

# GRAND PROJET ML

Mohamed Kallel - Simon-Pierre Rodner - Jean-Christophe Rigoni

## 1. Introduction

### ● Project Context

As SDA students, and as part of our Machine Learning courses, we have chosen to work on the Recommendation System for Movies. Nowadays this topic has become a prime concern in many fields such as online retail, streaming services, and digital advertising. Their primary objectives are to increase sales and enhance user engagement by leveraging a deeper understanding of customer preferences. Given the abundance of data that companies already possess, it is mandatory to utilize them to keep up with competition in these fast-paced industries. Machine learning provides the best automation to optimize these systems. This approach not only maximizes the value of existing data but also provides a scientific and precise method to achieve business goals.

### ● Dataset Overview

Movielens is a non-commercial, personalized movie recommendation platform online. It is run by GroupLens, a research lab at the University of Minnesota studying data exploration.

**Website:** <https://movielens.org/>

**Dataset:** <https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset/data>

The dataset contains 6 csv :

- rating.csv : contains users ratings of movies
- movie.csv : contains movie information, title, genres, year
- link.csv : contains identifiers to link to IMDb and TMDb
- tag.csv : contains tags about the movie theme and style written by users
- genome\_scores.csv : contains movie-tag relevance rates
- genome\_tags.csv : contains sorted tags and their identifiers

The Dataset contains information that we can condense in this presentation:

Users of the platform rate movies, a very small part of the users also write tags related to the movie, some are relevant, some are not. The platform provides a relevance rate for these 1128 unique tags.

The 4 most important informations are:

Movies = 27K

Users = 138K

Ratings = 20M

Tags = 38K

On a second level, we also learn that there are:

Users writing tags = 7.8K (out of 138K / 5.4%)

Movies being tagged = 19K (out of 27K / 70%)    Movies being rated = 26K (out of 27K / 96%)

- **Objectives**

The primary objectives of this project are to create efficient recommendation systems adapted to different use cases. A good system is not just one algorithm that understands it all. It's a set of many tools for as many use cases we can encounter. Here are some of these cases:

**Established user and movie:** In this case users have already rated movies, and we use the key approach **Collaborative Filtering**. This method needs existing user-item interactions to analyze and predict user preferences. It is most effective when a bigger amount of data is available and is particularly adapted for established users with a significant history of interactions.

**New-user or new-movies:** In this case we use the key approach **Content-Based Filtering**. This approach uses the characteristics of movies to recommend new movies to users or movies to new users. It relies only on metadata and, while not perfect, is very effective in addressing cold start cases where no user data is available.

We can also provide **segmented recommendations** like this:

Because you watch this movie (Hybrid Collaborative-Content with pearson correlation)

Blockbusters (filtered most popular movies)

Decade Preference (within a time signature)

Confidential movies (good ratings but not popular)

In order to implement all these cases, we are going to use different tools like Pearson Correlation, NLP embedding, Collaborative Filtering like SVD and LightFM.

Because there are no good models without **good data**, we are going first to explore them and attempt to extract the most relevant information out of them.

## 2. Data Exploration

- **Loading and Describing Data**

The most important CSV is rating.csv: 20 millions rows, and 3 columns, userId (int), movieId (int), rating(float). These three pieces of information are sufficient to operate collaborative filtering machine learning.

The second most important CSV is movie.csv. It contains titles, year of release and list of genres separated with pipes.

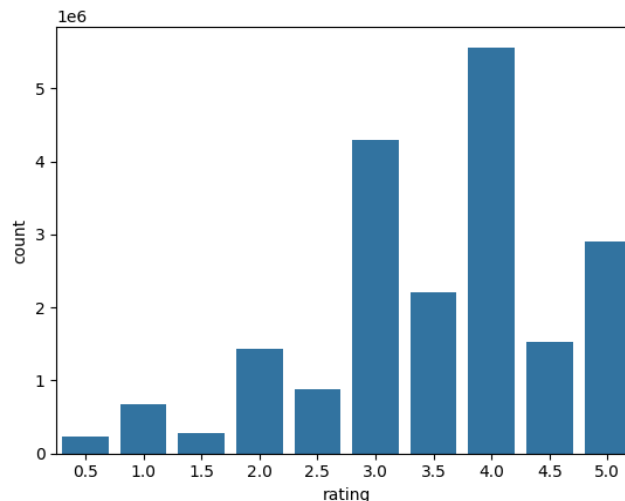
- **Descriptive Statistics Analysis and Visualizations**

We have made scatter plots and distribution plots in the Exploratory Data Analysis notebook. They help to understand the relationship between users and movies, but also the behavior of different types of users.

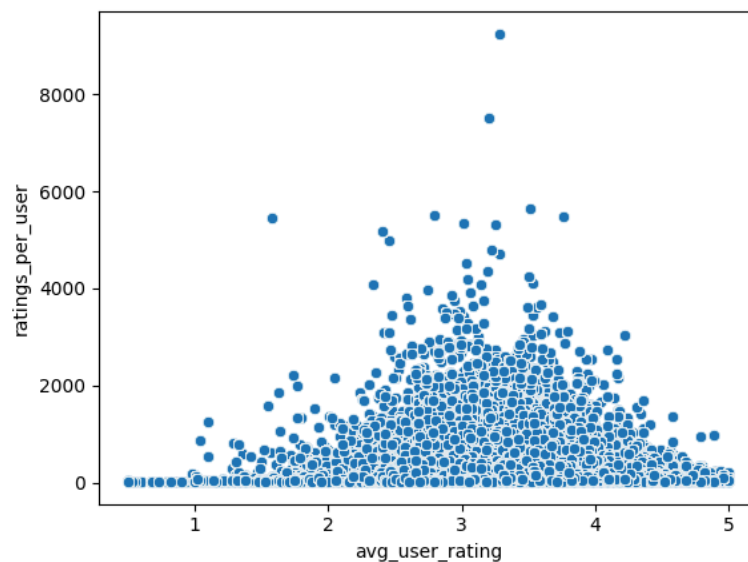
**Rating Distribution:**

Most ratings are distributed around the average

The average rating is 3.52



**Users average rating Distribution:**

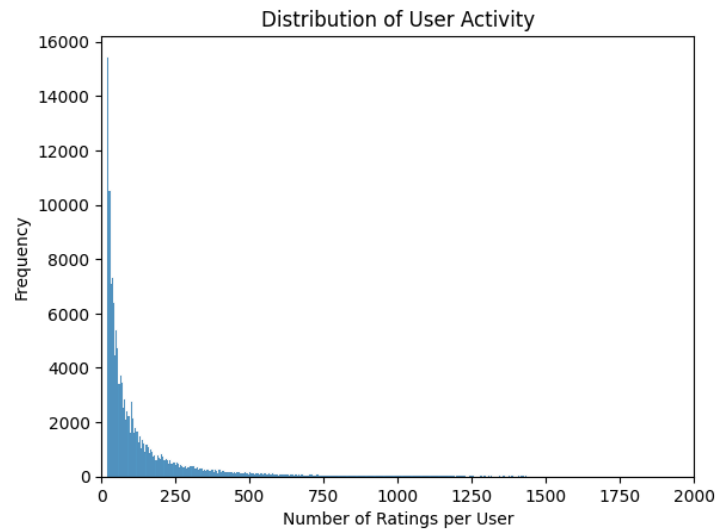


The scatterplot shows a tendency for users to get closer to the general average (3.52) as the number of their ratings rise. “The more you rate, the more average you get”

**Number of ratings per user Distribution:**

Most users (70%) do not rate more than 125 times.

The top rating users reach 9254 movie ratings.



### **Users Pattern Discovery:**

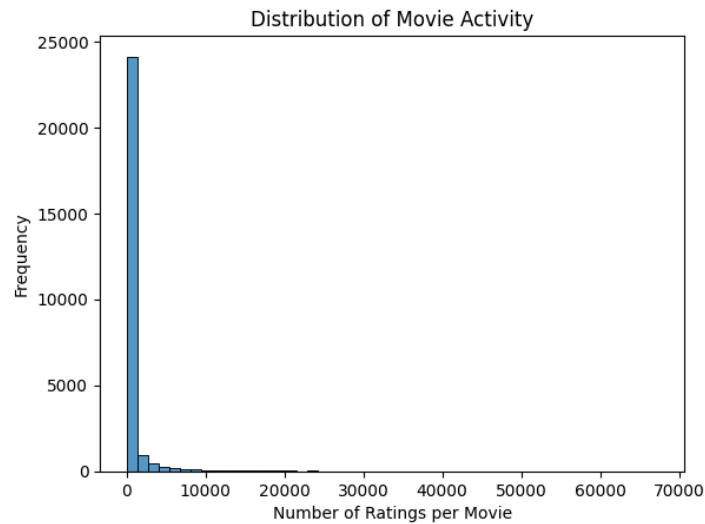
We separate this behavior in 3 main groups, bottom, middle, and top users, based on the quantity of ratings they have done, spreading from 20 to 9254 ratings.

<b><i>TOP ACTIVE USERS</i></b>	<b><i>MIDDLE ACTIVE USERS</i></b>	<b><i>LESS ACTIVE USERS</i></b>
Range : 644 to 9254 ratings	Range : 124 to 644 ratings	Range : 0 to 123 ratings
nb of ratings : 4993925	nb of ratings : 9962336	nb of ratings : 4999034
percentage of ratings : 25 %	percentage of ratings : 50 %	percentage of ratings : 25 %
nb of users : 4673	nb of users : 38105	nb of users : 95396
percentage of users : 3 %	percentage of users : 28 %	percentage of users : 69 %
nb of movies : 25748	nb of movies : 19658	nb of movies : 15175
percentage of movies : 96 %	percentage of movies : 74 %	percentage of movies : 57 %

### **Movie Pattern Discovery:**

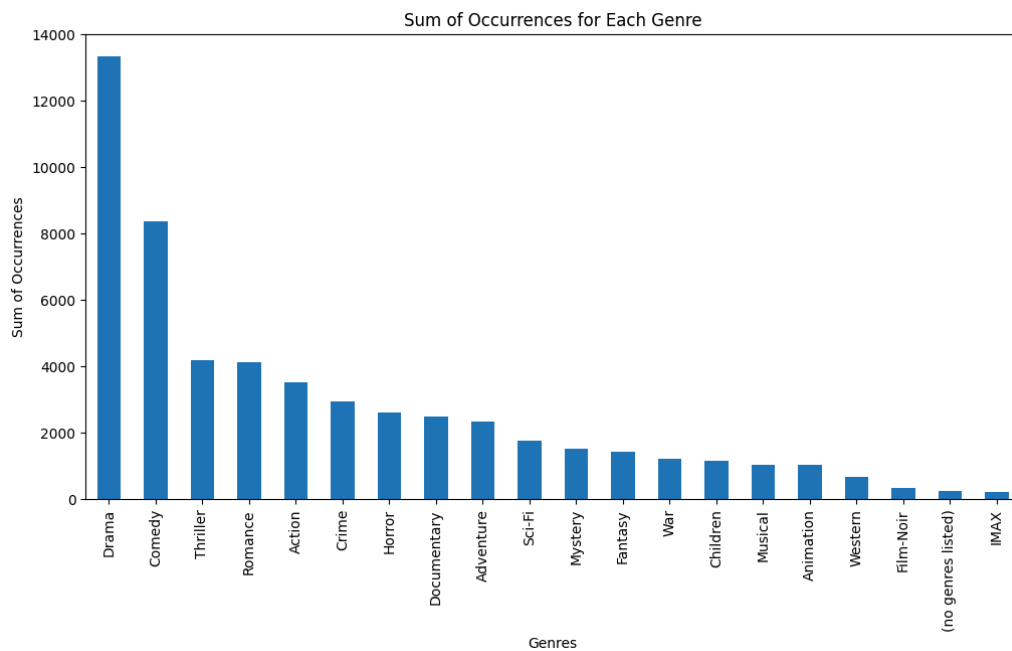
We separate movies in 2 main groups, Blockbusters and Confidential.

Most movie have a very low number of ratings.



### Genres Pattern Discovery:

Drama and Comedy are the two main genres, while other genres are more niche.



## 3. Data Preprocessing and Feature Engineering

- **Data Cleaning:**

The Data can be considered clean, there were very few NaN and 96% of all movies have ratings. We only had to drop a few rows with missing values.

- **Feature Selection, Data Balancing and Transformation:**

We have considered that not only the rating was important, other factors like age and popularity could be useful. We have created new features to implement that.

First the year of release of the movies is turned into an age, and this age is turned into a “youth rate”, a positive number between 0 and 1. The youngest movie gets 1 and all the movies released before 1965 gets 0. This was a choice, the movie list goes back to 1891, but we consider movies before the sixties to be appreciated by the very few.

Secondly we have counted the occurrences of ratings per user and movie. Certain ratings are overrepresented in both ways. This means that a very small fraction of the movies are very popular( 5%), and a smaller fraction of the users are very engaged (3%). A balance of the data to avoid biases needed to be operated.

We also introduced a "popularity rate" based on rating frequency to balance our dataset and improve our machine learning results. However, since this approach was not effective, we opted to apply filters instead, which yielded better results.

Since we have been working remotely, we have created a tool to filter the data frames rating and movie so we don't have to transfer heavy csv, with just 4 parameters, it's a notebook called: “Dataframe\_Filter.ipynb” that you can test yourself.

## 4. Model Building and Evaluation

- **Technology, Libraries, Models:**

Python, Pandas, Numpy, Re, Seaborn, Plotly, Scikit Learn (SVD), LightFM, Scipy,

- **Model 1: Matrix Factorization with Pearson similarity**

This first System is called: [Because you watch this movie](#)

This recommendation system is designed to suggest movies based on a user's current selection, employing a hybrid approach that combines collaborative filtering with content-based filtering to recommend a list of ten movies.

The collaborative filtering component uses matrix factorization, creating a DataFrame where each row represents a user and each column represents a movie, with ratings in the corresponding cells. A target movie is identified, and Pearson correlation is calculated with other movies to assess similarity. On the content-based side, recommendations are refined using a separate DataFrame that includes movie titles and dummy-encoded genre columns. This hybrid approach merges Pearson similarity scores from user preferences and movie genres into a combined table, which is then sorted in descending order to extract the top ten recommendations.

Its default is to be a little slow and needs to be runned while the user is watching a movie.

Although this method is straightforward and lacks advanced machine learning techniques like train/test splits or loss metrics calculation, it provides a solid foundation for understanding the data

and the complexities involved in building recommendation systems. The results, while broad, offer a reasonable starting point for generating movie recommendations.

- **Model 2: Singular Value Decomposition (SVD)**

**SVD Introduction:**

After reviewing various machine learning models, SVD (Singular Value Decomposition) stood out as the one offering the best results. During our research, we also tested well-known models without any prior optimization, under the same conditions, to get an initial evaluation of their effectiveness. The results are shown below:

ML/Metric	MAE	RMSE
Mf	0.6488	0.8142
PMF	0.8207	0.9796
NMF	0.8620	1.1044
SVD	0.6481	0.8136

These preliminary tests highlighted the significant potential of the SVD model, consistent with the literature. Based on our goals and various sources, including a representative study [1].

**Table 1.** Comparison of RMSE results.

Dataset	RMSE			
	SVD	RSVD	SVD++	BLS-SVD++
MovieLens 1 M	0.891	0.874	0.831	0.763
MovieLens 10 M	0.864	0.845	0.822	0.734
FilmTrust	0.903	0.887	0.853	0.826

**Table 2.** Comparison of MAE results.

Dataset	MAE			
	SVD	RSVD	SVD++	BLS-SVD++
MovieLens 1 M	0.736	0.724	0.706	0.675
MovieLens 10 M	0.694	0.673	0.651	0.585
FilmTrust	0.790	0.758	0.725	0.707

We aim to achieve an RMSE of 0.734 and an MAE of 0.585.

**SVD Overview:**

SVD, or Singular Value Decomposition, is a matrix factorization technique that reduces the dimensionality of a matrix while preserving its essential features. Here's how it works, without diving into the math:

1. **Matrix Decomposition:** SVD takes a matrix (e.g., a rating matrix where rows represent users and columns represent movies) and breaks it down into three matrices: one for users, one diagonal matrix with the importance of components, and one for movies.
2. **Dimensionality Reduction:** By keeping only the main components (those with the largest singular values), we reduce the complexity of the original matrix while maintaining significant relationships.

### Why SVD is Useful for Movie Recommendations:

SVD is particularly effective for movie recommendations because:

1. **Latent Relationship Discovery:** SVD uncovers hidden factors, like movie preferences, that users share but aren't explicitly defined.
2. **Noise Reduction:** By reducing the matrix's dimensionality, SVD filters out noise or anomalies in the data, leading to more accurate recommendations.
3. **Matrix Completion:** In recommendation systems, the user-movie matrix is often incomplete (not all users have rated all movies). SVD helps estimate missing ratings by leveraging latent relationships.
4. **Personalized Predictions:** Using SVD, we can predict what rating a user might give to a movie they haven't seen yet, based on discovered similarities in latent factors.

### Applying SVD:

Initially, we used the DataFrame obtained after our first data processing steps, containing columns like `movieId`, `userId`, `rating`, `movie_youth_rate`, and `popularity_rate`.

Given the large size of the dataset, it's resource-intensive and may produce biased results due to the presence of very distinct cases (as discussed in data processing). We filtered based on the minimum number of ratings per user and per movie:

python

Copier le code

```
min_user_ratings = 124
```

```
min_movie_ratings = 1000
```

This decision was made after studying the data. Once the filter was applied, we created a pivot table with `userId` as rows, `movieId` as columns, and `rating` as values.

We tested several scenarios and compared the results:



1. Using the raw DataFrame
2. Weighting ratings based on movie age
3. Weighting ratings based on movie popularity
4. Weighting ratings based on both movie age and popularity

For each test, the pipeline was as follows:

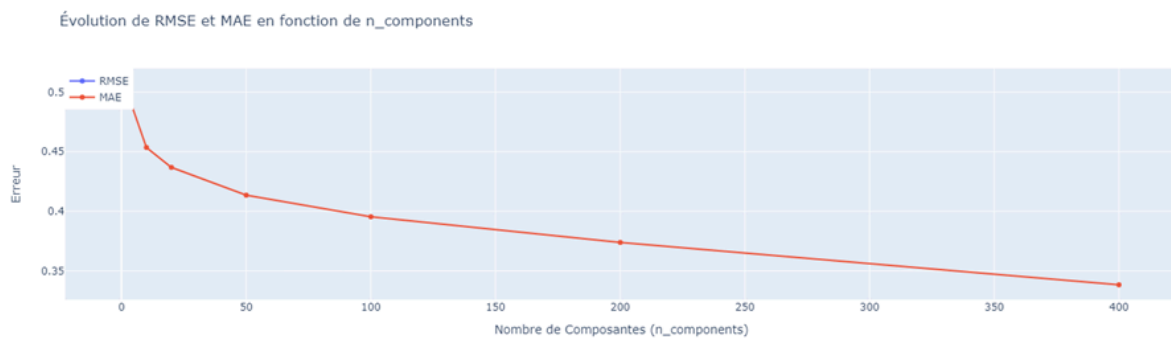
1. Data normalization
2. Grid search to determine the SVD sub-dimensionality (**n\_component**)
3. Observation of cumulative explained variance and metrics like RMSE and MAE

We found that weighting had no visible effect, which was surprising. After investigation, we discovered this was due to normalization done after applying the weights, which negated their influence.

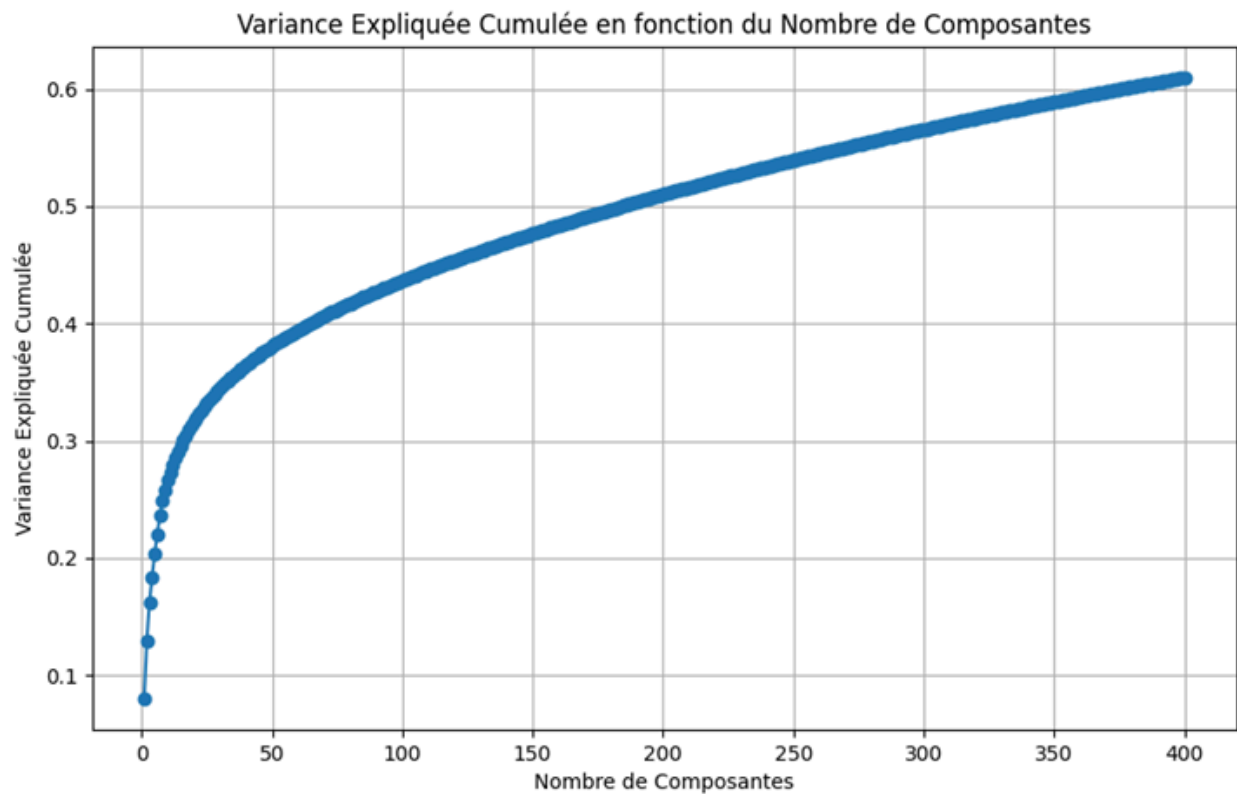
We repeated the tests with the following plan:

1. Using the raw DataFrame with standardization
2. Using the raw DataFrame without standardization
3. Weighting based on movie age without standardization
4. Weighting based on movie popularity without standardization
5. Weighting based on both age and popularity without standardization

The results improved but became excessively good, with MAE of 0.018 and RMSE of 0.024, indicating overfitting. A deeper study of the grid search was then conducted.



We clearly observed overfitting as the number of sub-dimensions increased, with less data in each.



By examining explained variances and metric evolution, we chose an `n_component` parameter that balanced underfitting and overfitting while providing reasonable metrics.

Test/Metric	RMSE	MAE
<b>Sans Pond. &amp; Avec Norm.</b>	0.9301	0.4516
<b>Sans Pond. &amp; Sans Norm.</b>	0.7944	0.4149
<b>Pond. Age &amp; Sans Norm.</b>	0.6438	0.3342
<b>Pond. Pop &amp; Sans Norm.</b>	0.0281	0.0184
<b>Pond. Pop/Age &amp; Sans Norm.</b>	0.0354	0.0268

We still noticed some inconsistencies, similar to overfitting, even though the problem was supposed to be resolved. We explored another approach: the system was recommending "classics," suggesting that ratings were heavily influenced by movie popularity.

We then tested an inverse weighting approach (dividing ratings by popularity instead of multiplying). The results confirmed that the information was redundant. However, the scores remained

suspiciously high, so we compared the top 10 movies seen by the user with the top 10 recommended movies.

Test/Metric	RMSE	MAE
Sans Pond. & Avec Norm.	0.9301	0.4516
Sans Pond. & Sans Norm.	0.7944	0.4149
Pond. Age & Sans Norm.	0.6438	0.3342
Pond. Pop & Sans Norm.	<b>59.2180</b>	<b>18.3648</b>
Pond. Pop/Age & Sans Norm.	<b>46.4349</b>	<b>14.3595</b>

After comparing the results from our function outputs, we notice two points:

1. The recommended movies are often blockbusters.
2. It often recommends movies that users have already seen.

We have therefore decided to apply filters to users to include only those within the interquartile range (IQR). To achieve this, we used a new library, Cornac, which supports SVD and is significantly faster for processing large volumes of data.

We obtained the following initial results with randomly chosen parameters:

```
svd = SVD(k=50, max_iter=10, learning_rate=0.001, lambda_reg=0.01)
```

With these parameters, we get the following results:

```
| MAE | RMSE | Precision@10 | Recall@10 | Train (s) | Test (s)
```

```
----- + ----- + ----- + ----- + ----- + ----- + -----
```

```
SVD | 0.5829 | 0.7416 | 0.0951 | 0.0200 | 4.7163 | 71.1660
```

With this filtered dataset, we observe that we are very close to the objectives mentioned in the literature, with an RMSE of 0.734 and an MAE of 0.585.

The goal now is to use hyperparameter tuning to optimize recall and precision, which are relatively low.

After the several kind of optimization we have these parameters.

```
svd = SVD(k=300, max_iter=23, learning_rate=0.0026, lambda_reg=0.008)
```

And we have these results :

```
| MAE | RMSE | Precision@10 | Recall@10 | Train (s) | Test (s)
```

```
--- + ----- + ----- + ----- + ----- + ----- + -----
```

```
SVD | 0.5814 | 0.7414 | 0.1504 | 0.0313 | 27.2483 | 90.3920
```

Thanks to the iterative loop on the 4 hyperparameters of the SVD model, we were able to improve the RMSE, MAE, recall, and precision scores. However, the recall and precision results are too low to have confidence in this model. Given the time constraints, we will keep these values to test the function on several different users. In the next step, we will test another model (LightFM) to improve the precision and recall scores. Additionally, we will compare the results of these two models and conduct a benchmark to evaluate their efficiency.

Below is an example with the user as an example:

Top 10 films vus par l'utilisateur 1 :

	title	genres	rating
0	Freaks	Crime Drama Horror	5.0
1	Lord of the Rings: The Fellowship of the Ring,...	Adventure Fantasy	5.0
2	Lord of the Rings: The Two Towers, The	Adventure Fantasy	5.0
3	Lord of the Rings: The Return of the King, The	Action Adventure Drama Fantasy	5.0
4	Star Wars: Episode V - The Empire Strikes Back	Action Adventure Sci-Fi	4.5
5	Raiders of the Lost Ark (Indiana Jones and the...	Action Adventure	4.5
6	Spider-Man 2	Action Adventure Sci-Fi IMAX	4.5
7	Constantine	Action Fantasy Horror Thriller	4.0
8	Watership Down	Adventure Animation Children Drama Fantasy	4.0
9	Dragonslayer	Action Adventure Fantasy	4.0

Les recommandations faites sont les suivantes :

	title	genres	score
1	Jaws	Action Horror	5.280891
7	Snatch	Comedy Crime Thriller	5.241471
4	Pi	Drama Sci-Fi Thriller	5.185878
8	Before the Devil Knows You're Dead	Crime Drama Thriller	5.160434
5	Dirty Dozen, The	Action Drama War	5.089878
2	Truman Show, The	Comedy Drama Sci-Fi	4.994098
6	Being John Malkovich	Comedy Drama Fantasy	4.937565
3	Primary Colors	Comedy Drama	4.933546
9	Chain Reaction	Horror	4.904399
0	Wizard of Oz, The	Adventure Children Fantasy Musical	4.873779

By studying the movies and their genres, we can observe that the recommendations are quite consistent. However, we notice predictions with values higher than the maximum. This code still needs further optimization.

But we will try another approach using the LightFM model.

---

[1] SVD++ Recommendation Algorithm Based on Backtracking by Shijie Wang, Guiling Sun, and Yangyang Li, 21 July 2020.

[2] Behavior's Study of some Classic SVD-Models with Noisy Data in Movie Recommender Systems Cupertino Lucero-Alvarez ´ 1,2, Perfecto Malaquías Quintero-Flores2 , Edgar Moyotl-Hernandez ´ 3 , Carlos Artemio Ortiz-Ramírez1 , Patricia Mendoza-Crisostomo ´ 1 , Maria Vazquez-Vazquez ´ 1, 01 september 2023.

- **Model 3: A hybrid latent representation recommender model: Light FM**

### **Introduction:**

The model learns embeddings (latent representations in a high-dimensional space) for users and items in a way that encodes user preferences over items. When multiplied together, these representations produce scores for every item for a given user; items scored highly are more likely to be interesting to the user.

The user and item representations are expressed in terms of representations of their features: an embedding is estimated for every feature, and these features are then summed together to arrive at representations for users and items. For example, if the movie 'Wizard of Oz' is described by the following features: 'musical fantasy', 'Judy Garland', and 'Wizard of Oz', then its embedding will be given by taking the features' embeddings and adding them together. The same applies to user features.

The embeddings are learned through stochastic gradient descent methods.

### **Implementation:**

We used the ratings  $\geq 4$  as a positive interaction and created a coo matrix from this interaction matrix.

The model works with 4 loss functions, after several tests, we decided to use the warp function:

WARP: Weighted Approximate-Rank Pairwise [2](#) loss. Maximizes the rank of positive examples by repeatedly sampling negative examples until rank violating one is found. Useful when only positive interactions are present and optimizing the top of the recommendation list (precision@k) is desired.

Tests also led us to reduce the data used in the training set as we suspected the model was overfitting due to a huge imbalance between the train and test results.

This allowed us to get our test\_precision@10 results from 0.1 to 0.4

We also reduced the data imbalance by eliminating users that had huge number of ratings in contrast of other users

The cold/warm start case:

LightFM was designed to also address the problem of cold/warm starts by incorporating features for both users and movies:

- **Users:** We calculated the distribution of their viewings across movie genres for each user and integrated that as a feature.
- **Movies:** We used one hot encoded genre for each movie using and integrated those as a feature matrix. Although we tried vectorizing the genres using word2vec and tf-idf that did not yield any significant results

This allows us to provide somewhat closer recommendations to users with sparse/non existent interaction data

- **Cold Start Scenario:** We propose as a solution to the cold start scenario to integrate a page to select a few like genres for new users that will be added to their user profiles to enable recommendations with no interaction data by the LightFM model


### Cold Start Helper


welcome!


To get started, tell MovieLens about your preferences by distributing 3 points among your favorite groups of movies below.

Next ➔


chick flick, feel-good, touching


Titanic


Dead Poets Society

Slumdog Millionaire


classic, masterpiece, quotable


The Godfather


Psycho

The Graduate


dramatic, good acting, intense


Forrest Gump


Million Dollar Baby

The Social Network


action, fun movie, special effects


True Lies


The Mask

Men in Black II


computer animation, good versus evil, mythology


Toy Story


The Lord of the Rings: The Fellowship of the Ring

Harry Potter and the Philosopher Stone

blood, dark humor, social commentary

Pulp Fiction

Kill Bill: Vol. 1

American History X

- **Prediction**

The model provides a ranking score for user/movie pairs using the dot product of the learnt embeddings. The higher the score the higher the rank. Results are sorted and a chosen number of recommendations are shown.

- **Evaluation:**

We essentially used precision as the main metric for evaluation as it represents the main goal of our project. Deliver k relevant recommendations to the user. Measure the precision at k metric for a model: the fraction of known positives in the first k positions of the ranked list of results. A perfect score is 1.0.

- We enhanced the precision@k through the various proceedings mentioned in the implementation from a starting 0.1 to 0.69
- We also conducted cross evaluation and varied the training part (as it is less than the test) through 4 folds, obtaining similar results meaning the model generalizes well .
- We also used recall and roc-auc.

Roc-auc :Measure the ROC AUC metric for a model: the probability that a randomly chosen positive example has a higher score than a randomly chosen negative example. A perfect score is 1.0.

We obtained a score of 0.96 on the test dataset meaning our model ranks well.

- Recall:the number of positive items in the first k positions of the ranked list of results divided by the number of positive items in the test period. A perfect score is 1.0.
- This was low as the k number (chosen 10) is really low compared to the number of available positive items in the set.
- **Model Comparison:** LightFM offered overall better performance on the various cases than the other models
- **Model Interpretability:** Although the predicted scores from LightFM are hard to interpret, we could work with the user and movie embeddings to analyze patterns and clusters and understand the recommendations.

## 5. Conclusion

- **Summary of Key Findings**

Recommending items is a multifaceted task that requires various approaches depending on the context. No single, highly powerful algorithm can address all scenarios. Instead, multiple models are needed to cater to different use cases. Data filtering is crucial, and understanding the audience is essential to effectively target recommendations.

- **Practical Implications**

Improving precision reduces the likelihood of recommending movies the user won't enjoy, while enhancing recall increases the chance of suggesting movies the user might discover and appreciate.

In our view, precision is more vital in recommendation systems because it minimizes the risk of presenting the wrong movies. The users are going to judge the quality of the streaming platform based on what they see, not based on hidden possibilities that could have been. It is a major issue for society since it wraps users in a filter bubble, but the streaming platform is a capitalistic company, its primary purpose is to generate revenue, not “educate” people.

- **Limitations and Future Work**

We were unable to incorporate actors' and directors' names into our recommendation system, despite their potential to significantly enhance the model.

Additionally, the features we created, such as youth and popularity rate, provided limited improvement. Instead, filtering the DataFrame proved to be a far more effective approach.

Finally, embedding tags, titles, and genres did not yield the desired results, as the vast amount of data generated by these vectors was too challenging to process. This raised the question: "Is it worth it?"

Could a simpler and more cost-effective approach produce results that are reasonably comparable?

## 6. Appendices

- **Source Code:** [https://github.com/jcrigoni/grand\\_ml\\_project](https://github.com/jcrigoni/grand_ml_project)
- **Data** <https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset>