# Recommender system

## Context of the project

The project started because an application of technology watch was in built and needed a recommendation algorithm. As the database was not already made, we have decided to create a POC of recommender system using the MovieLens dataset available online. We have chosen this dataset because it has the same characteristics as our future dataset (theme of items, user-item interactions, date, few tables…)

## Hypothesis and exploratory analysis

For the purpose of the technology watch application, we have decided to make 3 types of recommendation :

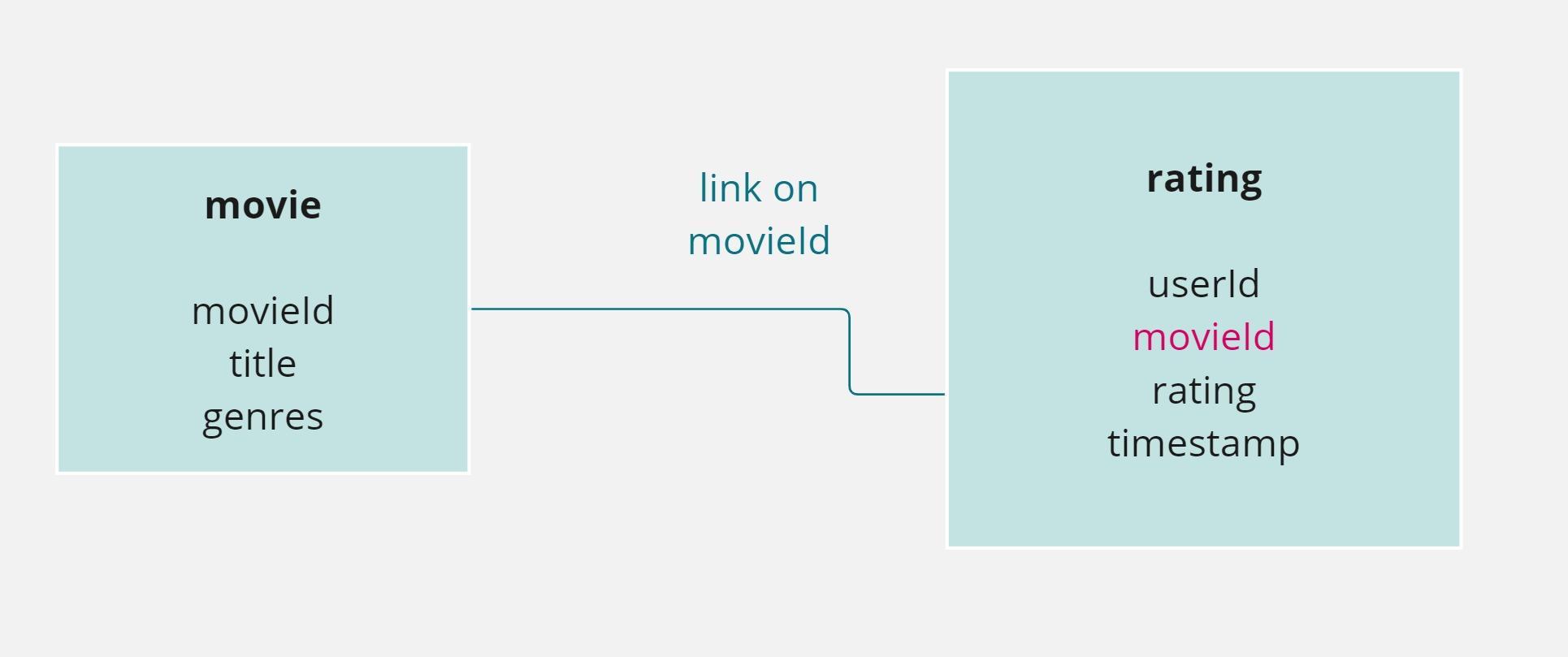
* Content based filtering
* Collaborative filtering
* Levenshtein

And of course, it means 3 types of data modeling that we are going to see.

1. Presentation of the dataset

We are using the [MovieLens dataset](https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset?select=rating.csv), originally composed of 6 tables on the topic of movies. We have chosen this dataset because it is one of the best for recommender system, as it has the most important data we will need, as well as a good data quality.

There 6 tables in the dataset, and we are going to use 2 of them :



We have :

* 20 000 263 ratings given by a userId to a movieId
* 1342 genres to classify the movies
* 26 744 movies
* 138 493 users

1. Content based filtering

For Content-Based Filtering, we will be using the 2 tables mentioned above, to use the movie genre and user interactions with them. Therefore, we simply add the "genre" column to the "rating" table.

In this approach, we assume that users are more likely to enjoy movies with similar features to the ones they have liked in the past. The features can include genres, cast, directors, and other descriptive elements of the movies.

**HYPOTHESIS:** Users are more likely to enjoy movies with similar characteristics to the ones they have rated positively in the past. We focus on the content features, such as genres, to make personalized recommendations. This approach is particularly useful in scenarios where user-item interaction data is limited, making it effective for handling the "cold start" problem for new users or items.

1. Collaborative filtering

For the Collaborative filtering, it will be easier: we only need to use the rating table.

But we need to take into account that we are talking about explicit data : in fact, we have the rating given by a user for a movie. But we don’t have data when the user did not give any rating. It means that we don’t know if the user did not watch the movie, or if he just did not rate it. So we can make the following assumptions :

**HYPOTHESIS** : I assume that a user did not watch a movie, if the user did not rate the movie.

1. Levenshtein distance

On Levenshtein distance, we will only use the movie table because it contains all the information we need: namely, only the names of the films. Here, the goal is simply to retrieve all distinct film names.

In our exploratory analysis, we hypothesize that leveraging Levenshtein distance can enhance the accuracy of our system when handling potential variations or typos in movie titles. The assumption is that movies with similar titles, even if there are slight differences or misspellings, might share content similarities. By using Levenshtein distance as a metric for string similarity, we aim to discover patterns in user queries and adapt our recommendation system to variations in title inputs. This exploratory approach allows us to accommodate a broader range of user interactions and potential title discrepancies, ultimately improving the overall user experience in the content-based recommendation system.

**HYPOTHESIS:** Leverage of Levenshtein distance in string matching will improve the system's ability to accurately identify and recommend movies, accommodating variations or typos in user-inputted movie titles.

## Modeling

Now that our data are ready for our models, we can deep dive in each of them.

### Content based filteringUne image contenant texte, capture d’écran, diagramme, conception Description générée automatiquement

Content-based filtering recommends items to users based on the characteristics or features of the items and the preferences of the users. It focuses on describing items and creating user profiles to match items with similar features to the user's preferences.

This approach is especially useful for handling the "cold start" problem, where there is limited information about new users or items.

The disadvantages are that we do not use the complete set of known user-item interactions (each user is treated independently), and we need to know the metadata information for each item and user. It may also lack in diversity in suggestions

In our modeling, we processed movie data, extracting titles and years while handling missing values appropriately. We explored genres, using TF-IDF to convert them into a numerical representation. Our recommendation system employs a content-based approach, finding similar movies based on genres.

The collaborative filtering counterpart considers the entire user-item interaction matrix, leveraging the behavior of similar users for recommendations. We specifically utilize User-Based Collaborative Filtering, employing a neighborhood-based method to calculate user similarities. This technique complements content-based filtering, ensuring a well-rounded recommendation system that excels in accuracy and personalization.

One crucial aspect is the use of TF-IDF (Term Frequency-Inverse Document Frequency) to convert item features, such as genres, into numerical representations.

TF-IDF works by assigning weights to terms based on their frequency in a document (term frequency) and their rarity across documents (inverse document frequency). In our case, TF-IDF transforms movie genres into a matrix, capturing the importance of each genre in describing a movie's content.

In the code, the TfidfVectorizer from scikit-learn is used for this purpose. The fit\_transform method is applied to the 'genres' column, creating a TF-IDF matrix. This matrix, along with feature names, becomes the numerical representation of movie genres used in the recommendation system.

The recommendation system then utilizes this TF-IDF-transformed matrix to find similar movies based on genres. The linear\_kernel function from scikit-learn calculates the similarity matrix, considering the cosine similarity between movies. This matrix guides the system in suggesting movies with similar content.

Additionally, the code efficiently filters and sorts the list of similar movies, ensuring that the recommendations provided are both relevant and ranked by their similarity scores.

In summary, TF-IDF is employed to quantify the importance of genres in describing movies, and this numerical representation, combined with cosine similarity calculations using linear\_kernel, contributes to the effectiveness of the content-based recommendation system in your code

In conclusion, content-based filtering proves valuable for its ability to handle the cold start problem and provide personalized recommendations based on item features. Integrated with collaborative filtering, our recommendation system achieves a balanced and effective approach, catering to diverse user scenarios and preferences.

### Collaborative filtering Une image contenant texte, capture d’écran, diagramme, conception Description générée automatiquement

Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users.

It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.

The advantage of this approach is that the entire user-item interaction (i.e., the matrix rᵤᵢ) is used, which generally allows for greater accuracy than the use of content-based models. Thus, these types of models are highly efficient in providing personalized content but also able to adapt to changing user preferences.

The drawback of this approach is that it requires some user interactions before being able to adjust the model. It is the cold start problem. For a first time user, we can use the previous technique “Content based filtering”.

There are 2 types of collaborative filtering :

User Based collaborative filtering : used to determine the best item recommendations for a target user.

Item Based collaborative filtering : used to determine the best user recommendations for a target item

Considering our context, we will use the 1st approach, the user based collaborative filtering.

To do that, we will use the neighborhood-based method. The neighborhood-based collaborative ﬁltering algorithms are based on the fact that similar users tend to show similar patterns of rating behavior and similar items receive similar ratings.

We will start by creating a matrix based on our dataset consisting of the reactions given by a set of users to some items from a set of items. Each row would contain the ratings given by a user, and each column would contain the ratings received by an item.

Then, to determine the similarity between two users, we will use the Nearest Neighbor function to compute the distance between the ratings users have given to common items. The ratings by each user are treated as vectors in a multi-dimensional space, with each dimension representing an item. We will use the Cosine similarity to take into account the angle between these user vectors. After computing similarities for a target user with all other users, a neighborhood of similar users is selected based on a fixed number of nearest neighbors (depending on how many we want to retrieve in the technology watch application).

### Levenshtein distance

Finally, we will use the Levenshtein distance to identify closely related items when a person does not search for the exact name of the article in the technology watch application.

The incorporation of Levenshtein distance in our modeling approach is geared towards handling potential variations or typos in movie titles. Unlike the TF-IDF and similarity calculation used in content-based filtering, Levenshtein distance focuses on the string-level similarity between movie titles.

In our implementation, the Levenshtein distance is encapsulated in the matching\_score function, which quantifies the similarity between two strings. This function is applied when determining the closest title to a given input title in the find\_closest\_title function.

By leveraging Levenshtein distance, our hypothesis is that the system becomes more adept at accommodating variations in user-inputted titles. The exploratory analysis suggests that this approach enhances the model's adaptability to diverse user queries and title discrepancies, ultimately contributing to a more user-friendly and accurate content-based recommendation system.