | 0.5 |

Fig 21: Neo4j output for movie dataset