SINGIDUNUM UNIVERSITY

POSTGRADUATE STUDIES DEPARTMENT

COLLABORATION AND CONTENT-BASED RECOMMENDER SYSTEMS EVALUATION

MASTER THESIS

Mentor:

Prof. dr Milan Milosavljević

Student:

Mihailo Joksimović

Belgrade, 2019.

Table of Contents

1 Abstract

The idea of this thesis is to implement and evaluate multiple recommender system categories against the publicly available MovieLens dataset. Primary objective is to see how using various implementations and algorithms affects the recommendations and end-user experience.

We will be using both the Collaborative filtering algorithms (both neighborhood and model-based) and Content-based recommender system implementations.

For evaluation metrics, the main focus will be on comparison between predicted and actual outcomes. Nonetheless, an attempt will be made in order to evaluate predicted users experience.

The outcome of this thesis should be a clear overview of which implementation brings the best results and what steps could be taken to further enhance the outcome.

2 Recommender Systems

2.1 Introduction

Recommender systems have evolved as a natural response to ever growing amount of information available to wide audiences of users. With the rise of Internet and general availability of it to common users, it has become necessary to help users navigate the content and steer them towards the new options they might have not anticipated in the first place.

Strictly technically speaking, Recommender systems represent a subclass of Information filtering systems whose main purpose is to predict the “ratings” or “preferences” a user would give to an item [1].

2.2 History

There does not seem to be an official information on when the recommender systems have been mentioned for the first time.

One of the earliest mentions of Recommender systems as such dates to early 1990’s where the official term has been mentioned in a technical report written by Jussi Karlgren at Columbia University [2].

Since then, especially with the rise of Cloud computing and Big data, when processing of vast amounts of data became possible, recommender systems have become an important aspect and indispensable commodity of all successful businesses.

It is especially important to mention the efforts of GroupLens working group [3]. GroupLens is a group of scientists from the department of computer science and engineering in University of Minnesota. This group has pushed forward the efforts towards both publishing various research papers on recommender systems and publishing the freely available datasets to be analyzed and used for educational and research purposes. Some of their featured projects include MovieLens – a web site that helps people find movies to watch, Cyclopath – an editable map where anyone can find maps and routes for riding bicycle and LensKit – open source toolkit for building, researching and studying recommender systems. MovieLens is also the dataset that this thesis is using for evaluation of results.

2.3 Overview

It has been stated before that one of the main driving factors for introduction of recommender systems was development of Internet and World wide web. Once the data became publicly available and many users gained access to it, there was a need to help users browse the content and steer their attention towards the items that are curated for them.

Another highly important catalyst that has driven this development is the ease with which users are able to express their preferences. Namely, today, users are able to demonstrate their liking or disliking of a certain product by a single click of mouse. This one second of users feedback multiplied by number of users and amount of items they interact with, results in enormous volumes of data which are then used to recommend even more fine-grained items. This loop repeats itself.

Charu C. Aggarwal in his Recommender systems book [4] takes a Netflix [5] as an example. He suggests to take content providers, such as Netflix [5], as an example. *"In such cases"*, he further states, *"users are able to easily provide feedback with a simple click of a mouse. A typical methodology to provide feedback is in the form of ratings, in which users select numerical values from a specific evaluation system (e.g., five-star rating system) that specify their likes and dislikes of various items."*.

The aforementioned approaches for feedback collection are usually referred to as "Explicit ratings" or "Explicit feedback". This naming stems from the fact that the rating or the feedback of the item being recommended was explicitly specified by user. This also means that user is most likely consent to share his preferences with the service provider and is looking forward towards getting more curated content.

In contrast to explicit ratings, there is a group of so called "Implicit ratings" or "Implicit feedback". Namely, this is the kind of feedback that can be derived based on the actions and behavior of user in question. Perfect example of such scenario is Amazon.com. As Aggarwal [4] states, a simple act of a user buying or browsing an item may be viewed as an endorsement for that item. Another example would be a YouTube [6]. Having a user watch the video from beginning to an end is a perfect example of a positive feedback where users expresses his interest towards the item being watched. On the other hand, user skimming through the video being played is an indirect act of providing negative feedback. This negative feedback should be treated with caution as it can happen that user is not interested only at that moment, but might be looking forward to interacting with same item on another occasion. It is up to the designer of system to make sure that the feedback is evaluated properly and all relevant things are being taken into consideration.

3 Basic principles

Before starting to dig into the basic principles of recommender systems, it is rather important to introduce a basic terminology that will be used throughout this thesis.

Broadly speaking, the entity to whom the recommendation is being made, and based on whose feedback is the decision being based on, is called *user*. User does not have to, necessarily be a human, but can really be any entity that is interacting with system that we are currently predicting the recommendations against.

Product that is being recommended to aforementioned user is called *item*. As mentioned above, what is said for users is also valid for items – this does not have to be a physical item, but can rather be a content, commodity or any other type of service being offered by system in question.

Broadly speaking, no matter which type of recommender system are we referring to, Aggarwal [4] identifies two distinct categories:

1. Prediction version of problem – in this version, we are dealing with incomplete *m x n* matrix, where rows of the matrix *m* represent users, and *n* columns represent the items. This matrix is incomplete because not all users have specified ratings for all items. The goal here is to find the best-fitting values that would make this matrix complete. This problem is usually referred to as *matrix completion problem*.
2. Ranking version of problem – in contrast to prediction, ranking problems are usually concerned with selection of top-k items that user might be interested in. This sort of systems is usually found in e-commerce websites and online shops.

Whichever category is being used, recommender systems are usually implemented in order to increase the user's engagement with the service provider, with the end result of increasing the sales and overall profit.

3.1 Beyond accuracy

Accuracy of the recommendation system is one of the most important evaluation metrics. Bad or inaccurate recommendations would surely lead to user's dissatisfaction and lost profit.

However, even though it is one of the most important metrics, there are others non-directly observable, which have a strong influence on quality of recommendations.

Kaminskas et al. [7] mention the following important metrics:

1. *Relevance* – as mentioned before, this is, indeed, one of the most important metrics that has to be measured. What's more, having training and test sets, this is one of the easiest metrics to be evaluated. However, it's not important to treat this in isolation and influence of other metrics is important as well
2. Coverage – refers to a degree to which the recommender system covers the specter of available items. The more the available items are included in recommendations, the higher the coverage is.
3. Diversity – if recommender system keeps recommending the most popular items, user might either get overwhelmed or stop liking them, and the risk of negative feedback increases. Diversity refers to a degree on to which the recommender system is able to break apart common recommendations, while still being able to suggest relevant items to the user.
4. *Novelty* – this is an important metric that refers to recommending something that user hasn't seen or experienced in the past
5. *Serendipity* – even though it sounds similar to novelty, this is a metric that, on certain occasions may have even stronger influence than novelty. While the former one refers to a content that user hasn't seen before, it is still somewhat expected. Serendipity, on the other hand, refers to a content that user might even consider as being lucky to have found. This metric is usually related to, so called "latent interests", which user might not even have been aware of in the first place.
   We'll use Youtube [6] as an example here. Let's consider user who mostly enjoys the techno music and occasionally enjoys watching travel channels and videos with nature. Recommending new DJs or new travel videos that he hasn't seen before can be considered as *novelty.* However, combining those two interests and recommending a video with DJ playing at a concert in the nature, would be truly serendipitous experience. On the other hand, as Aggarwal [4] states, serendipitous algorithms often end up recommending irrelevant content. Still, long term benefits of such experiences seem to outweigh the short-term disadvantages.

It is sufficient to say that, with the exception of relevance, all other factors are harder to measure and evaluate, as they are mostly related to users experience and interaction with the system.

Nevertheless, the attention should be put towards trying to increase all of the mentioned metrics.

In the end, it is important to mention that not all services and recommender systems are inclined towards increasing the sales and overall revenue. Instead, there are instances where recommendations are used with sole purpose of increasing users engagement with a product. Let’s take Facebook.com as an example. Facebook uses the recommender systems for recommending new friends and groups to the user, which, in turn, directly increases users engagement with the product. This engagement results in user spending more time on the website and more ads being displayed, which finally leads to more revenue. As can be seen, this is an example where recommender system is not directly driving the revenue, but rather passively, by increasing users engagement.

3.1.1 Netflix Prize

Netflix [5] is one of the biggest and most popular on-line movie subscription rental services. In October, 2006, in order to improve their recommender systems, they opened the Netflix Prize contest. As part of this contest, they released a dataset containing 100 million movie ratings that were previously anonymized. The challenge was to build the most reliable rating prediction algorithm, which challenged experts from data mining, computer science and machine learning universes. The promised award was $1 million USD.

Up to that date, Netflix was using a Cinematech recommendation system [8]. This recommendation system automatically analyzes the accumulated movie ratings on a weekly basis using a variant of Pearson’s correlation. After the user provides his rating for the movie he just watched, system computes a multivariate regression based on these correlations to determine a unique, personalized prediction for each predictable movie based on those ratings [8].

When the context started, the dataset was released that contained 95.91% of all the ratings they had in their system, up to that date. Another 1.36% was used as a validation set. This validation set wasn’t revealed publicly, but was used in order to build the publicly available ratings boards where users could see how their algorithms ranked against other competitors. The remaining ~ 3% of the test was never published and was to be used as a final evaluation for the winner solution.

The contest itself as well as it's solutions have been heavily studied in [8] [9] [10]. The main idea of the contest was to build a system that could beat the accuracy of Cinematech one. The accuracy metric that was used for validation was Root mean squared error (RMSE) [11].

Winner of the contest was the team called *Bellkor's Pragmatic Chaos*, which was a team combined of multiple participating teams who joined together in order to provide a solution for the final part of the contest. Their solution which was thoroughly discussed in Töscher et al. [12] and relies on matrix factorization techniques combined with three years of work and research on the dataset.

Interestingly enough, even though the winning algorithm has beat the Cinematech's accuracy by 10.10%, it was never reported to be put to use by Netflix.

3.1.2 Recommender systems and software engineering

One of the interesting areas that is actively being researched is using recommender systems in software engineering [13]. Specifically, recommender systems are capable of providing stakeholders with support throughout the whole software development process. Some of the examples include but are not limited to – *recommendation of methodological knowledge* [14, 15], *recommendation of requirements* [16] and the *recommendation of code* [17].

Recommendation of methodological knowledge – appropriate choice of methodology is usually the crucial step in ensuring the successfulness of the project. As an example, waterfall process is applicable only for risk-free types of projects, but is not applicable for high risk one. These systems usually rely on knowledge-based or content-based systems, since there usually are well-defined rules and criteria that needs to be met in order to ensure the appropriate method selection.

Recommendation of code – due to frequently changing development teams, the intelligent support for discovering, locating, and understanding code is crucial for efficient software development [13]. Some of the example use-cases are code recommendations, recommending relevant code fragments and tailoring the displayed software artifacts to the current development context.

Recommendation of requirements – Poor requirements engineering is the most usual reason that leads to failure of software projects [18]. This makes the requirements engineering one of the most complicated steps in the software development. Activities involved in this process are *elicitation & definition, quality assurance, negotiation, and release planning*. Recommender systems can be used in all of these activities. For example, the recommendation of requirements to stakeholders working on similar requirements. Content-based systems are usually used for these purposes.

As a final remark, one interesting application is *use of recommender systems for persuasive technologies*. Persuasive technologies [19] aim to trigger changes in a user's attitudes and behavior on the basis of the concepts of human computer interaction. The impact of persuasive technologies can be significantly increased by additionally integrating recommendation technologies into the design of persuasive systems. Such an approach moves persuasive technologies forward from a one-size-fits all approach to a personalized environment where user-specific circumstances are taken into account when generating persuasive messages. By creating *Persuasive Software Development Environments*, we might be able to persuade the programmers to write cleaner and better code by suggesting the improvements that could be made or suggesting reuse of code fragments or patterns that were used previously. Pribik et al. [20] introduce such an environment which has been implemented as an Eclipse plugin [21].

3.1.3 Future of Recommender systems

Amatriain et al. did a thorough study [22] of where recommender systems are now and what the potential future is. Netflix prize competition launched in 2006 seems to have been a turning point in this area. Namely, before this competition, not much research and interest was involved in experimenting and developing the recommender systems. However, once the competition launched with it's generous USD $1 million award, it not only started a revolution but seems to have put the wheel of research in the moving motion.

Currently, many of the online services rely on usage of these systems, ranging from Netflix [5] to online web shops. It has been noticed that, broadly speaking, everything can be personalized and made a recommendation. Netflix uses them to recommend movies to watch, Quora [23] uses them to make personalized feeds, Facebook [24] and Twitter [25] use them for making personalized timelines and for suggesting new people to follow.

When it comes to the future of these systems, some of the interesting directions [22] seem to be:

1. *Full page optimizations* – making home pages fully personalized towards the users liking. Optimizing the diversity and freshness is another potential direction. While some users prefer seeing the new and popular content, others might have different

personalized preferences. Taking all this into account is one possible area that needs to be explored and experimented with.

2. *Personalizing how we recommend* – aside from personalizing the content itself, the way the content is presented to the user is important as well. For example, in the Netflix case, it could mean deciding whether or not to show explanations, reviews, average ratings, or whether to play the trailers or not.
3. *Indirect Feedback* – even though most of the time the explicit feedback provided by user is being used, the power of implicit ratings has yet to be leveraged properly. For example, having user read the description of the movie of specific genre, and doing so for a number of them is a form of implicit feedback that tells us that user might have a preference for given genre. One of the shortcomings of this approach is that we are only able to record the positive feedback, but capturing the negative one is problematic. Some of the solutions have been proposed, such as Collaborative Competitive Filtering, which leverages the info of whether the given item was even displayed to the user.
4. *Value-aware recommendations* – even though the main goal of recommender systems is to recommend the content that is relevant to the user, from perspective of the retailer or service owner, items that have similar relevance to the user might have marginally different value to the owner. For example, the item that is more expensive should generally be preferred over the cheaper one, even though the both have same relevance for the end user. Special care also needs to be taken for algorithms not to become biased against certain kinds of users, for example not reinforcing prejudices. As such, it is important to keep user's values in mind.

3.2 Approaches

3.2.1 Basic models

Generally speaking, there are roughly two distinct kinds of recommender systems – ones that are working with incomplete matrices of users and ratings and ones that are modeling against the attributes of the item being recommended. Former ones are referred to as *Collaborative Filtering* methods, because other users collaborate in order to derive the predicted rating for item being rated. The latter ones are usually referred to as *Content-based filtering* methods because the attributes of the item itself are what drives the recommendation. Content-based filtering methods are usually user-centric as the predictor is built for every single user and usually considers users previous behavior mixed with item's attributes. There is also a third category of recommender systems which is known as *Knowledge-based* systems. These are, as the name suggests, systems that are based solely on *user's requirements* which are explicitly specified by user before the recommendation process is started at all. Finally, there is a group of so called *hybrid systems* which combine multiple approaches in order to derive the best possible rating for user in question.

Some examples of Collaborative-filtering methods include popular services like Netflix.com, Youtube [6] and Spotify [26]. These are all the services which use ratings of other users in order to drive predictions for the current user. When it comes to Content-based filtering, a perfect example would be a movie recommendation service that learns their users preferences based on their previous feedback. Such an example is a MovieLens project, a movie recommendation service developed by GroupLens research group [3].

3.2.2 Collaborative filtering

Collaborative filtering represents a family of recommender systems where recommendations are solely based on so called "collaboration" between users or items. Therefore, the predicted recommendations are based solely on how other users have rated the same item (in case of user-based collaborative algorithms) or how the majority of items was rated (in case of item-based collaborative algorithms). Main challenge with collaborative filtering algorithms seems to be the fact that underlying rating matrices are usually incomplete [4].

We'll take an example of a Movie recommendation engine. In systems like this one, users usually express their liking or disliking of a particular movie, or, in some cases, specify the numerical rating. The main problem that collaborative filtering algorithms suffer from is the fact that, out of the whole universe of available movies, only a handful of them is actually rated by users. The remaining majority is usually either sparsely rated or not rated at all.

The basic premise of collaborative filtering algorithms is that the missing ratings can be imputed by observing the ratings of similar users or products, depending on what is being observed. What's more, one of the main premises that drives these engines is the assumption that if two users have a similar taste and have similarly rated number of items, then, they must have the similar taste and one can infer the ratings from the other.

Let's take as an example two users Alice and Bob. We will assume that both Alice and Bob have a similar taste in movies and we conclude this by observing the ratings they provided for the same set of items. Let's further assume that Bob is looking for a new movie to watch.

Supposing that they have a similar taste and that Alice has liked Terminator movie which Bob hasn't rated yet, we can assume that Bob might be interested in watching Terminator as well.

These algorithms, unfortunately, suffer from so called *cold start* problem. This topic which will be discussed in greater length, generally refers to the case where there are either no previous ratings to suggest, which is a case in newly built systems, or to a case when a new item is added which was never rated before. This matter will be discussed at greater length in later text.

3.2.2.1 Types of Collaborative Filtering systems

There are two distinctively different categories of Collaborative filtering systems [4]:

1. *Memory-based methods* and
2. *Model-based methods*

Memory-based methods, also referred to as "lazy" methods are the simplest and one of the earliest experimented with and implemented methods of recommendation systems. Their process of working is relatively simple – for any given user, find the set of users who appear to be similar and make recommendations based on what other users have liked.

These can be further split into two categories:

1. *User-based collaborative filtering* and
2. *Item-based collaborative filtering*

In User-based systems, the idea is to find the like-minded users and base the recommendations on that. As mentioned in previous chapter, if two users, Alice and Bob, have like the same movies in the past, then we can assume that movies that were positively rated by Alice but have not been rated by Bob are going to be positively perceived by him as well. Generally speaking, the underlying assumption is that top-k users who are most similar to Bob can actually predict the movies that Bob will like.

In Item-based systems, the idea is generally similar. In order to predict item B to user A, we first need to find a set of top-k items that are most similar to B and based on that we can conclude and infer the rating of user A for item B.

For example, let's assume that Bob has expressed positive liking towards the movies Thor, Avengers and Captain America. We can make a conclusion that Bob likes Marvel's movies and, based on that, we look for movies that are similar and one of the movies that can be recommended is Guardians of the Galaxy.

As can be seen, both user-based and item-based methods work on one-by-one basis, and that is exactly why they are also referred to as being “lazy”. This comes from the fact that the ratings are known only at the time of evaluation and not before that.

The advantages of memory-based techniques is that they are very simple to implement and resulting recommendations are often very easy to explain [4].

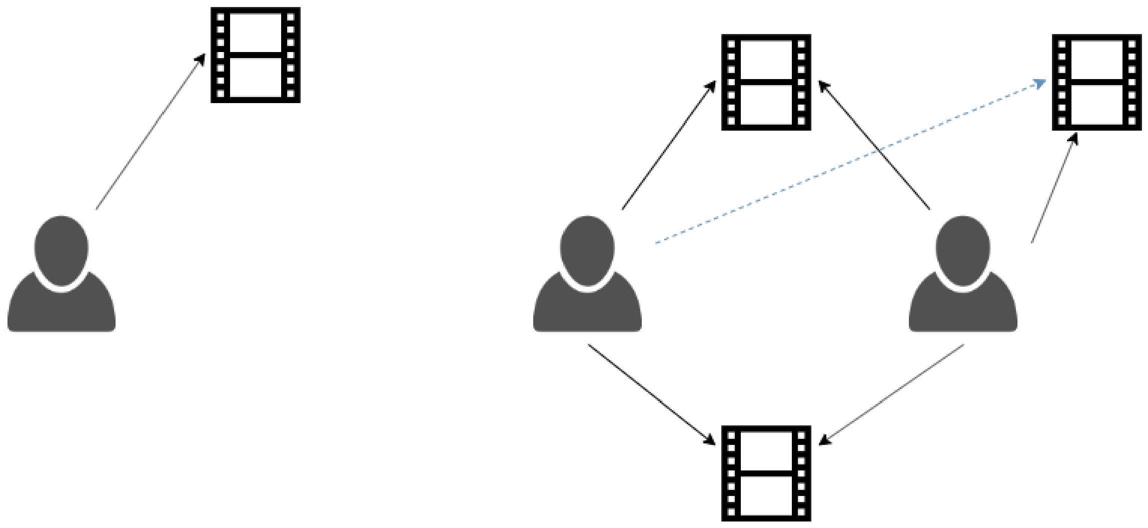
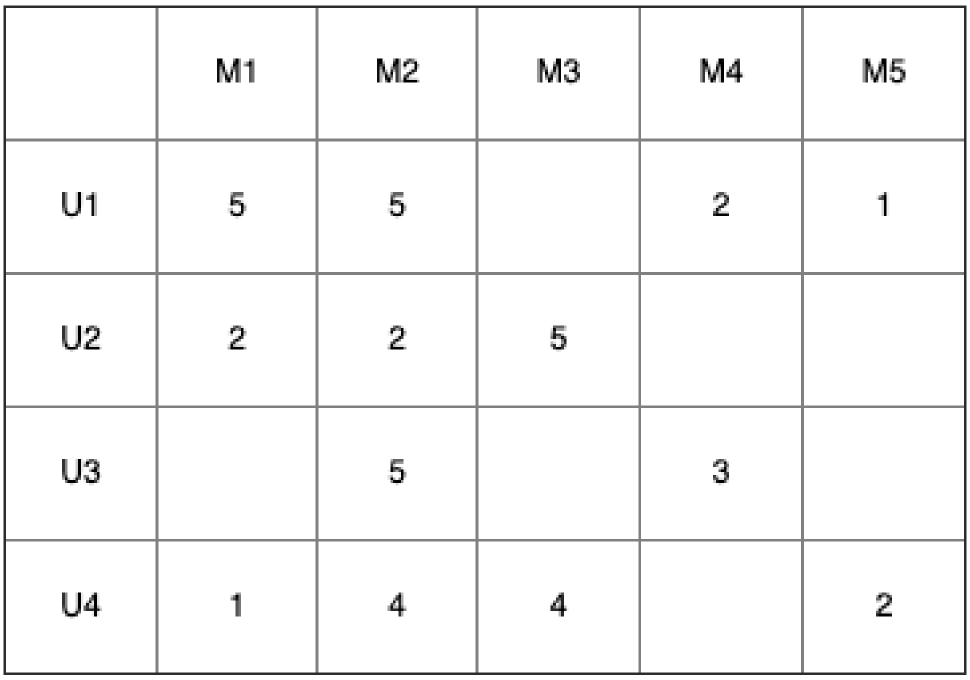


Figure 1 – User 2 and User 3 like the same items and item 4 being recommended based on that

In the model-based filtering systems, the emphasis is on using the prediction algorithms in order to fill-in the missing ratings. Some of the example algorithms that are usually used are decision trees, rule-based models, Bayesian methods and latent factor models [4].

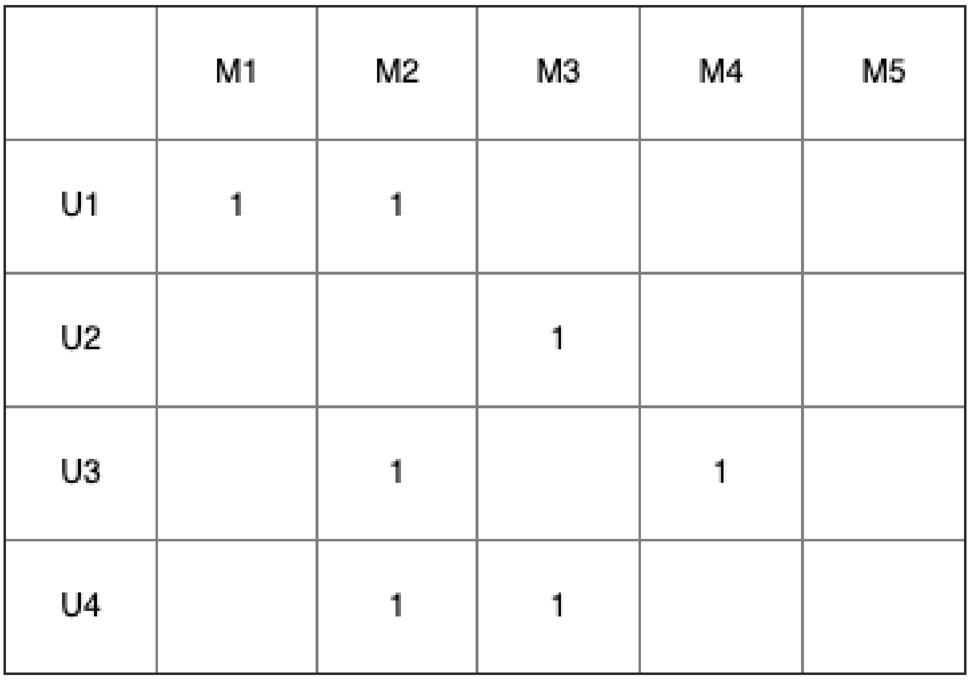
Bayesian networks allow creation of models within a matter of hours or days and as such are useful when big datasets are present. Generally speaking, these models are perfect for environments where user’s preferences are slowly changing over time. In these circumstances, they can be pretty fast and almost as precise as neighborhood-based methods [27]. On the other hand, in fast-changing environments, alternate algorithms might be better suited.

Another possible approach is by using the clustering techniques. These algorithms work by identifying clusters of similar users and use those clusters for identifying related items to be recommended. Even though they have great performances in terms of recommendation speed, they suffer from lack of personalization and are shown to have worse accuracy than nearest-neighbor algorithms [28].



	M1	M2	M3	M4	M5
U1	5	5		2	1
U2	2	2	5		
U3		5		3	
U4	1	4	4		2

Figure 2 - Example of an incomplete rating matrix



	M1	M2	M3	M4	M5
U1	1	1			
U2			1		
U3		1		1	
U4		1	1		

Figure 3 - unary ratings matrix

Matrices displayed in Figure 1 and 2 represent a typical setup for a collaborative filtering method. Rows (U1, U2, …, Un) represent Users, while columns (M1, M2, …, Mn) represent movies. The goal is to predict the missing ratings by either using neighborhood-based or model-based methods.

In case of unary ratings, it is often recommended to do the analysis by treating the missing ratings as 0s [29].

3.2.2.2 Types of ratings

Design of recommender systems is usually influenced by the type of ratings that are being used. Most-commonly used rating scheme is one used by Netflix [5], Youtube [6] and others, which offers a rating from the following set {1, 2, 3, 5}. 1 and 2 are usually considered as negative, while 3, 4 and 5 are usually treated as positive feedback. There are also examples of system using a three-star ratings, where 1 represents a dislike, 2 is neutral and 3 expresses liking towards the particular item.

Aforementioned rating systems are referred to as *interval ratings*. There are also examples of where ratings are provided in terms of ordered categorical variables, such as {Strongly disagree, Disagree, Neutral, Agree, Strongly agree}. These types are usually referred to as *ordinal ratings*.

Unary ratings, where feedback is provided in terms of “Like” or “Dislike” are particularly common in the case of *implicit feedback data sets* [30] [29] [31].

Explicit vs Implicit ratings

Ratings that are intentionally specified by the user himself are called *explicit*. This is because user was determined and explicit in his intention to provide a feedback for a specific item. As mentioned above, these can be either numerical, binary or unary.

In contrast to explicit ones, ratings can also be *implicit*. These types of ratings are usually in the unary form.

For example, in the example of a web shop, user adding the item to the shopping cart can be treated as a positive feedback towards the given item. Also, on a movie website, user spending some time to read the description of the movie, check the director, actors and other type of information can also be treated as a positively inclined rating.

The power of implicit ratings lies in the fact that number of users actions and behaviors can be converted into a rating that can further be used for producing tailor-made content for given user.

3.2.3 Content-based recommender systems

Content-based systems represent a group of Collaborative filtering systems. In contrast to aforementioned mentioned algorithms, they don't rely on ratings of other users. Instead, they rely solely on *content* that describes the item being evaluated. These systems combine behavior of users and items content in order to predict the likelihood of users being satisfied with the predicted item.

Let's take as an example user who watched *The Lord of the Rings: The Fellowship of the Ring (2001)* and liked it. Let's also assume that ratings of other users are not available, which effectively rules out the aforementioned collaboration filtering algorithms. In this scenario, we can rely on content that describes the movie itself. Specifically, for the movie in question, one of the possible content descriptions would be it's genres – *Adventure, Drama, Fantasy* [32]. We could also use the name of the movie as a way to describe it, as *Lord of the Rings* is actually a name of the trilogy. We could further extend this and add a flag on whether it's based on a book or not. This could proceed forever, but should carefully be evaluated because adding too many features could worsen the quality of predictions [33].

Using the solely it's genre and movie title, we can quickly generate list of predictions that are similar to the target one: *The Lord of the Rings: The Two Towers (2002), The Lord of the Rings: The Return of the King (2003)* and *The Matrix (1999)*.

As can be seen, we relied solely on the content that describes the target item and generated predictions based on that.

Content-based systems can be built in two different ways – either as a solely collaborative filtering technique where the item's characteristics are used as primary recommendation driver. The alternative possibility is creating a user-specific matrix where columns represent the content characteristics and rows represent user's feedback on each of the movies. Last column can either represent users actual numerical rating or can indicate a positive/negative feedback. Organized this way, we can either user classification or regression algorithms in order to create user-specific models for making predictions.

These systems do have some advantages in making recommendations, especially for new items where sufficient rating data is not available. This is because other items with similar attributes might have been already rated by active user. As such, the supervised model is able to leverage these ratings in conjunction with the item attributes in order to make recommendations even when there is no history of ratings for that item [4].

These systems, however, do have some disadvantages as well:

1. Considering the fact that they mostly rely on keywords, the recommended items tend to be obvious ones. As seen in the example above, first part of the *Lord of the Rings* would surely recommend second part. The problem is that this behavior tends to produce a narrow set of recommendations and actually reduces the diversity and serendipity, which are rather important factors to consider [34] [35] [36].
2. Even though they are pretty good for providing recommendations for *new items*, they tend to be rather bad when providing content for *new users*. This is due to the fact that these systems usually require history of user's previous behavior and ratings in order to make good recommendations.

As such, they generally have different trade-offs to collaborative-filtering ones.

In specific cases, content-based systems can be combined with specific knowledge about a user. For example, user might explicitly express interest in certain movie genres or movie directors, and this knowledge can be leveraged in order to further narrow down and tailor the recommendations. These systems, however, tend to be categorized as *knowledge-based systems* due to the fact that they use the upfront knowledge about a *new user*. They tend to be a great alternative for solving the cold-start problem.

For example, let's consider a new user who has just registered on our website. We do not have any history of his previous ratings. Options that we have at that moment are:

1. Recommending the top-k most popular items. This is generally useful approach because most popular items tend to be liked by general public. The downside is that user might already have interacted with those, which we can leverage again by asking user to provide his feedback for those which further helps us to curate his profile.
2. Another approach is explicitly asking user about his preferences. In case of movie recommender engine, we might ask user to specify which genres he likes the most. Once he does, based on his input, we can further ask him whether he watched some of the top-k most popular movies from each of the genres he specified and then use that knowledge to build his profile.

Both methods provide one of the solutions to the cold start problem by relying solely on the content that describes the item. The cold start problem will be further discussed in a separate chapter.

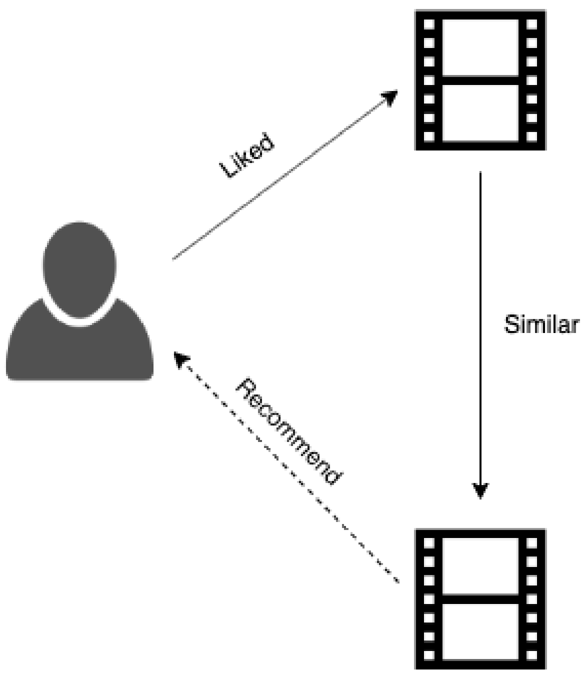


Figure 4 - Content-based recommender systems. If user has liked Movie 1, recommend a movie whose content is similar to it

3.2.3.1 Vectorization techniques

Before making recommendations based on the content that describes the items, we first have to turn the content into a vector. This process is referred to as *vectorization*.

Number of vectorization techniques is available. One that we will be using in our recommender system is called "TF-IDF" vectorizer. *TF* is an abbreviation for *Term Frequency*, while *IDF* refers to *Inverse Document Frequency*. The concepts of *Term Frequency* and *Inverse Document Frequency* originate from information retrieval systems and are widely used in content-based filtering systems and text mining algorithms. These two are used to determine the relative importance of a document that is being searched or evaluated against. In our case, the document is the movie that is being evaluated.

TF represents the frequency of a word in a document. *IDF* is the inverse of the document frequency among the whole corpus of documents. *TF-IDF* combination represents a multiplication of *Term Frequency* and *Inverse Document Frequency*. The end result of the multiplication is that words or tokens that occur frequently across the universe of the documents gain lower score than the words that occur less frequently.

$$tfidf_{i,j} = tf_{i,j} * log\left(\frac{N}{df_i}\right)$$

$$tf_{i,j} = total\ number\ of\ occurences\ of\ term\ i\ in\ document\ j,$$

$$df_i = total\ number\ of\ documents\ containing\ term\ i$$

$$N = total\ number\ of\ documents\ in\ corpus$$

Equation 1 - TF-IDF formula

For example, in our case of movie recommender engine, we can take movie genres as a content that will be vectorized. If we take each genre as a word and calculate the frequencies across all movies, the results that we might end up with is that movies that have a *Sci-fi Drama* genre are far less common than movies that have *Action* in their genre [1]. Therefore, we are safe to assume that if user liked a *Sci-fi Drama* movie, that he is more likely to like another movie that has the same genre combination.

Simply put, TF-IDF adds more weight to less frequent terms that are able to uniquely identify classes of documents.

After calculating the TF-IDF scores, we have successfully converted our text content into a numerical matrix. Next step is similar as in collaboration-filtering models – we calculate the similarities between movies using one of the similarity functions, for example Cosine Similarity. Once we have the similarities, the process is the same as in all other cases.

3.2.4 Latent Factor Models

All previously mentioned models were based on the data at hand, and either used variations data classification algorithms, or used neighborhood based methods in order to determine the predicted values. Those models are usually made to work with sparse matrices. However, there are approaches [37] where the dimensionality reduction techniques can be leveraged in order to create fully-specified datasets. These algorithms are playing only an enabling role of creating a more convenient data representation.

Latent factor models are said to be the state-of-the-art algorithms in recommender systems. The main idea comes from the matrix factorization properties. Namely, it is assumed that rows

[1] These assumptions are not validated. They are taken as an potential example that TF-IDF vectorizer might return.

and columns, or, in case of recommender systems – users and items – are usually highly correlated. This correlation can be exploited by creating a lower-rank representation of the matrix. These methods not only provide the full matrices, but also expose certain latent properties that were not obvious from the full-rank matrix.

Let's take two users as an example – Alice and Bob. Let's further assume that Alice has given a rating of 5 out of 5 starts to *Titanic (1997)* and *The Notebook (2004)* and she gave a rating of 2 stars to movies *The Terminator (1984)* and *The Dark Knight (2008).* Bob, on the other hand has rated *Titanic (1997)* with 1 star, but has rated *The Teminator (1984)* and *The Dark Knigh (2008)* with 5 stars. In this case, the latent properties that would have been discovered are that Alice has a preference for *Romantic Drama* movies, while Bob has a strong preference for *Action Sci-fi* movies. We can further use this knowledge in order to predict the movies for Alice and Bob to watch. What's interesting is that, even though they never explicitly stated their preferences, by running the matrix factorization and breaking down their preferences into basic components, we discovered their *latent features*.

The way these algorithms work is, roughly speaking, by finding the *eigenvectors* and *eigenvalues* of the matrix. Roughly speaking, *eigenvectors* represent the basis vectors of the matrix which can be used to reconstruct all values in the matrix. By decomposing the matrix and taking only top-k *eigenvectors* we have successfully discovered the *latent features* of the matrix.

Most popular approaches used for matrix factorization are *Singular Value Decomposition (SVD) and Principal Component Analysis (PCA)*. The latter one usually refers on former one to find the eigenvectors of the matrix.

Singular Value Decomposition states that every matrix *M* can be represented as a multiplication of three lower-rank matrices:

$$M = U * \Sigma * V^T$$

Equation 2 - Singular Value Decomposition Formula

M represents the original matrix that is being decomposed, *U* represents the *m x m* unitary matrix which holds user coefficients, *Σ* is a diagonal *m x n* matrix used for recreating the original matrix and V^T is a transposed *n x n* unitary matrix that holds item coefficients.

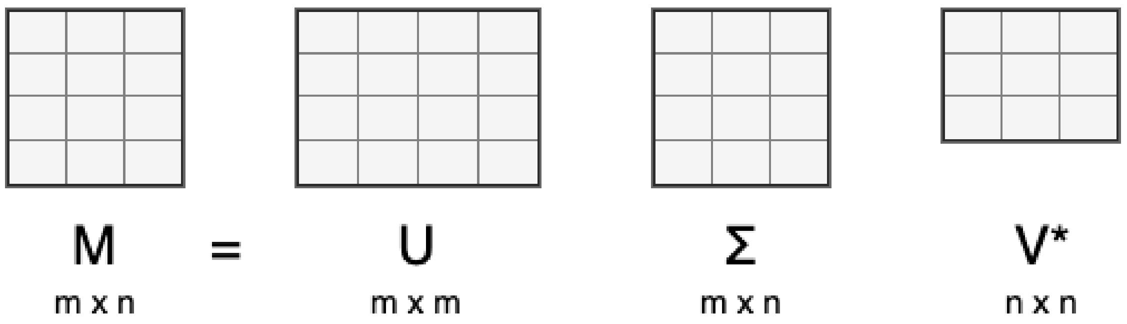


Figure 5 - Singular Value Decomposition of a matrix M

One nice property that SVD has is that it can decompose and put back the full matrix all at once. That means that, no matter how sparse the matrix is, it can be decomposed and recomposed in a single run, providing us with predictions immediately.

It is important to mention that, original SVD approach does not support sparse matrices by itself. Instead, what is usually done is that they are either set to value of 0, or they get set to mean value of rows or columns. The best approach is usually case-specific and therefore, it has to be experimented in order to see which one brings the best results.

One variation of SVD that got popularized during the Netflix Prize competition is SVD approach proposed by *Simon Funk*, usually referred to as *Funk SVD* [38]. His idea took 9th place in the competition. Roughly speaking, what he proposed is to simply ignore the missing value when decomposing the matrix. This can be done by running the error minimization algorithm with the known values only. He later proves that the factorized and predicted values are almost identical as when the fully specified matrix is used.

As such, Funk's algorithm is a great replacement when working with sparse matrices, which usually is the case in recommender systems.

3.2.5 Knowledge-based recommender systems

Knowledge-based recommender systems represent a category of systems which are particularly useful in cases where items being recommended are not being interacted with very often. Some of the examples include expensive luxury goods, financial services and real estate. The main problem is that items are not being bought very often and there is insufficient amount of historical data which prevents collaborative techniques from being used. Similar problem is being encountered in cold-start environments.

In the example of real estate, user is more interested in specifying the list of constraints that the listing needs to satisfy. Conditions like – minimum number of rooms, area range, neighborhood and minimal or maximal price are just some of the constraints that user might be willing to set. In these cases, where item domain tends to be complex in terms of its varied properties, knowledge-based recommender systems are the perfect choice.

The recommendation process is done by having user specify his requirements upfront and then doing the similarity calculations between the initial requirement and each product in the database.

Such systems can be classified in two categories [4]:

1. *Constraint-based recommender systems* – in systems like these, user specifies the requirements upfront and the system compares the items against these. Once the results are returned, user can further refine the search results or change the search criteria completely.
2. *Case-based recommender systems* – these systems ask user to specify an example of the item he's interested in. After that, the recommendations are based on calculating the similarities between items in database and the example that was specified. The returned results are often used as a new target cases, such as that, after receiving them, user might wish to take one of the resulting items, make minor modifications and submit a new query. In that sense, *case-based recommender systems* are said to be interactive.

It is well worth mentioning that both content-based and knowledge-based systems rely heavily on characteristics of the items. The main difference between them is that content based systems leverage the user's past behaviors, while knowledge-based systems rely on user's active specification. As such, in most of literature, knowledge-based systems are said to be a distinct category of content-based recommender systems.

3.2.6 Other approaches

Aside from collaborative and content-based systems, there are two other distinct categories called *Demographic recommender systems* and *Hybrid and ensemble-based recommender systems*.

Demographic recommender systems rely on knowledge of users demographic information. One example is recommending the restaurant to have dinner at by leveraging the information from user's GPS device. Although they are not much useful as standalone algorithms, they are usually used in combination with knowledge-based systems in order to increase their robustness.

Hybrid and ensemble-based systems are said to harness the power of all aforementioned techniques. Namely, collaborative techniques rely on other users ratings, content systems rely on items characteristics and knowledge-based systems leverage users explicit set of requirements. The theoretical problem with all these is that they rely only on a single dimension of input. This is exactly the area that ensemble methods try to leverage by combining multiple techniques in order to provide the most relevant recommendations.

3.3 Cold start problem

Cold start problem refers to an issue of not having the previous data for basing the predictions on. Collaborative filtering algorithms are ones that are most affected by this issue. Namely, these algorithms rely on previous ratings of similar items or similar users. In case that there are no previous ratings to rely on, these algorithms can not function.

One typical example would be a new user who just registered at our website. Initially, we don't have any information about him and using collaborative filtering techniques is not an option.

Potential solutions have been proposed in chapter 3.2.3. These are:

1. Starting with top-k most popular items. We can either recommend these to user or ask user to provide a feedback on them. The chances are high that user has already interacted with those so that he can provide the feedback immediately. Based on this we can curate the content immediately and further narrow down the users preferences.
2. Explicitly asking user for input. For example, in case of movie recommendation engine, we could ask the user what movie genres does he prefer and then rely on recommending top-k movies from those genres.

There has been a lot of research involved in this topic and number of solutions have been proposed [39, 40, 41, 42]. Most of the proposed solutions work exactly as mentioned above – by first working out to get some feedback from user about his preferences, and then working towards building the best proposals.

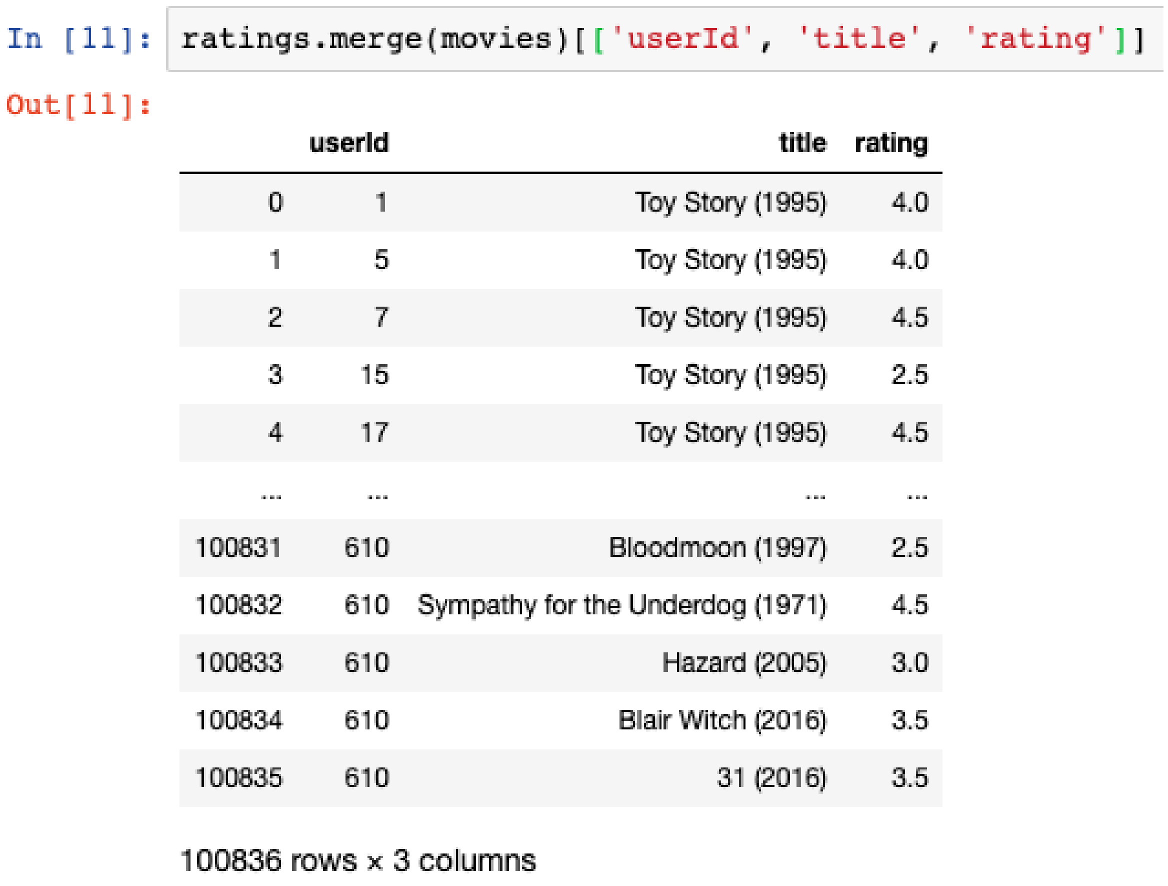
4 Methodology

We are going to evaluate the Movies dataset against the following algorithms:

1. Collaborative filtering using neighborhood-based techniques
2. Collaborative filtering using model-based techniques
3. Content-based filtering
4. Matrix factorization

4.1 Dataset

All the evaluations that will be done are going to use the MovieLens data set. MovieLens is a publicly available dataset provided by GroupLens research group [3]. Given dataset contains 100.000 ratings in total. These ratings were provided by 610 users against 9700 movies in total.



```
In [11]: ratings.merge(movies)[['userId', 'title', 'rating']]
Out[11]:
```

	userId	title	rating
0	1	Toy Story (1995)	4.0
1	5	Toy Story (1995)	4.0
2	7	Toy Story (1995)	4.5
3	15	Toy Story (1995)	2.5
4	17	Toy Story (1995)	4.5
...	...	...	...
100831	610	Bloodmoon (1997)	2.5
100832	610	Sympathy for the Underdog (1971)	4.5
100833	610	Hazard (2005)	3.0
100834	610	Blair Witch (2016)	3.5
100835	610	31 (2016)	3.5

100836 rows × 3 columns

Figure 6 - Dataset samples

Ratings values are defined in the [0.5, 5] range. Average rating value is 3.5.

```
In [22]: ratings.rating.plot.hist(bins=10)

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2159d358>
```

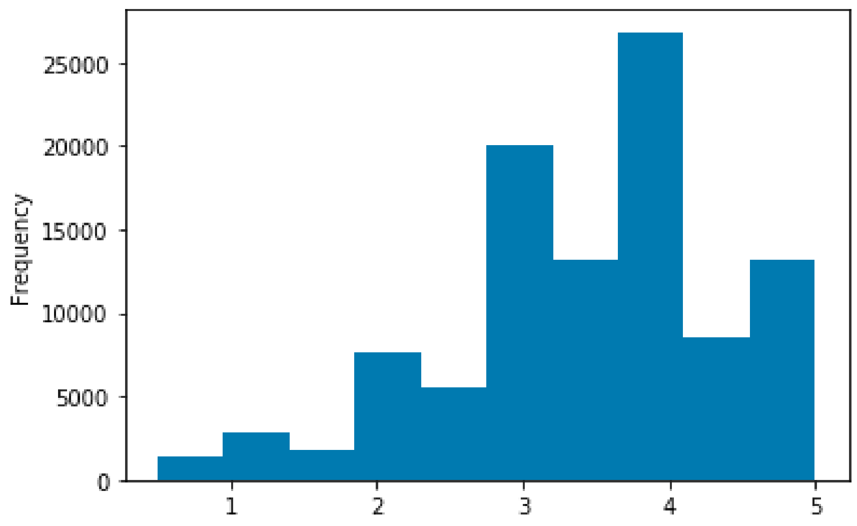


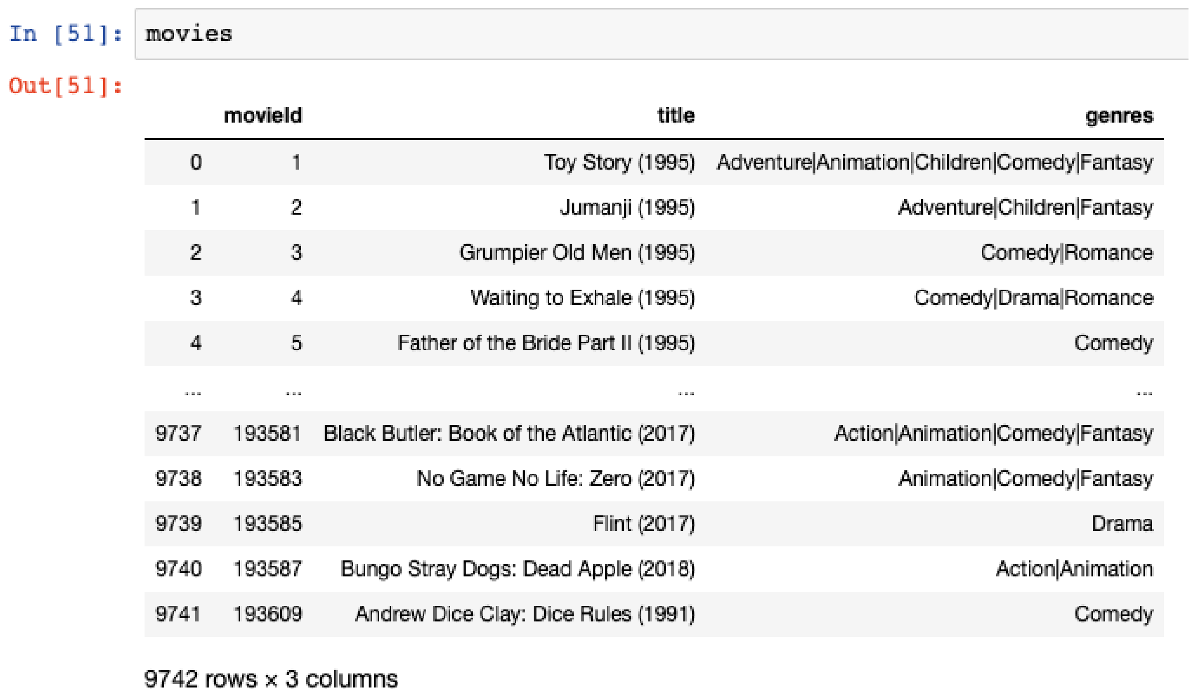
Figure 7 - Ratings distribution

Movie with highest number of ratings is *Forrest Gump* with 329 ratings in total and average rating of 4.2. Movie with highest number of positive ratings is *The Shawshank Redemption* with total of 274 rows who rated it with a positive rating[2]. The most disliked movie is *Ace Ventura: Pet Detective* having total of 49 negative ratings[3].

Aside from providing users and their associated ratings for each movie, MovieLens also provides *genres* and *tags* for each movie. Genres represent officially published data of movie genre, while tags represent user-submitted values for each movie. As such, tags have to be treated as unofficial source of information. Both genres and tags will be used for Content-based filtering and calculation of similarities between movies.

[2] Positive rating is considered as a rating in range (3, 5]

[3] Negative rating is considered a rating in range of [0, 3)



```
In [51]: movies

Out[51]:
        movieId                                  title                                        genres
   0          1                       Toy Story (1995)   Adventure|Animation|Children|Comedy|Fantasy
   1          2                         Jumanji (1995)                    Adventure|Children|Fantasy
   2          3                Grumpier Old Men (1995)                                Comedy|Romance
   3          4               Waiting to Exhale (1995)                          Comedy|Drama|Romance
   4          5     Father of the Bride Part II (1995)                                        Comedy
 ...        ...                                    ...                                           ...
9737     193581   Black Butler: Book of the Atlantic (2017)         Action|Animation|Comedy|Fantasy
9738     193583             No Game No Life: Zero (2017)                 Animation|Comedy|Fantasy
9739     193585                             Flint (2017)                                    Drama
9740     193587       Bungo Stray Dogs: Dead Apple (2018)                        Action|Animation
9741     193609       Andrew Dice Clay: Dice Rules (1991)                                  Comedy

9742 rows × 3 columns
```

Figure 8 - List of movies and genres

4.2 Environment

All experiments and evaluations are run on MacBook mid-2015, with 16GB of DDR3 RAM memory and 2.8 GHz Intel Core i7 processor.

Programming language of choice is Python 3.7.3 with Anaconda 4.7.12 and Jupyter 4.6.0.

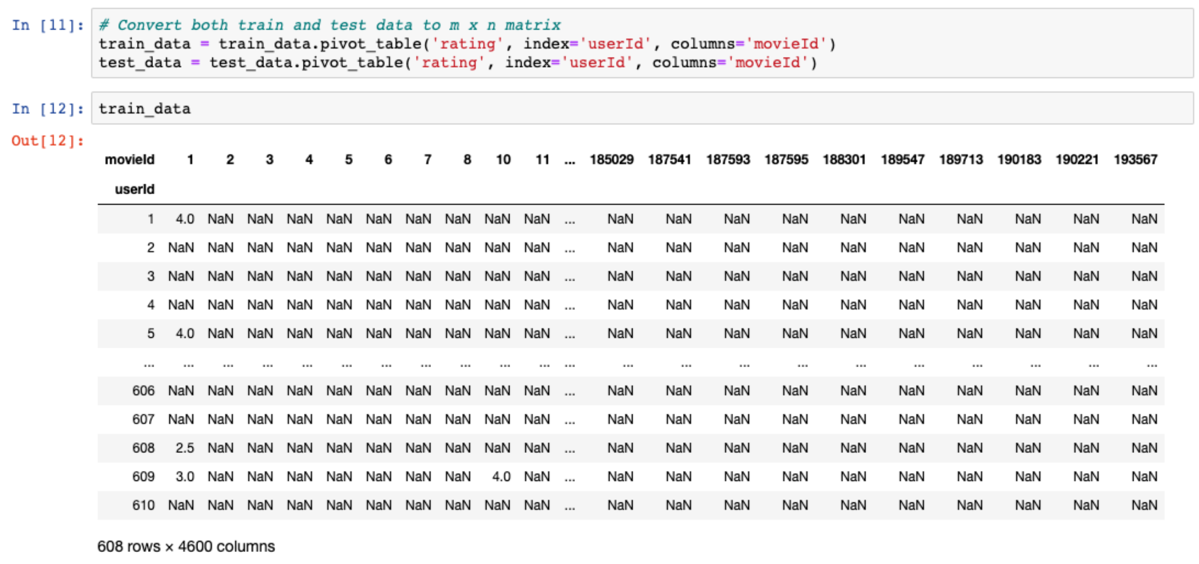
All the reported results originate from the aforementioned environment. All the code used for experiments is available on GitHub. Link can be found in Appendix.

4.3 Collaborative filtering

4.3.1 User-based collaborative filtering

For the first evaluation we are going to be using the Collaborative filtering techniques. As mentioned before, these algorithms are one of the fastest and simplest to implement. Therefore, they present a good starting point to build our evaluations upon.

The next step is converting the long-format into a wide format. Specifically, we are going to turn the original matrix where each row represents a vector for single rating of single user for single movie into a *m x n* matrix where rows represent users and columns represent ratings for movies. This will be a sparse matrix but for sake of presentation we will keep it as a normal *DataFrame*.



```
In [11]: # Convert both train and test data to m x n matrix
         train_data = train_data.pivot_table('rating', index='userId', columns='movieId')
         test_data = test_data.pivot_table('rating', index='userId', columns='movieId')

In [12]: train_data
Out[12]:
```

movieId	1	2	3	4	5	6	7	8	10	11	...	185029	187541	187593	187595	188301	189547	189713	190183	190221	193567
userId																					
1	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
608	2.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
609	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
610	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
608 rows × 4600 columns
```

Figure 9 - m x n ratings matrix

As can be seen, most of the ratings are unspecified. This is normal since an average user has specified around 165 ratings [4].

In order to remove the user's bias [43] we will also do a mean normalization and make sure that mean value for each user is set to 0. In order to do this, first we calculate the mean value for each user and then we subtract these values from each of user's ratings.

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \qquad \forall_u \in \{1 \ldots m\}$$

Equation 3 - mean rating value

Theoretically, this process of mean removal should be done on user-to-user basis, such as that when comparing two users, we first find the set of items that were rated by both users, and then we do mean calculation using those movies only. However, for sake of efficiency, we will not be doing that. It was also found that using the former approach by doing mean removal initially on the whole set doesn't introduce much of a difference and output results remain similar [4].

[4] Average number of ratings per user has been calculated by finding an average value of total number of ratings per user

Next step is calculating the similarities. The ideal scenario is finding the top-K users who are most similar to target user and using those as source of recommendation for target user. However, the problem with finding top-k users is that amount of ratings they specified themselves might vary significantly. In order to solve that, we are going to find top-K users who rated the specific movie and predict the rating based on that. Even though this approach could sound less intuitive, it has been observer that in practice this approach doesn't make much of a difference in recommended data, but does lead to more efficient calculations and faster algorithms.

When it comes to similarity rating, there are number of approaches that are available [44]. Most popular ones are:

1. Pearson correlation – similarity between two vectors is calculated using the Pearson's similarity coefficient. It represents a measure of the linear correlation between two variables *X* and *Y*. The value of this coefficient ranges from -1 to +1, where +1 is total positive linear correlation, 0 means no correlation at all and -1 is total negative correlation.
2. Cosine similarity – similarity is expressed as a cosine of the angle between two vectors A and B. The theory behind it is that vectors with smaller angle have larger cosine value. Value of this coefficients ranges from 0 to 1, with 0 representing no similarity at all (i.e. vectors with an angle of 90°) and 1 represents completely aligned vectors (i.e. angle is 0°). We will be using cosine similarity in content-based systems evaluation.
3. Jaccard similarity – similarity is based on the number of users which have rated item A and B divided by the number of users who have rated either A or B. It is typically used where we don't have numeric rating but just a boolean value.
4. Number of alternative approaches are suggested by Aggarwal et al. [44] such as *PIP Similarity, New Heuristic Similarity Model, Spearman Rank Correlation* and *Kentall's Tau correlation.*

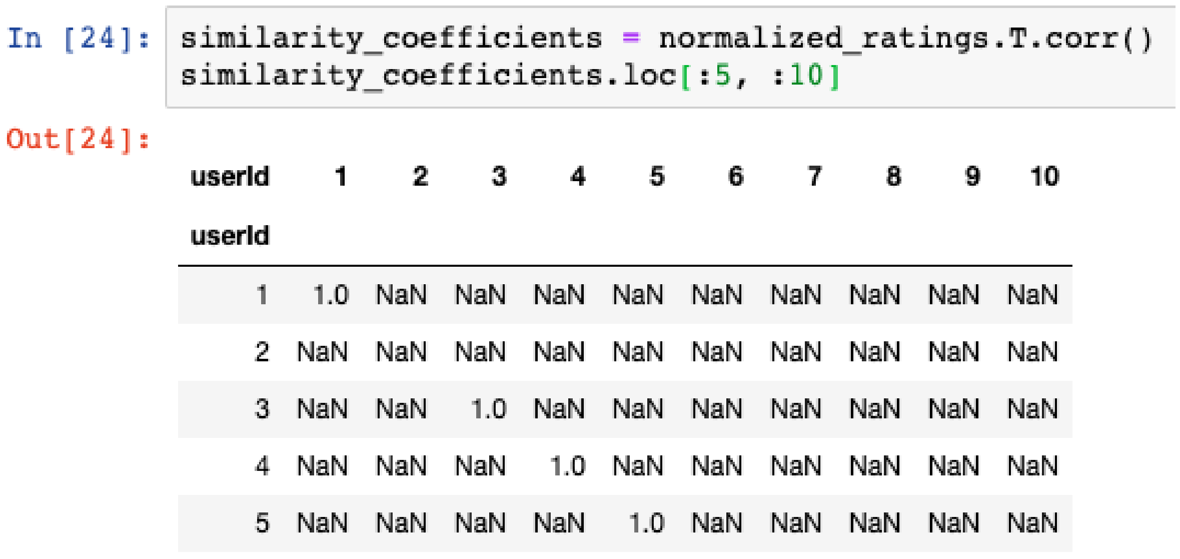
For the purposes of this evaluation, we will be using Pearson correlation coefficients:

$$Pearson(u,v) = \frac{\sum_{k \in I_u \cap I_v}\left((r_{uk} - \mu_u) * (r_{vk} - \mu_u)\right)}{\sqrt{\sum_{k \in I_u \cap I_v}(r_{uk} - \mu_u)^2} * \sqrt{\sum_{k \in I_u \cap I_v}(r_{uk} - \mu_u)^2}}$$

Equation 4 - Pearson correlation coefficient

Pearson correlation is most commonly used method in collaborative filtering models [45]. This method works by finding the linear correlation between two vectors, resulting in a value between -1 and +1. -1 represents a negative correlation while +1 represents high positive correlation. 0 value shows no correlation at all, which is sometimes referred to as zero order correlation.

It's worth mentioning that Agarwal et al. [44] have concluded that using generic traditional correlation coefficients, such as cosine similarity, mean squared difference and Pearson correlation is usually not enough to describe the diversity in users being correlated. This is especially true when it comes to users who specified only a small number of ratings in total. However, for purposes of this evaluation, we will use Pearson correlation because it provides an easy and reasonably fast way of calculating the coefficients.



```
In [24]: similarity_coefficients = normalized_ratings.T.corr()
         similarity_coefficients.loc[:5, :10]
Out[24]:
```

userId	1	2	3	4	5	6	7	8	9	10
userId										
1	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN

Figure 10 - Similarity coefficients between samples

Due to sparsity of the matrix, number of coefficients are set to NaN, as the coefficients can't be calculated.

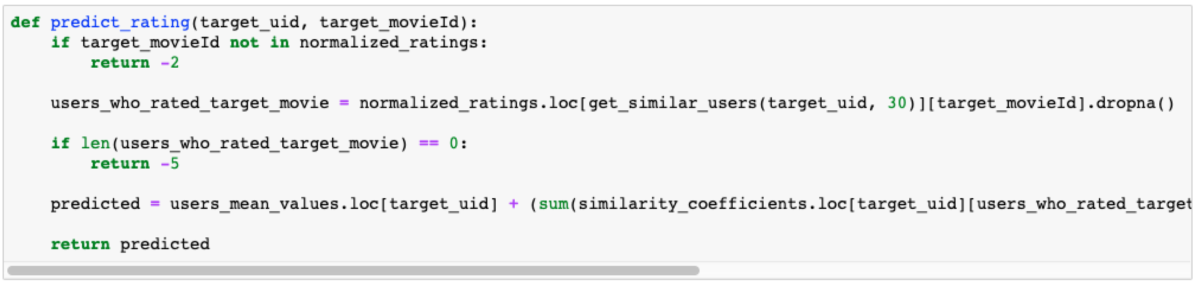
Next, we define a method for predicting the ratings. This method takes two parameters on input – *target user* and *target movie*. As mentioned above, in order to make the algorithm more efficient, we will be doing predictions against specified user and specified movie.

The predicted rating *r* of given movie for given user *u* and movie *j* is calculated using the following formula:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} Pearson(u,v) * (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Pearson(u,v)|}$$

Equation 5 - Formula for calculating the predicted rating of given user for given movie

What we are doing is finding the users most similar to target user who have also rated the target movie. Then, for each user we multiply the correlation coefficient and the mean-centered rating of target user. We sum all these multipliers, divide by total number of similar users and finally, in order to get a rating in the original [0.5, 5] range, we add back the mean value for target user. There seem to be number of alternative approaches for calculating the predicted rating, but this one has been found to predict solid results [4].

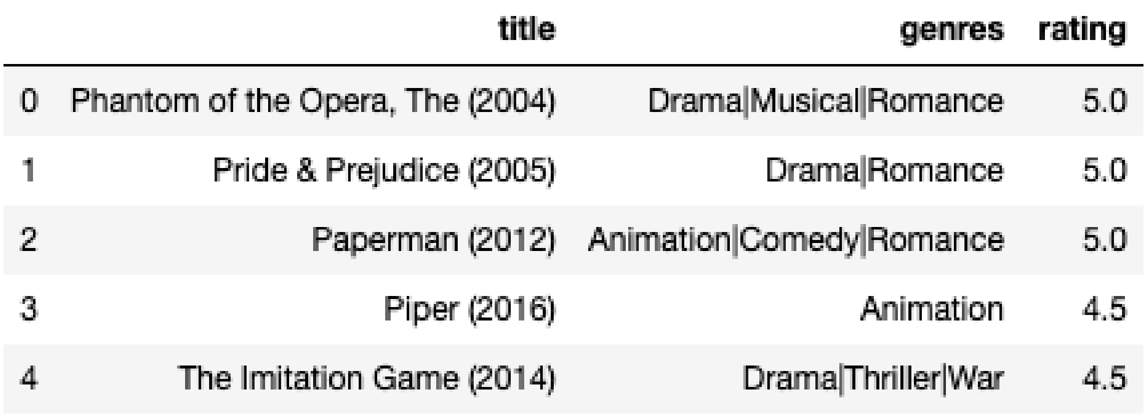


```python
def predict_rating(target_uid, target_movieId):
    if target_movieId not in normalized_ratings:
        return -2

    users_who_rated_target_movie = normalized_ratings.loc[get_similar_users(target_uid, 30)][target_movieId].dropna()

    if len(users_who_rated_target_movie) == 0:
        return -5

    predicted = users_mean_values.loc[target_uid] + (sum(similarity_coefficients.loc[target_uid][users_who_rated_target

    return predicted
```

Figure 11 - Python function for predicting the ratings

Let’s see how that works on a sample user. We’ll use user 148 as a reference user. This user has rated 48 movies in total with mean rating of ~4. Let’s see some of the movies he rated with a high values[5].

[5] High value is referred to any ratting above 4.

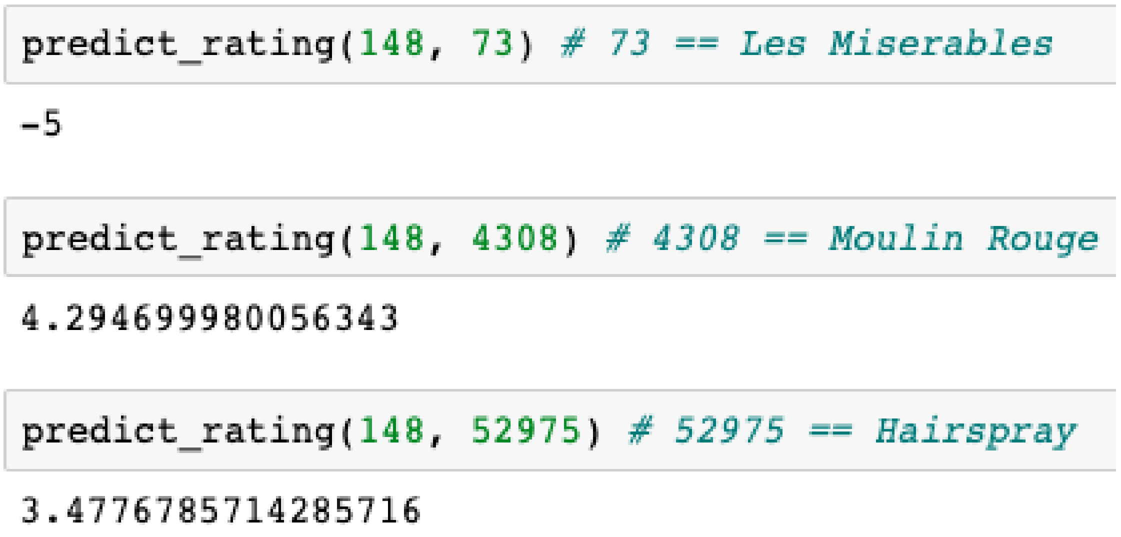


	title	genres	rating
0	Phantom of the Opera, The (2004)	Drama|Musical|Romance	5.0
1	Pride & Prejudice (2005)	Drama|Romance	5.0
2	Paperman (2012)	Animation|Comedy|Romance	5.0
3	Piper (2016)	Animation	4.5
4	The Imitation Game (2014)	Drama|Thriller|War	4.5

Figure 12 - movies that were rated with a high value by observed user

It seems that this user has highest preference for Romance and Drama. Using IMDb [32] we find that some of the movies similar to *The Phantom of the Opera (2004)* [6] are *Les Misérables (2012), Moulin Rouge (2001)* and *Hairspray (2007)*.

Here are the evaluation results:



```
predict_rating(148, 73) # 73 == Les Miserables
```

```
-5
```

```
predict_rating(148, 4308) # 4308 == Moulin Rouge
```

```
4.294699980056343
```

```
predict_rating(148, 52975) # 52975 == Hairspray
```

```
3.4776785714285716
```

Figure 13 - sample predicted ratings

This is a perfect example since where we can immediately see one of the problems that Collaborative filtering suffers from – if there are no neighboring users who have rated the same movie, rating can't be predicted. This is what happened with movie *Les Misérables (1995)*. Value of -5 indicates that there are no neighboring users that could be used for prediction.

[6] Movies that are listed are the moves that exist in MovieLens dataset as well

The other two ratings show positive results which potentially do align with what user might like [7].

In order to evaluate the efficiency of evaluation algorithm, we are going to run it against all samples in test set. We will use the predicted ratings and compare them to actual ratings. It is important to note that we are evaluating only the ratings that we were able to predict[8].

The reported root mean squared error (RMSE) on test set is 1.09. However, the problem stems from the fact that algorithm was able to predict rating for 30% of movies only. RMSE on test set is 0.29.

Overall, even though this approach gives very good results in terms of accuracy, it is not very efficient as we need to provide both the user and the target movie. Therefore, recommending top-k items that user might like is an approach that makes more sense. This approach, referred to as *Item-based collaborative filtering* is going to be evaluated next.

4.3.2 Item-based collaborative filtering

Item-based collaborative filtering is similar to user-based collaborative filtering in terms of how the ratings are predicted. The main difference between these two approaches is that, in case of item-based, the similarity coefficients are calculated between the items itself. For the similarity measure, instead of using Pearson correlation coefficient, we will be using Adjusted Cosine Similarity. Although similar, it has been shown that cosine similarity gives better results [4].

First step before doing the analysis is doing the mean-centering by removing the mean value. This is done in the same way as it has been done in User-based filtering. We find the mean rating value for each user and remove it from all of his ratings.

Once the mean removal is done, next step is similarity coefficient calculation. As has been mentioned, we will be using adjusted cosine similarity formula:

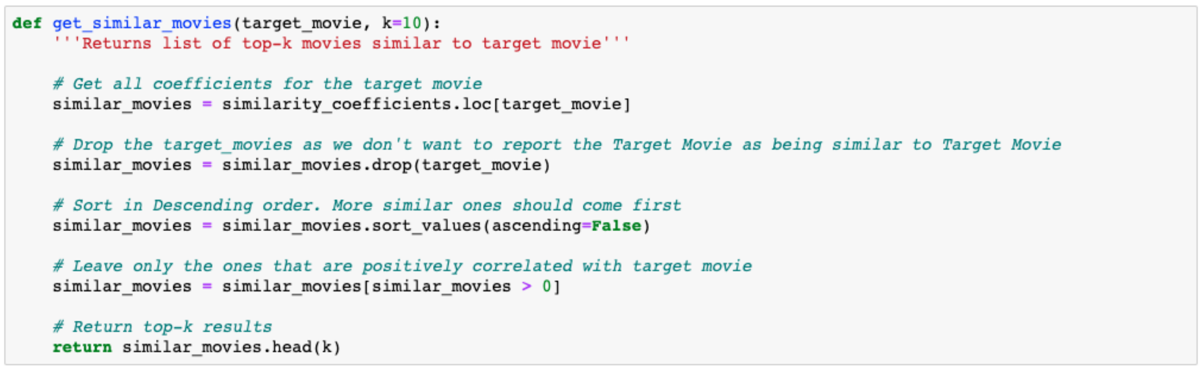
$$AdjustedCosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} * s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} {s_{ui}}^2} * \sqrt{\sum_{u \in U_i \cap U_j} {s_{uj}}^2}}$$

[7] This is under assumption that we consider rating 3 and higher as positive liking.

[8] Out of 20.168 test samples in total, we were able to predict ratings for 6.756 samples only

Equation 6 - Adjusted cosine formula

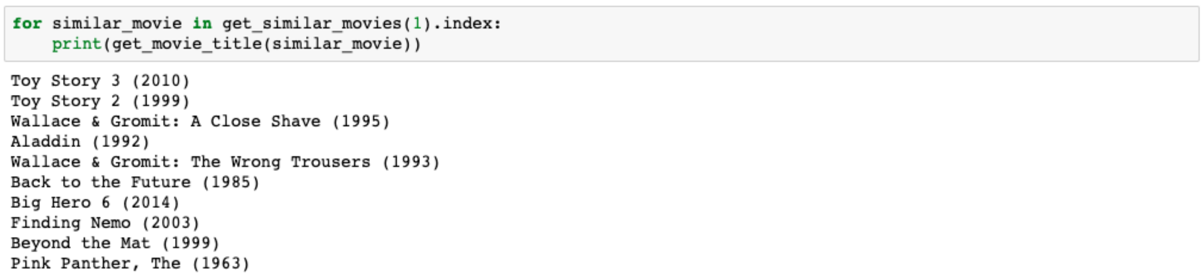
Next step is to define a function for finding movies similar to target movie. The function works by taking move ID on the input and returning the top-k movies with highest similarity coefficient:



```python
def get_similar_movies(target_movie, k=10):
    '''Returns list of top-k movies similar to target movie'''

    # Get all coefficients for the target movie
    similar_movies = similarity_coefficients.loc[target_movie]

    # Drop the target_movies as we don't want to report the Target Movie as being similar to Target Movie
    similar_movies = similar_movies.drop(target_movie)

    # Sort in Descending order. More similar ones should come first
    similar_movies = similar_movies.sort_values(ascending=False)

    # Leave only the ones that are positively correlated with target movie
    similar_movies = similar_movies[similar_movies > 0]

    # Return top-k results
    return similar_movies.head(k)
```

Figure 14 - Python function for getting list of movies that are similar to the target movie

Running this function against a *Toy Story Movie* with ID 1, gives the following results:



```python
for similar_movie in get_similar_movies(1).index:
    print(get_movie_title(similar_movie))
```

```
Toy Story 3 (2010)
Toy Story 2 (1999)
Wallace & Gromit: A Close Shave (1995)
Aladdin (1992)
Wallace & Gromit: The Wrong Trousers (1993)
Back to the Future (1985)
Big Hero 6 (2014)
Finding Nemo (2003)
Beyond the Mat (1999)
Pink Panther, The (1963)
```

Figure 15 - list of movies that are siimlar to the movie "Toy Story"

Next step is ratings prediction. The easiest way for doing the predictions is by predicting a rating of specific movie for specific user. Formula for predicting the rating is:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} AdjustedCosine(j,t) * r_{uj}}{\sum_{j \in Q_t(u)} |AdjustedCosine(j,t)|}$$

Equation 7 - formula for calculating the predicted rating of given user for given movie

The predicted rating is calculated as a product of users ratings on movies similar to target one[9].

[9] Due to longevity of prediction function, it has been omitted from this document. You can find the sample implementation at [TK: Add link to where the prediction function can be found]

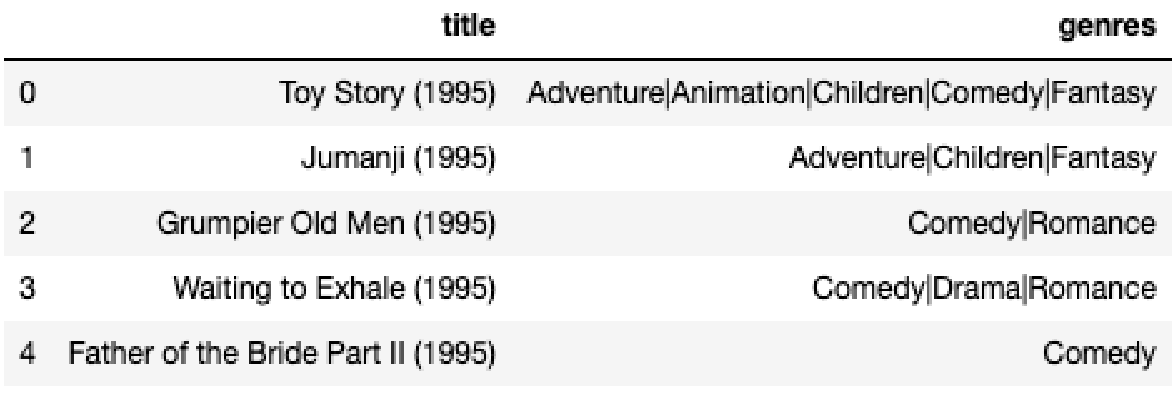
After running the evaluation against the test data with 20.168 the reported RMSE is 0.89 with 76% coverage.

4.3.3 Conclusion

We conclude that even though the predictions are pretty good, the problem is that for lots of items the ratings can't be predicted and on top of that we have a cold-start problem (i.e. user who didn't rate anything can't really be recommended anything).

4.4 Content-based filtering

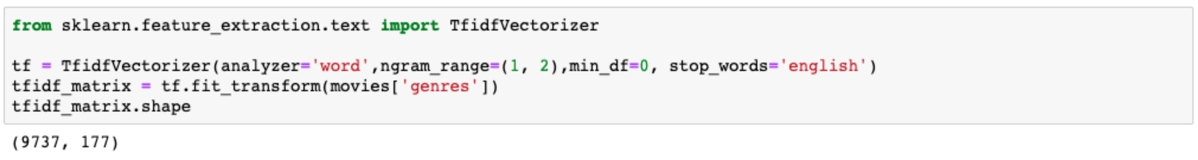
In case of content-based filtering, items are being recommended based on the content they are built of. For the purpose of this evaluation, we will be using the movie genres as features of describing each movie in the data set. The following genres are available in the dataset: *Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, IMAX, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western*.



	title	genres
0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	Jumanji (1995)	Adventure|Children|Fantasy
2	Grumpier Old Men (1995)	Comedy|Romance
3	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	Father of the Bride Part II (1995)	Comedy

Figure 16 - List of movies with their genres

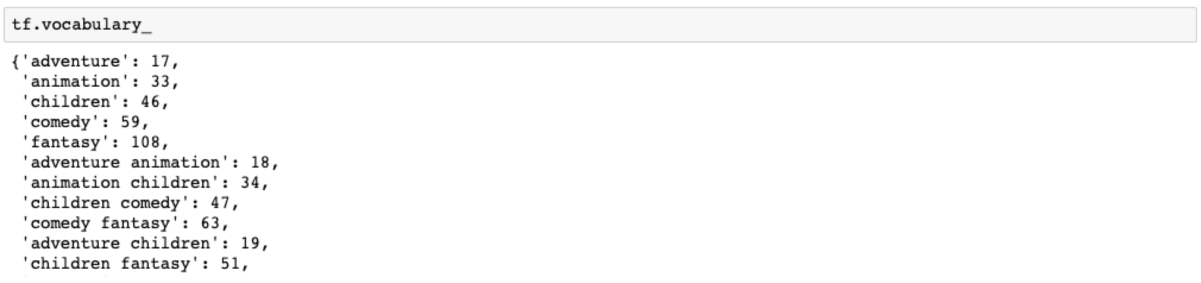
We are going to turn the genres into vectors using the TF-IDF vectorizer. This means that genres that more occurring genres will get lower importance, while the genres that are less occurring will gain higher importance.



```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(movies['genres'])
tfidf_matrix.shape
```

```
(9737, 177)
```

Figure 17 - TF-IDF vectorization of movies list



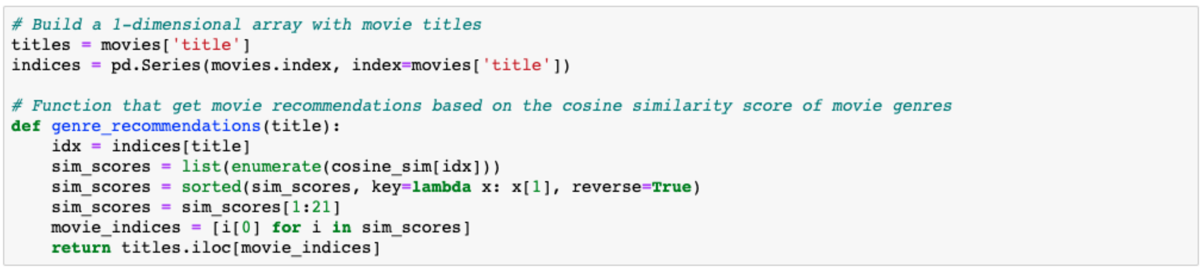
```
tf.vocabulary_
```

```
{'adventure': 17,
 'animation': 33,
 'children': 46,
 'comedy': 59,
 'fantasy': 108,
 'adventure animation': 18,
 'animation children': 34,
 'children comedy': 47,
 'comedy fantasy': 63,
 'adventure children': 19,
 'children fantasy': 51,
```

Figure 18 - TF-IDF vocabulary after vectorization

As can be seen, the movies have been vectorized using unigrams and bigrams. We could have used trigrams as well, but for the purpose of this evaluation we will keep the former two only.

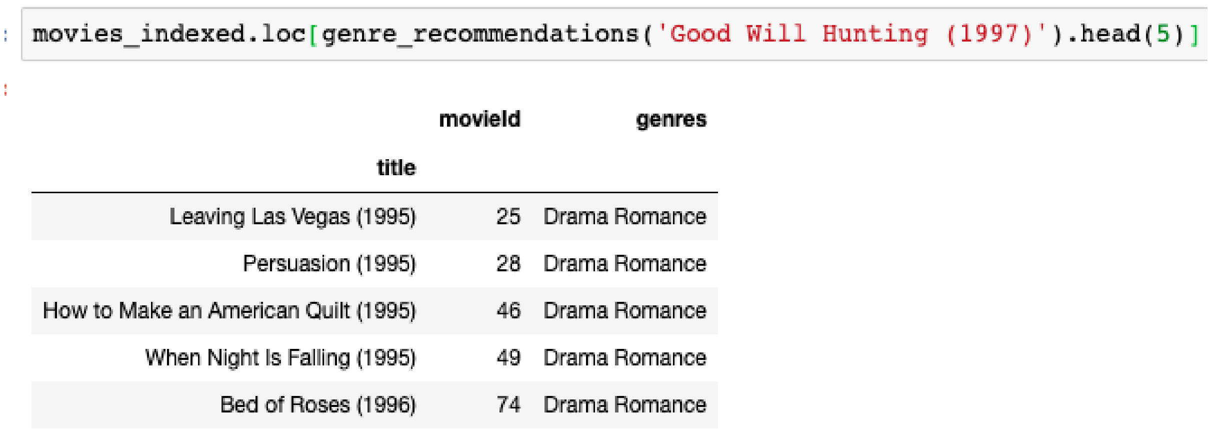
The main idea of content-based recommender engines is to suggest the items that have content similar to the one being evaluated against. This means that we will build a similarity matrix by calculating the similarities between the previously vectorized content and use this matrix to predict similar items.



```
# Build a 1-dimensional array with movie titles
titles = movies['title']
indices = pd.Series(movies.index, index=movies['title'])

# Function that get movie recommendations based on the cosine similarity score of movie genres
def genre_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:21]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

Figure 19 - Python function for predicting the similar movies based on the target movie

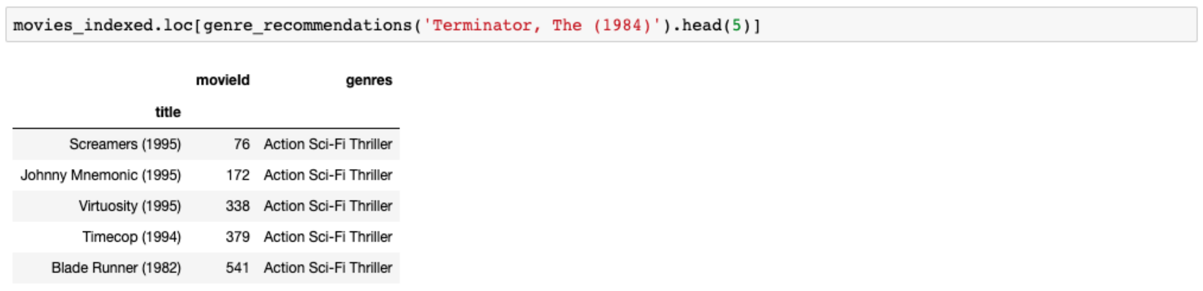
Here’s an example of how it works:



```
movies_indexed.loc[genre_recommendations('Good Will Hunting (1997)').head(5)]
```

title	movieId	genres
Leaving Las Vegas (1995)	25	Drama Romance
Persuasion (1995)	28	Drama Romance
How to Make an American Quilt (1995)	46	Drama Romance
When Night Is Falling (1995)	49	Drama Romance
Bed of Roses (1996)	74	Drama Romance

Figure 20 - List of movies that were predicted, which are similar to the input movie



```
movies_indexed.loc[genre_recommendations('Terminator, The (1984)').head(5)]
```

title	movieId	genres
Screamers (1995)	76	Action Sci-Fi Thriller
Johnny Mnemonic (1995)	172	Action Sci-Fi Thriller
Virtuosity (1995)	338	Action Sci-Fi Thriller
Timecop (1994)	379	Action Sci-Fi Thriller
Blade Runner (1982)	541	Action Sci-Fi Thriller

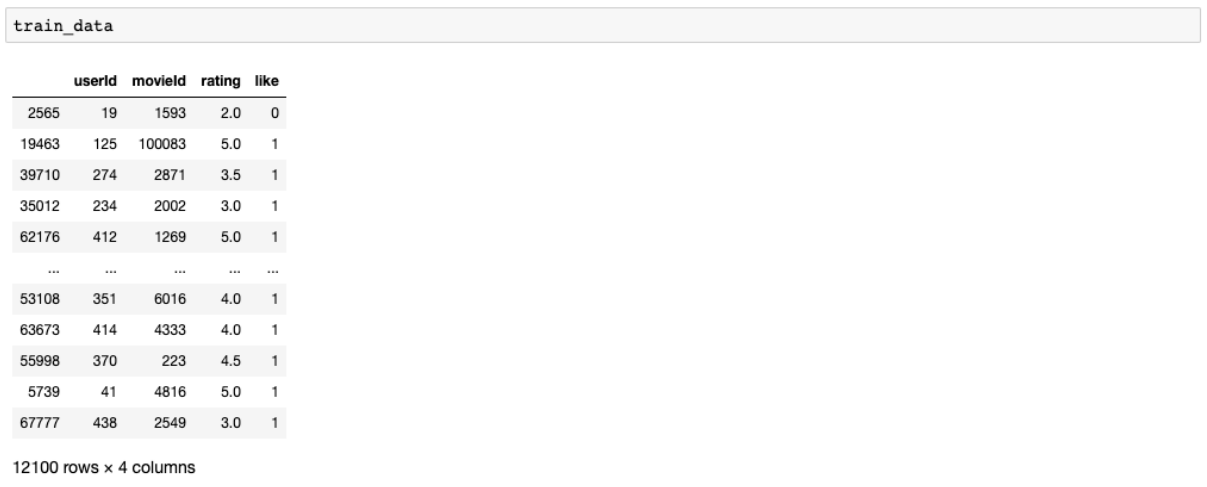
Figure 21 - Movies similar to movie Terminator

As can be seen, the movies that are recommended are exactly the same in genre as the one being evaluated against. It’s important to note in the second example that, for movie *Terminator (1984)*, we don’t see the *Terminator 2: The Judgement Day (1991)* in the list of top-5 recommendations. This is because we didn’t include movie name in the vector, but we rather used movie genre only. Upon further inspection of the recommendation list, we do find the Terminator 2 among the first top-100 recommendations. For the purpose of this evaluation, we will keep the prediction engine as is.

4.4.1 Evaluation

In case of content-based filtering, as the recommendations are being made based on the content, without any reference to users ratings, we have to evaluate the results based on whether user did actually like the movie being recommended. For the purpose of evaluation, we will treat ratings from 3 to 5 as positive feedback, and ratings and everything below 3 as negative feedback.

Due to limited computing power of the environment where this evaluation is being ran, we will take only 20.000 training samples. Each sample represents a single rating of single user for a single movie:



```
train_data
```

	userId	movieId	rating	like
2565	19	1593	2.0	0
19463	125	100083	5.0	1
39710	274	2871	3.5	1
35012	234	2002	3.0	1
62176	412	1269	5.0	1
...	...	...	...	...
53108	351	6016	4.0	1
63673	414	4333	4.0	1
55998	370	223	4.5	1
5739	41	4816	5.0	1
67777	438	2549	3.0	1

12100 rows × 4 columns

Figure 22 - Train data list

For each movie that was rated in the aforementioned set, we will find a list of 50 movies that are predicted based on it. Out of those 50 movies, we will consider as positive hit ones that were positively rated by user, and as a negative hit ones that were negatively rated. Predicted movies that were not rated by the user in question will be ignored. Finally, we calculate the accuracy score based on the number of positive and negative hits.

The evaluation algorithm can be summarized in the following diagram:

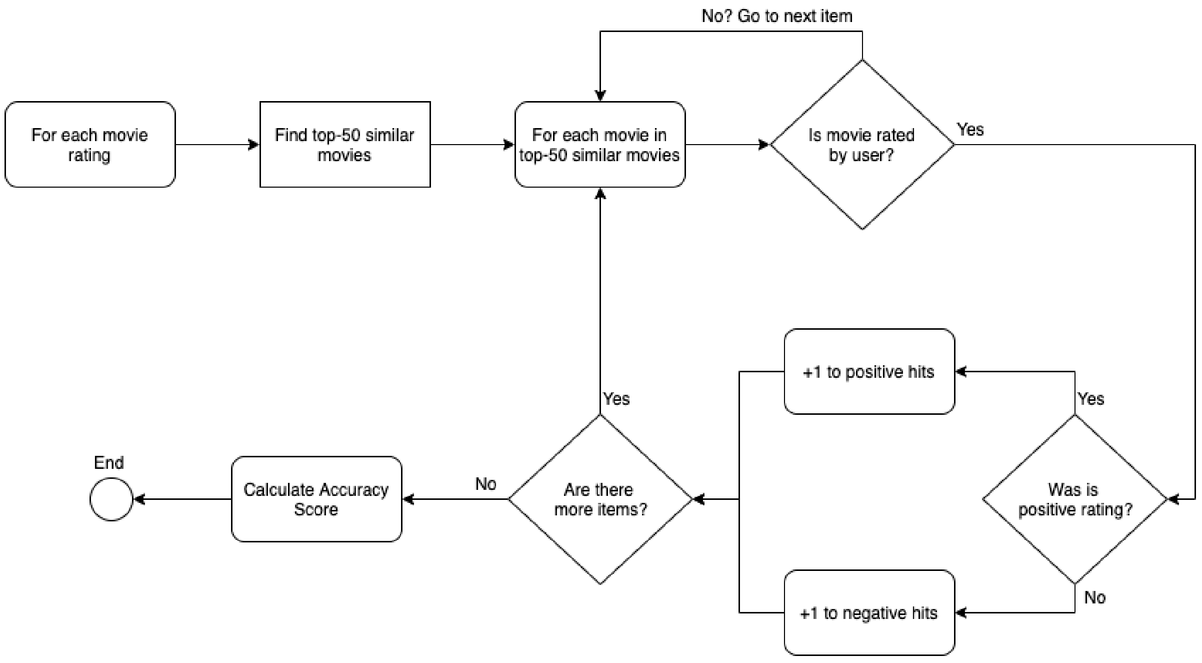


Figure 23 - ratings evaluation algorithm

4.4.2 Conclusion

Out of 20.000 items that were evaluated, 7986 were actually rated, which gives a ~40% evaluation coverage. The reported accuracy score is 0.89, while the reported F1 score is 0.94. One downside is the number of ratings that were actually identified and used for comparison, but the ones that were rated seem to have a pretty good accuracy.

We conclude that content-based systems are perfect for situations where previous data about user preferences is not available. They don't suffer from cold-start problem, don't rely on previous ratings and generally provide a really good coverage of the item universe.

With those characteristics in mind, they can be used as a great first step for onboarding new users and selecting the content to recommend for them.

4.5 Singular Value Decomposition

Final method that will be evaluated is *Singular Value Decomposition (SVD)*. The main idea behind *Singular Value Decomposition* and matrix factorization techniques is that each matrix can be split into smaller matrices representing the *base vectors*. These base vectors represent the, so called, *latent factors*, which refer to user's hidden interests.

In case of movies rating matrix, the rows represent users and columns represent movies and the values represent users ratings for the specified movies. Decomposing that matrix into singular components reveals the users hidden interests behind the given ratings. In this specific example, latent factors may represent the movie genres. After decomposing the matrix into singular components and coefficients, each rating of each user can be treated as a composition of interest of all users and how much of that interest contributes to the given rating.

As an example, we can say that User 1's rating for movie *Terminator (1984)* can be represented as *(User's preference towards Action Movies)* + *(User's preference towards Drama movies)* + *(User's preference towards Comedy movies)*. These preferences and their coefficients are exactly the base vectors, or, so called – latent factors.

$$M = U * \Sigma * V^T$$

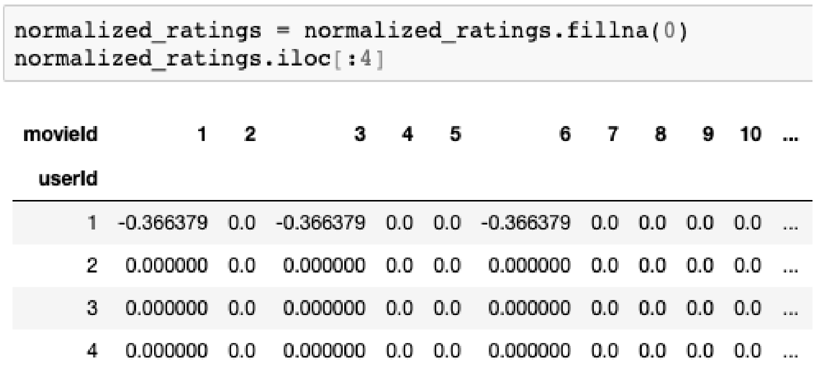
Equation 8 - SVD formula

As has been mentioned before, *SVD* relies on the fact that the data actually has some inner correlations between features. If data is not correlated at all, the *SVD* will simply fail to produce any useful results.

Finally, one great benefit that *SVD* offers is that the whole ratings matrix can be decomposed and predicted at once. There is no need to predict ratings one by one. This really helps both with efficiency and speed.

4.5.1 Evaluation

In order to evaluate the SVD approach, we will first normalize data by doing the mean-centering of rows. Next step is setting the missing values to 0. This is one of the possible approaches, while some other potential approaches include mean-centering the columns and mean-centering both rows and columns. It is generally advised [4] that all approaches should be evaluated as there is no single best solution.



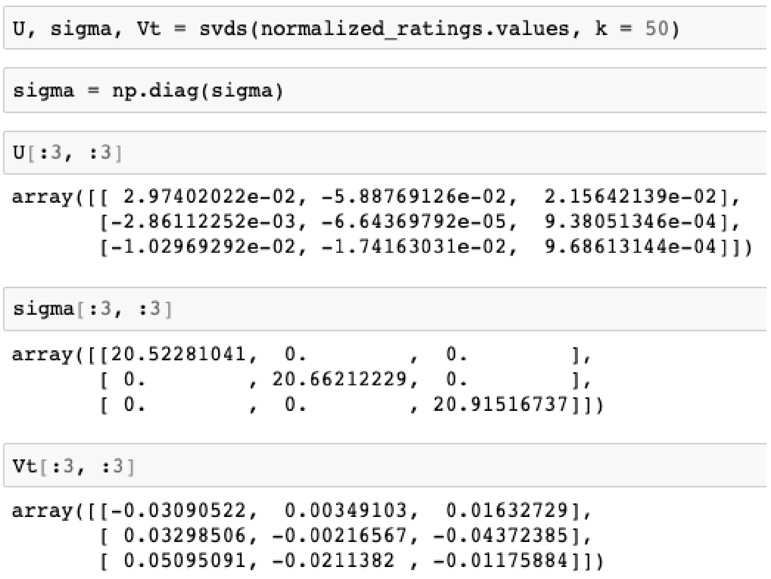
```
normalized_ratings = normalized_ratings.fillna(0)
normalized_ratings.iloc[:4]
```

movieId	1	2	3	4	5	6	7	8	9	10	...
userId											
1	-0.366379	0.0	-0.366379	0.0	0.0	-0.366379	0.0	0.0	0.0	0.0	...
2	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...
3	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...
4	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...

Figure 24 - mean-centering rows

After having the data mean-centered, next step is to run the decomposition. We will be using SciPy's [46] SVD function, as it allows us to easily specify number of components that we want to keep. In our case, we will stick to 50 components[10].

[10] Number of components to use is case-specific and should be evaluated. One of the possible approaches is using the *Principal Component Analysis* to plot the graph of reconstruction errors using a varied number of components. For the purpose of this thesis, we stick to 50 components which brings solid results



```
U, sigma, Vt = svds(normalized_ratings.values, k = 50)
```

```
sigma = np.diag(sigma)
```

```
U[:3, :3]
```

```
array([[ 2.97402022e-02, -5.88769126e-02,  2.15642139e-02],
       [-2.86112252e-03, -6.64369792e-05,  9.38051346e-04],
       [-1.02969292e-02, -1.74163031e-02,  9.68613144e-04]])
```

```
sigma[:3, :3]
```

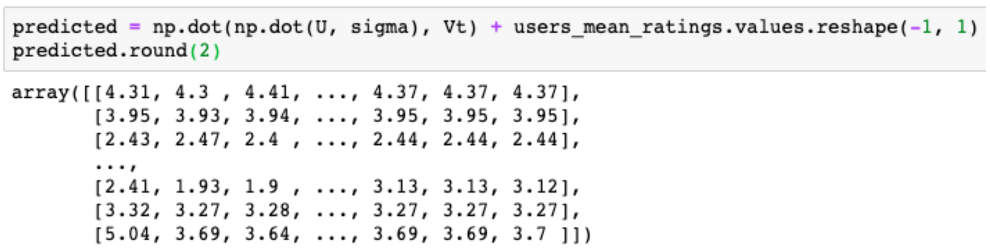
```
array([[20.52281041,  0.        ,  0.        ],
       [ 0.        , 20.66212229,  0.        ],
       [ 0.        ,  0.        , 20.91516737]])
```

```
Vt[:3, :3]
```

```
array([[-0.03090522,  0.00349103,  0.01632729],
       [ 0.03298506, -0.00216567, -0.04372385],
       [ 0.05095091, -0.0211382 , -0.01175884]])
```

Figure 25 - U, Sigma and V values after decomposition. For clarity only first 3 rows and columns are displayed*

As can be seen in Figure 25, we decompose the matrix in one go. Next step is recomposing back the matrix. We use a simple dot product to multiply the vectors.



```
predicted = np.dot(np.dot(U, sigma), Vt) + users_mean_ratings.values.reshape(-1, 1)
predicted.round(2)
```

```
array([[4.31, 4.3 , 4.41, ..., 4.37, 4.37, 4.37],
       [3.95, 3.93, 3.94, ..., 3.95, 3.95, 3.95],
       [2.43, 2.47, 2.4 , ..., 2.44, 2.44, 2.44],
       ...,
       [2.41, 1.93, 1.9 , ..., 3.13, 3.13, 3.12],
       [3.32, 3.27, 3.28, ..., 3.27, 3.27, 3.27],
       [5.04, 3.69, 3.64, ..., 3.69, 3.69, 3.7 ]])
```

Figure 26 - Recomposed matrix with Predicted values filled in

As can be seen, after converting the matrix back to it's original form, we see that all the missing values are already present.

After running the evaluation against the test dataset, we obtain a RMSE of 0.92. The results seem slightly worse than in case of collaboration-based filtering. However, we have to take into account that no prior optimization has been done.

4.5.2 Conclusion

Even though the resulting RMSE is higher than in case of collaborative-based techniques, we still have to keep in mind that no other optimizations have been done. One step usually recommended in the literature is to take the original matrix and replace the missing values with predicted one, and run the SVD again. The convergence is usually reached [4] after 4 to 5 iterations which, it has been reported, usually produces much better results.

It is also worth mentioning that the algorithm that we used represents the most basic version of *Singular Value Decomposition*. Number of alternatives exists, such as SVD++, Funk SVD and others.

One important factor that has to be considered is the speed efficiency. In all the other approaches, calculating the predicted value relied either on running the predictions on case-by-case basis, or by building the user-model upfront. This, obviously, introduces the additional time and space complexity. In case of *Matrix Factorization* approaches, the whole matrix is decomposed in one go, and the missing values are estimated immediately after recomposing the matrix back. This makes it a great choice for estimating the number of ratings in a timely and efficient manner. This characteristic also makes them the state-of-the-art systems in recommender system universe.

5 Conclusion

In this thesis, we have evaluated and implemented the following recommender system approaches:

1. User-based collaborative filtering
2. Item-based collaborative filtering
3. Content-based collaborative filtering
4. Matrix Factorization using Singular Value Decomposition

Resulting metrics are displayed in the table below:

Technique	**RMSE**	**F1**
User-based c.f.	1.09	-
Item-based c.f.	0.89	-
SVD	0.92	-
Content-based c.f.	-	0.94

Table 1 - model evaluation results

As can be seen, the best accuracy was achieved with Item-based collaborative filtering. However, one downside of collaborative techniques is that each missing rating has to be predicted separately, which leads to added time and space complexity.

Therefore, we conclude that SVD algorithms represent the best possible ratio of accuracy and speed and time complexity. The resulting error of 0.92 can be further reduced and improved by varying the number of components and running multiple iterations of decomposition.

Finally, the cold-start problem from which all the algorithms suffer from can be effectively leveraged using the Content-based techniques, that have an F1 score of 0.94. By carefully combining the content-based techniques for new users and SVD algorithms for existing ones, we can effectively maximize the outcomes and likelihood of recommending the best possible items for users to interact with.

6 Appendix

6.1

6.2 Future research resources

https://github.com/YuyangZhangFTD/awesome-RecSys-papers/blob/master/RecSys18/A%20Field%20Study%20of%20Related%20Video%20Recommendations-%20Newest%2C%20Most%20Similar%2C%20or%20Most%20Relevant%3F.pdf

TK: Add reference to the code by this guy https://github.com/khanhnamle1994/movielens

6.3 Programming code

TK: Add code samples in a pseudo language.

6.4 Jupyter notebooks

TK: Add links where all the Jupyter notebooks can be found

7 Bibliography

[1] F. R. a. L. R. a. B. Shapira, Introduction to Recommender Systems Handbook, Springer, 2011.

[2] J. Karlgren, An Algebra for Recommendations, Syslab Working Paper 179, 1990.

[3] GroupLens research project, https://grouplens.org/.

[4] C. C. Aggarwal, Recommender systems: The Textbook, Springer, 2016.

[5] "Netflix," [Online]. Available: http://netflix.com.

[6] "YouTube," [Online]. Available: http://youtube.com.

[7] M. Kaminskas and D. Bridge, Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems, ACM Trans Interact Intell Syst, 2016.

[8] J. Bennett and S. Lanning, The Netflix Prize, Proceedings of KDD cup and workshop. Vol. 2007, 2007.

[9] Y. Zhou, D. Wilkinson, R. Schreiber and R. Pan, Large-scale Parallel Collaborative Filtering for the Netflix Prize, Berlin: International conference on algorithmic applications in management. Springer, 2008.

[10] Y. Koren, The BellKor Solution to the Netflix Grand Prize, Netflix prize documentation 81.2009, 2009.

[11] J. L. Herlocker, J. A. Konstan, L. G. Terveen and J. T. Riedl, Evaluating collaborative filtering recommender systems., ACM Transactions on Information Systems (TOIS) 22, no. 1, 2004.

[12] A. Töscher, M. Jahrer and R. M. Bell, The bigchaos solution to the netflix grand prize., Netflix prize documentation, 2009.

[13] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank and S. Reiterer, oward the next generation of recommender systems: applications and research challenges, Multimedia services in intelligent environments, Springer, 2013.

[14] R. Burke and M. Ramezani, Matching recommendation technologies and domains, Recommender systems handbook, Springer, 2011.

[15] B. Peischl, M. Zanker, M. Nica and W. Schmid, Constraint-based recommendation for software project effort estimation, Journal of Emerging Technologies in Web Intelligence 2, no. 4, 2010.

[16] A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maalej, D. Pagano, L. Weninger and F. Reinfrank, Group decision support for requirements negotiation, International Conference on User Modeling, Adaptation, and Personalization, 2011.

[17] D. Cubranic, G. C. Murphy, J. Singer and K. S. Booth, Hipikat: A project memory for software development, IEEE Transactions on Software Engineering, 2005.

[18] H. F. Hofmann and F. Lehner, Requirements engineering as a success factor in software projects, IEEE software 4, 2001.

[19] B. J. Fogg, Persuasive technology: using computers to change what we think and do., Ubiquity, 2002.

[20] I. Pribik and A. Felfernig, Towards persuasive technology for software development environments: an empirical study., International Conference on Persuasive Technology, 2012.

[21] "Eclipse IDE," [Online]. Available: www.eclipse.org.

[22] X. Amatriain and J. Basilico, Past, Present, and Future of Recommender Systems: An Industry Perspective, Proceedings of the 10th ACM Conference on Recommender Systems, 2016.

[23] "Quora," [Online]. Available: http://quora.com.

[24] "Facebook," [Online]. Available: http://facebook.com.

[25] "Twitter," [Online]. Available: http://twitter.com.

[26] "Spotify," [Online]. Available: http://spotify.com.

[27] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, Item-Based Collaborative Filtering Recommendation Algorithms, University of Minnesota, 2001.

[28] J. Breese, D. Heckerman and C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1998.

[29] Y. Hu, Y. Koren and C. Volinsky, Collaborative filtering for implicit feedback datasets, Eighth IEEE International Conference on Data Mining, 2008.

[30] H. Cho-Jui, N. Natarajan and I. Dhillon, PU Learning for Matrix Completion, ICML, 2015.

[31] O. Douglas and J. Kim, Implicit feedback for recommender systems, Proceedings of the AAAI workshop on recommender systems. Vol. 83. WoUongong, 1998.

[32] http://imdb.com.

[33] R. Van Meteren and M. Van Someren, Using content-based filtering for recommendation, Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop , 2000.

[34] P. Castells, N. J. Hurley and S. Vargas, Novelty and diversity in recommender systems, Recommender Systems Handbook, pp. 881-918, 2015.

[35] M. Kunaver and T. Požrl, Diversity in recommender systems–A survey, Knowledge-Based Systems 123, 2017.

[36] F. Ricci, L. Rokach and B. Shapira, Introduction to recommender systems handbook, Recommender systems handbook, 2011.

[37] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, Application of dimensionality reduction in recommender system-a case study. No. TR-00-043, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[38] S. Funk, "Netflix Update: Try This at Home," 2006. [Online]. Available: https://sifter.org/~simon/journal/20061211.html.

[39] A. I. Schein, A. Popescul, L. H. Ungar and D. M. Pennock, Methods and metrics for cold-start recommendations., Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 253-260. ACM, 2002.

[40] X. Lam, T. Vu, T. Duc Le and A. Duc Duong, Addressing cold-start problem in recommendation systems., Proceedings of the 2nd international conference on Ubiquitous information management and communication, pp. 208-211. ACM, 2008.

[41] L. Blerina, K. Kolomvatsos and S. Hadjiefthymiades, Facing the cold start problem in recommender systems., Expert Systems with Applications 41, no. 4, 2014.

[42] Y. Zhou, S.-H. Yang and H. Zha, Functional matrix factorizations for cold-start recommendation., Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, 2011.

[43] M. M. Milosavljević, Veštačka Inteligencija, Univerzitet Singidunum, 2015.

[44] A. Ajay and M. Chauhan, Similarity measures used in recommender systems: a study., International Journal of Engineering Technology Science and Research IJETSR, ISSN, 2017.

[45] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl, GroupLens: an open architecture for collaborative filtering of netnews, Proceedings of the 1994 ACM conference on Computer supported cooperative work, pp. 175-186. ACM, 1994.

[46] M. Balabanović and Y. Shoham, Fab: Content-based, Collaborative Recommendation, Communications of the ACM, 1997.