Interesting title here

Interesting subtitle here

Clezia Costanzo

A.Y. 2022/2023

To blah blah blah — Dedication goes here!

Abstract

With the great technological advancements of the last two decades, artificial intelligence became one of the most studied and utilized tools in the last few years. I personally decided to tackle on a relatively popular subject: recommendation systems applied to movies, one of my greatest passions. This master's degree thesis in Data Science aims to provide an example of an implementation of a recommendation system engine, using the famous open source dataset provided by GroupLens, the MovieLens dataset. The project for the thesis is that of combining two different approaches, namely: machine learning and deep learning. The first will be used to apply an hybrid user-item collaborative filtering algorithm, while the second aims to find similar movie posters using a convolutional neural network in order to train the dataset.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence. 14
CF	Collaborative Filtering. 28
CNN	Convolutional Neural Network. 29
CSV	Comma-Separated Values. 34
CV	Computer Vision. 30
EDA	Exploratory Data Analysis. 34
IMDb	Internet Movie Database. 39
MAE	Mean Absolute Error. 32
ML	Machine Learning. 19
RMSD	Root Mean Square Deviation. 32
RMSE	Root Mean Square Error. 15, 32
RS	Recommendation system. 18
SVD	Singular Value Decomposition. 16
TF-IDF	Term Frequency - Inverse Document Frequency. 41
TMDB	The Movies DataBase. 34
WR	Weighted Rating. 39

1 Introduction

A recommendation system (or recommender system) is a class of machine learning that uses data to help predict, narrow down, and find what people are looking for among an exponentially growing number of options.

It's an artificial intelligence or AI algorithm, usually associated with machine learning, that utilizes big data to suggest a given product to consumers. These recommendations can be based on various criteria, namely past purchases, search history, demographic information, and other factors.
These systems are trained to understand the preferences, previous decisions, and characteristics of people and products using data gathered about their interactions. These include impressions, clicks, likes, and purchases. [1]
Nowadays, these systems are so commonplace that we don't even realize when we're using them. They play a huge role in the way we consume content: they improve our experience by filtering it for us and expose us to new items that we wouldn't otherwise discover. Some famous examples of recommendation engines include product recommendations on Amazon, Netflix suggesting movies and TV shows in each user's feed, recommended videos on YouTube, music on Spotify, the Facebook newsfeed and Google Ads.

Given the exponential growth of the amount of content which can be found online and the increasing amount of users who have access to online services and platforms, people felt that there was a need to filter information that might appeal to users to deal with the problem of information overload.
These system are, in fact, beneficial to both the users and the service providers, in the sense that they reduce transaction costs of finding and selecting an item that a potential consumer might like and users benefit from the fact that they have a selection of products tailored to their liking.

Recommender systems were introduced as a means of assisting and augmenting the social process of using the recommendations of others to make choices where there is no sufficient personal knowledge or experience of the alternatives. [2]
Again, they handle the problem of information overload that users normally encounter when presented with a huge amount of possible products to choose from. Recently, multiple approaches for building recommendation systems have been developed, in the data mining field, and artificial intelligence, too. The sheer amount of available information and the number of users on popular platforms pose some difficulties for recommendation systems. The main challenges are the following: producing good recommendations for users, and performing well in terms of computation and coverage given the sparsity of the data. In traditional statistical methodologies, the amount of work and computational time increases with the number of users in the system. Though recent development in the computing field made it possible to decrease significantly the amount of time it takes to fully train and fit an algorithm, the use of deep neural networks is still time consuming when the number of inputs is really high. In addition,

the amount of content one can consume increases a lot on a daily basis: a person is not able to watch every single piece of content each platform makes available, so a recommender engine makes it so that users don't feel overwhelmed. As of 2013, a study by McKinsey [3] reported that 35 percent of what consumers purchase on Amazon and 75 percent of what they watch on Netflix come from product recommendations based on such algorithms.
Below, a closer look at the recommendation system that Netfix utilizes to provide recommendations for movies and tv shows to its users.

1.1 State-of-the-Art: Netflix recommendation engine

Recommendation algorithms are one of the main features of Netflix, the popular movies and TV shows streaming platform. They provide personalized suggestions to the users, mainly in order to reduce the amount of time they need to look for a movie or show they might like. On their website, [4] they state that they track what content the users consume and how they interact with the service to improve their system. The improvement comes in the form of A/B testing: a technique by which they provide a multitude of algorithms to different users, and end up using the one that works best for the majority - this helps the developers understand what they are doing wrong and how to improve the algorithm.
In 2007, Netflix promoted a contest in which they would give one million dollars to anyone capable of improving their recommendation algorithm. In order to accomplish this, they provided a training data set of around ten million ratings that around 400,000 users gave to almost 18,000 movies. Each training rating is represented by a quadruplet of the form ¡user, movie, date of rating, rating¿. The training is built so that the average user rated over 200 movies, and the average movie was rated by over 5000 users. Despite this fact, there is strong presence of variance in the dataset. The error metric chosen by Netflix is RMSE. Assuming that the rating matrix A_{mxn} with m users and 17K movies, n, the matrix is extremely sparse: it only contains 100 million ratings, implying that 99% of other possible ratings are missing. This is incredibly common in cases in which a dataset of user interactions are given: generally, there's data about the most popular items taken into consideration, there rarely is data about rare items people review or buy. Given the incredibly large amount of data and the sparse nature of the matrix, the teams which took part in the Netflix prize competition decided to tackle the problem by applying some sort of *dimensionality reduction*, which turned out to be the winning strategy. One way to perform dimensionality reduction is by using matrix factorization, which is useful when explicit feedback is not available because it's possible to infer users' preferences using *implicit feedback*, which is an indirect representation of what the userbase of a platform likes and dislikes based on the behavior of said users: if a user decides to watch a movie, the inferred information is that the user liked the movie enough to watch it.

The winning algorithm which outperformed the other ones used SVD to perform matrix factorization.
Again from what the company states on its website [5], the recommendations system strives to help users find a show or movie to enjoy with minimal effort. This happens by estimating the likelihood that a given user will watch a particular title in their catalog based on a number of factors including:

> Past interactions with their service (such as viewing history and how a user rated other titles);
>
> Other members with similar tastes and preferences in their database;
>
> Information about the titles, such as their genre, categories, actors, release year, etc.

In addition to this information, they keep track of the following:

> The time of day a given user watches a title;
>
> The devices users watch Netflix on;
>
> How long a user watches for.

All of these pieces of data are used as inputs in their algorithms. Worthy of note (in my opinion) is the fact that the recommendations system does not include demographic information (such as age or gender) as part of the decision making process in order to cluster similar users in terms of age groups and occupation.
Of course, if none of the recommended items are appealing to the user, they can just look at the entire catalog - which differs country to country - or enter a search query. The top results shown are based on the actions of other members who have entered the same or similar queries.

Below is a description of how the system works over time, and how these pieces of information influence what they present to their users. When a new Netflix account is created, or a new profile is added to an already existing account, they ask the user to choose a few titles that they like. These titles are then used to *jump start* the recommendations. The step just described is optional. If a user chooses to forego this step then a diverse and popular set of titles are presented to the user to give them something to watch immediately. Once you start watching something, this information will replace the initial optional title that a user may have provided. In addition, the titles that have been watched more recently will hold more weight than the titles watched in the past in order to drive the algorithm in the newer direction in terms of users' likings.

Not only Netflix's algorithm chooses which titles to include in the rows on the users' Netflix homepage, their system also ranks each title within the row, and then ranks the rows themselves, using complex systems to provide a personalized experience. Basically, they rank their items on each of their user's homepage to improve as much as possible their viewing experience: they present an ordering that supposedly maximizes the probability that the items ranked more highly are appealing to their userbase.

In each row there are three layers of personalization:

> The choice of row (e.g. Continue Watching, Trending Now, Award-Winning Comedies, etc.);
>
> Which titles appear in the row;
>
> The ranking of those titles.

The most strongly recommended rows go to the top. The most strongly recommended titles start on the left of each row and go right.
In addition to the collaborative filtering - based on both the items and the users - Netflix developed a machine learning approach based on reinforcement learning to create an optimal list of recommendations considering a finite time budget for the user. In the recommendation system approach, the factor of finite time to make a decision is often ignored; Netflix added this dimension to its recommendation system and in general in decision problems.

The evaluation problem can be measured in an amount of time because the users take some time to choose what they want to see: reading trailers, watching the preview, and so on, and different shows require different amounts of evaluation time. This time budget can be considered in the recommendation system, so the recommendation model should construct the recommendation list considering the relevance of the items and the evaluation cost for the user.

The user's time budget may not be directly observable (like the preferences), but the goal of the recommendation algorithm is to construct a recommendation list that has a higher chance of engagement for the user. The recommender system needs to learn the user's time budget in addition to the user's latent preferences.

2 Formal definition of the problem

When dealing with RS, data comes in form of interactions between a given user and an item. By *interaction* between an item and a user, one means that a user has interacted with a given item, whether watched it, bought it, reviewed it, or rated it. The items could be anything that could potentially be recommended to another person. The interactions can be implicit or explicit. For the former, it could be a click on a page: this information tells us that a user is interested enough in something to click on it (or hover over it on a smartphone). On the other hand, explicit interactions means that users give a rating to an item or review it. Usually, it's better to deal and have explicit feedback because it's an information explicitly given by the user and it offers higher quality data that can, in turn, make it easier to predict what that user might like. On the other hand, implicit feedback only relays the absence of presence of interest, making the information itself binary (1 if the user is interested, 0 otherwise).

2.1 Notation

Below, a table to summarize the notation used throughout the thesis.

I	The item matrix (shape m x m)
U	The user matrix (shape n x n)
R	The ratings matrix (shape m x n)

Table 1: General notation

Given a set U of n distinct users and a set I of m distinct items, the user-user matrix has dimension $n\ x\ n$ and the item-item matrix has shape $m\ x\ m$. The user matrix, U contains the information about the users present in the dataset. In the case at hand, one knows about their age - this information is coded in clusters - more about this later - and their occupation. These kinds of info are useful when the objective is to make subsets of users with similar characteristics: the inferred point is that users of similar age and occupation *might* enjoy similar content. A user-item matrix can also be constructed, M and it has $m\ x\ n$ dimension. The ratings matrix, R contains all the interactions, whether implicit or explicit that each user has with each item. R has shape $|U|x|I|$, meaning that each row represents a user and each column represents a given item. Of course, the interaction - or ratings - matrix is incredibly sparse, meaning that each user interacts only with a really small subset of the items present in a dataset. [6] In terms of interaction, a tuple is usually defined in the form of ⟨u, i, r ⟩, where u is a *user_id*, i is an *item_id* and r is the observed *response*. The interaction matrix represents the main source of information of a recommendation system, in the sense that it gives the algorithm the rating that each user assigns to each item.

Given the fundamental sparse nature of the matrix, when it's loaded in the system it provides data in the following way.

$$rating = \begin{cases} r_{ui}, & r_{ui} \in \mathbb{R} \\ NaN, & \text{otherwise} \end{cases}$$

where r_{ui} specifies the rating given from a user u to a given item i and it is a real number, usually from 0.5 to 5 in case of explicit rating, and *NaN* is still a real value, but unknown.
When dealing with a ML problem, data is usually split into a training set and a validation (or test) set: a subset of the whole dataset which is not used to train the model but it's utilized to test how well the algorithm performs on new data. The union of the training and validation set creates the whole dataset. Further details on this will be provided in the implementation section.

2.2 Machine learning perspective

From a machine learning point of view, a recommender system is a *prediction problem*. The objective is to predict the possible rating or score that a given user might assign to a certain item, in this case movies. Once this is done, with various possible techniques and results, the problem turns into a ranking one: given n amount of items that the user might like, which ones are going to be presented first, and which ones after? The *ranking problem* is not trivial as the attention span of a user is limited: the top-k of the n elements presented have to be the ones that almost surely the user will like. For example, the Netflix homepage is catered to each user in terms of both items and their internal ranking; in addition, they operate on the movies and shows posters to appeal to as many users as possible.
Of course, the recommendation issue can also be viewed as a *classification problem*: given a user and an item, we have to classify whether a user would like a given item - class 1 - or dislike it - class 0. The two points of view change the way we have to think about how to implement a system, but both are correct. In this thesis the prediction approach is implemented.

2.3 Biases and potential problems

Though recommendation systems have proven to be resourceful, they do have their own limitations as well. One of the biggest issues associated with the content-based analysis is that it can lead to over specialization and sparsity of data. Over specialization happens when, using a content-based filtering, the recommendation systems tend to systematically recommend the same items to similar users. [7]
This implies that the diversity in the results is minimal, which could be perceived as a problem, either by the user or by the product sellers. All these recommendation methods suffer from the same tendency to foster the same kind of items to the vast majority of users, a situation known as the overspecialization

problem [8] or the serendipity problem [9]. This problem arises from the lack of reviews for new items or from the tendency to recommend the most general products instead of more specialized ones. A personalized recommendation implies more diversity in the possible outcomes while fostering the unexpectedness, in particular in entertainment and cultural recommendations.

2.3.1 Bias

According to one of the main expert in the concept of bias in recommendation systems, one can define three main types of bias [10]:

> Statistical: it occurs when there is a systematic deviation from a prior distribution;
>
> Cultural: given by the interpretation and judgement that each person has;
>
> Cognitive: whenever a deviation occurs in a pattern of events that are considered the norm.

It's quite clear that, even though the before mentioned biases are the three main kind, people who work and research these systems focus on cultural bias the most, including gender, racial, sexual, age, religious, social, linguistic, geographic, political, educational, economic, and technological.
In addition, often people do not take into account statistical biases: they scrape data from the web or extrapolate some results from a data manipulation process without considering how the data is gathered, sampled, validated, the presence of noise etc. In addition, there is cognitive bias when measuring bias.
Most web systems, including the automatized ones, are optimized by using implicit user feedback. However, user data is partly biased by the choices that these systems make. For example, we can only click on things that are shown to us, and what is shown to us is highly dependant on many factors that are out of the control of the users. A clear example of this is what is displayed on the first page of results of a given query search. Because these systems are usually based on machine learning, they learn to reinforce their own biases: instead of positive reinforcement learning, the machine learning algorithm is fine-tuned on wrong assumption it makes itself on the information it receives as input, or the developers themselves feed to the network. Personalization and filter bubbles for users can create echo chambers for recommender systems.

Let us consider a user who only watches horror movies and each time they log on a given platform, they filter out any movie that is not a horror. In this case, the recommender system only learns one information: the user in question likes only horror movies and never recommends anything else. This leads to another popular form of bias in recommending systems: niche products and items are often never recommended to users, leading to the fact that it becomes even harder (in a huge amount of data) to find a niche product that one might like. The concept explained in the previous sentence is called *popularity bias*, and it happens when a recommender engine only provides recommendations for a few most popular items on a website. This action implies that the sales for new or niche items that are not particularly famous on a given platforms are cut. Many recommender systems ignore unpopular or new items in a dataset that only have few ratings and focus only on those items having enough ratings to be easier to use in the recommendation algorithms. Granted, such unpopular or newly introduced items can remain in the system but would require special handling using various cold start methods, as described in [11]. Using the same terminology as in [12], these unpopular or new items belong to the *long tail* of the item distribution, as shown in Figure 1. Following the spirit of extensive research on the long tail phenomena [12], these types of items should not be discarded or ignored but gainfully utilized in recommendation methods. [13]

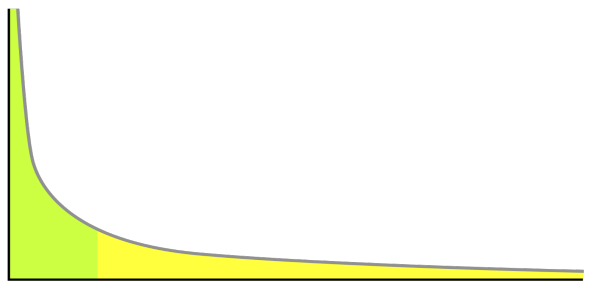


Figure 1: Example of a distribution with a long tail

Picture by Hay Kranen/PD

Given a distribution of items, which could be movies, books, or any product which could be recommended to another user, the most popular item, which happen to be the ones that have most ratings, or interactions, are the ones most likely to be picked and recommended to other users.
This is mainly because they provide more information to the recommender engine. There were several partial solutions to solve the *popularity bias* – particularly in systems that use personalization. This includes replacing one or more of the popular items that you're presented with today with other items that improve the diversity, novelty, and serendipity of what gets presented. If recommender engines never show less popular items, the majority of the users never discover said new items: no one has the time and willingness to go through an entire catalogue on a platform; in addition, there are so many options to choose from, that anyone would feel overwhelmed - the amount of content produced online is too much. Other than popularity bias, there are many other kind of biases one needs to take into account, namely:

Presentation bias: the items that receive more exposure are more likely to be picked by users;

Position bias: items that are in the upper part of a web page are more likely to be chosen;

Social bias: users are more likely to pick items that have higher ratings or better reviews;

Interaction bias: meaning which items catch our eyes when scrolling a web page;

Ranking bias: users tend to assume that items with a higher ranking are better;

Click bias: the amount of clicks or interactions that an item gets tends to be considered as a positive feedback;

Mouse movement: the more time users spend hovering over an item, the better the item.

Of course, fine-tuning an algorithm does not imply that any of the biases mentioned above is automatically solved, some of them come from the way users are influenced and think. One thing users can do is to be aware of all these points and try to change the way they think about what recommendations they get. On the other hand, developers and people who work and research these machine learning solutions should be aware of these biases and manipulate the implementation accordingly.

2.3.2 Cold start

Related to the long tail problem is the cold start one. The cold start issue occurs when the system lacks rating information to make reliable recommendation at that specific time.
The point of a recommendation engine is to filter information for users such that each person has a list of content that is tailored to their liking. This usually happens by comparing a user's profile to some other profile or reference characteristics. These characteristics can be based on the items themselves (content-based filtering) or to the users' past behavior and similarities to other users (collaborative filtering).
There are three cases of cold start: [14]

New community: when, although a list of items might already exist, almost no users are present and the lack of user interaction makes it very hard to provide reliable recommendations;

New item: a new item is added to the system, it might have some content information but no interactions are present;

New user: a new user registers and has not provided any interaction yet, therefore it is not possible to provide personalized recommendations.

The new community problem, or *systemic bootstrapping*, refers to the initial state of the system, when virtually no information the recommender can rely upon is present. This case presents the disadvantages of both the new user

and the new item case, as all items and users are new. Due to this some of the techniques developed to deal with those two cases are not applicable to the system bootstrapping.
The item cold-start problem refers to when items added to the catalogue have either none or very little interactions. This constitutes a problem mainly for collaborative filtering algorithms due to the fact that they rely on the item's interactions to make recommendations. If no interactions are available then a pure collaborative algorithm cannot recommend the item. In case only a few interactions are available, although a collaborative algorithm will be able to recommend it, the quality of those recommendations will be poor. [15] This raises another issue, which is not anymore related to new items, but rather to unpopular items. In some cases (e.g. movie recommendations) it might happen that a handful of items receive an extremely high number of interactions, while most of the items only receive a fraction of them. This is referred to as popularity bias, explained in the previous section.

2.3.3 Sparsity

A sparse dataset refers to a matrix with many observations set to zero. They differ from matrices where most of the values are non-zero, which are called dense. In addition, a sparse matrix is not the same as a dataset with many missing values: an observation not present in the dataset does not imply that its value is 0. Below, a visual example of a sparse matrix next to a dense one.

$$S = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 1 & 0 & \cdots & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \tag{1}$$

Generally speaking, data sparsity is common in many datasets that count the occurrences of a certain event or that count the activity of items or users. From this, we gather that the interaction matrix that keeps track of ratings given by some users to a few movies is incredibly sparse. Sparsity can pose some challenges because very large datasets are hard to tackle in terms of complexity. In particular, they are complex in terms of time and space: a really large matrix needs quite a bit of space to fit into memory, and if one manages to load it in memory, if one wants to perform some operations on it, they will take a lot of time to process.
In addition, sparsity causes unique challenges for machine learning, which will be explained briefly. First of all, overfitting occurs when a model becomes too complex and starts to capture noise in the data instead of the underlying patterns.

In sparse data, there may be a large number of features, but only a few of them are actually relevant to the analysis. This can make it difficult to identify which features are important and which ones are not.

As a result, a model may overfit to noise in the data and perform poorly on new data.

One of the biggest problems with sparse data is that it can lead to the loss of potentially useful information.

When we have very limited data, it becomes more difficult to identify meaningful patterns or relationships in that data. This is because the noise and randomness inherent to any data set can more easily obscure essential features when the data is sparse.

Furthermore, because the amount of data available is limited, there is a higher chance that we will miss out on some of the truly valuable patterns or relationships in the data. This is especially true in cases where the data is sparse due to a lack of sampling, as opposed to simply being missing. In such cases, we may not even be aware of the missing data points and thus may not realize we are losing valuable information.

That's why if too many features are removed, or the data is compressed too much, important information may be lost, resulting in a less accurate model.

Memory problems can arise due to the large size of the dataset. Sparse data often results in many features, and storing this data can be computationally expensive. This can limit the amount of data that can be processed at once or require significant computing resources.

The time problem can also occur due to the large size of the dataset. Sparse data may require longer processing times, especially when dealing with a large number of features. This can limit the speed at which data can be processed, which can be problematic in time-sensitive applications.

Below, the visual depiction of the sparsity of the dataset used in the final project.

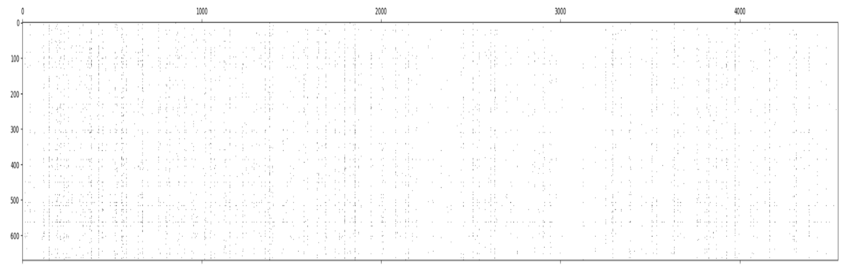


Figure 2: Sparsity of the users-movies matrix

3 Methodologies and metrics definitions

In this section, an introduction to the three different methodologies included in the implementation will be presented. The first part is the one based on machine learning, the second on on deep learning and the third on a special case of reinforcement learning. All three are regularly used together with other methodologies in many state-of-the-art systems, including the one adopted by Netflix, as described in the introduction 1.1. In this thesis, a simplified version of the combined approach using all three will be presented.

3.1 Collaborative filtering

In order to produce good recommendation for users, it's usually preferred to implement an item-based collaborative filtering approach. Item-based techniques first analyze the item matrix to identify some relationship between items - this is based on some parameters - and then utilize these relationships between items to give recommendations to users. On the other hand, user-based techniques analyze the user matrix, comparing the rating a user assigns to a certain item, in order to find other users in the database with similar taste. A hybrid approach combines the two methodologies to compare both similarities between the users and the items.

Regarding the second main category, content-based filtering, these methods analyze the content of products, services or encounters a customer has rated. Based on these ratings, content-based filtering methods draw up a user profile based on the characteristics of the ratings and use these profiles to recommend items that are similar to the ones that were positively rated [16]. One of the major pitfalls with these kind of methods is that they tend to give overspecialized recommendations including only items that are very similar to the ones already rated or bought, as mentioned in 2.3.

The recommendation problem can be formalized in the following way [17]. Let U $(u_1, u_2, \ldots, u_m)$ be the set of all available users in a recommender system and consider I $(i_1, i_2, \ldots, i_n)$ be the group of all items users have access to in the system. Let $f : UxI \rightarrow R$, where R specifies an entirely ordered set be a utility function such that $f(u_m, i_n)$ computes the usefulness of item i_n to user u_m. Then, for every user $u_m \in U$, the system chooses an item $i_{max,u_m} \in I$, unknown to the active user, which maximizes the utility function f.

3.1.1 Recommender system taxonomy

The recommender system core function is to determine which products, or items, are worth recommending to a user. In order to accomplish this task, a recommender system must examine items which are valuable for suggesting to the target user. The system must be able to predict how much a user will enjoy some of them, or at least evaluate the utilization of some products and then

choose which products are suitable to suggest depending on this evaluation. Recommendation methods have a variety of possible categories [18, 19].
To provide an exhaustive review of the different kinds of recommender systems, a taxonomy offered by Burke [20] that has become a traditional way of identifying between recommender techniques will be utilized in this thesis. Burke [20] differentiates between six different classes of recommendation approaches as:

Content-based Content-based recommender system try to suggest products that are similar to the ones that the user liked, or interacted with, in the past. The similarity between different items is given by analyzing traits associated with the compared content. For example, if a user usually gives a really high rating to horror movies, the system will understand to suggest other movies from this genre. In addition, content-based recommenders will treat recommendations as a user-specific problem, meaning that they will provide unique suggestions tailored to each user, depending on the trait of the analyzed product. According to [21], content-based recommender system will evaluate a set of details for each item previously ranked or rated by a given user and develop a user profile based on the features that a given user likes. In [22], the authors define a general structure of how content-based recommender engine operate, namely:

1. Construction of a user profile from previous rating assigned from said user to different items;
2. Identify similar users who rate different items in a similar manner to a target user using a similarity metric such as cosine similarity, Pearson correlation coefficient, or distance-based similarity;
3. Recommend the top k items that the like-minded users enjoyed after their ratings are predicted as an average of ratings given on an item by the previously identified similar users.

Collaborative filtering According to [23], this strategy dictates to recommend to target users items that similar-minded users liked in the past. The *likeness* measure is given with regards to the ratings given by different users in the past. All collaborative filtering methods are capable of using past ratings of users to predict or recommend new items that a person might like. There is a strong underlying assumption in this type of modeling: likeness between users and between items - where the similarity between elements of the two sets is given by some metric - is provided by some implicit level of agreement between past preferences and current ones. Two variants of the collaborative filtering algorithm can be classified as *user-based* and *item-based*. In [24] it is stated that collaborative filtering methods can be further classified into two groups: memory-based and model-based. Memory-based methodologies are heuristics that predict ratings that depend on the whole collection of items rated by users in the past. As the name suggests, these algorithms require that all ratings, items, and users' data reside in memory.

The immediate drawback of this is that only fairly small datasets can utilize this algorithm, as scalability is a huge issue. Model-based methods [25] use the group selection of ratings to learn a model, which will then be used to make predictions on the ratings.

Demographic filtering This type of system suggests items depending on the user demographic profile. The assumption is that different recommendations should be suggested to different clusters of people. A simple example would be a recommender engine that suggests content based on the age of the user. The advantage of this methodology is that the amount of past ratings needed for this approach is way less than the amount needed in a classical approach. Even though it does present this advantage, there aren't many applications that make use of this method.

Knowledge-based In this type of recommender engine, the model recommends items based on the particular domain knowledge about how well some features that specific items present are able to fulfil the needs and preferences of the users in the database. Knowledge-based recommender systems are usually case-based, meaning that it is well suited to many product recommendation domains where individual products are described in terms of a well defined set of features. These representations allow case-based recommenders to make judgments about product similarities in order to improve the quality of their recommendations. [26] A knowledge-based model is based on a similarity function that takes into account the users' needs and the solution of the problem (recommendations that match the needs) and estimates how much the user needs them. Knowledge on a particular domain is the foundation for these systems: the knowledge itself models how a particular item meets a particular user's needs and can draw connections between a need and possible recommendation to satisfy it. In this case the user profile can easily be seen as a source of knowledge to support the inference mentioned before. Knowledge-based recommendation systems are not based on the ratings given by some users, therefore they do not suffer from the sparsity problem and the early rater problem. For this reason, this system can be a complement to other recommendation approaches.

Community-based They work on the preferences of the user's close contacts to recommend items. It's a fact that people tend to value more highly the opinion of someone close to them rather than the recommendation of a machine or anonymous users. [27] Community-based methodologies are also connected to the graph structure of social networks: they build a cluster of similar users based on the different communities they are a part of. The recommender system models the information from the relationships between groups of similar users and the preferences of the user's friends in that group specifically. Instances of statistical and graph-based methodologies are Bayesian generative models [28], graph clustering approaches, hierarchical clustering and modularity-based models. [29]

Hybrid recommender systems These systems are basically a combination of two or more of the approaches mentioned above. The main goal of these algorithms is to overcome the shortcomings that each of them have: by combining their capabilities, more accurate predictions are produced. Seven categories of hybrid recommendation systems exist, namely weighted, switching, mixed, feature combination, feature augmentation, cascade, and meta-level. [20]

Collaborative filtering methods uses the opinion of other community for recommendation to target user. Generally, prediction for target user is made based on many other similar users by gathering taste information [30]. Therefore, collaborative filtering is based on the assumption that users who agreed in the past will tend to agree in the future. Within the last decade CF has been enhanced constantly and lastly became one of the most popular customization methods in the area of recommendation techniques. The objective of a collaborative filtering algorithm is to recommend new products or to estimate the utility of a certain product for a specific user depending on the customer's past likings and the views of other like-minded users. There are two tasks that a CF can perform, leading to two unique kinds of result. The first task is the rating prediction process, which implies predicting the rating that a given unseen product will have for the target user. The predicted rating, $r_{i,j}$, can be defined as a numerical value that indicates the predicted likeliness of item $ij \notin I$, where I is the set of items for the target user u_k. It's worth mentioning the the new predicted rating that the engine assigns to a give item by a certain user has to maintain the same scale used previously by other users in the dataset (e.g., from 1 to 5). If multiple scales are present in the dataset, normalization is required to make sure that each dataset contains the same values.

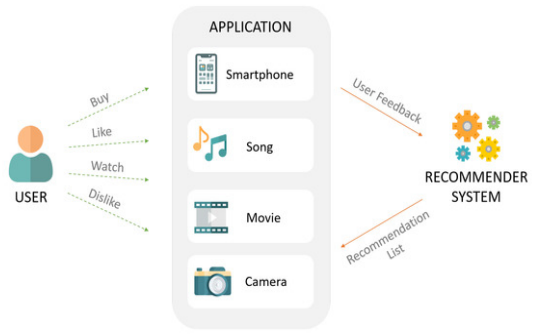


Figure 3: The schema of a simple RS

Picture taken from www.mdpi.com

3.2 Feature extraction on movie posters

A convolutional neural network, CNN, will be used in this part of the project to find similar movie posters for the recommendation process. The use of images in recommendation systems, especially the ones which operate on a dataset for movies, is fairly uncommon, at least in this particular task. Many repositories exist on both GitHub and Kaggle in which the goal is to classify the movie genre based on the movie poster, but this is not the case. My motivation for adding this particular task is given by the fact that I came across one article online [31], stating that many state-of-the-art engines do make use of the images present in their database to choose the ones that would make the product look more appealing to the users. Netflix does this too, with their movies and TV shows posters. The problem of finding similar images has been well documented and studied in the past years, with many papers and research done in the field of computer vision.

When talking about image similarity, the problem can be further split into two different tasks that have been studied and developed over the years: feature extraction and similarity measure. Feature extraction refers to the process of transforming raw data - in this case images - into numerical features that can be processed while preserving the information in the original dataset. [32]
This can be done manually or automatically. Manual feature extraction requires identifying features that are relevant for a given problem and implement a way to extract those particular features. Automated feature extraction, on the other hand, implies the use of specialized algorithms or deep neural networks to extract features automatically.

With the development and growing popularity of deep learning, feature extraction for images has become a common task performed by the first layers of a deep neural network. The main objective of feature extraction is to represent the most relevant part of a picture as a compact feature vector. In particular, recent literature focused on the use of CNN to perform this task. The reason behind this is that convolutional neural networks are hierarchical in nature, in the sense that extract a hierarchy of features from every image passed as input, from low level features in the first layers to high level features in the last layers. [33] The goal of this approach is to extract a feature embedding for each image: this is basically a vector of numbers that aims at representing the image in a more compact way. The neural network will then compare these vectors of numbers and assess how similar two or more pictures are. This computation is performed by the ANN itself: the pixels in a given image are transformed by the fully connected layers into feature embeddings that will then be used to performed different tasks, one of which could be classification. The logic of this process is that by comparing the feature embeddings alone, it's possible to infer how similar two images are. When dealing with the concept of *similarity*, various measures for computing vector similarity exist. The most widely used is *cosine similarity.*

To map discrete variables into continuous numbers, embeddings are utilized. Embeddings are low-dimensional vectors that represent categorical variables. Embeddings produced by neural networks allow for searching neighboring elements in the data point space and make recommendations based on this representation. Embedding visualization is useful as it easily creates relations between different elements in a given dataset by reducing dimensionality without losing information about what is being represented. To generate image embeddings, the most important aspect is to accurately select a set of features that will represent the images in the best way possible. One way to do this is to build up a set of descriptors, but this representation lacks the ability to perform a more *symbolic* search.
Deep learning convolutional networks can be used to extract features from pictures that are invariant to both geometric transformation and in the instance itself, meaning that two images belonging in the same category will always have the same representation.

3.2.1 Convolutional Neural Networks

Generally speaking, a convolutional neural network, *CNN*, is a particular kind of artificial neural network, ANN, built specifically for signal processing, regardless of whether it's an image, an audio or a video. For this reason, its importance is critical in the CV field, specifically for object classification and pattern recognition tasks. To identify patterns within a picture, CNN leverages mathematical principles from linear algebra, such as matrix multiplication. The architecture of a CNN ultimately mimics the connectivity patterns of the frontal lobe of the human brain, the area responsible for processing visual stimuli. The neurons of the convolutional neural network are arranged in a way built to ensures that images can be fed to the network in their original resolution, without any loss. For this reason, a CNN delivers better performance with image inputs compared to any other kind of artificial neural network. A CNN consists of three layers that increase in complexity - this complexity allows the network to successfully identify larger portions and more complex features of an image.

Convolutional layer The majority of the computation is handled in the convolutional layer, the core building block of this architecture. More than one convolutional layer can be stacked. The process of convolution implies the presence of a kernel - or filter - inside of this layer moving across the receptive fields of the image, in order to check whether a given feature is present or not. Over multiple iterations, the kernel slides through each pixel in each row, until the entire image is swept. After each iteration, a dot product is computed between the input pixels and the filter. The output of this procedure is a feature map or *convolved feature*. Finally, the image is converted into numerical values that the CNN can interpret so that it can extract relevant patterns and features from it.

Pooling layer Where the previous layers extract features from pictures, the pooling layers consolidate these features previously learned. The reason behind this is to shrink the spatial dimension of the pictures' dimensions to minimize the number of parameters and computation in the network. Specifically, these layers downsample the feature map to introduce *translation invariance*, which is necessary because the feature map produced by the convolutional layers is location-dependent: if an object in a given image is slightly shifted or rotated, it might not be recognizable again. The pooling layer achieves translation invariance by iterating through the pictures by means of a kernel. Two types of pooling exist:

Max pooling: it works by selecting the maximum value from every pool. Max pooling retains the most prominent features of the feature map, and the returned image is therefore sharper than the original one;

Average pooling: it works by getting the average of the pool. Average pooling retains the average values of features in the feature map. It smoothes the image while maintaining the essence of the features in an image.

The main objective of the pooling layer is to reduce the number of parameters that were provided as input, resulting in some information loss. This is necessary in order to improve the efficiency of the the CNN, which in turn reduces overfitting for the CNN model.

Fully connected layer The fully connected layer is the one responsible for the image classification computations, based on the features extracted in the previous layers. This layer contains the same number of units as the number of classes and the output activation function, such as *softmax* or *sigmoid*. *Fully connected* refers to the fact that all the nodes from one layer are connected to every activation unit of the next layer. The rest of the layers in the CNN are not fully connected because that would make the ANN too dense, implying that the computational load would increase, affecting the losses and the output quality.

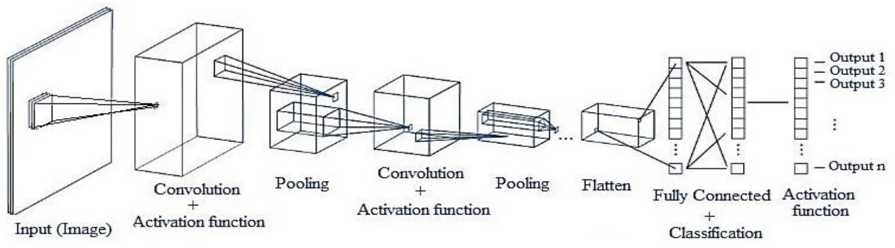

Input (Image)
Convolution + Activation function
Pooling
Convolution + Activation function
Pooling
Flatten
Fully Connected + Classification
Activation function
Output 1
Output 2
Output 3
Output n

Figure 4: Representation of a CNN architecture

3.3 Performance metrics definition

In this sections, all the different metrics used to measure the performance of the model will be briefly explained.

3.3.1 RMSE

RMSE, root-mean-square error or RMSD, root-mean-square deviation, is a measure of difference between the sample (or population) values predicted by a model and the values observed. [34] It computes the square root of the second sample moment of the differences between the values predicted by the model and the ground truth values.
The RMSE is an aggregated measure of the magnitude of all the errors in the prediction model across all the data points into a single measure. RMSD measures the accuracy, used to compare different forecasting errors given by different forecasting models for the *same* dataset, as it has the characteristic of being scale-dependent. [35]

RMSD is always non-negative, ($RMSD > 0$), and a value of 0 would imply a perfect fit to the data, meaning that the model captures the data points without fault. Generally speaking, a lower RMSD is better than a higher one. However, comparisons across different types of data would be invalid because the measure is dependent on the scale of the numbers used, as stated above.

RMSD is the square root of the average of squared errors. The effect of each error on RMSD is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSD. Consequently, RMSD is sensitive to outliers.
Below, the formula for the RMSE.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)}{T}}, \tag{2}$$

where $\hat{y}_t$ represents the predicted values for times t of a regression's dependent variable y_t with variables observed over T times, computed for T different predictions as the square root of the mean of the squares of the deviations.

3.3.2 MAE

MAE, Mean Absolute Error, is a measure of errors between pairs of predicted values and observed ones. [36] MAE is computed as the sum of absolute errors divided by the sample size. [37] Below, the formula used to compute MAE.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}. \tag{3}$$

It is thus an arithmetic average of the absolute errors $|e_i| = |y_i - x_i|$, where y_i is the prediction and x_i the true value.

The mean absolute error, as the root-mean-square error, uses the same scale as the data being measured. This implies that MAE too is a scale-dependent accuracy measure and therefore cannot be used to make comparisons between predicted values that use different scales.

4 Implementation and results

Before getting into the implementation methodologies, a brief explanation on how the datasets are structured, the used features, and finally some EDA. In terms of ratings, the main dataset used for the project is the 100K MovieLens one. Given the fact that the computations are performed locally on the Google Colab platform (which comes with 12 GB of RAM on its free plan), the smallest version of the dataset is utilized. Even though this version of the dataset is fairly outdated, it keeps being used in a large amount of projects.

The main dataset is made up of three different CSV files, namely the items one, the users one and the ratings one. Instead of using the provided file directly, I made use of the *tmdb 5000* dataset as my *items* database. This dataset is composed of two files, one for the movies and one for the credits. They can be combined by using the merge function available on the Pandas library. The merged dataset provides information for exactly 4803 movies, and the info include the following.
Note: only the used features will be listed below in order to keep the list brief.

Id - the TMDB identifier for each movie;

Title - the movie title (translated in English);

Year - the year each movie was released;

Genres - the genres for each film;

Keywords - some words that describe the movie themes;

Overview - the movie synopsis;

Popularity - a score that indicates how popular a movie is on the platform;

Vote average - the average rate given to each movie (from 1 to 10);

Vote count - the amount of ratings each movie has;

Cast - the list of actors present in each film;

Crew - the list of people hired by a production company whose purpose is producing the movie;

Poster link - this field is used to scrape the poster for each film;

In terms of information for each rating, the dataset provides the following.

User_id - the unique identifier for each user;

Item_id - the unique identifier for each movie;

Rating - a score from 1 to 5.

Finally, the information for each user will be listed below.

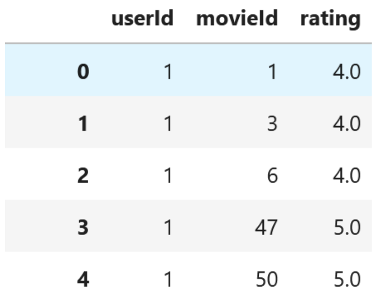
User_id - the unique identifier for each user;

Gender - the gender of each user;

Age - the age of each user;

Profession - the job each user has.

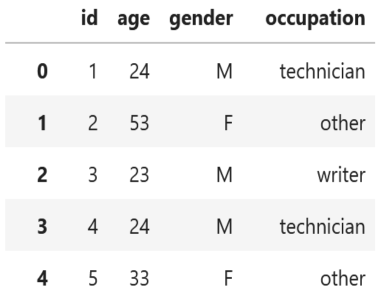
Before displaying some examples of extracted data and what information could be gathered from it, I wanted to provide some visual examples of how it was structured.



	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

Figure 5: Snippet of the ratings dataset

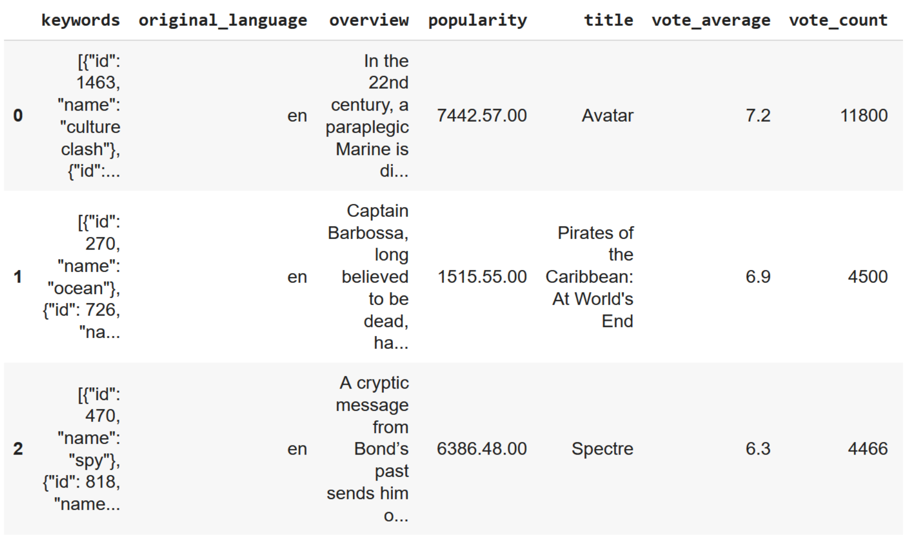
For each entry in the dataset, the unique *id* for each user and film are give, together with the rating (in this case each movie has a rating from 0.0 to 5.0).



	id	age	gender	occupation
0	1	24	M	technician
1	2	53	F	other
2	3	23	M	writer
3	4	24	M	technician
4	5	33	F	other

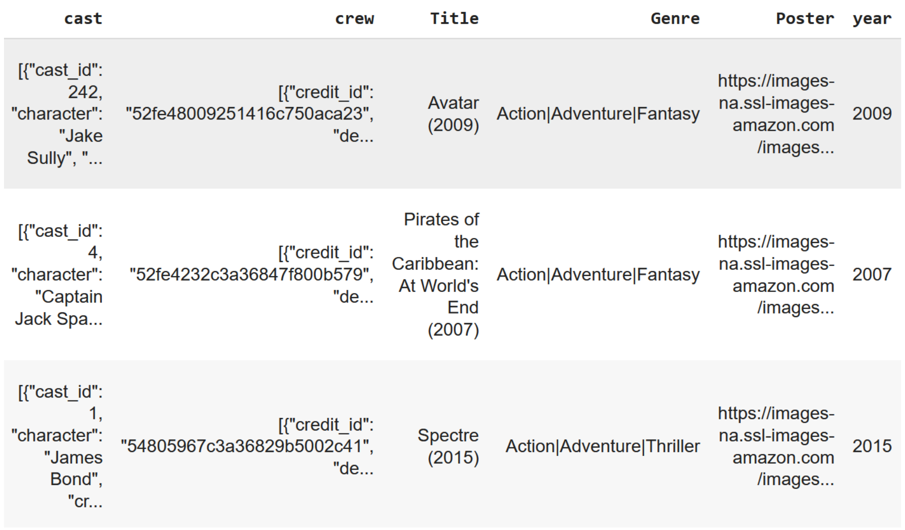
Figure 6: Snippet of the users dataset

For each user, on the other hand, information about their gender, their age and their occupation is provided. For example, the user with *id* 1 is a 24 years old man whose job is a technician.



	keywords	original_language	overview	popularity	title	vote_average	vote_count
0	[{"id": 1463, "name": "culture clash"}, {"id":...	en	In the 22nd century, a paraplegic Marine is di...	7442.57.00	Avatar	7.2	11800
1	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Captain Barbossa, long believed to be dead, ha...	1515.55.00	Pirates of the Caribbean: At World's End	6.9	4500
2	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	A cryptic message from Bond's past sends him o...	6386.48.00	Spectre	6.3	4466

Figure 7: Movies dataset - part 1



cast	crew	Title	Genre	Poster	year
[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...	Avatar (2009)	Action|Adventure|Fantasy	https://images-na.ssl-images-amazon.com /images...	2009
[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...	Pirates of the Caribbean: At World's End (2007)	Action|Adventure|Fantasy	https://images-na.ssl-images-amazon.com /images...	2007
[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...	Spectre (2015)	Action|Adventure|Thriller	https://images-na.ssl-images-amazon.com /images...	2015

Figure 8: Movies dataset - part 2

In terms of information about movies, the columns are listed above. Some of them come in a string format, some in a json format (keywords, cast, and crew) and another are numerical values.
In terms of data analysis, I wanted to show some information about the users and their occupation: the question I wanted to answer was how the occupations are distributed among the population.

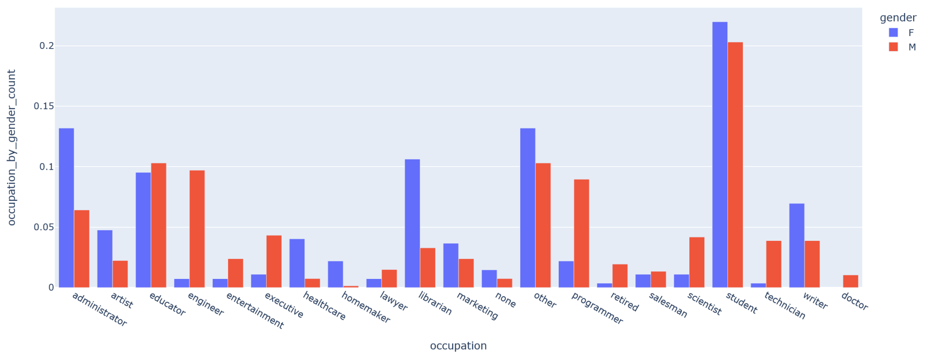


Figure 9: Distribution of the different occupations among users

From the above picture, one can gather that most of the users in the dataset are actually student, which is not so surprising once I think about the fact that young people tend to spend more time online so they are more likely to give ratings to movies (I know I do, at least). Another question that I wanted to answer was the following: is there a genre preference among users? As in, is there a correlation between what users tend to watch more and their gender?

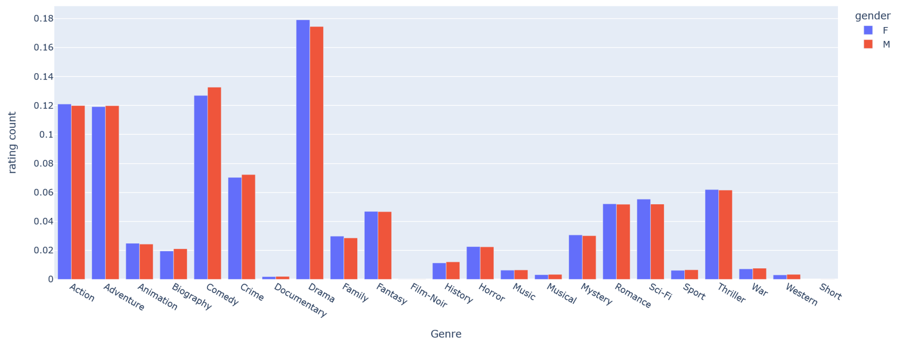


Figure 10: Total amount of movies watched for each genre divided by gender

Finally, I wanted to report the distribution of the movie releases over the years. As one could have guessed, most movies in the database are fairly recent:

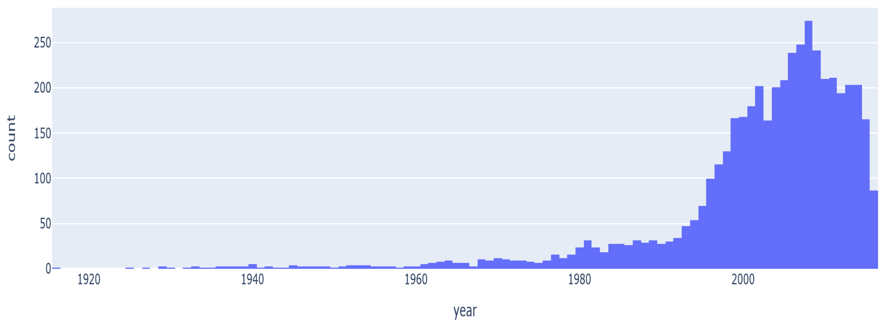


Figure 11: Movie released over the years

many are from the 80's and then there is a huge increase during the years, until a peak is reached in the early 2000s and 2010s.
Finally, a graph that shows how many movies are present for each genre.

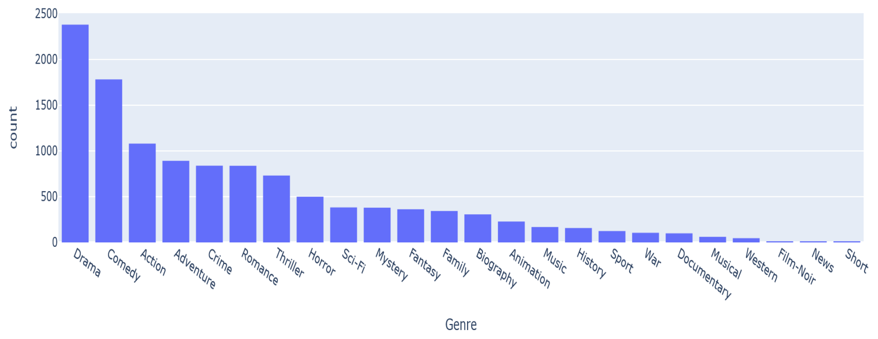


Figure 12: Movies divided by genre

The two most prominent genres are drama and comedy, followed by action. The genres that follow are pretty close in terms of numbers, until we reach almost zero movies for noir films, news and short, which are the least present movies in the dataset.

4.1 Methodology one: Collaborative filtering

Before starting with the analysis, some computations and checks were made in order to make sure that the columns needed for the implementations of the recommending engine were without null values. In addition, I removed all the columns that were not needed to save some memory space, and proceeded with some very basic statistics about the numerical features.

The first step in my analysis was to compute a *WR*, a weighted average rating given by the following formula,

$$weighted\,rating\,(WR) := \left(\frac{v}{v+m} * R\right) + \left(\frac{m}{v+m} * C\right), \tag{4}$$

where

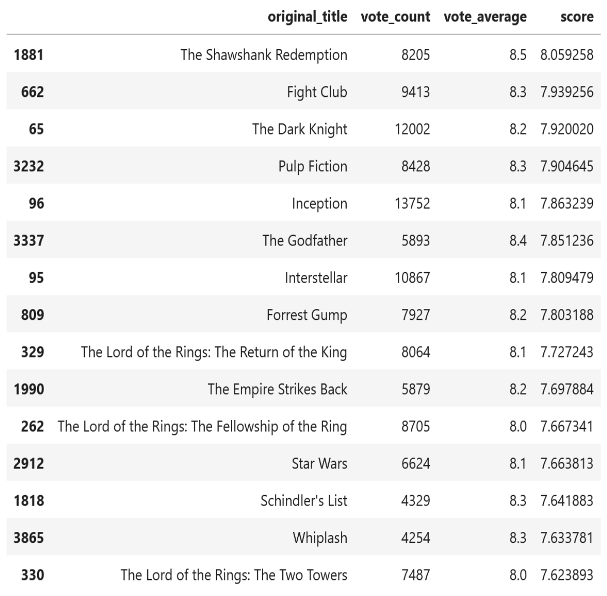
v is the number of votes for each movie (vote_count in the dataset);

m is the minimum amount of votes required t be listed in the chart;

R is the average rating of the movie (vote_average in the dataset);

C is the average vote across the whole chart.

The reason why this formula is utilized is because although IMDb accepts and considers all votes received by its userbase, not *all* of them have the same impact (or weight) on the final rating. This is done mainly to discourage unusual voting activity. [38]
Before computing the actual WR value for each movie, the values C and m need to be calculated. The non-trivial part is picking the *right* quantile so that the cut-off in the rating distribution is not so strict that not enough movies remain and not strict enough so that too many movies remain after having applied the filter. In my case, a 90% quantile was applied, meaning that a movie needs to have more votes than at least 90% of all movies to be considered a candidate in the *trending* set of movies. After having computed WR for each movie in the dataset, and having ordered the items such that the highest scoring movies end up on top, the result is the following.



	original_title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884
262	The Lord of the Rings: The Fellowship of the Ring	8705	8.0	7.667341
2912	Star Wars	6624	8.1	7.663813
1818	Schindler's List	4329	8.3	7.641883
3865	Whiplash	4254	8.3	7.633781
330	The Lord of the Rings: The Two Towers	7487	8.0	7.623893

Figure 13: Trending movies according to WR rating

From the chart above, what stands out is that the movies with the highest score are the ones that are considered to be incredibly popular in the western pop-culture. Now, this could be considered to be the equivalent of the *most popular of all time* section for the movies, where by this definition I mean the most popular in terms of how many ratings they received and the average rating (according to WR). This could be a safe attempt at overcoming the cold start problem 2.3.2: when no information about a new user are available, it's common to present the most popular items in a given dataset. Another option to overcome this issue is to display the most popular items over a given period of time. In that case, the presented options would be the ones in the graph below.

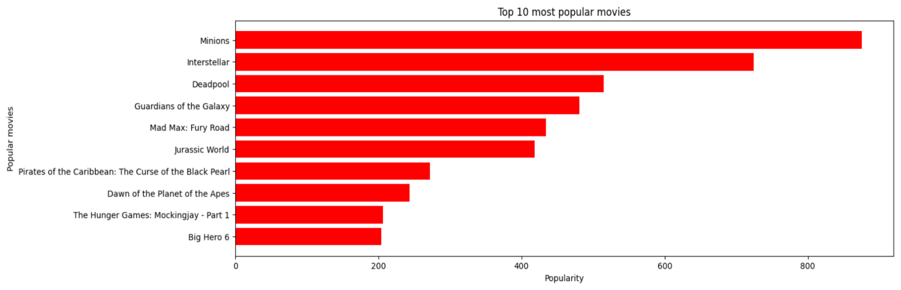


Figure 14: Popular movies in the dataset

Again, the movies above are *blockbusters*, movies which are successful both in terms of budget and sales, but also widespread in success among different demographics, differing from the first chart, where the presented films are considered to be *all time classics.*

To begin with the actual implementation of the recommender system, the first step is the *content-based filtering.* This is done by utilizing the overview feature for each movie, as it contains a brief synopsis for each entry. Below, as an example, the plot for the movie *Avatar.*

> In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between following orders and protecting an alien civilization.

In more detail, the function used is the Tfidvectorizer provided by Sklearn which computes the TF-IDF. The TF is the term frequency, so the amount of times each word appears in a particular document. The IDF, on the other hand, is the inverse document frequency, which is a measure of how common or rare a term is across the entire corpus of documents (overviews in this particular case). If a word is common and appears in many documents, the IDF value (normalized) will approach 0 or else approach 1 if it's rare. As per Sklearn online documentation, they implement this measure as the following.

$$IDF(t) = log_e \left[\frac{(1+n)}{(1+DF(t))} \right] + 1 \tag{5}$$

The result of this computation will then be transformed into a matrix of the dimensions $(4803, 20978)$. After this, the cosine similarity function is applied to compute how similar each synopses is to to all the other ones. Below, an explanation of what cosine similarity is and how it's computed.
Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. More in detail, it is the dot product of the vectors divided by the product of their lengths. From this, we know the cosine similarity doesn't depend on the magnitude of the vectors, but only on their angle. The cosine similarity resides in the interval $[-1, 1]$. When the component values of the vectors cannot be negative, the cosine similarity will be bounded in the interval $[0, 1]$. The biggest advantage of the cosine similarity is its low complexity, especially in the case of sparse vectors, which implies that this measure is particularly suited for recommendation systems.

Cosine similarity is defined as:

$$S_c(A, B) := cos(\theta) = \frac{\mathbf{A} * \mathbf{B}}{||\mathbf{A}||\,||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2 * \sum_{i=1}^{n} B_i^2}} \tag{6}$$

where A_i and B_i are the ith components of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively.

By using the overview information alone, recommendations will be based on movies with a similar plot. For example, by querying a movie similar to *Avatar*, the following movies are suggested.

Apollo 18;

The American;

The Matrix;

Obitaemyy Ostrov;

Tears of the Sun;

Hanna;

The Adventures of Pluto Nash;

Semi-Pro;

Supernova;

Blood and Chocolate.

After this, I decided to expand this method to other features other than plots, namely the top three actors in the cast, the director, the genres and the keywords. Again, taking *Avatar* as an example, the operation described above yields that Sam Worthington, Zoe Saldana, Sigourney Weaver are the three main actors, James Cameron is the director, *culture clash, future, space war* are the keywords and genres are action, adventure, and fantasy. These strings are then mixed together in order to create a longer line of texts that can potentially contain the most meaningful information about each film. Again by making use of the cosine similarity, we can get movies with similar features. Repeating the process as before with *Avatar*, this time the process yields the movies below.

Clash of the Titans;

The Mummy: Tomb of the Dragon Emperor;

The Monkey King 2;

The Sorcerer's Apprentice;

G-Force

4: Rise of the Silver Surfer

The Time Machine

The Scorpion King

Pirates of the Caribbean: At World's End

Spider-Man 3

From this list, it's easy to detect that the recommendations are now more diversified, meaning that serendipity is a positive factor present in the new items, which could be similar because of the genres, the cast, or the director.

Now, onto the collaborative filtering methodology. User based filtering implies that products that similar users liked are recommended to others. Again, cosine similarity or Pearson correlation can be used to measure how similar two users are. Item based collaborative filtering, on the other hand, recommends items based on their similarity with the items that the target user rated.

The collaborative filtering methodology has two main issues, namely scalability and sparsity, as mentioned in the previous sections.

To tackle these problems, one can leverage a latent factor model. Latent Factor models are a state of the art methodology for model-based collaborative filtering. The basic assumption is that there exist an unknown low-dimensional representation of users and items where user-item affinity can be modeled accurately. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations to the user-item matrix, where weights are used to hold out missing entries. There is a large family of matrix factorization models based on choice of loss function to measure approximation quality, regularization terms to avoid overfitting, and other domain-dependent formulations.

By doing this, the recommendation problem turns into an optimization one. One common metric in this case is the RMSE. (the lower the RMSE, the better the performance). SVD decreases the dimension of the utility matrix by extracting the latent factors. Essentially, each user and item is mapped into a latent space with dimension r. Below, an explanation of how SVD operates. Singular Value Decomposition is an operation performed on a matrix: in particular, it factorizes a matrix into the product of three matrices $A = UDV^T$ where the columns of U and V are orthonormal - a real square matrix whose columns follow the property $A^TA = AA^T = I$, where A^T is the transpose of A and I is the identity matrix. Let **A** be an nxd matrix with singular vectors $v_1, v_2, ..., v_r$ and corresponding singular values $\sigma_1, \sigma_2, ..., \sigma_r$. Then $u_i = \frac{1}{\sigma_i} A v_i$ for $i = 1, 2, 3, ..., r$, are the left singular vectors and, by theorem, A can be decomposed into a sum of rank one matrices as $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$. [39]

Theorem 1 *Let A be an n x d matrix with right singular vectors $v_1, v_2, ..., v_r$, left singular vectors $u_1, u_2, ..., u_r$, and corresponding singular values $\sigma_1, \sigma_2, ..., \sigma_r$.*

Then $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$.

In recommender systems, as well as other machine learning problems, the objective of the SVD algorithm is used to reduce the dimensionality of the problem. Especially in the recommender systems case, sparsity is an issue one has to face when coding these systems. In terms of the implementation, it's used as a collaborative filtering technique: it uses a matrix structure in which each row represents a user, and each column represents an item. The elements of the matrix are the ratings given by a certain user to a certain item, in this case movies.

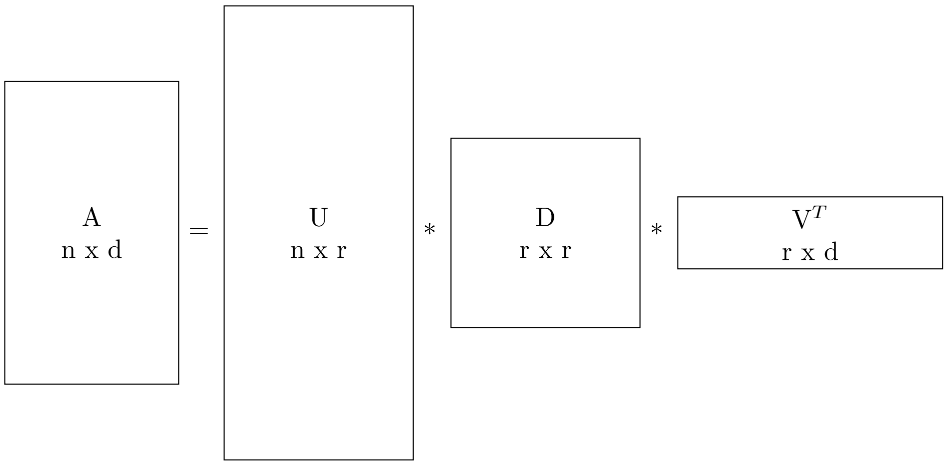


Figure 15: SVD decomposition

The factorisation of this matrix is done by the singular value decomposition. It finds factors of matrices from the factorisation of a high-level (user-item-rating) matrix. The singular value decomposition is a method of decomposing a matrix into three other matrices, Where A is a mxn utility matrix, U is a mxr orthogonal left singular matrix, which represents the relationship between users and latent factors, S is a rxr diagonal matrix, which describes the strength of each latent factor and V is a rxn diagonal right singular matrix, which indicates the similarity between items and latent factors. The latent factors here are the characteristics of the items, for example, the genre of the movie, or its plot. The SVD decreases the dimension of the utility matrix A by extracting its latent factors. It maps each user and each item into a r-dimensional latent space. This mapping facilitates a clear representation of relationships between users and items.
If the matrix A is rank r, than we can prove that the matrices A and A^T are both rank r. In singular value decomposition (the reduced SVD), the columns of matrix U are eigenvectors of $A * A^T$and the rows of matrix V$\hat{T}$ are eigenvectors of $A^T A$. What's interesting is that $A * A^T$ and $A^T * A$ are potentially in different size (because matrix can be non-square shape), but they have the same set of eigenvalues, which are the square of values on the diagonal of Σ. This is why the result of singular value decomposition can reveal a lot about the matrix A. Let's say that a person collects data about some movie reviews such that movies are columns and people are rows, and the entries are the ratings that a person gave to a film. In that case, $A * A^T$ would be a table of person-to-person in which the entries would mean that the the sum of the ratings one person gave match with another one. Similarly, there would be a table of movie-to-movie items in which the entries are the sum of the ratings a movie received matched with that received by another film. What can be the hidden connection between people and movies? That could be the genre, or the director, or something of

similar nature, but the actual connection is enclosed in a number assigned by the algorithm.
Note that, in the decomposition $A = U\Sigma V$ we know that the rows of U are the users and columns of V^T are movies, we cannot identify what are the meanings of the columns of U or rows of V^T (an equivalently, that of Σ). We know they could be genres, for example, that provide some underlying connections between the users and the films but we cannot be sure what exactly are they. However, this does not stop us from using them as features in our recommendation system, given that ultimately the machine learning algorithm will reconstruct the link between a user and an item.

5 Conclusions

6 Acknowledgements

References

[1] NVDIA. *What is a Recommendation System?* URL: https://www.nvidia.com/en-us/glossary/data-science/recommendation-system/.

[2] Paul Resnick and Hal R. Varian. "Recommender Systems". In: *Commun. ACM* 40.3 (1997), pp. 56–58. ISSN: 0001-0782. DOI: 10.1145/245108.245121. URL: https://doi.org/10.1145/245108.245121.

[3] Ian MacKenzie, Chris Meyer, and Steve Noble. *How retailers can keep up with consumers.* URL: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

[4] Netflix. *Research areas: Recommendations.* URL: https://research.netflix.com/research-area/recommendations.

[5] Netflix. *How Netflix's Recommendations System Works.* URL: https://help.netflix.com/en/node/100639.

[6] Michael D. Ekstrand and Joseph A. Konstan. *Recommender Systems Notation Proposed Common Notation for Teaching and Research.* URL: https://md.ekstrandom.net/static/pubs/recsys-notation.pdf.

[7] Sarika Jain et al. "Trends, problems and solutions of recommender system". In: (May 2015). DOI: 10.1109/CCAA.2015.7148534.

[8] Poonam B Thorat, Rajeshwari M Goudar, and Sunita Barve. "Survey on collaborative filtering, content-based filtering and hybrid recommendation system". In: *International Journal of Computer Applications* 110.4 (2015), pp. 31–36.

[9] Reza Jafari Ziarani and Reza Ravanmehr. "Serendipity in recommender systems: a systematic literature review". In: *Journal of Computer Science and Technology* 36 (2021), pp. 375–396.

[10] *Biases search in recommender systems.* URL: https://www.searchenginejournal.com/biases-search-recommender-systems/339319/.

[11] Andrew I. Schein et al. "Methods and Metrics for Cold-Start Recommendations". In: *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2002). DOI: 10.1145/564376.564421. URL: https://doi.org/10.1145/564376.564421.

[12] Chris Anderson. "10. The Long Tail". In: *The Social Media Reader* (2012). DOI: doi:10.18574/nyu/9780814763025.003.0014. URL: https://doi.org/10.18574/nyu/9780814763025.003.0014.

[13] Yoon-Joo Park and Alexander Tuzhilin. "The Long Tail of Recommender Systems and How to Leverage It". In: *RecSys'08* (2008). URL: https://pages.stern.nyu.edu/~atuzhili/pdf/Park-Tuzhilin-RecSys08-final.pdf.

[14] Jesús Bobadilla et al. “A collaborative filtering approach to mitigate the new user cold start problem”. In: *Knowledge-Based Systems* 26 (2012), pp. 225–238. ISSN: 0950-7051. DOI: `https://doi.org/10.1016/j.knosys.2011.07.021`. URL: `https://www.sciencedirect.com/science/article/pii/S0950705111001882`.

[15] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. “Facing the cold start problem in recommender systems”. In: *Expert Systems with Applications* 41.4, Part 2 (2014), pp. 2065–2073. ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2013.09.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0957417413007240`.

[16] Deuk-Hee Park et al. “A Literature Review and Classification of Recommender Systems on Academic Journals”. In: *Journal of Intelligence and Information Systems* 17 (Jan. 2011).

[17] G. Adomavicius and A. Tuzhilin. “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions”. In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), pp. 734–749. DOI: `10.1109/TKDE.2005.99`.

[18] Paul Resnick and Hal R Varian. “Recommender systems”. In: *Communications of the ACM* 40.3 (1997), pp. 56–58.

[19] J Ben Schafer, Joseph Konstan, and John Riedl. “Recommender systems in e-commerce”. In: (1999), pp. 158–166.

[20] Robin Burke. “Hybrid web recommender systems”. In: *The adaptive web: methods and strategies of web personalization* (2007), pp. 377–408.

[21] Cai-Nicolas Ziegler. “Semantic web recommender systems”. In: (2004), pp. 78–89.

[22] Michael J Pazzani and Daniel Billsus. “Content-based recommendation systems”. In: *The adaptive web: methods and strategies of web personalization*. Springer, 2007, pp. 325–341.

[23] J Ben Schafer et al. “Collaborative filtering recommender systems”. In: (2007), pp. 291–324.

[24] John S Breese, David Heckerman, and Carl Kadie. “Empirical analysis of predictive algorithms for collaborative filtering”. In: *arXiv preprint arXiv:1301.7363* (2013).

[25] David Goldberg et al. “Using collaborative filtering to weave an information tapestry”. In: *Communications of the ACM* 35.12 (1992), pp. 61–70.

[26] Barry Smyth. “Case-Based Recommendation”. In: 4321 (Jan. 2007), pp. 342–376. DOI: `10.1007/978-3-540-72079-9_11`.

[27] Yi Zhang, Jamie Callan, and Thomas Minka. “Novelty and redundancy detection in adaptive filtering”. In: (2002), pp. 81–88.

[28] Nishith Pathak et al. “Social topic models for community extraction”. In: (2008).

[29] Santo Fortunato. "Community detection in graphs". In: *Physics reports* 486.3-5 (2010), pp. 75–174.

[30] J Ben Schafer et al. "Collaborative filtering recommender systems". In: (2007), pp. 291–324.

[31] Analytics Vidhya. *Contextual multi-armed bandit — (Intuition behind Netflix Artwork Recommendation)*. URL: `https://medium.com/analytics-vidhya/contextual-multi-armed-bandit-intuition-behind-netflix-artwork-recommendation-b221a983c1cb`.

[32] MathWorks. *What is Feature Extraction?* URL: `https://www.mathworks.com/discovery/feature-extraction.html`.

[33] KNIME. *Convolutional Neural Networks & Computer Vision*. URL: `https://www.knime.com/blog/convolutional-neural-networks-computer-vision`.

[34] Wikipedia. *Root-mean-square-deviation*. URL: `https://en.wikipedia.org/wiki/Root-mean-square_deviation`.

[35] Rob J. Hyndman and Anne B. Koehler. "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22.4 (2006), pp. 679–688. ISSN: 0169-2070. DOI: `https://doi.org/10.1016/j.ijforecast.2006.03.001`. URL: `https://www.sciencedirect.com/science/article/pii/S0169207006000239`.

[36] Wikipedia. *Mean absolute error*. URL: `https://en.wikipedia.org/wiki/Mean_absolute_error#cite_note-:0-1`.

[37] Cort J. Willmott and Kenji Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". In: *Climate Research* (2005). DOI: `10.3354/cr030079`. URL: `https://www.int-res.com/abstracts/cr/v30/n1/p79-82/`.

[38] IMDb. *Weighted Average Ratings*. URL: `https://help.imdb.com/article/imdb/track-movies-tv/weighted-average-ratings/GWT2DSBYVT2F25SK?ref_=ttrt_wtavg#`.

[39] Reza Bagheri. *Decomposition and its Application in Data Science*. URL: `https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d`.