When a user interact with large catalog of items. There are two ways to two ways to interact with such large catalogs.

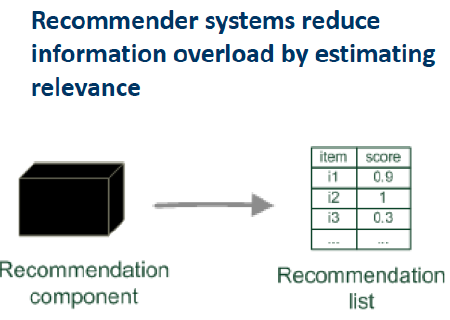
* Search
* Recommendation

With the rapid development of the internet, the volume of data has grown exponentially. Because of the overload of information, it is difficult for users to pick out what interests them among a large number of choices. To improve the user experience, recommender systems have been applied for scenarios such as music recommendation [1], movie recommendation [2], and online shopping [3].

Recommendation Algorithms mainly categorized into:

1. collaborative filtering (CF)-based recommender systems,
2. content-based recommender systems, and
3. hybrid recommender

**Recommendation Systems Overview**

One common architecture for recommendation systems consists of the following components:

* candidate generation
* scoring
* re-ranking
  1. Might be context dependent
  2. Relevance score used for ranking

**Candidate Generation**

In this first stage, the system starts from a potentially huge corpus and generates a much smaller subset of candidates. For example, the candidate generator in YouTube reduces billions of videos down to hundreds or thousands. The model needs to evaluate queries quickly given the enormous size of the corpus. A given model may provide multiple candidate generators, each nominating a different subset of candidates.

**Scoring**

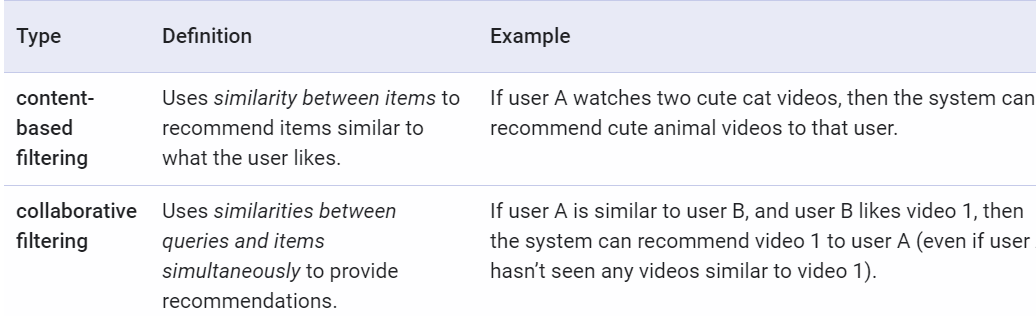
Next, another model scores and ranks the candidates in order to select the set of items (on the order of 10) to display to the user. Since this model evaluates a relatively small subset of items, the system can use a more precise model relying on additional queries.

**Re-ranking**

Finally, the system must take into account additional constraints for the final ranking. For example, the system removes items that the user explicitly disliked or boosts the score of fresher content. Re-ranking can also help ensure diversity, freshness, and fairness.

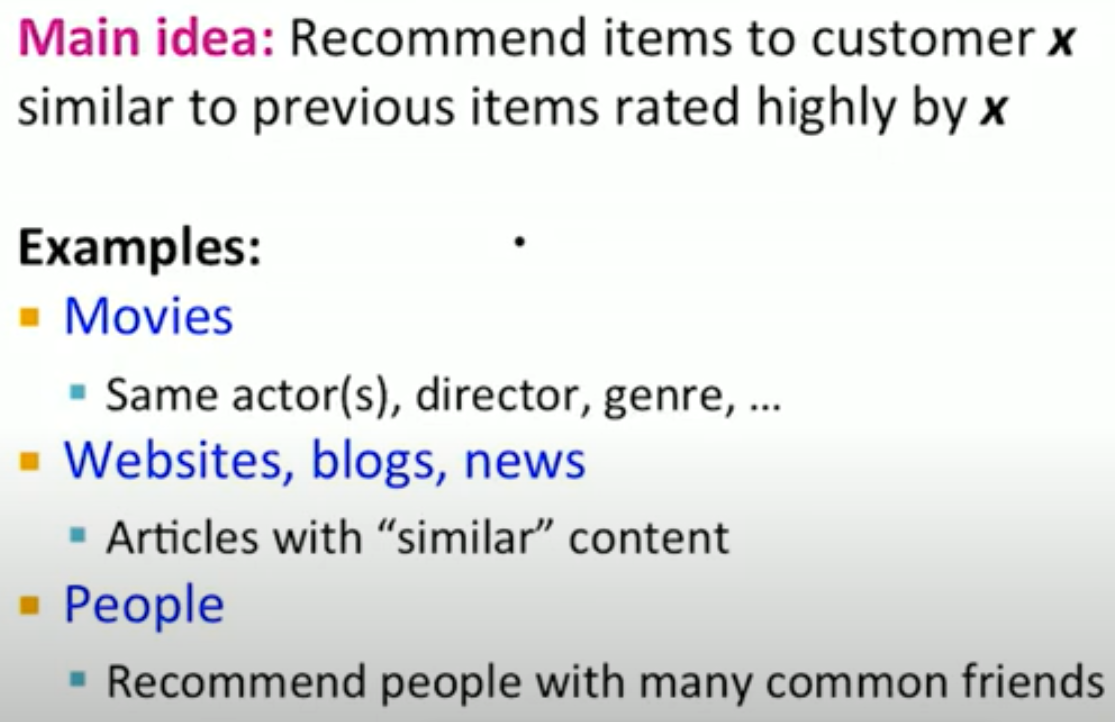
**Candidate Generation Overview**

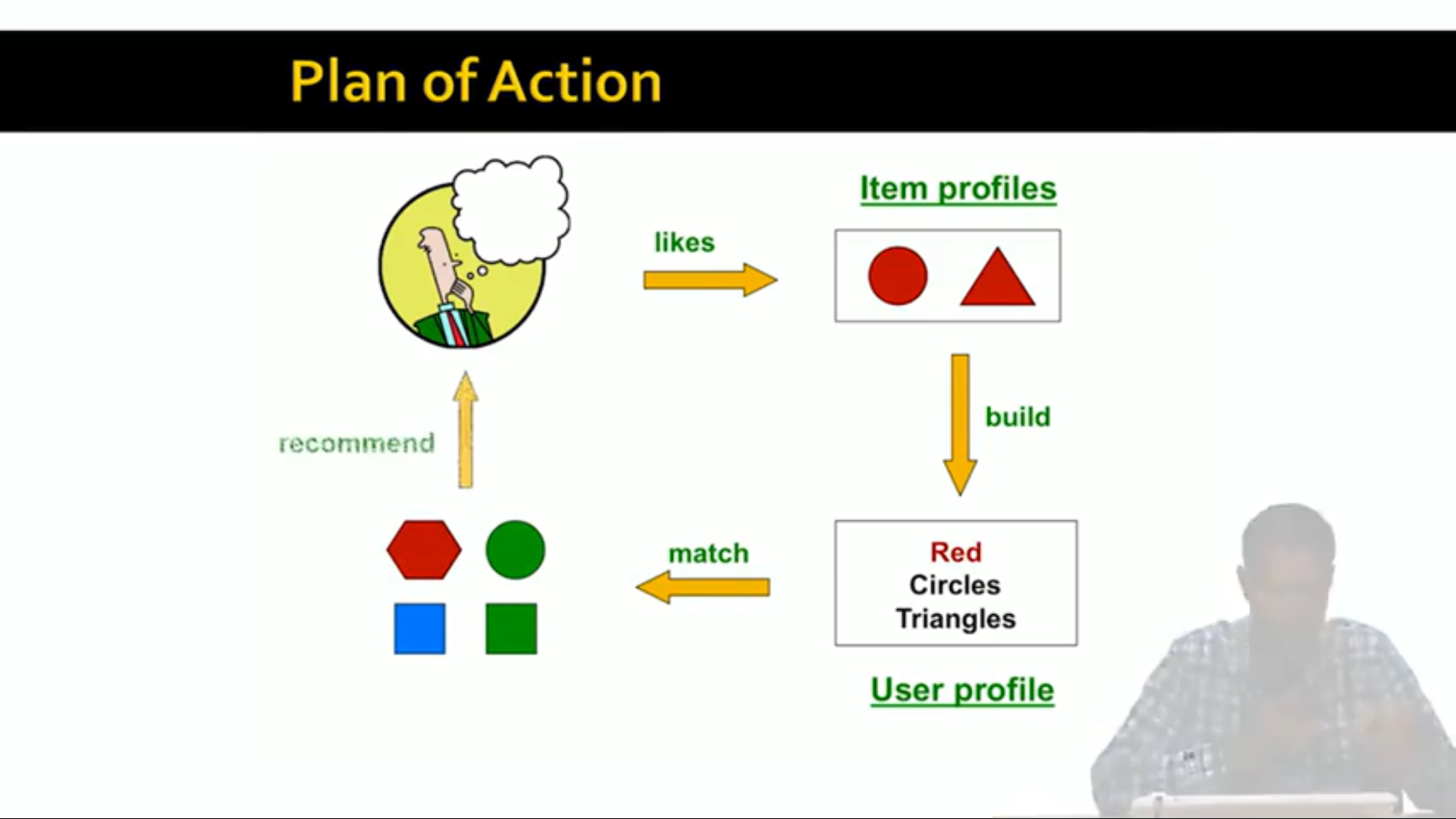
Candidate generation is the first stage of recommendation. Given an Item (Query), the system generates a set of relevant candidates. The following table shows two common candidate generation approaches:



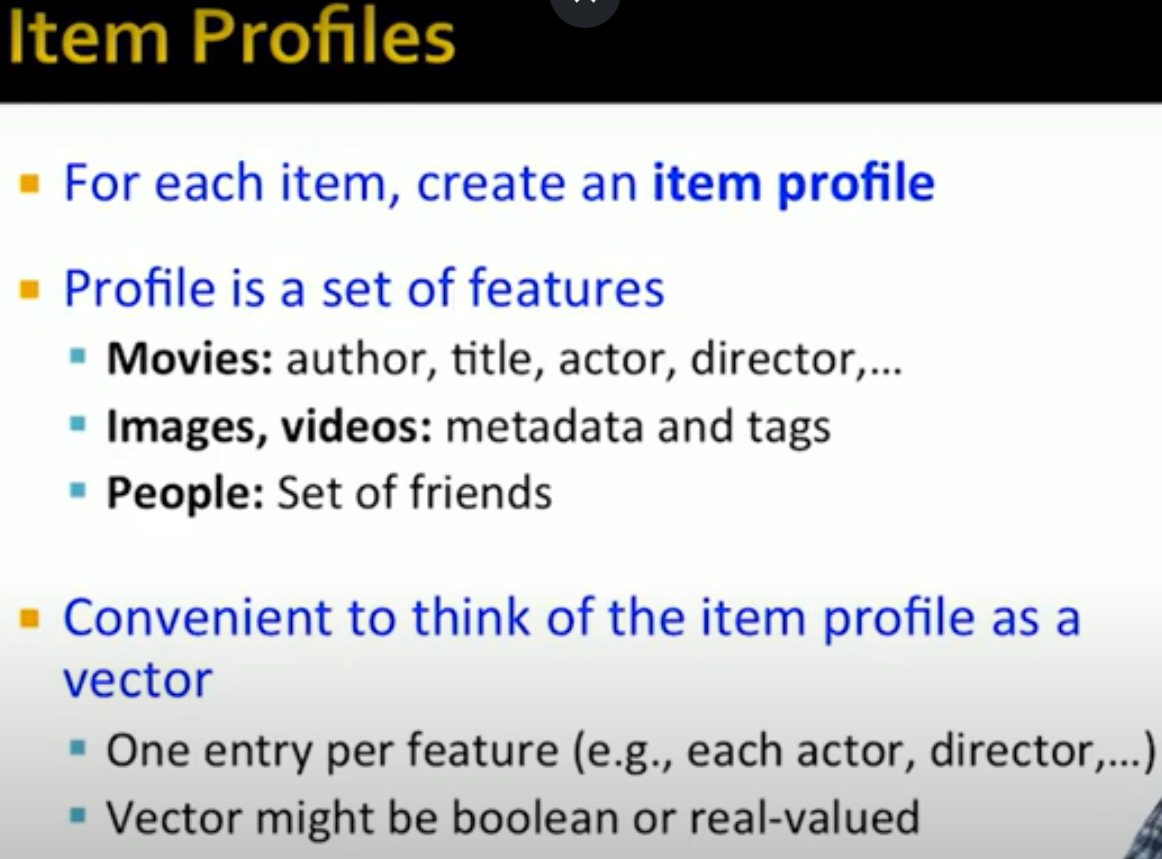
**Content Based Filtering**

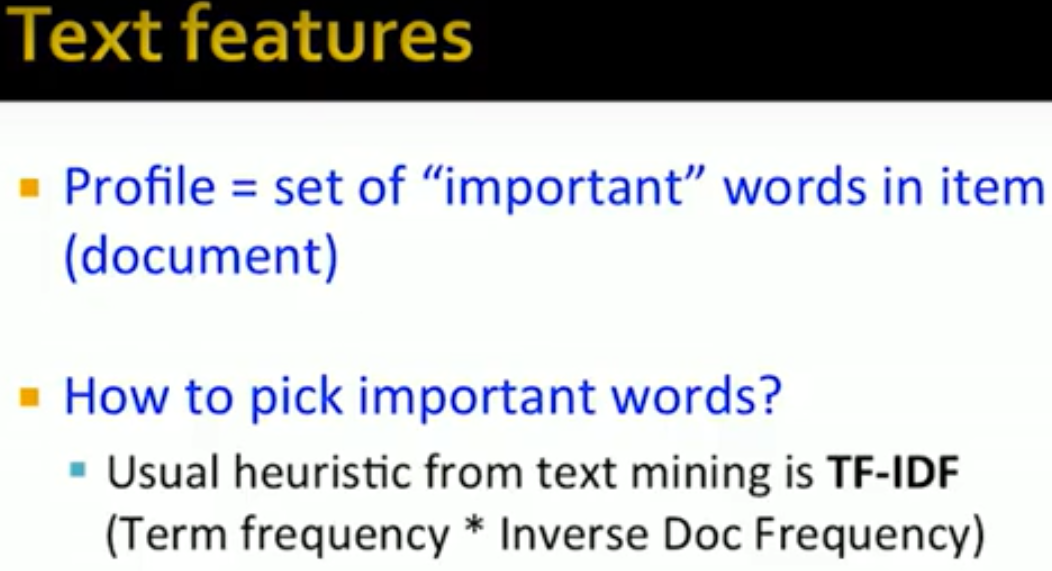
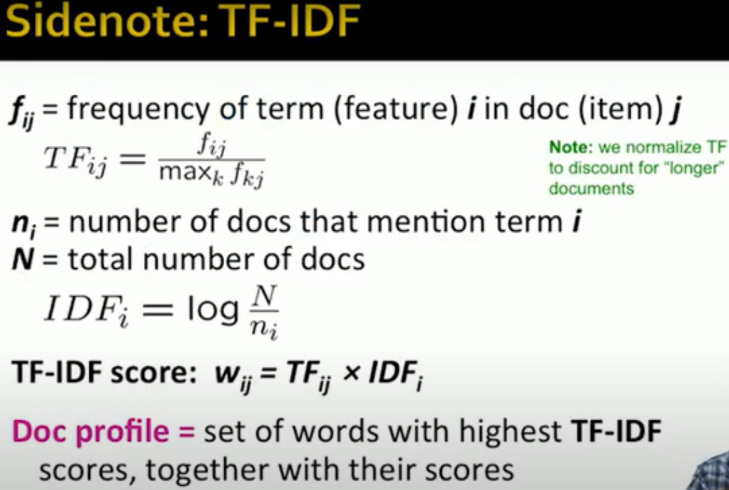
Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.





How to build these item profiles?

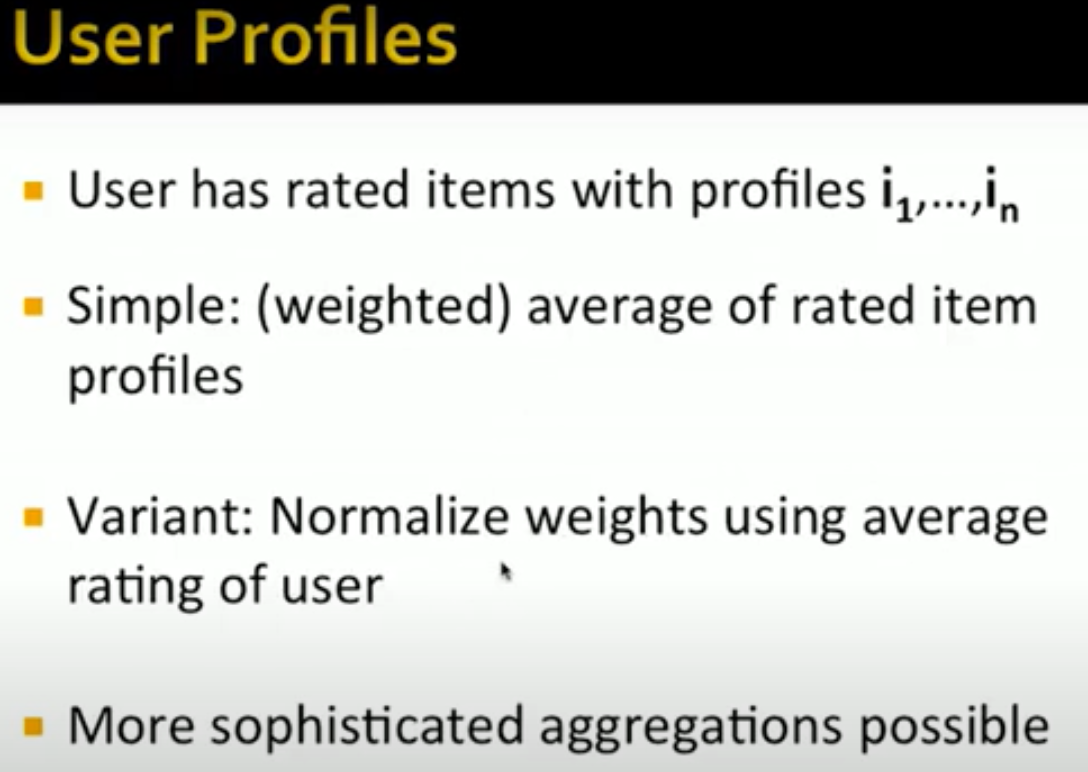
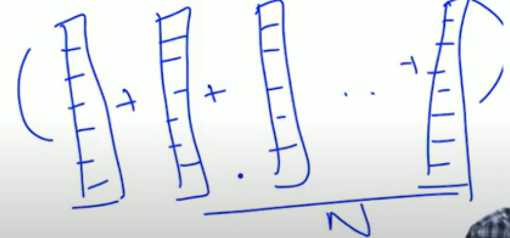




**User Profiles:**

Simplest way to construct from set of items is just to **average** the item profiles.

“i” is a vector for an item with profile. i1 through in are vectors.



This does not account user likes and dislikes. Therefore we might add weight average of item profiles.

Examples:

**Advantages**

* The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.
* The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.

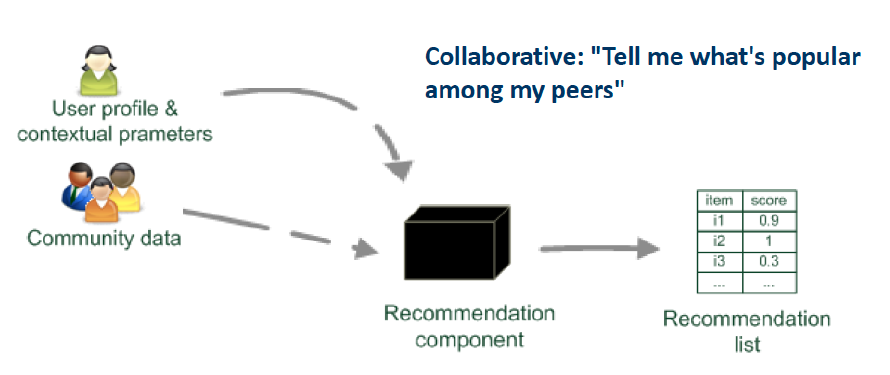
**Disadvantages**

* Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features.
* The model can only make recommendations based on existing interests of the user. In other words, the model has limited ability to expand on the users' existing interests.

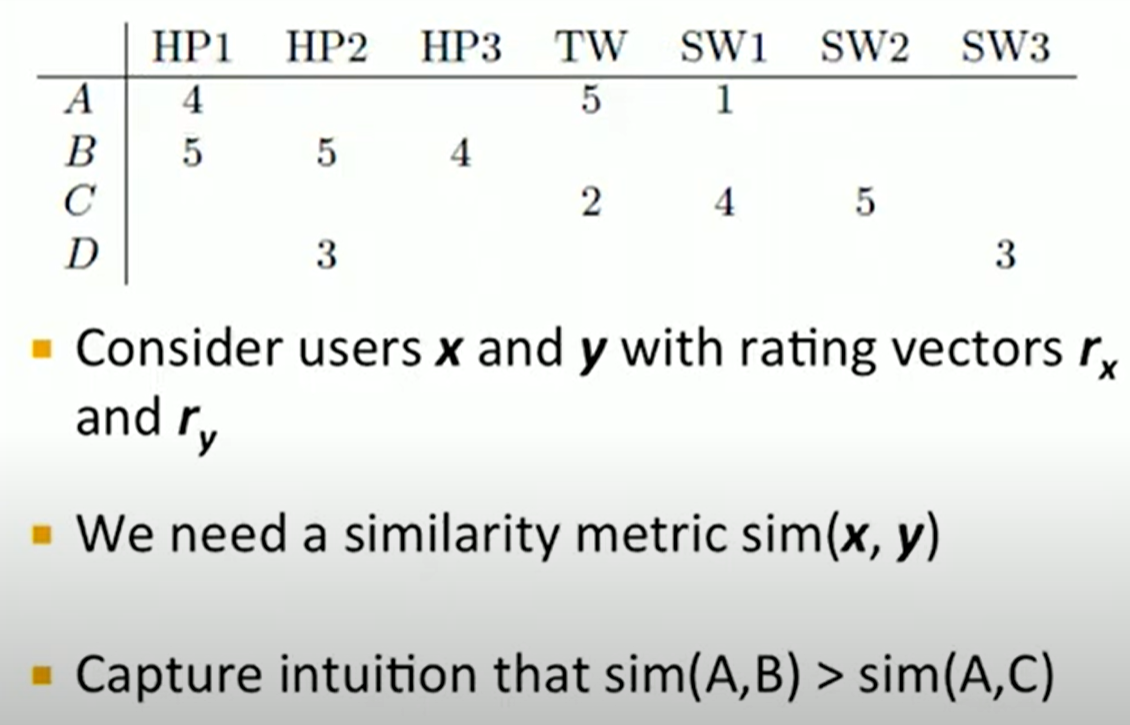
**Collaborative Filtering**

To address some of the limitations of content-based filtering, collaborative filtering uses *similarities between users and items simultaneously* to provide recommendations. This allows for unanticipated recommendations; that is, collaborative filtering models can recommend an item to user A based on the interests of a similar user B.

“Collaborative filtering uses the known **preferences** of a group of users to make recommendations or predictions of the unknown preferences for other users”.



Community is also called Neighborhood of users. Estimate the user preference based on the neighborhood users. Key is to find the Neighborhood of users similar to user X and find the movies that are liked by the lot of users in that community and recommend those items to user X. We need to define the notion of similarity between users.

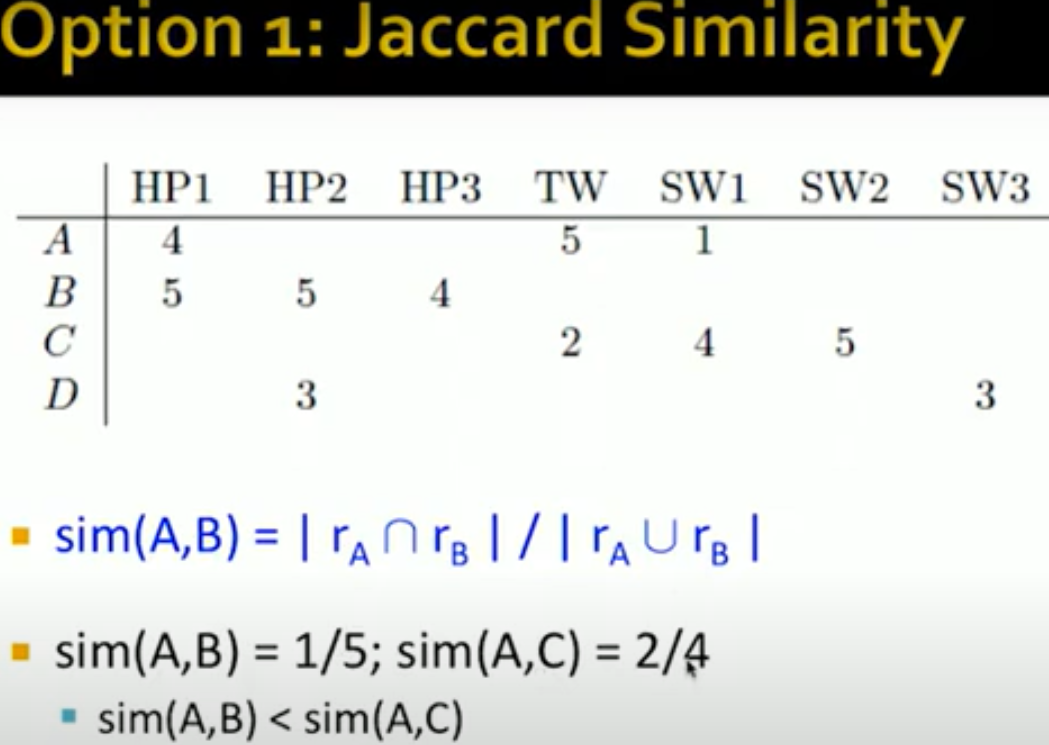


It should captures the intuition that the users with similar taste have higher similarity than the users with dissimilar tastes.

Ex: User A & B have rated HarryPorter, However they rated with fairly high.Where as A &C actually rated two movies in common buth their ratings are dissimilar. C likes StarWar but A does not. Intuitively, A and C are dissimilar while A and B are similar. We would like to capture this intuition while defining the notion of Similarity

**Option 1: Jaccard similarity**

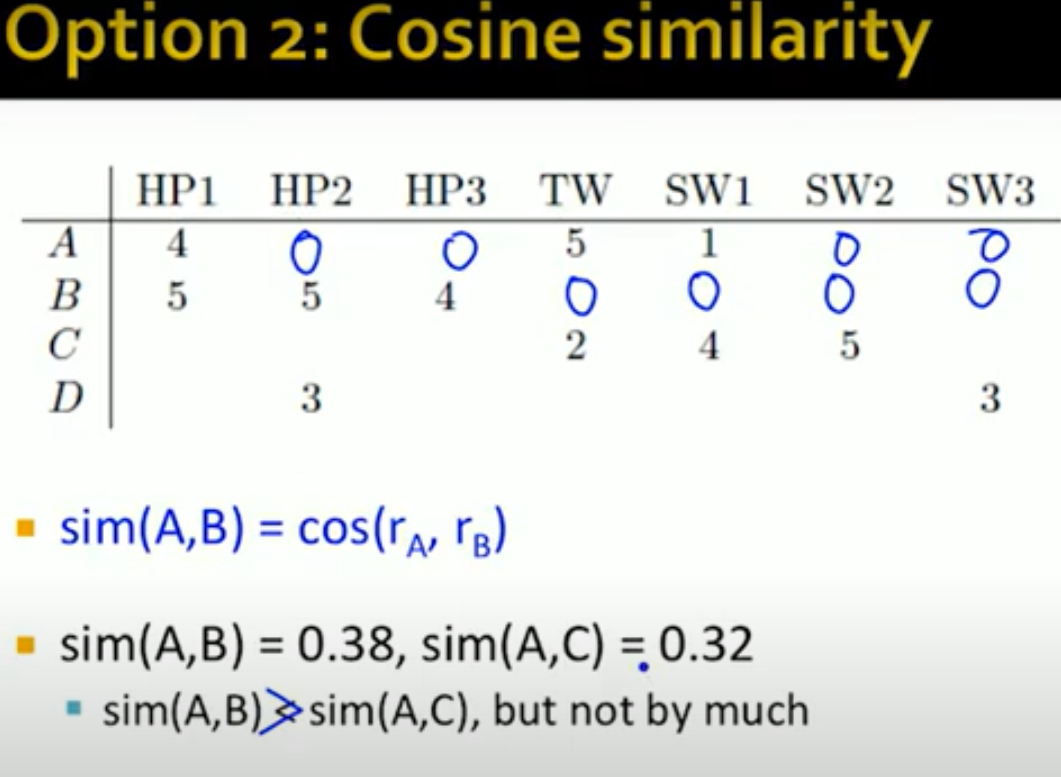
Jaccard similarity is nothing but 1-Jaccard distance.



This is counter to the intuition that we wanted to capture. Jaccard ignores the rating values.

**Option 2: cosine similarity**

Replace the known values to zeros to compute the cosine similarity.

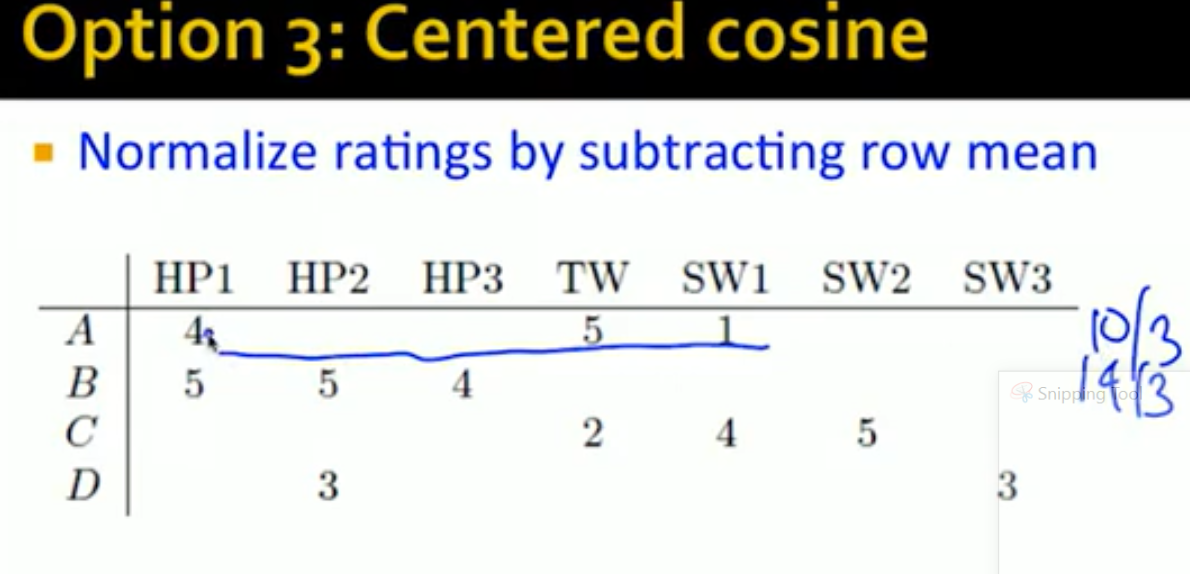


AB is actually marginally greater than the similarity of AC. But it actually Doesn’t really captures A & B are much more similar than A &C.

Problem with cosine similarity is, It treats missing ratings as negative ratings. In 0-5 rating scale, 0 is worse possible rating. Actually user A has not rated HP2, we are assuming 0 rating given the fact that they actually liked HP1. This is the problem with cosine that we will fix with Centered cosine.

**Option 3: Centered Cosine**

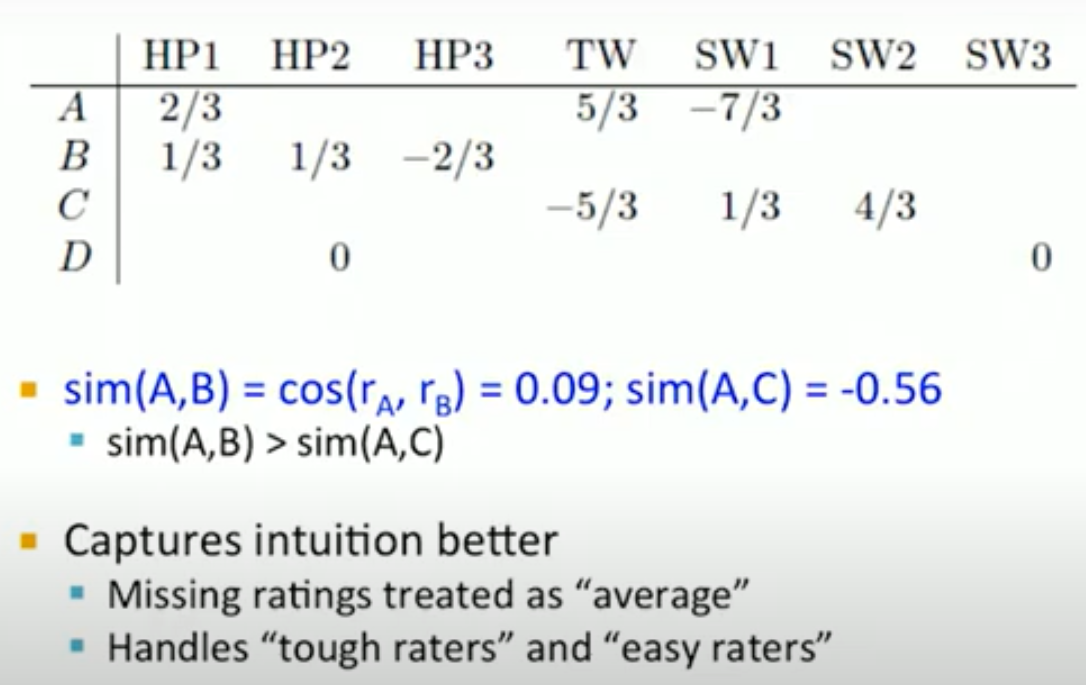
Normalize the ratings by subtracting row mean for a given user.



Blank ratings are treated as zero



If we sum of any ratings across row, it will be zero. What we have done is we have centered rating of each user around zero. Zero become average rating for every user and +ve rating indicates user liked the movie more than average. –ve ratings indicates user liked the movie less than the average. Magnitude of ratings shows how much he liked or disliked the specific movie. Once centering is done, we compute the cosine similarity on these centered ratings.



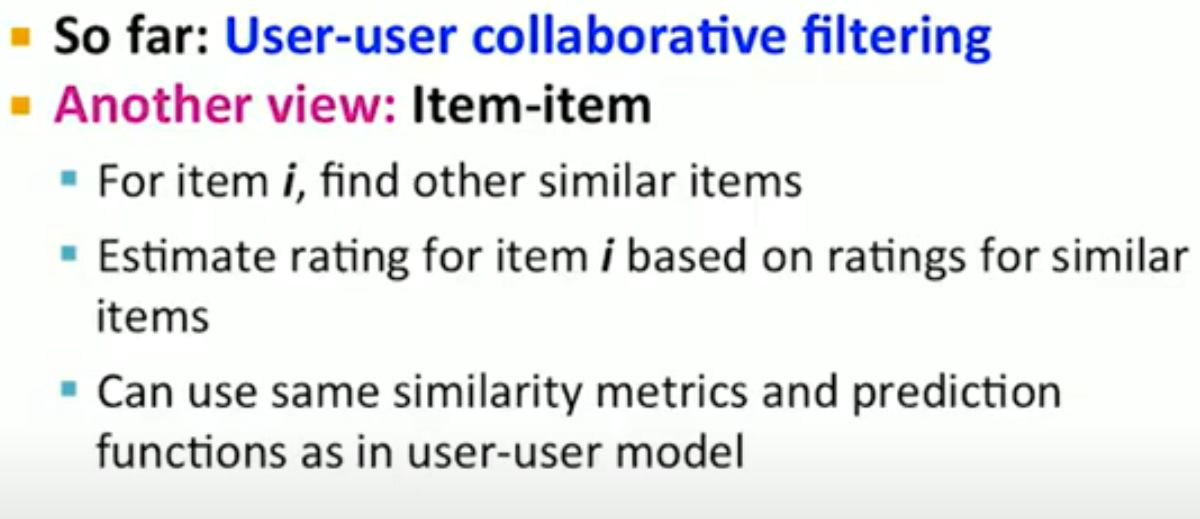
Notice we observe big gap between AB and AC and captured out intuition better. Centered cosine similarity technique is also known as “**Pearson** correlation”

**Rating Prediction**

Consider N users who have rated the item “i”



So far User-User collaborative filtering.



YES

To solve the information explosion problem and enhance user experience in various online applications, recommender systems have been developed to model users preferences. Although numerous efforts have been made toward more personalized recommendations, recommender systems still suffer from several challenges, such as data sparsity and cold start. In recent years, generating recommendations with the knowledge graph as side information has attracted considerable interest. Such an approach can not only alleviate the abovementioned issues for a more accurate recommendation, but also provide explanations for recommended items. In this paper, we conduct a systematical survey of knowledge graph-based recommender systems. We collect recently published papers in this field and summarize them from two perspectives. On the one hand, we investigate the proposed algorithms by focusing on how the papers utilize the knowledge graph for accurate and explainable recommendation. On the other hand, we introduce datasets used in these works. Finally, we propose several potential research directions in this field.

-------------------------

1. **Recording user events**

This page describes how you record user events. Recommendations AI uses real-time user events to generate next-item recommendations. Recording as many types of user events as possible with valid product information increases the recommendation quality.

**User event types**

There are several user event types that you can record as users browse your retail site

|  |  |
| --- | --- |
| User event name | User action |
| [add-to-cart](https://cloud.google.com/recommendations-ai/docs/user-events#add-to-cart) | Adds product to cart |
| [add-to-list](https://cloud.google.com/recommendations-ai/docs/user-events#add-to-list) | Adds product to list |
| [category-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#category-page-view) | Views special pages, such as sale or promotion pages |
| [checkout-start](https://cloud.google.com/recommendations-ai/docs/user-events#checkout-start) | Starts checkout process |
| [detail-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#detail-page-view) | Views product detail page |
| [home-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#home-page-view) | Views homepage |
| [page-visit](https://cloud.google.com/recommendations-ai/docs/user-events#page-visit) | Views generic page not included in other page view events |
| [purchase-complete](https://cloud.google.com/recommendations-ai/docs/user-events#purchase-complete) | Completes checkout |
| [refund](https://cloud.google.com/recommendations-ai/docs/user-events#refund) | Returns or is refunded for a purchased product |
| [remove-from-cart](https://cloud.google.com/recommendations-ai/docs/user-events#remove-from-cart) | Removes product from cart |
| [remove-from-list](https://cloud.google.com/recommendations-ai/docs/user-events#remove-from-list) | Removes product from list |
| [search](https://cloud.google.com/recommendations-ai/docs/user-events#search) | Searches the catalog |
| [shopping-cart-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#shopping-cart-page-view) | Views shopping cart |

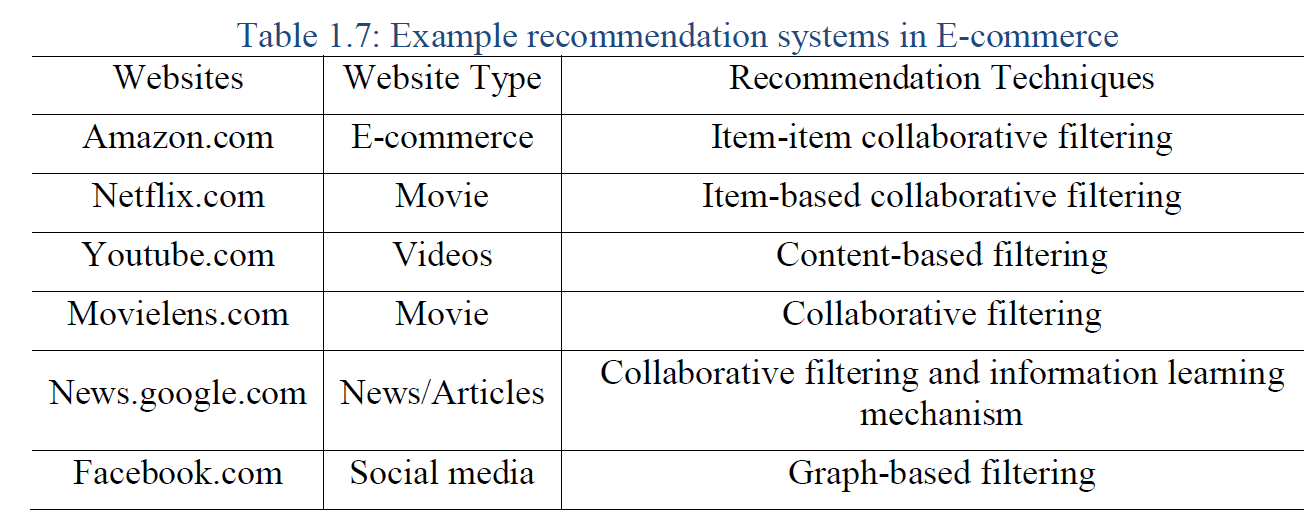
### Event type priority

For the highest quality recommendations, we recommend that that you record user events for all event types. The following table describes the priority of the different user event types. You must log the highest priority user events to achieve quality data models.

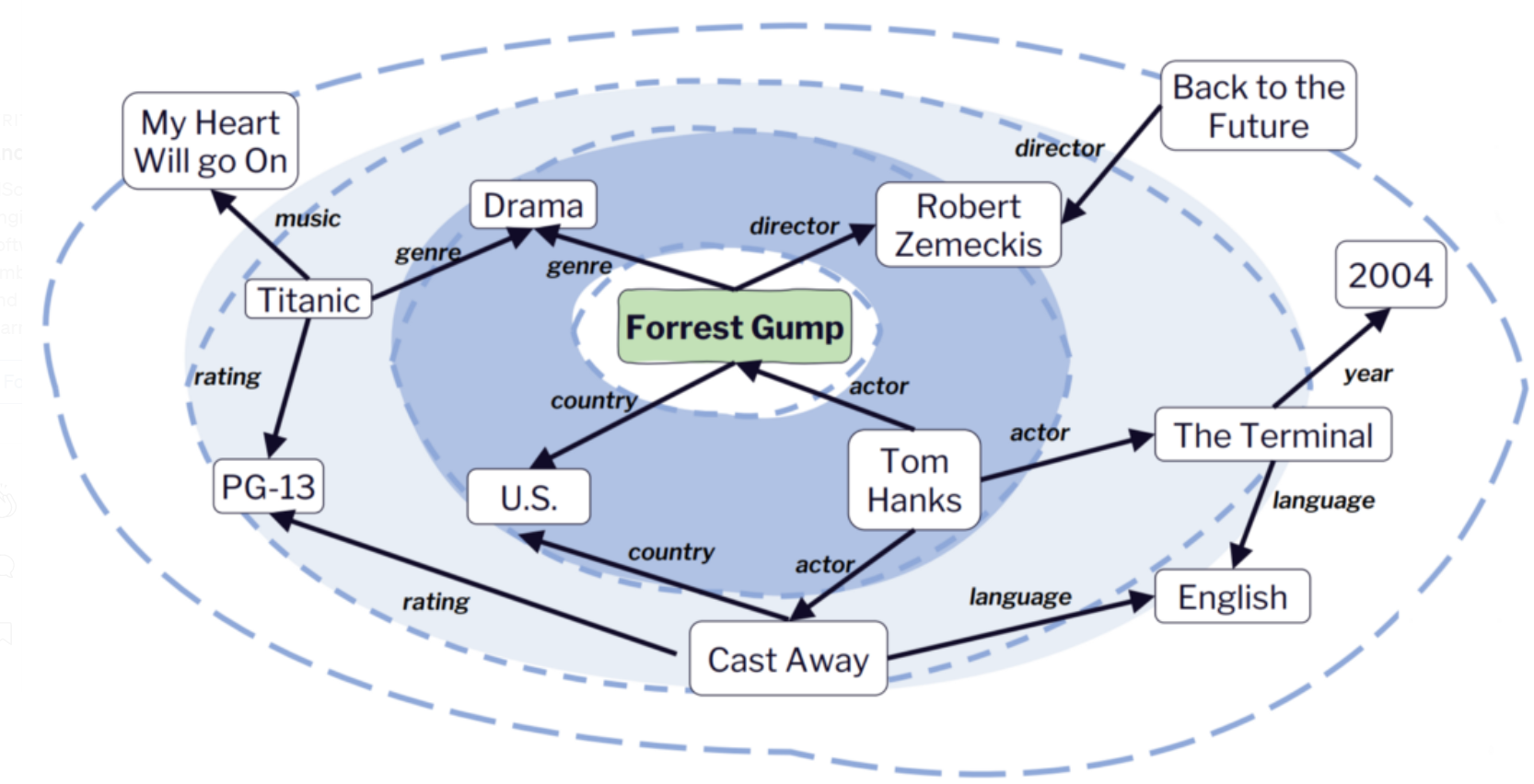
|  |  |
| --- | --- |
| Priority | User Events |
| Required for initial live experiment | [add-to-cart](https://cloud.google.com/recommendations-ai/docs/user-events#add-to-cart)  [detail-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#detail-page-view)  [home-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#home-page-view)  [purchase-complete](https://cloud.google.com/recommendations-ai/docs/user-events#purchase-complete) |
| Important for improving model quality over time | [checkout-start](https://cloud.google.com/recommendations-ai/docs/user-events#checkout-start)  [category-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#category-page-view)  [remove-from-cart](https://cloud.google.com/recommendations-ai/docs/user-events#remove-from-cart)  [search](https://cloud.google.com/recommendations-ai/docs/user-events#search)  [shopping-cart-page-view](https://cloud.google.com/recommendations-ai/docs/user-events#shopping-cart-page-view) |
| Nice to have | [add-to-list](https://cloud.google.com/recommendations-ai/docs/user-events#add-to-list)  [page-visit](https://cloud.google.com/recommendations-ai/docs/user-events#page-visit)  [refund](https://cloud.google.com/recommendations-ai/docs/user-events#refund)  [remove-from-list](https://cloud.google.com/recommendations-ai/docs/user-events#remove-from-list) |

1. **Creating recommendation model**

***Some Recommendation Systems***



**Recommendations powered by Knowledge Graphs**



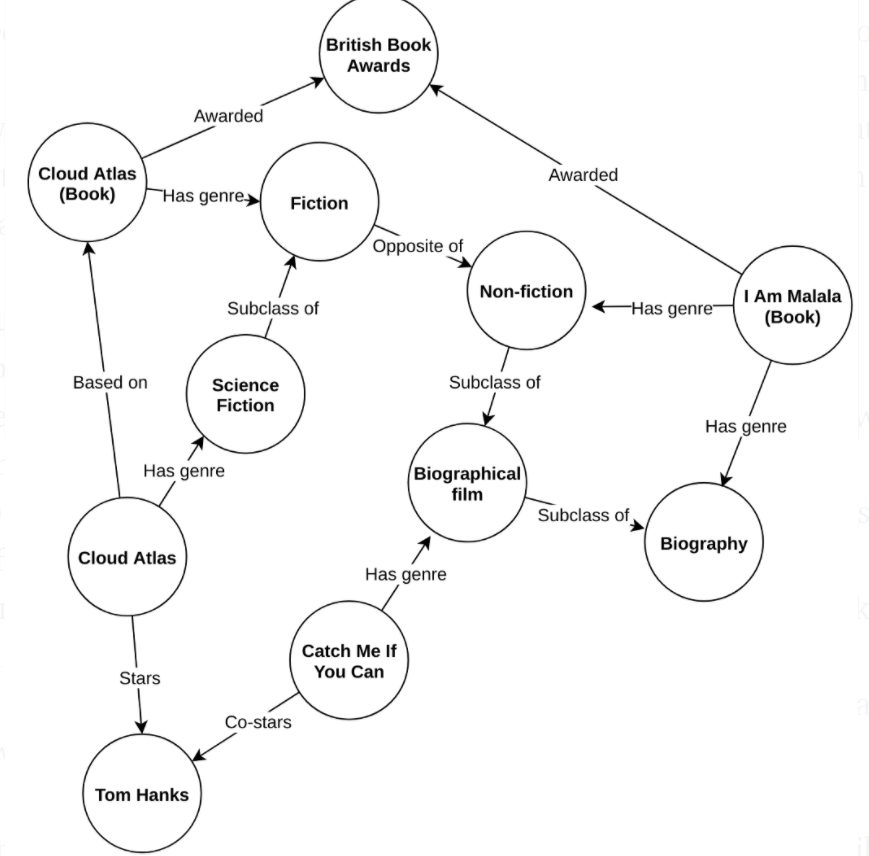
Netflix uses a powerful **recommendation system** to generate this list. Based on what you have watched and rated, it builds a profile of your tastes in terms of genres, plots, actors and more, and uses this profile *to recommend movies that fit to your taste*.

We’re going to build a content-based recommender that uses a user’s information as well as a knowledge graph (powered by a Neo4j graph database) for recommending products to users. First, however, it’s worth discussing why a knowledge graph and a graph database is necessary at all in the first place.

**Why graphs?**

Intuitively, for implementing a content-based recommender, we should be able to model all movies as simple objects with a list of properties (for instance, genres, actors, and subjects) in an SQL database. This, indeed, is easily implemented with a few tables connected through appropriate relationships.

The power of graph databases becomes clear once we start considering connections other than Movie→HasProperty→Property. In fact, we want to express a much richer model where we represent inter-relations between properties - effectively allowing properties to have properties. This also allows us to explicitly model the nature of each relationship. In this case, the expressiveness of the graph model becomes clearer:



The above is an example **knowledge graph** representing movies and books as well as actors, genres and the complex interelationships among them. In a knowledge graph, not only do we know what items are related to what properties, we know how they are related and impose no restrictions on what can be related and how.

With such a graph structure, we suddenly have many new ways of describing the items we want to recommend. This translates to more complex reasoning about what a given user might appreciate and why when we compare two items.

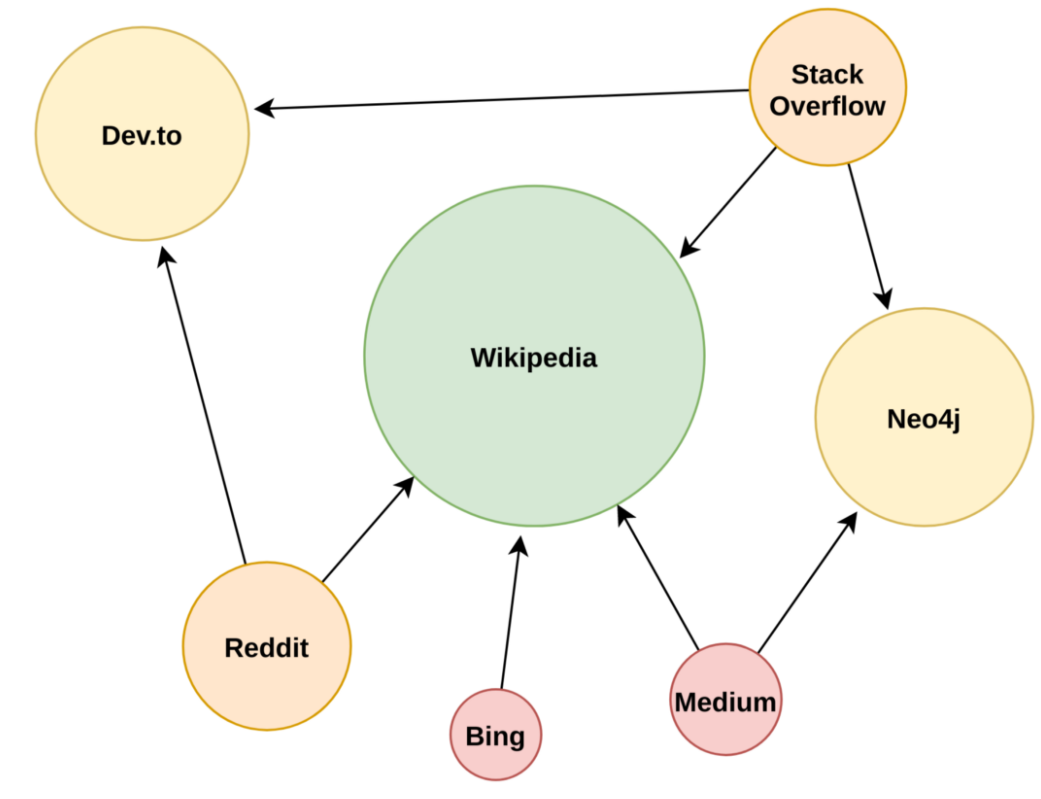
For example, if a user likes “Cloud Atlas” (the movie), they might like “Catch Me If You Can” because Tom Hanks stars in both of them. On the other hand, they could be looking for something different from fiction. If they’re looking for a book to buy, they might like “Cloud Atlas” (the book), and if they also liked “Catch Me If You Can”, maybe they would like the “I Am Malala” book as it is also a biography and won awards similar to the Cloud Atlas book.

While modelling this with standard SQL technologies is definitely possible, it is usually very difficult because of the rich structure. Instead, in a graph database, modelling such structure is [more straightforward](https://neo4j.com/blog/data-modeling-basics/?ref=blog). Also, querying a lot of relationships in an SQL database like this is [not exactly a](https://neo4j.com/whitepapers/overcoming-sql-strain-graph-databases/?ref=blog) [very efficient](https://neo4j.com/whitepapers/overcoming-sql-strain-graph-databases/?ref=blog) [operation](https://neo4j.com/whitepapers/overcoming-sql-strain-graph-databases/?ref=blog). What’s more is that in a graph database, we are free to extend the structure of our database graph as we’d like and to represent an ever-evolving domain.

**Making recommendations**

To suggest items to users, it is common to deploy very complex machine learning models. Here, we will instead be exploiting the full power of graphs by using a variant of the ***PageRank* *algorithm for making recommendations*** *for our users*. PageRank is an algorithm that is at the core of Google’s ranking algorithm for web-pages. It is used to rank the most relevant and important pages on the internet based on how they are connected. *This means that it is used to evaluate the importance of a page.*

The algorithm models a random web-surfer navigating the web by following links between individual web-pages. Web pages are presented as nodes and the connections (the edges) are created when a page contains a link to another page. The PageRank of a given website, i.e., a node in the web-graph, is given by how likely would be a user to end up on a specific web page if browsing the web aimlessly.



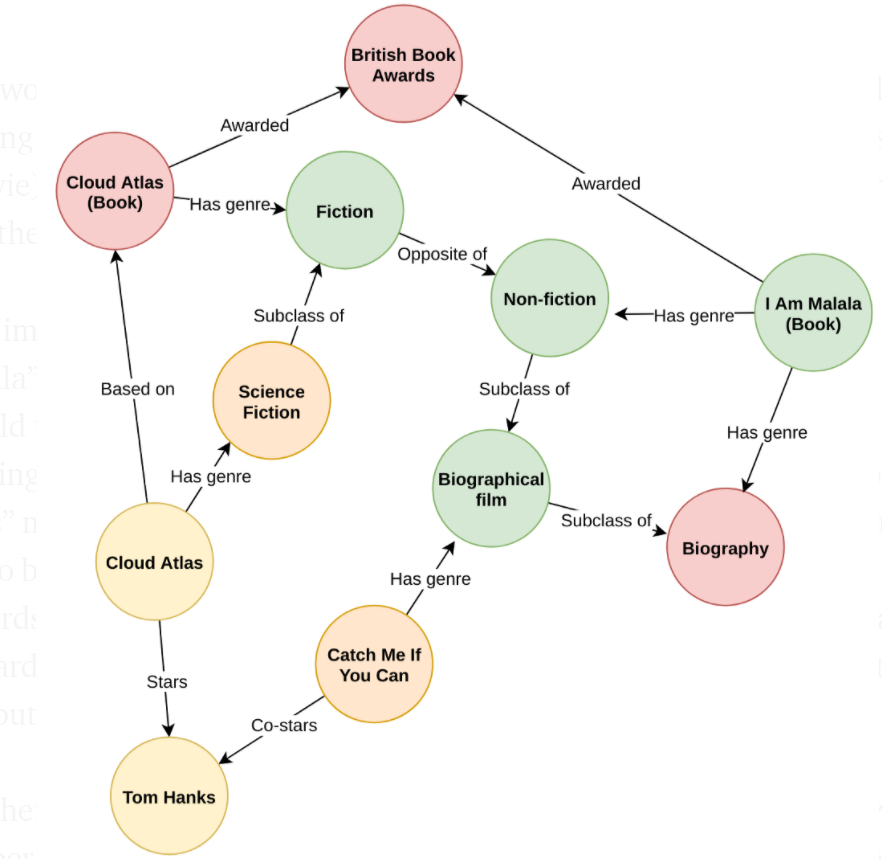
In the graph in the figure, the most important web-page would be Wikipedia, followed by Neo4j and Dev.to, followed by Google and Reddit, and so on.

In the PageRank model, we assume that the random web-surfer can teleport to any page in the entire network at any time. This is analogous to the surfer simply typing in a different URL in the browser instead of following the links on a page. In a variant called **Personalized PageRank**, we limit the target pages the surfer can teleport only to a specific set of graph nodes (this is called the preference set or the *personalized set* because they represent the pages a specific user likes the most). For example, if we “personalize” the PageRanks by only allowing the surfer to teleport to Medium, we get the following rankings:



Note that the random-surfer model makes no requirement for what the graph is modelling. In the end, what we obtain is a ranking of nodes in the graph according to their relevance and importance, regardless of what the nodes represent.

So, we should be able to do something similar without movie-graph database, right? Yes! The global PageRank of the previous knowledge graph gives us the following rankings:

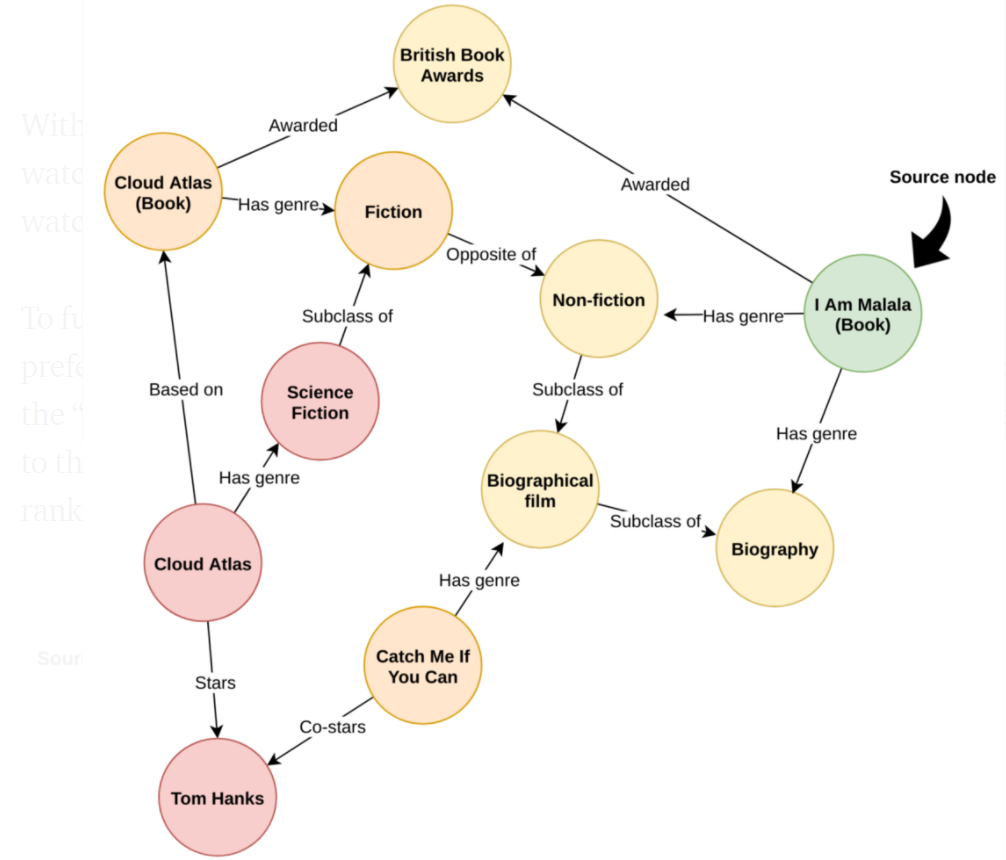


This would be the rankings we would use to present products to a newly visiting user, yielding a top-three of (1) “I Am Malala”, (2) “Cloud Atlas (movie)”, and (3) “Catch Me If You Can”. As such, we would recommend that the user reads “I Am Malala”.

Let’s imagine that the user accepts our recommendation, reads “I Am Malala” and enjoys it. What information does that give us? Also, how should the recommendation change as a result of this information? If nothing changes, we would recommend that the user watches the “Cloud Atlas” movie next, but perhaps the fact that they liked “I Am Malala” can be put to better use. An idea could be to simply personalize the PageRank towards “I Am Malala”. This will push nodes closely related to “I Am Malala” upwards through the ranks. As an added bonus, this allows us to limit the computation to the locally affected nodes.

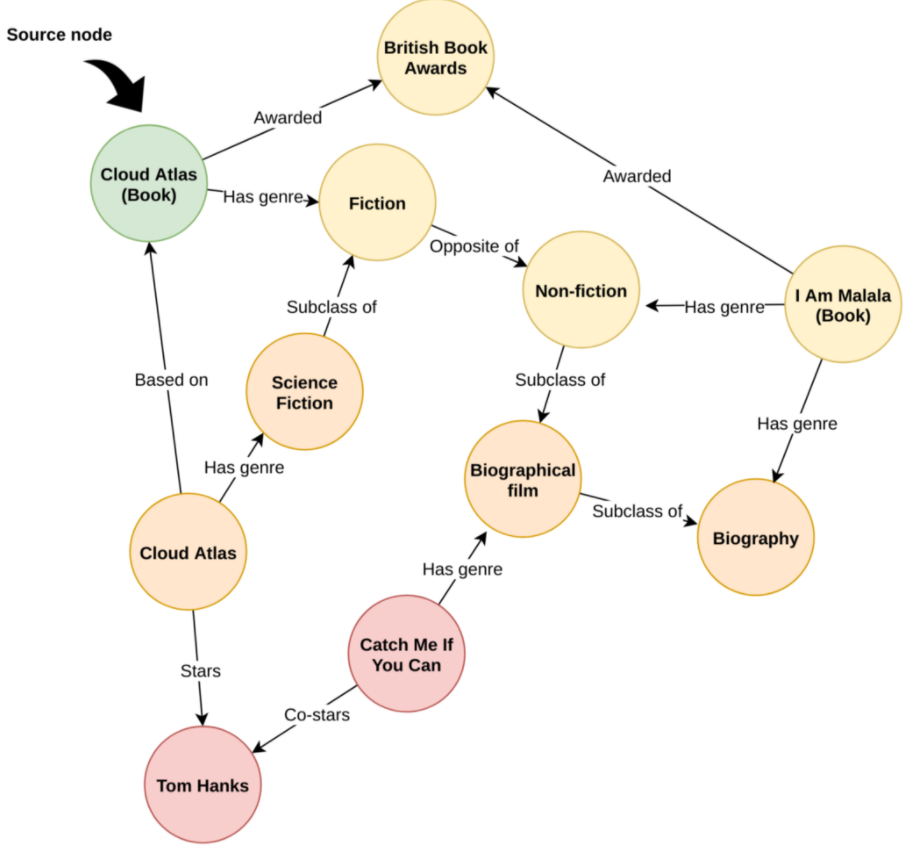
Another quite significant advantage of Personalized PageRank is that we can personalize the ranks *even further* by assigning user-specific relation weights. For example, if a user likes seeing the same actors in different movies, we could weigh the Stars and Co-stars relations highly for that user.

Running Personalized PageRank over the same graph with “I Am Malala” as the only source node, we get the following rankings:



With that small change, we would now recommend that the user either watches “Catch Me If You Can” or reads “Cloud Atlas (Book)” instead of watching “Cloud Atlas”.

To further demonstrate Personalized PageRank’s ability to adapt to user preferences, let’s instead assume we have a user who has read and enjoyed the “Cloud Atlas” book. In this case, we simply change the personalized set to that containing only “Cloud Atlas (Book)” and get the following rankings:



So, with no further intervention from our side, we now have a personalised top-three for this user: (1) “I Am Malala (Book)”, (2) “Cloud Atlas”, (3) “Catch Me If You Can”.

[Personalized PageRank](https://en.wikipedia.org/wiki/PageRank) has been proven to be a very effective ranking tool in the context of personalized recommendations ([Shams et. al 2016](https://arxiv.org/pdf/1604.03147.pdf)), and is even used by Twitter to present users with accounts they may want to follow ([Gupta et. al 2013](https://dl.acm.org/doi/10.1145/2488388.2488433)). Unfortunately, in it’s most basic form, PageRank is not a scalable algorithm as it requires several traversals over a potentially huge graph. Luckily for us, [Gallo et. al 2020](http://people.cs.aau.dk/~matteo/pdf/EDBT20-particle-filtering.pdf) presents a way to use particle filtering to very efficiently approximate PageRank over a knowledge graph. We will use this approach in the implementation later.

Reference:

https://towardsdatascience.com/movie-recommendations-powered-by-knowledge-graphs-and-neo4j-33603a212ad0#:~:text=In%20a%20knowledge%20graph%2C%20not,items%20we%20want%20to%20recommend.