Topic: Synthesis and Evaluation of various Recommender System approaches against a Movie database dataset

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 as an example. He suggests to take content providers, such as Netflix, 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.com. 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. [5] 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.com 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, Last.fm and Pandora

Write a bit about how recommender systems were/are used at these

3.1.2 Recommender systems and software engineering

Read more in https://www.researchgate.net/profile/Gerald_Ninaus/publication/261263565_recommender_systems_future/links/00463533bd544bf3c3000000/recommender-systems-future.pdf

3.1.3 Recommender systems and Big data

Search for some work in this area and write a bit about it …

3.1.4 Future of Recommender systems

Write a bit about how we see the future of Recommender systems, etc.

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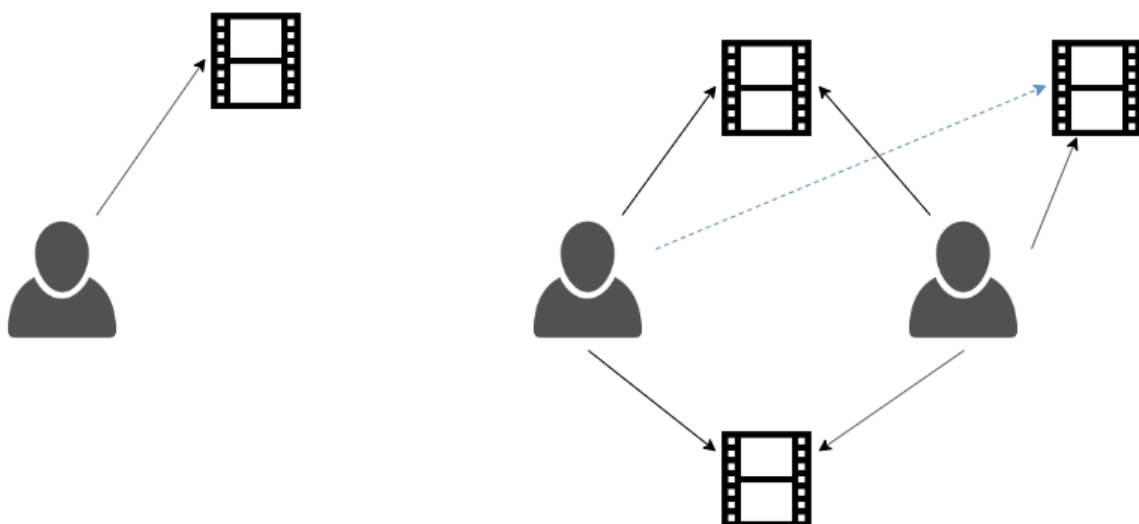
Figure 1 – User 2 and User 3 like the same items and item 4 being recommended based on that

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 [6]. 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 [7].

	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 [8].

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, Youtube 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* [9] [8] [10].

Explicit vs Implicit ratings

3.2.3 Content-based recommender systems

Write about Content-based recommender systems, about TF-IDF vectorizer, etc.

3.2.4 Knowledge-based recommender systems

3.2.5 Other approaches

3.3 Evaluation

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

Primary evaluation metric that will be used is the accuracy score.

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

TK – Figure Name Here!

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>
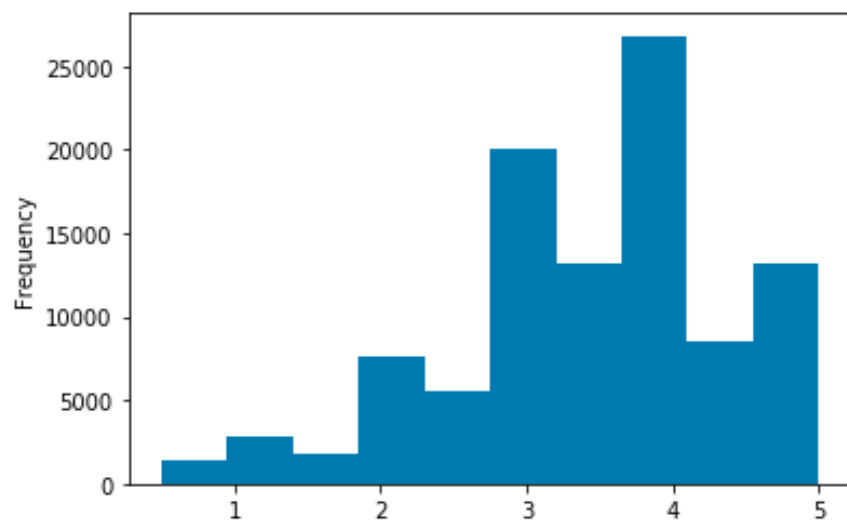
TK – Figure Name Here!

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[1]. The most disliked movie is *Ace Ventura: Pet Detective* having total of 49 negative ratings[2].

 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.


```
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

TK – Figure Name Here!

4.2 Environment

Write about the laptop that will be running the tests, etc. Also write about Python, Jupyter, etc.

---

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

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

4.3 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 [TK: Who came up with this naming]. 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

TK – figure name here!

As can be seen, most of the ratings are unspecified. This is normal since an average user has specified around 165 ratings [3].

In order to remove the user's bias (TK: Any literature about this) 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.

TK: ADD Mean formula here (2.1 from book)

---

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

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].

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 [11]. Most popular ones are:

1. Pearson correlation (TK: a bit more about it)
2. Cosine similarity (TK: a bit more about it)
3. Jaccard similarity (TK: a bit more about it)

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

TK: Add Pearson correlation formula here!

Pearson correlation is most commonly used method in collaborative filtering models [12]. 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. [11] 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

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:

TK: Add formula 2.4 from book

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].

```
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
```

TK – Figure name here!

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 value[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

It seems that this user has highest preference for Romance and Drama. Using IMDb we find that some of the movies similar to *The Phantom of the Opera (2004)* [5] 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
```

TK – Add figure name here!

---

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

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

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.

The other two ratings show positive results which potentially do align with what user might like [6].

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[7].

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.

4.3.1   Item-based collaborative filtering

TK: Write about item-based collaborative filtering here

4.3.2   Conclusion

Well 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*,

---

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

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

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

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.

```python
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)
```

TK: Add figure name here!

```python
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,
```

TK: Add figure name here!

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.

```python
# 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]
```

TK: Add figure name here!

Here's an example of how it works:

```python
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

TK: Add figure name here!

```python
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

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

TK: Add figure name here!

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:
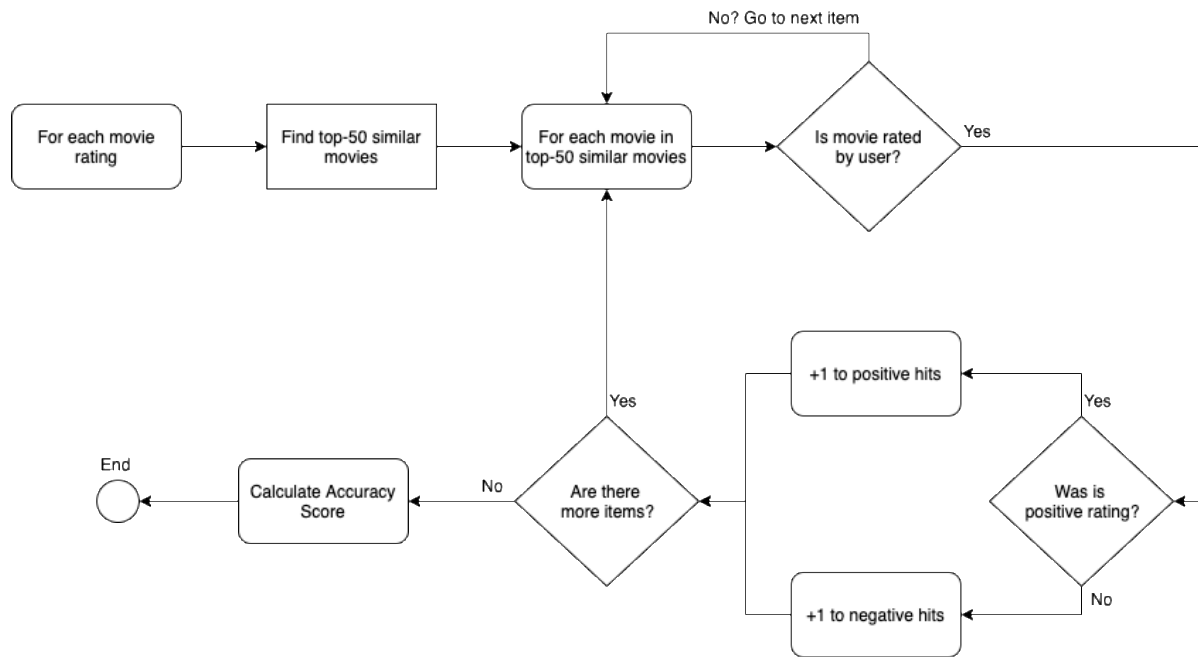
TK: Add figure name here!

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 obvious 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.

TODO: Add following references to the END of bibliography: YouTube, MovieLens, IMDb, my github repo

5 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] 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.

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

[7] 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.

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

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

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

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

[12] 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.

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