Actor Recommendation System

Hao Wang, Shuo Feng, Jintao Jiang

University of Illinois Urbana-Champaign,
{haow4, sfeng15, jjiang43}@illinois.edu

Abstract. With the increase of film production year by year, many movie recommendation systems are set up recent years to help users find good movies. But there is no actor recommendation system to recommend actors, helping users find movies based on actors. We build an actor recommendation system to help users find the actors who are similar with users' favorite movie star and provide a directed graph that shows similarities and movie genres among actors to help users explore more information.

1 Introduction

Movies have become an indivisible part of everybody's life. Especially in the era of big data and artificial intelligent, people are expecting more and more handy functions to help them explore information that they may be interested in. Current systems like IMDB and Amazon Prime Video, afford many useful search functions based on movie genres, actors or other filters. However, for recommendation they only recommend movies.

1.1 Recommend by Actor

People may be a big fan of certain movie star but they already watched all movies acted by that actor and want to find other movies. The movies acted by similar movie stars with their favorite actor are expected to recommend. Our recommendation system is the application to help user find similar movie stars so that they can know whose movies they should watch.

1.2 Target Users

1. **Fans of certain actors**: Many people are biased when choosing movies to watch. They are apt to watch whatever movies their idols featured in. So simply recommending movies to them wont attract them. However, our tool chooses to recommend to them the similar actors to their idol. In this way they are more likely to accept our recommendation and be able to explore more actors and more movies that fit in their tastes.

2. **Movie directors**: Many movie directors are upset about choosing a cast. Sometimes they can find actors they really admire and are very suitable to their films but the problem is that they cant always reuse these actors. Our tool provides solution to them. By using our tool, they can find similar actors to actors they admire so that they can have good chance to cooperate with other good actors that are suitable for their movies.

2 Related work

There are many recommender systems on the market in the movie industry. Suppose there is a movie lover who wants to find other actors similar to their idols. If they go to IMDB and type in their favorite actors for instance. As shown in figure, IMDB can recommend a variety of movies related to this actor but they do not support recommending actors directly. It is the same for other websites such as Google or Ranker.com.

Fig. 1. IMDB & Ranker

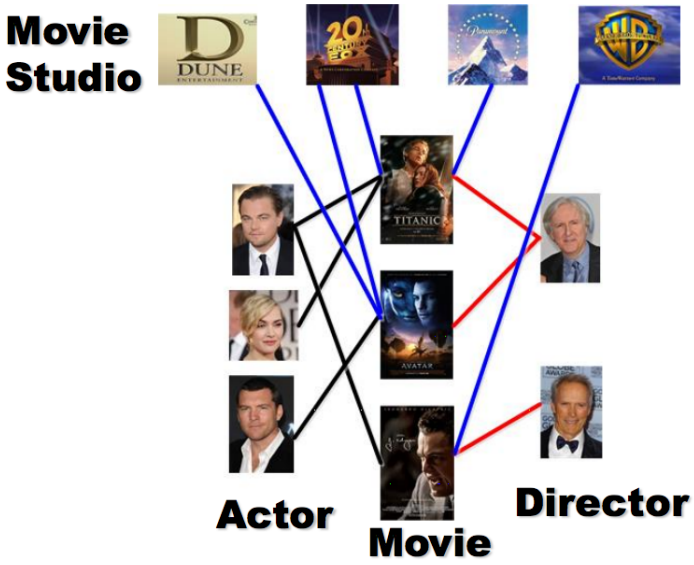
3 Framework & Method

Crawler We write a script to crawl the data from the web. Initially we intended to use APIs to download the data, but we found out that these APIs only afford information about movies rather than actors, so we decided to Google actors directly. First of all, we find a website that list top 100 ranked actors. We crawl the names into an array and use these names to query by requesting URL "http://www.google.com/search?q= name + genre + movies". After 100 iterations on different genres, we finished our data collecting.

Similarity Algorithm We use PathSim[1] as our principal similarity algorithm to compute similarities among actors. PathSim is a similarity algorithm to computer top-k similar results based on heterogeneous information networks.

1. **Heterogeneous Information Networks**: Given a directed graph, $G = (V, E)$ with an object type mapping function $\phi : V -> A$ and a link type mapping function $\psi : E -> R$ where each object $\nu \subset V$ belongs to one particular object type $\phi \subset A$, and each link $e \subset E$ belongs to a particular relation $\psi(e) \subset R$. [1] When the types of objects $A > 1$ or the types of relations $R > 1$, we can define this information network as heterogeneous information network.[1]

Fig. 2. IMDB Network



2. **Meta-Path**: A meta-path P is defined on a graph of network schema $T_G = (A, R)$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, which defines a composite relation $R = R_1 - R_2 - ... - R_l$ between type A_1 and A_{l+1}, where $-$ denotes the composition operator on relations.[1]
3. **PathSim**: PathSim is based on heterogeneous information network. It compute similarity according to various meta-path. Given a meta-path P on heterogeneous information network and two same type objects x and y, we compute their conjunct elements:
$s(x, y) = \frac{2*|\{P_{x->y} : p_{x->y} \subset P\}|}{|\{P_{x->x} : p_{x->x} \subset P\}| + |\{P_{y->y} : p_{y->y} \subset P\}|}$[1]
For our project, we try many meta-path such as Actor - Movie Genre - Actor, Actor - Awards - Actor, Actor - Period - Actor ... to find the best path to recommend actors. After plentiful measurements, we find Actor - Movie Genre - Actor is the best meta-path to generate actors' similarities.

Front-end Javascript Template

Fig. 3. Bibliographic network schema and meta paths

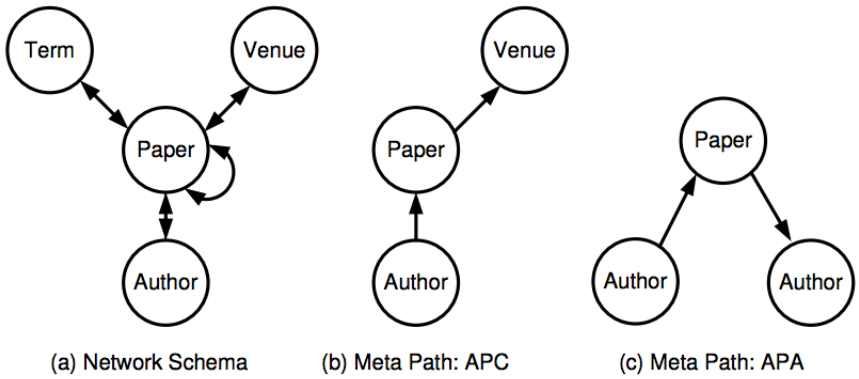


Fig. 4. Author-Conference-Author Example

	SIGMOD	VLDB	ICDE	KDD
Mike	2	1	0	0
Jim	50	20	0	0
Mary	2	0	1	0
Bob	2	1	0	0
Ann	0	0	1	1

	Jim	Mary	Bob	Ann
P-PageRank	**0.3761**	0.0133	**0.0162**	0.0046
SimRank	**0.7156**	0.5724	**0.7125**	0.1844
RW	**0.8983**	0.0238	**0.0390**	0
PRW	**0.5714**	0.4444	**0.5556**	0
PathSim	0.0826	**0.8**	**1**	0

1. **Graph**: The way we solved the problem is to build a website to visualize the similarities among different actors. The main part of our system is the graph representation. The graph consists of nodes and edges. Each node represents a actor. Each edge between two nodes means these two actors are closely related to each other. On the center of our system is a node which we are searching. Meanwhile, there are two types of nodes surrounding the center node. The first type of nodes are closer to the center node. They are the ones most related to the center actor. The second type, peripheral nodes, are scattered farther from the center node. These nodes are less related to the center node. By looking at the graph, movie watcher are able to see actors recommended to them. In addition, we allow movie watchers to click each node and see more details. For example, they can see the genres of movies these stars are in. Additionally, nodes has different sizes. The larger the size of the node, the more popular this actor is among the general public. This is a good way to combine the opinions from the whole society and users personal ideas. Different nodes The color of the nodes are used to distinguish the genres of the movies. If users are not interested in a specific genre, they can easily ignore those irrelevant ones.
2. **Filters and search functions**: Besides the basic graph, we also provide users with a handful of helper functions. The first one is a filter used to choose whether we make all the nodes scattered all around the canvas or not. The default setting is yes. However, if users only like a specific type of movies, they can use this filter to group actors into different clusters where

each clusters are the same genres. In this case, users can simply disregard the actors whose main genres are not their types. The second filter is to adjust the threshold of how many relevant actors are displayed. This allows users to see as much recommended actors as they want. This could make it less overwhelming if there are so many related actors. The final helper function we have is the search. Users are given a search bar so that they could type in keywords. For instance, if users type in Leonardo, nodes with keyword Leonardo Dicaprio will be be highlighted. If users type in Captain, Captain America will be highlighted. Users can also input their own data to expand their function.

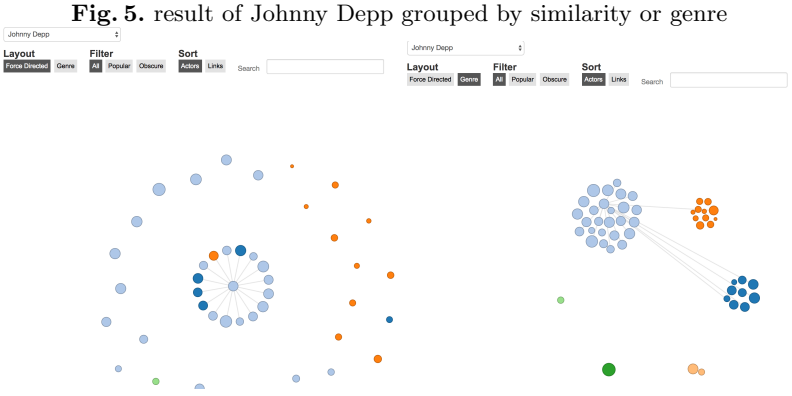
3. **Use of D3.js**: We used D3 to make the user interface more responsive and better. The whole graph can be dragged to different shapes. It is totally responsive. If movie watchers use their mobile devices to use our system. the whole website will be much more compact.

Backend Database On the backend, we used MySql to store all the data. Our databaes schema consists of three parts. The first one is the field for actors' names. The other fields are varied genres. Each one records the total amount of movies a specific actor plays. These fields are used to perform the similarity calculations. The last field is the total number of movies. This field enables us to extract this number and use that to determine the size of each node.

4 Sample Results

We import 50 famous actors as sample data and group them by different approaches. As figure shows below, we compute the similarities between target movie star(Johnny Depp) and other actors to link them based on the similarities greater than threshold(0.8).

Fig. 5. result of Johnny Depp grouped by similarity or genre



5 Conclusions & Future Work

Our system achieves our goal to recommend actors or actresses to movie watchers and at the same time provides a good interface. We built front-end, back-end, databases, and crawlers to finish our project. In terms of future work, there are a few directions which can be further explored. The first one is to try other similarity measures. There are many classical and more cutting-edge measures which might lead to better results. Besides, we could extend the system to allow movie watchers to enter data into the database. Another thing we would improve is to add Admin page so that users can register and log into our web. They can create their own profile. They could save their favorite collections so that they could view them later. In terms of visualization, there is still some room to improve. For example, the first thing is to display the image of each actor when users hover their mouse over a node. Also, we could display the poster of the each movie. At the same time, we could also provide links so that they can be routed to movie theater websites to see tickets information once they decide to watch an actors' movies.

6 Appendix: Individual Contributions

Jintao Jiang Implement crawler and build database to store data.

Hao Wang Implement PathSim algorithm to compute similarity among actors.

Shuo Feng Implement visualization with d3.js; In charge of presentation.

References

1. Sun, Yizhou, et al. "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks." Proceedings of the VLDB Endowment 4.11 (2011): 992-1003.