Movie Recommender System

Frederico Cardoso[1] , **Tomás Ferreira**[2]
[1,2]University of Coimbra
[1]fnc@student.dei.uc.pt , [2]tomasferreira@student.dei.uc.pt

Abstract

With the last decade's emergence of streaming services and e-commerce, especially during and after the Covid-19 pandemic [Guthrie *et al.*, 2021], people are consuming more virtual content than ever before. Services like these, now, hold a powerful tool in regard of the consumer data and preferences. With this type of information, these types of services are able to make a great usage of it in order to suggest to their users other content or products that they might like. To further research these topics, a movie recommender system was developed to evaluate its performance and discuss the obtained results.

1 Introduction

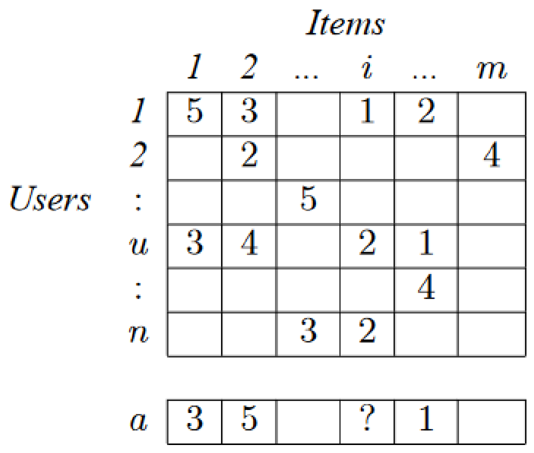
A **Recommender System** is a Machine Learning algorithm that analyses user's data in order to achieve recommendations to an . With that in mind, the aim of this project is to develop a program to recommend movies to an user, based on its previously watched movies and genre preferences. To build a reliable **Explainable Recommender System**, with a **Human-in-the-Loop** approach, and having information about users and their personal ratings of movies they watched, we will apply different types of information filtering to study how can these rating be related to a particular user. Overall, this system should be able to recommend a movie to an user, based on how much he liked (rated) the movies he watched, the similarity between other movies of the same genre, and a mix of both.

2 Background

Recommender systems have been around since the beginning of the web and their evolution accompanied the evolution of the internet [Bobadilla *et al.*, 2013]. The first generation of these systems were based on the demographic information about the users and their preferences. It is expected that, in the future, they also take in account implicit, local and personal information from the **Internet of Things**. For this project, we will only focus on building and evaluating a system that takes in account relevant information about a user's preferences and the content itself. Our aim was to build a modern system, by today's standards, that is reliable and robust, mixing the various properties of the provided information from the dataset and understand the obtained results.

2.1 Recommendation task

For the vast majority of recommender systems, the first and base task is to create the User Rating Matrix (URM) (Fig. 1). This structure is built by taking all the users and items and creating a squared matrix where each individual cell $r_{u,i}$ corresponds to the rating of the user *u* to the item *i*.



Items

Users	*1*	*2*	...	*i*	...	*m*
1	5	3		1	2	
2		2				4
:			5			
u	3	4		2	1	
:					4	
n			3	2		
a	3	5		?	1	

Source: [Melville and Sindhwani, 2010]

Figure 1: The User Rating Matrix.

This matrix is, usually, very sparse, since most users do not rate most items. The **recommendation task** in this project is to predict the missing rating $r_{u,m}$ of a movie *m* for the active user *u*, using methods of filtering the relevant information and extracting features of the users and movies from the selected cells.

2.2 Feature extraction and filtering

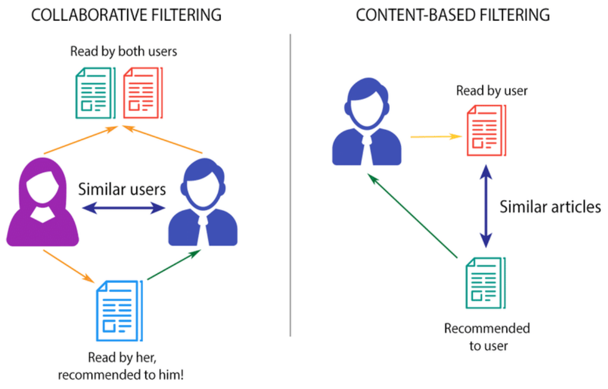
Some former works [B. Thorat *et al.*, 2015][Li *et al.*, 2005], that explored this resulting URM and the task of recommendation, have distinguished these techniques into four classes:

Demographic filtering
The recommendation is established on a demographic profile of the user, based, therefore, on personal information provided by the user, like its age, gender, nationality, etc.

Collaborative filtering
The recommendation is received by comparing the preferences of other users who have rated items in a similar way as the active user. The similarities considered are between users (Fig. 2).

Content-based filtering
The recommendation depends on the user former choices, being selected the ones that have more similarities with the ones that the user enjoyed more. It takes in account the similarities between the movies (Fig. 2).

Hybrid filtering
The hybrid filtering method combine more than one of the above. It is used to overcome some problems that may occur with some filtering methods selected, like sparsity or scalability. It may improve the accuracy and efficiency of the recommendation process.



Source: [Tondji, 2018]

Figure 2: Collaborative vs Content-based filtering.

3 Dataset

In this implementation, it was used the **MovieLens** [Harper and Konstan, 2015] dataset, contained in the TensorFlow [Abadi *et al.*, 2016] project, which is a Python library for developing and analysing Machine Learning (ML) models. This dataset in particular is very often used for recommender systems, as we could see in some relevant articles.

The MovieLens dataset has 5 current versions included:

- **25m:** the largest and latest stable version, with no demographic data, recommended for research purposes;
- **latest-small:** a subset of the latest version;
- **100k:** the oldest version of the dataset, that contains demographic data;
- **1m:** the largest dataset that contains demographic data;
- **20m:** the subset most used dataset in academic papers.

Movie ID	Title	Genres
1	Toy Story (1995)	Adventure ... Fantasy
2	Jumanji (1995)	Adventure ... Fantasy
...	...	...
62422	A Girl Thing (2001)	(no genres listed)

Table 1: Movies dataset.

User ID	Movie ID	Rating	Timestamp
1	296	5	1147880044
1	306	3.5	1147868817
...	...	...	...
162541	63876	5.0	1568666558

Table 2: Ratings dataset.

For the scope of this project, there is no interest in creating recommendations based on any demographic information. It were used the movies (Table 1) and ratings (Table 2) information from the various sub-datasets included in the version **25-m** of MovieLens. This selection of information contains about 25 million ratings and 62 thousand movies.

The relevant features selected were the following:

- **Movie ID:** the unique identifier of a movie;
- **Movie title:** the title (and year) of a movie *m*;
- **Movie genres:** all the genres of a movie *m*;
- **User ID:** the unique identifier of an user;
- **User rating:** the rating that an user *u* gave to a movie *m*.

With these features, the data-processing phase can begin and all the subsequent filtering methods can be applied.

4 Methodology

4.1 Data pre-processing

The first thing to do is to pre-process the information extracted to be used. With these features the URM can be created, to be further subjected to the 3 different types of filtering that don't rely on demographic data.

Discarding redundant values
The MovieLens dataset is actually quite clean. There were very few entries with repeated ID's or titles, and also can be said about empty values. Nevertheless, we removed those few entries from the original dataset, before proceeding to other operations.

Dimensionality reduction
As the 25 million entries would be unfeasible to use, and for performing a later analysis on if the data size can provide better and more robust results, we considered a reduction to the first 5 hundred, 5 thousand and 50 thousand items. This reduction to, at most, 0.2% of the original dataset is essential, as the algorithms often work with $n \times n$ matrixes, which limits the storage and processing power of the average computer [Altman and Krzywinski, 2018].

	Movie 1	Movie 2	...	Movie *m*
User 1	$r_{1,1}$	$r_{1,2}$	...	$r_{1,m}$
User 2	$r_{2,1}$	$r_{2,2}$	...	$r_{2,m}$
...	...	...	...	...
User *u*	$r_{u,1}$	$r_{u,2}$	...	$r_{u,m}$

Table 3: Structure of the URM.

	Movie 1	Movie 2	Movie 3	Movie 4
User 1	4	5	?	?
User 2	?	?	?	1
User 3	?	5	5	5

Table 4: Example of an incomplete URM.

4.2 User similarity approach

The user similarity approach, also called **Collaborative Filtering**, it's a technique that filters items that a user might like, considering other users reaction to that item, *i.e.,* it is based on the assumption that people like things similar to things that other people, with similar taste, like [Grover, 2020].

In this case, since we already have the users and movies information, we start by building the URM matrix (Table 3). This matrix will, as explained above, have a lot of missing values, but it contains valuable information to predict them.

Singular Value Decomposition

The **SVD** is a linear algebra method of factorization of a real or complex matrix. Mathematically, the singular value decomposition of a matrix R is a factorization of the form

$$R = U\Sigma M^T$$

In this project, as the scope is not to understand fully the algebra behind it, let us simply consider that:

- R is a matrix $m \times n$, which is our URM with missing values (Table 5);
- U is a rectangular matrix $m \times r$ that considers the most relevant features for all users;
- Σ is a diagonal matrix $r \times r$ that has some values that help the multiplication between U and M^T.
- M^T is a transposed rectangular matrix $r \times n$ that considers the most relevant features for all movies;

In modern ML algorithms, the SVD is applied to predict the missing values of the URM matrix (represented by R), based on the values that already exist. This prediction is more accurate the more information and training is given to the model [Mavuduru, 2020]. For implementing this, we used the Surprise Scikit [Hug, 2020] that already has this matrix factorization method implemented.

However, two of the main problems of recommender systems are data ***scalability*** and ***sparsity***. SVD methods have been successful in dealing with scalability, but they have not efficiently solved the problem of sparsity [Nguyen, 2016]. This sparsity, very present in the example of this dataset, often leads to lower accuracy in prediction in these systems.

Keras Deep Learning Library

To try to tackle the problems described above, we implemented a normal neural network that applies methods of collaborative filtering to later analyse and compare the results.

We used the Interface for the Keras Deep Learning Library [Arnold, 2017], which is an **API** of neural-network models, that also have some interesting functions and plots. With some methods and models from Keras, we divided the dataset in training and testing subdatasets, to see what predictions of the missing values from the URM were suggested.

For example, with Keras, we can very easily understand, what was the **model loss** (the indication of how badly the model's prediction was on a simple example) over epochs (Fig. 3). Being a neural network, it is a much more complex model than the SVD one, but requires much more time processing power. Also, we can have a axial representation of the position of the movies, using the **T-distributed Stochastic Neighbor Embedding** (TSNE). With this method, and with a bit more of exploring, we could have analysed the clusters and categorized these movies by genres, changing their respective colour in the graph (Fig. 4).

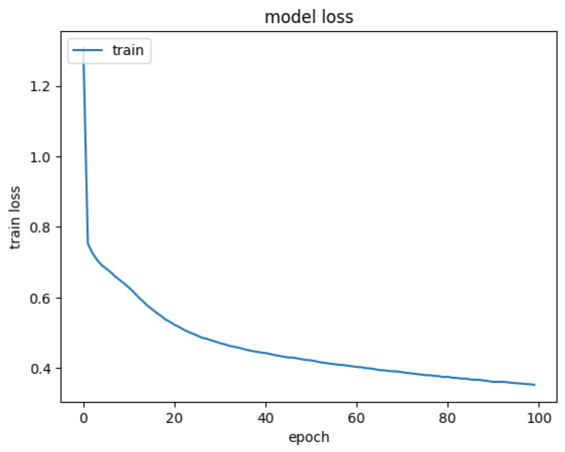


Figure 3: Model loss.

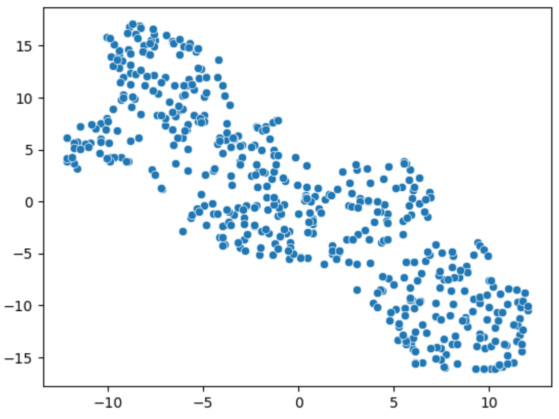
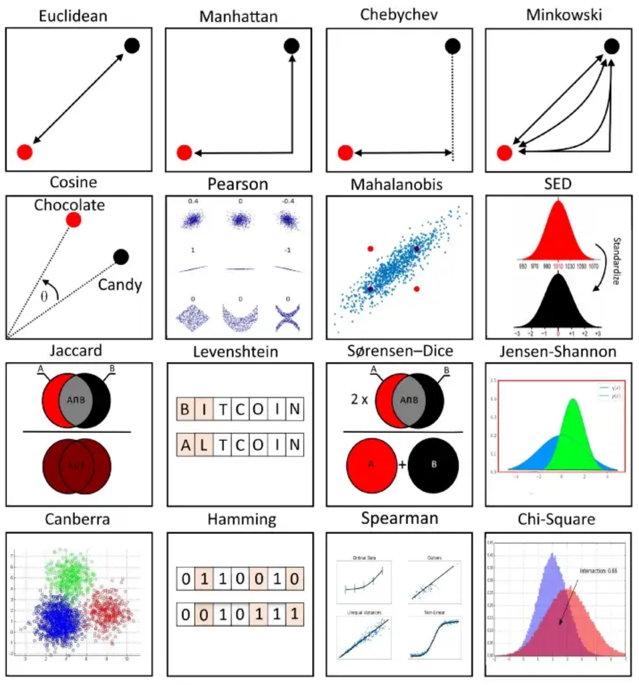


Figure 4: Initial phase of clustering the first 500 entries.

4.3 Item similarity approach

Many data mining applications, like clustering, outlier detection and classification, require the determination of similar and dissimilar objects, patterns, features, etc. in the data. All those applications are subjected to a methodical way of quantifying how these samples are related to each other or how distinct they are (Fig. 5)[Charfi *et al.*, 2020].



Source: [Harmouch, 2021]

Figure 5: Various ML similarity metrics.

These techniques may be applied to relevant features of the information about the movies. For the scope of this project and of the content-based approach, it will be considered the **cosine similarity** [B. Thorat *et al.*, 2015] between the genres.

Cosine similarity

The similarity between two items is given by the angle θ formed between them. As it can be seen in the Figure 5, the lesser the angle, the more similar the items are, at least in that dimension [Singh *et al.*, 2020], *i.e.,* if the value of θ is equal to 0, the items are identical. This similarity is given by:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|_2 \, \|B\|_2}$$

We applied the cosine similarity methods present in the Sklearn library [Pedregosa *et al.*, 2011] to our movies data. We only considered one distinguishing factor, as we wanted our user, after rating a movie, to be recommended the top movies of that genre, not by year or title similarity.

After creating the cosine similarity matrix, we simply sort all the movies in a descending order by value of similarity (tipically, the movies with the exact same genres) and return the top n to the user.

	Movie 1	Movie 2	Movie 3	Movie 4
User 1	4	5	?	?
User 2	?	?	?	1
User 3	?	5	5	5

Table 5: Example of an incomplete URM.

4.4 Hybrid approach

The hybrid approach appears when there is not enough data to relate the users to the items. In these cases, this third method may be applied to try to eliminate these deficiencies. There are various types, in which all take account the predictions obtained in the two types of filtering [chiang, 2021], like:

- **Weighted:** the predictions for each model are given a score (weight) and the result is their sum;
- **Switching:** evaluates and choses one of the predictions;
- **Mixed:** generates different of candidate datasetsand then applies a score system to decide which one outputs.

We applied to the collaborative and content-based systems implemented a weighted score system. This system simply sums, for all the movies, the indexes on which that movie recommendation appears to the user, on both filtering types, and the result is then sorted by these new values (Table **??**).

5 Evaluation metrics

5.1 Mean Squared Error

The results that derive from the models are not exactly the same of the user ratings. To measure what is the difference from a true rating of an item and the prediction, it is used the **squared error**, *i.e.,* the squared distance from those two values. To validate a model, we consider the mean of all those errors, which is given by:

$$MSE = \sum_{i=1}^{D}(x_i - y_i)^2$$

The Mean Squared Error, on par with the **Mean Absolute Error** and **Root Mean Squared Error**, which is the root of the formula above, are techinques of quantifying how much distant is the predicted results from the original (correct) rating. These metrics are used almost in all recommender systems [Salam Patrous and Najafi, 2016].

5.2 Mean Reciprocal Rank

Another method was applied to validate the models. Instead of considering the *error* between the predictions and actual values, we take in account the sorted list of an user's movies' ratings and the movies' predictions to that user and compare it by indexes, with:

$$MRR = \frac{1}{\|Q\|} \sum_{i=1}^{\|Q\|} \frac{1}{rank_i}$$

This is a statistical way to evaluate by position of the prediction instead of by value. In some cases, it can be a more

	500	5.000	50.000
SVD 5	0.5935	0.5136	0.50212
SVD 50	0.5047	0.4668	0.43227
Keras 10	1.8374	1.3696	1.0942
Keras 100	1.3776	1.3689	1.0828

Table 6: MSE results for different methods and dataset size.

”realistic” approach, as even if those two lists coincide, the MSE will not return error of zero, but the MRR can return a perfect result, as it is oredered by probability of correctness.

6 Results

6.1 MSE comparison

As we can see in Table 6, the SVD, as expected, has better results, as it resolves the problem of sparsity better than the Keras model. For the Keras model, in this case, is a neural network that is being fed a lot of useless information. To be more efficient, a lot of dimensionality reductions should be applied, but it is not a simple task, as the relevant results are very sparse and can not be easily discarded.

We can also observe, and also as expected, that the number of entries of the dataset is fundamental to achieve better results, as it consolidates the predictions of these models. The same is true for the number of **cross validation** iterations in the SVD model and the number of epochs (the number of times the entire dataset is passed through) of the Keras model.

This results, naturally, to a exponentially increase in the processing power of the machine used for calculation. For example, for the Keras with 100 epochs and 50k entries of the dataset, the calculation of the MSE took about 5 minutes, for the 5k only one and a half and for 500 entries about 40 seconds.

With the Keras API, we can observe, graphically, the MSE straight line through the predicted per real results (Fig. 6).

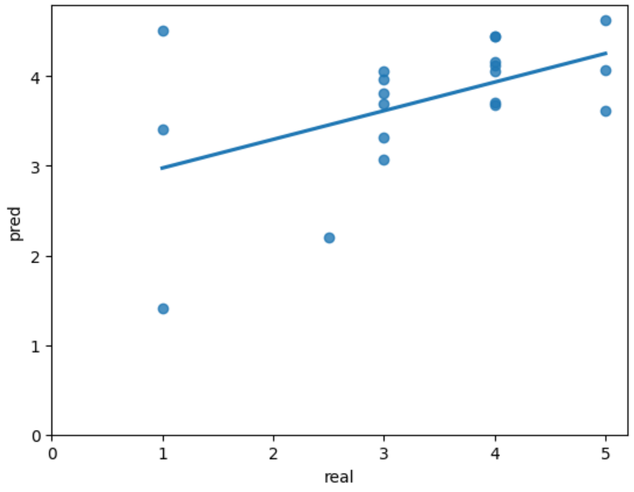


Figure 6: Keras model MSE straight line for the first 20 entries.

6.2 MRR and dataset analysis

As for the SVD model, for this example, we calculated the MRR to evaluate if the recommendations are well sorted.

$$MRR_{svd} \approx 0.382$$

We can conclude that it is not a very good result, means that the predictions are a little bit off in relation to where they should be. This is due to the dataset size and sparsity of items. For example, it is difficult to correctly recommend items to an user that only rated two or three movies and, on top of that, in the Right order. With more work on the selection of the training dataset, or the inclusion of more features, this value could go up and have more precise recommendations.

6.3 Overall comparison between models

We start by comparing these two images (Fig. 7)(Fig. 8), related to the predictions and real values of the two models.

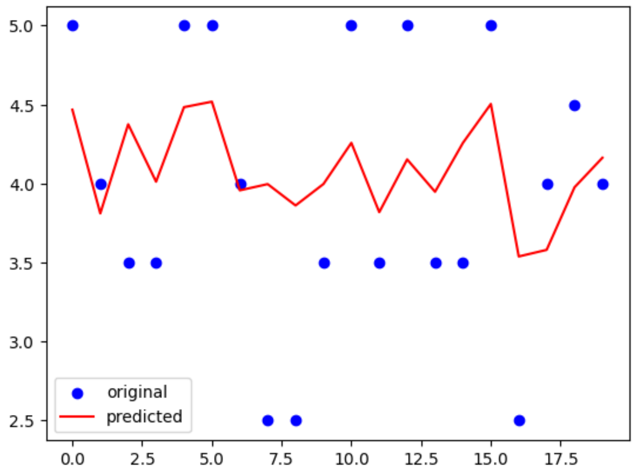


Figure 7: SVD predictions vs real values.

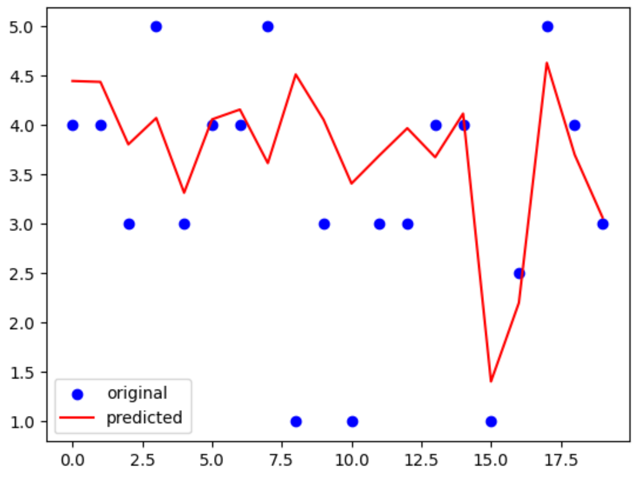
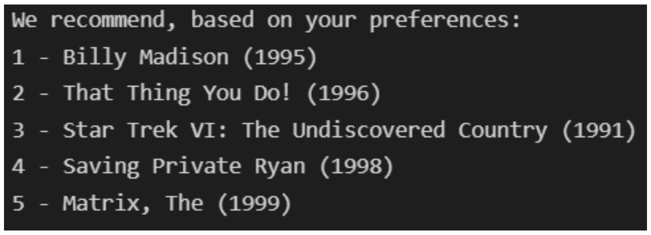


Figure 8: Keras model predictions vs real values.

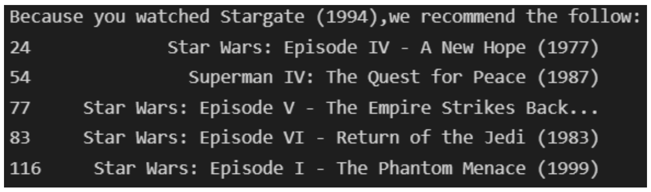
We can see that the SVD model is more consistent, in average, in its predictions, because is based on the resulting values of the decomposition of a very incomplete matrix, meaning that it has very little account of outliers. As for the Keras model, some of the predictions are very close to the real values, but others are way of. This was already proved and discussed by the MSE values in table 6.

Lastly, here are some examples of how the system works:



```
We recommend, based on your preferences:
1 - Billy Madison (1995)
2 - That Thing You Do! (1996)
3 - Star Trek VI: The Undiscovered Country (1991)
4 - Saving Private Ryan (1998)
5 - Matrix, The (1999)
```

Figure 9: SVD model recommendations.



```
Because you watched Stargate (1994),we recommend the follow:
24            Star Wars: Episode IV - A New Hope (1977)
54              Superman IV: The Quest for Peace (1987)
77      Star Wars: Episode V - The Empire Strikes Back...
83      Star Wars: Episode VI - Return of the Jedi (1983)
116      Star Wars: Episode I - The Phantom Menace (1999)
```

Figure 10: Cosine similarity model recommendations.

7 Conclusion

Recommender systems have been around for a long time but, nowadays, are being used a lot more. There are some techniques that are more fine-tuned than others, but the general consensus is that the quality of the dataset is the fundamental variable in evaluating the performance of a system.

On the topic of spatial and temporal complexity, we could observe that, as some models reliable on matrix operations, the amount of RAM space needed is often quadratic, which is a concern. As for time needed, the neural networks are being fed millions of entries (in our project, for a $m \times m$ matrix, considering $m = 50000$ there are 2.5 billion entries), which can be executed in our machines, but a full dataset would need a processing power of a supercomputer to be feasible.

We could conclude that, for a **Item-based filtering** approach, it would have to take a lot of features in account to achieve better results. We only worked with genres, which can work, for the scope of this project, but to consider, for example, demographic variables, title similarity, etc., it would require a lot more work and planning. Nevertheless, we could see good recommendations of top movies based on the genre alone. The **Collaborative filtering** method revealed much more efficient, therefore more discussed in this paper, because it could be considered a more "realistic" approach, in comparison to the previous model.

As said, the hybrid approach could solve some of these problems, as it takes in account both methods, and give a better overall recommendation.

References

[Abadi *et al.*, 2016] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning, 2016.

[Altman and Krzywinski, 2018] Naomi Altman and Martin Krzywinski. The curse (s) of dimensionality. *Nat Methods*, 15(6):399–400, 2018.

[Arnold, 2017] Taylor B Arnold. kerasr: R interface to the keras deep learning library. *J. Open Source Softw.*, 2(14):296, 2017.

[B. Thorat *et al.*, 2015] Poonam B. Thorat, R. Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110:31–36, 01 2015.

[Bobadilla *et al.*, 2013] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[Charfi *et al.*, 2020] Amal Charfi, Sonda Ammar Bouhamed, Éloi Bossé, Imene Khanfir Kallel, Wassim Bouchaala, Basel Solaiman, and Nabil Derbel. Possibilistic similarity measures for data science and machine learning applications. *IEEE Access*, 8:49198–49211, 2020.

[chiang, 2021] Jeffery chiang. 7 types of hybrid recommendation system, Jun 2021.

[Grover, 2020] Prince Grover. Various implementations of collaborative filtering, Mar 2020.

[Guthrie *et al.*, 2021] Cameron Guthrie, Samuel Fosso-Wamba, and Jean Brice Arnaud. Online consumer resilience during a pandemic: An exploratory study of e-commerce behavior before, during and after a covid-19 lockdown. *Journal of Retailing and Consumer Services*, 61:102570, 2021.

[Harmouch, 2021] Mahmoud Harmouch. 17 types of similarity and dissimilarity measures used in data science., Apr 2021.

[Harper and Konstan, 2015] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. 5(4), dec 2015.

[Hug, 2020] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[Li *et al.*, 2005] Yu Li, Liu Lu, and Li Xuefeng. A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in e-commerce. *Expert Systems with Applications*, 28(1):67–77, 2005.

[Mavuduru, 2020] Amol Mavuduru. How you can build simple recommender systems with surprise., Dec 2020.

[Melville and Sindhwani, 2010] Prem Melville and Vikas Sindhwani. Recommender systems. *Encyclopedia of machine learning*, 1:829–838, 2010.

[Nguyen, 2016] Anh Nguyen. Singular value decomposition in recommender systems. 2016.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Salam Patrous and Najafi, 2016] Ziad Salam Patrous and Safir Najafi. Evaluating prediction accuracy for collaborative filtering algorithms in recommender systems. 2016.

[Singh *et al.*, 2020] Ramni Harbir Singh, Sargam Maurya, Tanisha Tripathi, Tushar Narula, and Gaurav Srivastav. Movie recommendation system using cosine similarity and knn. *International Journal of Engineering and Advanced Technology*, 9(5):556–559, 2020.

[Tondji, 2018] Lionel Tondji. Web recommender system for job seeking and recruiting. 2018.