Contents lists available at www.journal.unipdu.ac.id

Register

Journal Page is available to www.journal.unipdu.ac.id/index.php/register

Research article

Movie recommender systems using hybrid model based on graphs with co-rated, genre, and closed caption features

Putra Pandu Adikara [a], Yuita Arum Sari [b], Sigit Adinugroho [c], Budi Darma Setiawan [d]

[a,b,c] *Department of Information Engineering, Universitas Brawijaya, Malang, Indonesia*
[d] *Graduate School of Information Science and Engineering, Ritsumeikan University, Japan*

email: [a] adikara.putra@ub.ac.id, [b] yuita@ub.ac.id, [c] sigit.adinu@ub.ac.id, [d] gr0450hp@ed.ritsumei.ac.jp

ARTICLE INFO

Article history:
Received 07 August 2020
Revised 01 September 2020
Accepted 08 September 2020
Available online 30 January 2021

Keywords:
closed caption
hybrid recommender system
movies
neo4j graph database

Please cite this article in IEEE style as:
P. P. Adikara, Y. A. Sari, S. Adinugroho and B. D. Setiawan, "Movie recommender systems using hybrid model based on graphs with co-rated, genre, and closed caption features," *Register: Jurnal Ilmiah Teknologi Sistem Informasi*, vol. 7, no. 1, pp. 31-42, 2021.

ABSTRACT

A movie recommendation is a long-standing challenge. Figuring out the viewer's interest in movies is still a problem since a huge number of movies are released in no time. In the meantime, people cannot enjoy all available new releases or unseen movies due to their limited time. They also still need to choose which movies to watch when they have spare time. This situation is not good for the movie business too. In order to satisfy people in choosing what movies to watch and to boost movie sales, a system that can recommend suitable movies is required, either unseen in the past or new releases. This paper focuses on the hybrid approach, a combination of content-based and collaborative filtering, using a graph-based model. This hybrid approach is proposed to overcome the drawbacks of combination in the content-based and collaborative filtering. The graph database, Neo4j is used to store the collaborative features, such as movies with its genres, and ratings. Since the movie's closed caption is rarely considered to be used in a recommendation, the proposed method evaluates the impact of using this syntactic feature. From the early test, the combination of collaborative filtering and content-based using closed caption gives a slightly better result than without closed caption, especially in finding similar movies such as sequel or prequel.

Register with CC BY NC SA license. Copyright © 2021, the author(s)

1. Introduction

The necessities of human are never adequate in fulfilling their self-satisfaction, likewise entertainment that is always needed in daily life. One of the interesting entertainments is watching movies. Movies generally attract people anywhere in the world, regardless the genres and the movie enthusiasts' age. This is the reason why the movie business is really profitable. By the time goes, many films or movies are released at the same time in cinemas in order to satisfy their viewers while gaining profit. However, some people are unable to watch all the released movies due to personal reasons, such as time or financial limitations. Thus, some of them prefer to watch movies later, and eventually, they may forget what to watch. To recall what they want to watch usually they will browse the Internet, for instance, online stores where people can rent or buy movies. It is now easy to find online video-on-demand services, not only available on the web but also smartphone by using particular video streaming apps. The latest smart televisions and set-top boxes also provide video streaming apps. One of the common features provided by these video-on-demand services is movie recommendations. The reason why a movie recommendation is important because they want to display not only the latest but also older

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

movies, so people will get attracted and satisfied while the business is able to keep selling movies in a long-running after the show time in cinemas is ended. Based on that reason, this paper proposes a new framework to make a recommender system according to the user's preferences.

In general, there are several approaches in building a recommender system, content-based filtering (CB), collaborative filtering (CF), knowledge-based (KB), or combinations of the two (hybrid model). CB recommendation is based on previous preference similarity which depends on feature and textual description of items. On the other hand, the CF approach is relying on certain information given by users who have similar favor and rating is one of the parameters. Meanwhile, the KB provides recommendation based on knowledge resources which are not exploited in both approaches, for instance, constraint-based or case-based recommendation [1].

Each method has its own disadvantages. Content-based filtering has several limitations such as limited content analysis, overspecialization, and new users [2]. Since this approach is not involving the user's preference, item's features are extracted using information retrieval methods, and it is not enough to provide information to distinguish which one the user likes or dislikes. Overspecialization is happened due to only items with high similarity to those items that are already rated that likely to be recommended to users. For example, a user that never read a romantic novel will likely never be recommended to the best romantic novel. Another issue is users have to assign a rating to several items so that the system can learn user preferences and give an accurate prediction. If the available ratings are limited in a small number, the recommender system cannot give accurate suggestions for new users [2], for instance, the movie recommendation using content-based filtering and genomic tags [3].

Collaborative Filtering also has weak points in dealing with new users, new items, and sparsity issues [4]. CF estimates the recommendation of items that are assigned by other users. However, the same as CB, this approach has a problem when encountering new users, known as cold start [2]. Moreover, new items need to be rated by many users, since it is required for building a precise recommender system. CF recommender system is relying on a large number of available information. Based on the previous experiments, the number of retrieved ratings is lower than the number of ratings that need to be predicted. For example, if a movie is rated only by small number of people, this movie will not be recommended even it has high rating. This is known as sparsity, in which user-item matrix become sparse. Sparse user-item matrix can diminish the performance of recommender systems [5]. Contextual factor also plays an important role as well to produce good recommender systems, for instance, location, time, and the purpose of purchase. Nevertheless, early recommendation, the use of contextual factors is burdensome since it needs massive training data and huge effort to make one. In addition, the contextual part is arbitrarily changed over time. One of the researches in movie recommendation using CF and user's behavioral historical data on large data sets [6].

Since those two problems exist in CB and CF approach, thus a hybrid method is proposed to overcome the drawbacks of each method [7]. The combination of CB and CF can be done, to construct user profile information, item features from the textual description, and from the community such as rating. One way to apply this recommender system was applied in a digital library by Huang et.al using a combination of hybrid and graph-based [8]. The graph-based model yields a better result over the matrix factorization model. For example, music recommendation using Last.fm data and achieve novel recommendation and keep the results relevant [9]. Another hybrid method is proposed [10] for movie recommendation using tags and ratings. The study used social features constructed from personal preference based on content annotation. They employed a singular value decomposition algorithm to build a personalized scoring system.

In this paper, an initial framework of a hybrid model for recommendation using graph is proposed. The graph-based model is applied regarding its advantages comparing to CB and CF. Previous research for movie recommendation already used a graph model [11], however their result still needs a deeper investigation. Another graph model using latent graph features has been deployed for movie recommendation system [12]. They used user-id, item-id, and rating for the features, but did not considering the closed caption to be used. Other researches are using visual features [13], affective features [14], genre correlation [15], sentiment analysis [16], and story-based features extracted from movie characters and their interactions [17], user interest and movie feature [18], however none of them investigated closed caption. In this study, a graph model for movie recommendation system using a

graph database, Neo4j. The features used are the combination of collaborative and content-based, in the form of textual descriptions and ratings. In this case, one of the textual descriptions of a movie is the closed caption. Closed caption describes what the content of the movie is, such as names, places, talks. Closed caption is added as feature of CB to find out similar movies based on the conversations and the story. Therefore, in this study, rating, genres, and closed caption are considered. Moreover, this study are focusing on syntactic rather than semantic.

2. Proposed Method

In this paper, a hybrid recommendation system is built by means of graph-based model which is involving closed caption/subtitle as one of the features in CB filtering. Closed caption is a conversation part that explains the content of a movie and by using some of which can be employed hypothetically for searching out similar movies, either pre-sequel or sequel of a movie. The hybrid method is applied by using a graph-based model with a combination of CF and CB filtering. Fig. 1 depicts the overall steps in building the graph database. Later, this graph database will be used to construct the query for making movie recommendations.

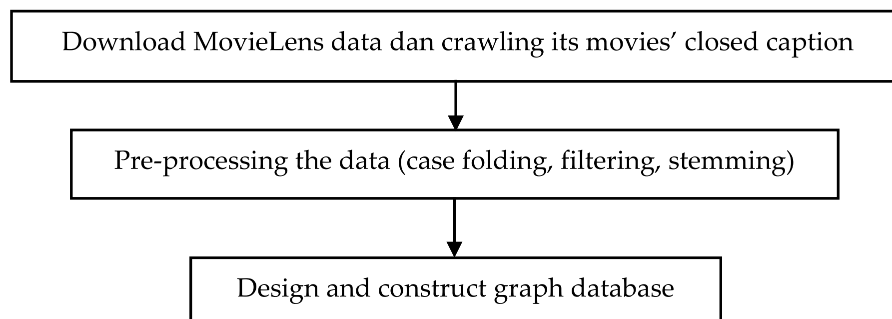

Fig. 1. Building a graph-based database

2.1. Dataset acquisition

The first dataset used in this paper is movies and ratings, 20M MovieLens that are available at MovieLens. This MovieLens dataset consists of 27.279 movies and 20 million ratings given by users. Most types of these movies are western movies (not tv series or drama) within 1891-2015. For this study purpose, two CSV files from MovieLens are used:

a. *Movies.csv*, contains movies data, with three columns:
   1. *movieId*: a unique identifier of each movie.
   2. *title*: the title of the movie.
   3. *genres*: the genres of the movie with delimiter "|" (pipeline symbol).

Example as follows:

```
1   movieId,title,genres
2   1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
3   2,Jumanji (1995),Adventure|Children|Fantasy
4   3,Grumpier Old Men (1995),Comedy|Romance
5   4,Waiting to Exhale (1995),Comedy|Drama|Romance
6   5,Father of the Bride Part II (1995),Comedy
```

b. *Ratings.csv*, contains user and rating data given to movies, with 5 columns:
   1. *userId*: an identifier of the user (anonymous).
   2. *movieId*: the identifier of the movie in which this user had given a rating.
   3. *rating*: the given rating, within range of 1-5
   4. *timestamp*: the timestamp when the user gave the rating with POSIX/UNIX time format.

Example as follows.

```
1   userId,movieId,rating,timestamp
2   1,2,3.5,1112486027
3   1,29,3.5,1112484676
4   1,32,3.5,1112484819
5   1,47,3.5,1112484727
6   1,50,3.5,1112484580
```

The second dataset used in this research is subtitle/closed caption from the movies. the closed captions are collected automatically by crawling from Subscene (subscene.com). It is challenging to

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

crawl from Subscene since they do not provide any API, thus and the download process used the movie title as the query. Pseudocode of the crawler described in Algorithm 1.

Algorithm 1 Pseudo-code of Subtitle crawler

1 *movies* = read file Movies.csv from MovieLens
2 **for** *m* **in** movies **do**
3 Search movie by title *m* and year then filter by language=English and *section type* in this priority:
  1. Exact: query and title are exact match
  2. Close: movie title is similar to query
  3. Popular: movie title is popular and similar to query
4 **if** *subtitle* is in one of *section type* **then**
5 *similarity* = calculate title similarity and query using Jaro-Winkler
6 *if similarity* > 0.8 **then**
7 Download the *subtitle* and save to a folder named movie ID *m*
8 Append to log that it's found
9 else
10 Append to log that it's not found

Finding an appropriate subtitle in Subscene sometimes give unexpected result, it happens due to similarity between returned movie title and query (and also with different movie year) but actually unrelated, which can be a false positive. There is also a possibility that the movie title is located not in the first position, even if the query and the title is an exact match. In this case, to reduce the false positive to get the correct subtitle, a similarity measure between query q and movie title τ in result set from Subscene is calculated using Jaro-Winkler similarity [19], in which improved from Jaro similarity (Eq. 1):

$$\boldsymbol{sim_w(q,t)} = \begin{cases} \mathbf{0} & \text{if } \boldsymbol{m} = \mathbf{0} \\ \frac{1}{3}\left(\frac{m}{|q|}\right) + \left(\frac{m}{|\tau|}\right) + \frac{m-t}{m} & \text{otherwise} \end{cases} \quad (1)$$

where: q = query, t = movie title from Subscene result set, $|q|$ = length of the string from the query, $|t|$ = length of the string from the movie title, m = the number of *matching* characters, t = half the number of transpositions. Two characters from q and t, are considered *matching* only if they are the same and not farther than $\left\lfloor\frac{\max(|q|,|\tau|)}{2}\right\rfloor - 1$.

Jaro-Winkler similarity formulated in Eq. 2.

$$\boldsymbol{sim(q,\tau)} = \boldsymbol{sim_w(q,\tau)} + \boldsymbol{\ell p}\big(\mathbf{1} - \boldsymbol{sim_w(q,\tau)}\big) \quad (2)$$

where $sim(q,t)$ = Jaro similarity for q and t, $\boldsymbol{\ell}$ = length of common prefix at the start of the string up to a maximum of four characters, p = constant scaling factor, the standard value is 0.1. In this case, a query and the movie title are considered correct if the similarity larger than the threshold of 0.8, permitting subtle differences, like diacritics, accent, or any translated language other than English.

From the crawled result, more than half of the movies failed to find its closed caption. Only approximately 13.144 closed caption that can be obtained from 27.279 movies (48%). One of the reasons is the possibility of many movies before 1990 (and also later) that do not have/include English subtitles or people who are interested to transcribe an English subtitle of the movie. From the statistic, there are more than 10.000 movies between 1891 to 1990 (~37%), more than 4.000 movies between 1990 to 2000 (~16%), and more than 12.000 between 2000-2015 (~44%). This statistic allows us to prove our initial hypothesis of missing closed caption. Later in building the graph database, only movies with English closed caption that are included in this research.

2.2. *Dataset pre-processing*

Specific text pre-processing is not applied on MovieLens dataset since its already well structured. Only movies that don't have closed caption are filtered out. The main text pre-processing step is applied to the collected closed caption dataset.

After the closed caption's time section is omitted, leaving only the text contains the conversation, pre-processing steps then applied on the resulted text:

- Remove formatting and hearing-impaired tags (opening and closing tag), such as <tag>, [tag], {tag}, and (tag).

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

- Remove the ads in the text, since some *fansubber* (people who transcribe the subtitle) also place ads to get commissions.
- Remove the numeric and non-word characters/ Unicode symbols.

The cleaned closed captions then saved in another file, in this case, a JSON file along with the movie ID of each closed caption belongs to. The resulted JSON file is approximately 500 MB in size.

2.3. *Constructing closed caption corpus*

The JSON file contains the movie ID and its closed caption then underwent another pre-processing stage. This pre-processing is important in preparing the corpus for Term Frequency-Inverse Document Frequency (TF-IDF) calculation. The pre-processing steps are case folding and lemmatization. Case folding is a process of converting all capital letters to lower case [20]. Stemming is not used, because stemmer tends to cut off the end of a word to reduce the inflection and resulting in an over-stemmed/under-stemmed word [20]. Hence, lemmatization is used to reduce the inflections but keep the word in original lemma to maintain its semantic later when in use as necessary. This step is important to use in order to reduce the number of terms or words that have similar meanings, even though there are some morphology differences. WordNet lemmatizer is used as part of Natural Language Toolkit (NLTK) Python module.

Once pre-processed is done, the term's weight in every document is calculated using TF-IDF. Later, the TF-IDF weight vector will be used to compute the similarity between two movies, the query and a candidate movie. Term weighting TF-IDF is usually used in information retrieval or text mining, specifically vector space model, in order to evaluate which terms have significance/importance in a document over the collection of documents (corpus). This importance is proportionally increasing as the number of times of a word t occurs in the document d ($tf_{t,d}$) as shown in Eq. 3. However, since more frequent a word is not always more important, considering the stop words, another measure is taken into account. Hence, $tf_{t,d}$ then multiplied by $\boldsymbol{idf_t}$, a measure of how important a word t over all documents in the corpus (Eq. 4).

$$\boldsymbol{tf_{t,d}} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{\boldsymbol{t,d}}, & \text{if } \mathrm{tf}_{\boldsymbol{t,d}} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where tf = number of occurrence (frequency) of term t in document d.

$$\mathrm{idf}_{\boldsymbol{t}} = \log_{10} \frac{\boldsymbol{N}}{\mathrm{df}_{\boldsymbol{t}}} \tag{4}$$

where N = total number of documents in the corpus, df_t = number of documents contain term t. Thus, the weight of term t denoted as $\boldsymbol{w_{t,d}}$ is shown in Eq. 5.

$$\boldsymbol{w_{t,d}} = \mathrm{tf}_{\boldsymbol{t,d}} \times \boldsymbol{idf_t} \tag{5}$$

Once pre-processed, TF-IDF calculation is done by utilizing Python module, TFIDFTransformer, and TFIDFVectorizer from Scikit Learn module. The resulted TF-IDF matrix then saved to another file using Pickle.

2.4. *Constructing the graph-based database*

After the pre-processing stage, the graph used for storing data is created. It contains vertices (nodes) and edges linked between vertices. The types of vertex are:

- ***Movie***. This vertex is composed of metadata of the movie, they are ID Movie, Title, Year, IMDB URL according to MovieLens data.
- ***User***. This vertex is composed of metadata of the user, they are ID User, Age, Gender, and Occupation as maintained by MovieLens.
- ***Genre***. This vertex is composed of properties of the genre. Each genre contains a single vertex so that 19 genres in MovieLens are generated into 19 genre vertices.

The edge of vertices can be described as follows:

- *Movie Genre*. The edge connects *Movie* vertex to *Genre* vertex if the *Movie* has *Genre* (the *Movie* has *hasGenre* relationship)
- *Movie-User*. The edge connects *Movie* vertex to *User* vertex if a *Movie* is rated by *User*. The rating is in a form of *stars*, with a possible value of 0-5.

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

The graph constructed from the dataset is stored in a NoSQL database optimized for the graph data structure. such as Neo4J, FlockDB, AllegroGraph, and GraphDB. A relational-based database is not used in this research since it does not support graph structure and its related operation like graph traversal. Graph traversal is conducted by the use of traversal language embedded in the database. For instance, Neo4J has Cypher. It is also possible to employ a specific graph traversal language that supports many databases such as Gremlin.

The graph traversal process is started from a movie node as a starting point to other vertices that are connected to each other. The traversal strategy is a random walk, starting from a start vertex to other vertices randomly. However, a certain filter is applied so that only vertices that meet specified criteria are traversed. The traversal is the basis for making a recommendation. Movie vertices traversed with a certain rating threshold given by a specific User (co-rated), overlapped genre, and overlapped terms are returned as a recommendation result. In the end, the recommendation result is sorted based on co-rated, the number of overlapped genres, and the number of overlapped terms. An example of a visualization of the graph between *Movie* vertex and *User* as well as co-rated Movie is shown in Fig. 2. One of the database management systems that is possible in saving and traversing this kind of graph data structure is Neo4j.

Neo4j is a database management system, but not a relational one. Neo4j has an advantage in term of speed than relational database. It is not because it has a very fast algorithm or sophisticated technology, especially since it's built upon Java which is slower than in native language like Assembly, C/C++. The speed that Neo4j has is due to its data structure, using graph-based instead of relational tables ***[21]***. Naturally, graph is fast in traversing, started from a certain node then visit interconnected nodes with speed that can be predicted. For example, the case is retrieving books in a library with a certain genre. In the relational database, all books need to be retrieved first then remove all books that are not in the predefined genre in the query. However, in graph-based, traversal is started from a book with the predefined genre, then visit to adjacent book until all books in the same genre cluster are visited. In this case, the need of retrieving all books in different genres is omitted naturally.

Neo4j offers application programming interface (API) so that Neo4j can be accessed, modified externally from another program no matter what the programming language is. Neo4j provides REST API so it can be accessed through HTTP protocol. For querying, Neo4j can be accessed via Cypher, its own query language, or more universally traversal/query language, Gremlin. Cypher is a declarative language that let users express what data to retrieve through pattern matching, like SQL statement. Gremlin, however is an example of imperative language, in which users explain how to traverse the graph. Cypher ease users to read the code, but applied on Neo4j only. In this study, Cypher is used to query the nodes.

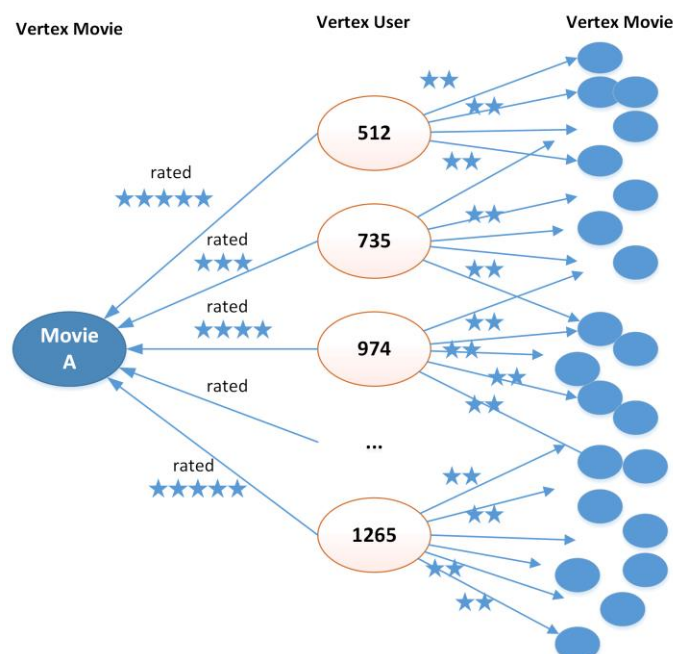

Fig. 2. Graph visualization for movie recommendation with co-rated

Cypher consists of 4 main parts that have specific roles *[21]*:

- **START**, to find the starting point in the graph, but it is deprecated in the latest Cypher, and the recommendation is to use MATCH.
- **MATCH**, to search for the pattern described in it, allowing to search for subgraph location.
- **WHERE**, to filter data based on criterions.
- **RETURN**, to return what to include in the query result set.

Given a case in which require to retrieve the first 10 users who had given rating to two movies with id=1 and id=2, Cypher syntax is written as follows.

```
MATCH (m:Movie {movieId:1}), (n:Movie {movieId:2})
MATCH (n)-[:hasRating]-(u:User)-[:hasRating]-(m)
RETURN m, n, u
LIMIT 10
```

First, from two starting points (movie nodes *m, n*), graph traversal begins to find users *u* who gave rating to movie *m, n*. This traversal then stops after finished finding 10 nodes and returns nodes *m, n, u* accordingly. Fig. 3 illustrates this Cypher query, with the blue nodes are the movie nodes, the pink nodes are the user nodes, and the edge from user to movie nodes is the rating given by the user to the corresponding movie. Neo4j has a built-in web application to show the result both in graph visualization or text-based.

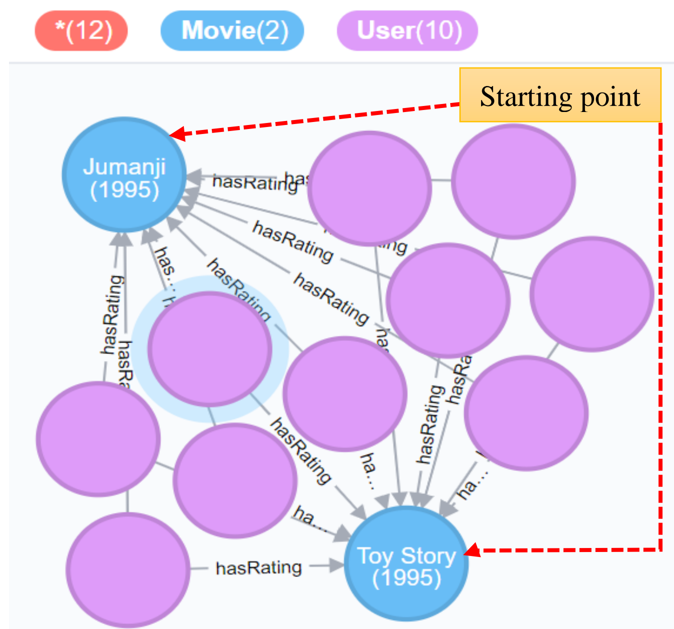

Fig. 3. Returned nodes from pattern matching using Cypher

In order to load the large movie dataset (in this case millions of ratings), instead of creating the node and relationship one by one manually, Neo4j provides direct load CSV using periodic commit. By using periodic commit, Neo4j will ensure the failure due to memory constraint will not happen unlike by using a single Cypher query. Example of the Cypher query to load CSV as follows. Note that the following example excludes the creation of Genre nodes and the edges to the corresponding Movie nodes.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'file:///ratings.csv' AS line
MATCH (m:Movie {movieId: toInteger(line.movieId)})
MERGE (u:User {userId: toInteger(line.userId)})
CREATE (u)-[r:hasRating {rating:toFloat(line.rating)} ]->(m)
```

To speed up the graph traversal in finding the nodes and edges, indices need to be created. This can be done after or before the nodes are created. In this case, the indices are for User nodes, Movie nodes, and Genre nodes, using an attribute or a property, *movieId, userID*, and *genre* terms respectively. These indices are constrained to be unique, since no more than one node with the same value (identifier) is allowed, and to enforce data integrity rule. The following example is a Cypher query to create the index in User nodes.

```
CREATE CONSTRAINT ON (u:User) ASSERT u.userId IS UNIQUE
```

After required CSV files are loaded into Neo4j database, the next important steps are how the recommendation is made and how to evaluate the results. The overall process of making the recommendation is shown in Fig. 4.

First, determine the movie *Q* that is used as the query. It can be any movie that exists in the graph database (Movie node). For example, given an auto-complete text box, after a few characters typed into the text box, it shows a list of movies with partial matches. After a movie is selected, the system will look for the subtitle. Unless the subtitle is retrieved online simultaneously when the query is made, pre-processing need not to be done.

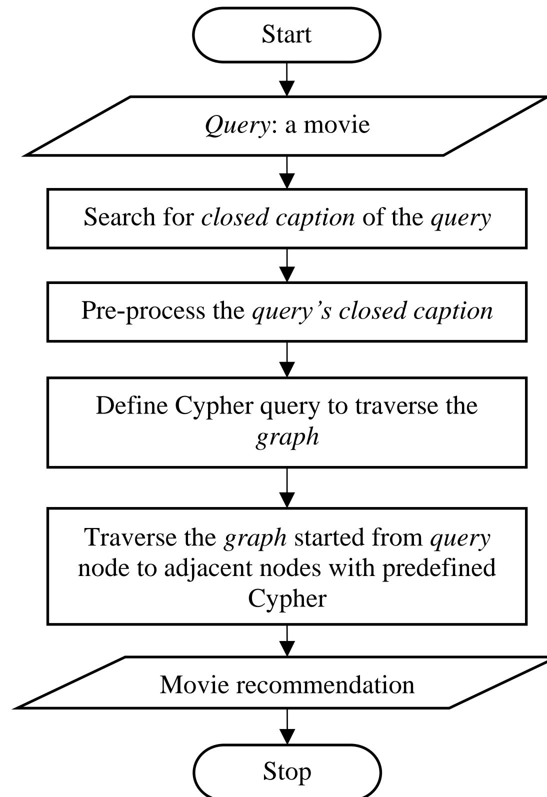

Fig. 4. Flowchart of creating the recommendation

Since the subtitles are collected beforehand, the searching of the subtitle is done offline by using the *movieId* of the query. The pre-processing of the subtitle can be done after the query is made or when all the subtitles are collected in the beginning. In this study, the pre-processed subtitles is used when constructing the corpus since it has the same pre-processing steps. This subtitle will be used in the last step of making the recommendation.

The next step after selecting a movie query, the pre-defined Cypher query is called to traverse the graph to get the matching nodes *M*. This Cypher query is used to test how well it performs using content or syntactically similarity based on the conversation instead of semantically similarity. Algorithm of the pattern matching is detailed in Algorithm 2.

Algorithm 2 Pseudo-code of Making a Movie Recommendation

1 ***M*** = Find Movie nodes with the highest co-rated by other users with rating given greater than or equal to 3 (in the scale of 0 to 5) with the same genre(s). Sorted by how many users co-rated the movie ($count_{rate}$) in descending order.
2 **for** *M* **in** *M* **do**
3 $score_m$ = calculate score based on $count_{rate(M)}$ and cosine similarity between query *q* and movie *m*
4 Sort movies *M* by $score_m$ in descending order
5 ***M*** = get top ***N*=10** of the movies
6 **return *M***

The similarity of two movies are calculated by using cosine similarity of their vector representations as in Eq. 6. This vector representation $\vec{Q}, \vec{M}$ are derived from the query movie subtitle and the candidate movie respectively, with one component in the vector for each term. This vector component is the value of TF-IDF weight.

39
ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

$$similarity_{\cos(Q,M)} = \frac{\vec{M} \bullet \vec{Q}}{|\vec{M}| \cdot |\vec{Q}|} = \frac{\sum_{i=1}^{t} (M_i \cdot Q_i)}{\sqrt{\sum_{i=1}^{t} M_i^2 \cdot \sum_{i=1}^{t} Q_i^2}} \quad (6)$$

where: Q = vector of term's weight in the query movie, M = vector of term's weight in the candidate movie.

The overall score used to sort the movies in the candidate list is simply formulated in Eq. 7.

$$score_m = count_{rate(M)} \times similarity_{cos(Q,M)} \quad (7)$$

where: $count_{rate(M)}$ = number of users co-rated the movie M with rating >= 3. In Eq. 7, if two movies are co-rated with the same number of users, it will likely to be adjacently recommended. By utilizing the cosine similarity, if the content of one movie is more similar than the other one to the query, then the former movie is likely to be more recommended than the later. Another approach is also possible by giving weight to the count rate and cosine similarity to give more flexibility. By giving the weight parameter to each $count_{rate(M)}$ and $similarity_{cos(Q,M)}$, the optimal parameter needs to be calculated, for instance using gradient descent method.

3. Results and Discussion

For reflecting the result of the system design based on the proposed method, a simple user interface is built. The user interface used in this research is web-based built using Python and using Django for the web framework. The user interface is built as simple as possible only to show the bare bones for proof-of-concept and not for a real-world case even though it is possible to improve the user interface. User is searching for a movie by typing into a text field and with the help of Javascript, showing a list of movies by which the user can select one of it. After a movie is selected and clicked on Suggest button, for example, Toy Story, a recommendation result should be shown as in Fig. 5. This recommendation result is later to be evaluated how well-performed the recommendation is.

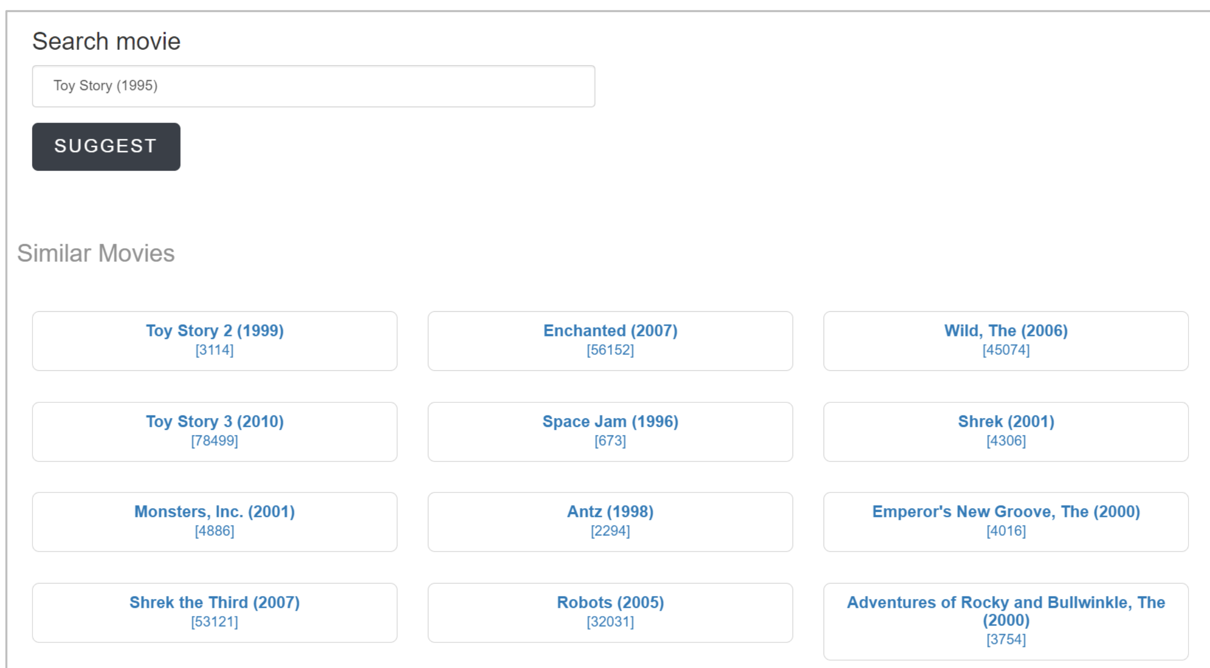

Fig. 5. Example of the recommendation result of Toy Story

Evaluating a recommendation is a difficult measurement since its really subjective and needs a broad test of subjects. There are so many things that affect this objectivity, such as age, sex, educational background, previous experience, etc. To simplify the evaluation and as a proof of concept, the result of the recommendation is evaluated and the accuracy is measured using Precision@K metrics. K used here is 10, since usually a recommendation is only shown in a small number. This is also usually used in information retrieval since no more than a few pages are evaluated. This evaluation was tested on a small subset of users and 4 different queries. Users are ranging from 20-40 years old. Even though it

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

cannot reach the whole broad population subject, a small result comparison is provided between the recommendation using Rating+Genre and Rating+Genre+Closed Caption. Mean Average Precision (MAP) is measured to sum up all the Precision@10 results from different tests as depicted in Fig. 6.

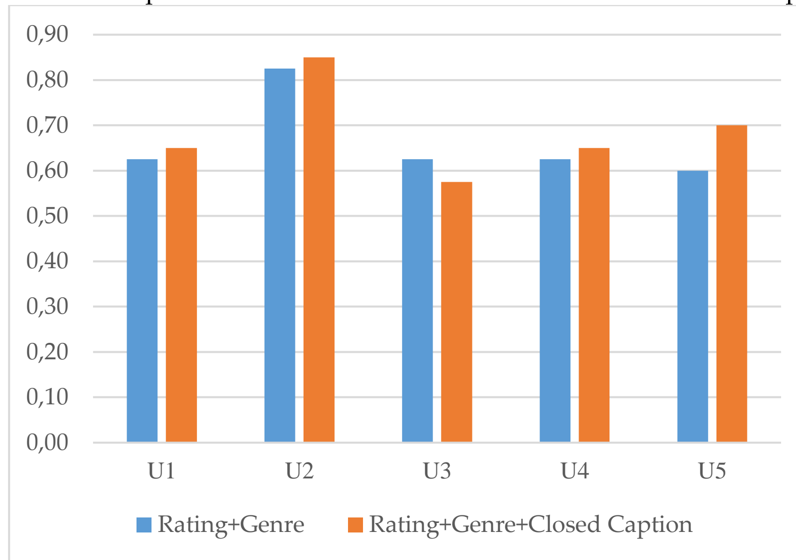

Fig. 6. Comparison of the recommendation results between Rating+Genre and Rating+Genre+Closed Caption

Table 1
Precision@10 of two recommendation results

User	Precision@10									
	Rating+Genre					Rating+Genre+Closed Caption				
	Q1	**Q2**	**Q3**	**Q4**	**Average**	**Q1**	**Q2**	**Q3**	**Q4**	**Average**
U1	0.5	1	0.5	0.5	0.62	0.4	0.9	0.6	0.7	0.65
U2	0.9	0.7	1	0.7	0.82	1	0.7	1	0.7	0.85
U3	0.6	0.4	0.6	0.9	0.62	0.6	0.2	0.6	0.9	0.57
U4	0.5	0.5	0.7	0.8	0.62	0.6	0.4	0.8	0.8	0.65
U5	0.6	0.5	0.8	0.5	0.60	0.7	0.5	0.9	0.7	0.70
				MAP =	0.66				MAP =	0.69

Table 1 shows the results from two different recommendation tests. It can be seen that the combination of three features gives better results even only a small fraction. Recommendation using Rating+Genre+Closed Caption promotes sequel or prequel movies more than Rating+Genre, since there are similarities in the content, such as names, places, or other proper nouns. This recommendation result is good for finding movies with a different title (partially similar or even totally different title) but have similarities in the content. Sometimes, people do not realize that movie A has a connection to movie B, unless the same or similar names come up in those movies. This is the reason why MAP of Rating+Genre+Closed Caption can be better than Rating+Genre, since the users think the recommended movies are still relevant to them. However, this outcome might not be preferred for people who want to get more varied movies, rather than just sequel or prequel.

4. Conclusion

From the experimental result, the use of features combination, Rating+Genre+Closed Caption produces a better MAP score than Rating+Genre, although only slightly better. A movie with similar content to the query has a higher rank such as sequel or prequel movies. The impact in the real world case however, still need to be evaluated, because a recommendation is very subjective to personal preference. A broader range of users such as background and demography variation, need to be considered for future works. Since this is an initial or preliminary test of this framework, as a proof of concept, many possibilities can be incorporated to give better accuracy. Overall scoring is one of them, by giving optimum weight to each score. Feature selection can also be utilized to select which terms are more important than the others to represent a movie. Further experiments can be interesting to be investigated by using contextual word representation with a pre-trained word embeddings method such as

http://doi.org/10.26594/register.v7i1.2081

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

Word2Vec or GloVe. Not to mention other possible features, like the actors, directors, and so on that can be used to make a better and more accurate recommendation.

Acknowledgment

This research was supported by DIPA Voucher research grant at Faculty of Computer Science, Universitas Brawijaya.

References

[1] D. Jannach, M. Zanker, A. Felfernig and G. Friedrich, Recommender Systems: An Introduction, New York: Cambridge University Press, 2011.

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

[3] S. M. Ali, G. K. Nayak, R. K. Lenka and R. K. Barik, "Movie Recommendation System Using Genome Tags and Content-Based Filtering," in *Advances in Data and Information Sciences*, Singapore, 2018.

[4] N. Mustafa, A. O. Ibrahim, A. Ahmed and A. Abdullah, "Collaborative filtering: Techniques and applications," in *International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, Khartoum, 2017.

[5] R. Zhang, Q.-d. Liu, Chun-Gui, J.-X. Wei and Huiyi-Ma, "Collaborative Filtering for Recommender Systems," in *Second International Conference on Advanced Cloud and Big Data*, Huangshan, 2014.

[6] T. Zhou, L. Chen and J. Shen, "Movie Recommendation System Employing the User-Based CF in Cloud Computing," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Guangzhou, 2017.

[7] J. Lu, D. Wu, M. Mao, W. Wang and G. Zhang, "Recommender system application developments: A survey," *Decision Support Systems,* vol. 74, pp. 12-32, 2015.

[8] Z. Huang, W. Chung, T.-H. Ong and H. Chen, "A graph-based recommender system for digital library," in *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries (JCDL '02)*, New York, 2002.

[9] K. Lee and K. Lee, "Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items," *Expert Systems with Applications,* vol. 42, no. 10, pp. 4851-4858, 2015.

[10] S. Wei, X. Zheng, D. Chen and C. Chen, "A hybrid approach for movie recommendation via tags and ratings," *Electronic Commerce Research and Applications,* vol. 18, pp. 83-94, 2016.

[11] G. Tüysüzoğlu and Z. Işık, "A Hybrid Movie Recommendation System Using Graph-Based Approach," *International Journal of Computing Academic Research (IJCAR),* vol. 7, no. 2, pp. 29-37, 2018.

[12] M. Shah, D. Parikh and B. Deshpande, "Movie Recommendation System Employing Latent Graph Features in Extremely Randomized Trees," in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (ICTCS '16)*, New York, 2016.

[13] Y. Deldjoo, M. Elahi, M. Quadrana and P. Cremonesi, "Using visual features based on MPEG-7 and deep learning for movie recommendation," *Int J Multimed Info Retr,* vol. 7, p. 207–219, 2018.

[14] J. K. Leung, I. Griva and W. G. Kennedy, "Making Use of Affective Features from Media Content Metadata for Better Movie Recommendation Making," arXiv, 2020.

[15] S. Reddy, S. Nalluri, S. Kunisetti, S. Ashok and B. Venkatesh, "Content-Based Movie Recommendation System Using Genre Correlation," in *Smart Intelligent Computing and Applications*, Singapore, 2019.

[16] H. Li, J. Cui, B. Shen and J. Ma, "An intelligent movie recommendation system through group-level sentiment analysis in microblogs," *Neurocomputing,* vol. 210, pp. 164-173, 2016.

ISSN 2502-3357 (online) | ISSN 2503-0477 (print)
7 (1) January 2021 31-42

[17] O.-J. Lee and J. J. Jung, "Explainable Movie Recommendation Systems by using Story-based Similarity," in *Explainable Smart Systems 2018 (ExSS '18)*, Tokyo, 2018.

[18] J. Li, W. Xu, W. Wan and J. Sun, "Movie recommendation based on bridging movie feature and user interest," *Journal of Computational Science,* vol. 26, pp. 128-134, 2018.

[19] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage," The Educational Resources Information Center (ERIC), Washington, DC, 1990.

[20] C. D. Manning, P. Raghavan and H. Schütze, An Introduction to Information Retrieval, Cambridge: Cambridge University Press, 2008.

[21] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox and J. Partner, Neo4j in Action, Greenwich, CT, United States: Manning Publications Co, 2014.

 http://doi.org/10.26594/register.v7i1.2081