Comparing collaborative filtering with content based filtering on Steam games

J.A. Simonse
STUDENT NUMBER: 2045505

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE OR DATA SCIENCE & SOCIETY
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

A.T. Hendrickson
I. Önal

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
December 2020

Preface

This thesis marks the end of my Master in Data science and Society.

First, I would like to thank my supervisor Drew Hendrickson, for the guidance and time investment he made every week. The weekly meetings were very helpful, and definitely contributed to me finishing my thesis.

I also want to thank my parents for their support throughout this process, as I could have been a bit grumpy over the course of the last few weeks.

Lastly, I want to apologize to my girlfriend and my friends, as I sacrificed some of their time in order to finish the thesis. I promise we will celebrate my graduation, when the time is there.

Jeroen Simonse

Tilburg, December 2020

Comparing collaborative filtering with content based filtering on Steam games

J.A. Simonse

Abstract

With the already 30000 unique game titles that are available on the Steam platform, and developers releasing more games every year, the average gamer might be overwhelmed by the abundance of game titles. To make sure the users of the platform don't get lost in the game store, there are systems working on the background, that make sure the users only gets to see relevant games. These systems are called recommender systems, and they try to recommend games to the user, based on the users' past interests. The two most popular methods for recommending items to users, are the collaborative filtering method and the content based filtering method. This study focuses on these two methods, and tries to determine which of the two methods provides the best recommendations for the users of Steam. The data used for this study contained information about the what games each user owned, how long each user played a game and the characteristics of each game. First the collaborative model was constructed, which combines the individual interests with the opinions of other users to predict recommendations. Then the content based model was constructed, which focuses more on the contents and the characteristics of a game. The results of this study showed that the collaborative filtering method was superior to the content based method, which corresponds with the research that already has been conducted on this topic.

Contents

1. Introduction

1.1 Context

As we spend more and more time on the internet, especially now during the pandemic, the amount of information that is being gathered and stored on each person is growing by the minute. One thing in particular we like to do on the internet, is shopping. The e-commerce industry is still rapidly growing, and they use all this stored information, by implementing recommender system. These systems can take the data on users and their preferences as an input, and predict the potentially interesting items for an user. This provides the user with a more personal shopping experience, and allows the e-retailer to target his customers better.

One very lucrative branch of e-commerce is the gaming industry. Before the explosive growth of e-commerce, people used to buy PC games at actual physical game stores. However, as we are shifting more towards online shopping, this also changed for games. One platform in particular that played a vital role in this transition, was Steam. Steam is an online platform that sells digital copies of games, which currently offers more than 30000 unique game titles, and has over 95 million monthly active users (Gough 2020).

With over 30000 unique titles and 95 million monthly users, Steam is a perfect example of a large e-commerce business that can benefit from an effective recommender system. Without any recommendations, a user may find it hard to decide on which games he wants to buy, as there are 30000 to choose from (Bolding 2019). The abundance of choice might be overwhelming to the user, and can lead to the user being indecisive. Adding a recommender system to the platform offers a great solution to deal with this issue.

In the context of recommender systems, there are two techniques that are widely used, namely collaborative filtering and content based filtering (Aggarwal and Aggarwal 2016). Collaborative filtering is a technique that can recommend items based on past interests of a user in combination with the opinions of similar, like-minded people. While this technique does not take the contents of a particular item into consideration, content based filtering focuses specifically on the contents of the item, and recommends items that are similar to the items that a user liked in the past. To determine these 'past interests' of a user, some sort of feedback is needed from the user. This can either be explicit feedback, or implicit feedback. Explicit feedback comes directly from the user, such as a rating or review, whereas implicit feedback contains information about the behavior of an user. Naturally, there is a more implicit feedback than explicit feedback which is also the case for Steam. One very interesting implicit metric they track, is the in-game time (or playtime) per game, as it is a very strong indicator of the actual preferences of a certain user(Parra and Amatriain 2011).

Since Steam has a very favorable policy regarding the data you can extract from their website, there are plenty of datasets that can be found on Steam users. This study uses two separate datasets to test both the collaborative filtering method and the content based method. The first dataset is a dataset that includes the playtimes of 12393 unique users, and can be downloaded from Kaggle. The dataset is provided by the Tamber Team (Tamber Team 2016). As stated before, playtime is a metric that can be seen as implicit feedback, since it isn't explicitly coming from the user himself. The second dataset is also downloaded from Kaggle, and includes all characteristics of 27033 unique games. This dataset is provided by Nik Davis (Davis 2019).

Insights gained during this study can contribute to the different studies that have already be conducted on this topic. This can be done by confirming whether or not the insights from this study correspond with findings of the other studies. If this is the case, the findings of those studies will become more reliable. Another contribution of this study is the cleaned dataset which can be uploaded to github, to provide the scientific community with a new dataset that is easier to work with and can be combined with other datasets.

1.2 Research questions

In order to define the scope of this study, one research question and two sub-research questions were formulated. The main research question captures the goal of this study in one question, while the sub-research questions supplement the main research question. The following research questions were formulated:

RQ: What type of recommender system performs best on the implicit data we have of the users?

The main goal is to objectively compare the collaborative model with the content based model, based on the chosen datasets.

SRQ1:What is the optimal number of latent factors for the collaborative model?

One very important aspect for the collaborative model, are the latent factors. These factors determine how complex or how basic your model is going to be. For recommender systems the optimal amount of latent factors may vary, since no dataset is the same. Therefore, this sub research question is included.

SRQ2:Does adding descriptions of games to the metadata impact the performance of the content based model?

For the content based model a selection of characteristics of the games is made, and for the most part these selected characteristics are very obvious. However, the model might actually perform better if it has more data, even though an entire description of a game seems very extensive.

1.3 Structure

In Chapter 2 I will discuss the research that already has been done regarding this topic and aditionally cover some of the core concepts and methods that will be used for study will also be discussed. Chapter 3 explains the datasets in more detail and discusses the setup of this study. Chapter 4 will then discuss the results of the models, and chapter 5 will interpret these results. Chapter 5 also describes what could have been done differently during the study, and what possibilities there are in terms of future research. To finalize the study, chapter 6 will simply answer the research questions, formulated in section 1.2.

2. Related Work

In this section the background information will be presented, as well as related research to the topic.

2.1 Steam

As stated in the introduction, Steam is a platform that sells digital copies of games, and allows users to play games via their platform. In 2003 Steam was developed and launched by Valve Corporation as a software client that allowed Valve to provide users automatic updates for their games. At the time, Steam provided this service for only seven unique games, but the platform slowly evolved into the biggest digital distribution platform for PC games. Currently, Steam offers a variety of over 30000 unique games and has approximately 90 million active users (Gough 2020). Since Steam also keeps track of a lot of interesting statistics and documents the characteristics of their games very well, their website is a great source for collecting data on games. Steam has a very easy to use API and a favorable policy regarding collecting and using data from their website. The result of this is that there are a lot of publicly available datasets already containing data on reviews, playtime, characteristics of games and so on. As stated in the introduction, the amount of games Steam offers, in combination with so many active users, makes Steam a perfect example of a platform that can benefit from the implementation of a recommender system.

2.2 Recommender systems

Recommender systems are tools that try to predict a users' possible future interests, by using the data that already exist on that specific user and his preferences. The concept of a recommender system was first introduced by Jussi Karlgren in 1990, where he explains in his paper that like-minded people might actually have very good hidden recommendations for each other (Karlgren 1990). Karlgren continued his work on the recommender system, and in 1994 published a paper where he applied the recommender system to predict recommendations for users of a news site (Karlgren 1994). This marked the beginning of the commercial application of recommender systems, as the potential benefits of an effective recommender were very promising. Some of these potential benefits included: selling more items, sell more diverse items and increasing the user satisfaction (Ricci, Rokach, and Shapira). Nowadays , almost all business that sell products online, make use of a recommender system in one way or another. Broadly speaking, there are two techniques used by these companies of recommending items to users. The first technique being the content based filtering method, which focuses on creating items profiles and user profiles, based on the contents of an item. The second technique being the collaborative filtering method, which looks for similar like-minded users, based their historical preferences.

2.2.1 Techniques. As mentioned before, the two main techniques for recommender systems are the collaborative filtering method and the content based filtering method. This section explains how both methods work and what their strengths and weaknesses are. Quick side note: this section mentions 'the active user' every now and then, which refers to the user we try to predict recommendations for.

2.2.2 Collaborative filtering. Collaborative filtering requires a user-item interaction matrix that contains information on the preferences of each user. This user-item interaction matrix has a MxN shape, where M is equal to the amount of unique users and N is equal to the amount of unique items. The values that are stored within the user-item interaction matrix are some measure of either explicit feedback, or implicit feedback. Explicit feedback is easier to use for recommender systems, as this type of data directly reflects the opinion of a user on an item, in contrast to implicit feedback (Hu, Koren, and Volinsky 2008).

When the user-item interaction matrix is constructed, there are two approaches to build a model that can generate recommendations. The first approach is the memory-based model which is a neighborhood-based prediction algorithm, and it makes use of the fact that like-minded individuals contain information about each other. It assumes that if we have two persons, and person 1 likes items X, Y and Z and person 2 likes items X and Y, that item Z is most likely a solid recommendation for person 2 (Hu, Koren, and Volinsky 2008). This concept is illustrated in figure 1. The memory-based model first computes the similarities between users, which is most commonly done by computing the pearson correlation coefficient between the given ratings (Herlocker et al. 1999). After the similarities are computed, the model will select the top K most similar users (or select all users above a certain similarity threshold), and compute the average ratings among them. Lastly, the highest rated items on average will be recommended to the active user.

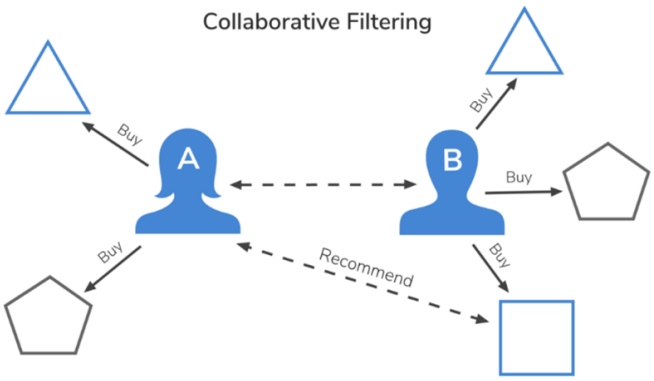

Figure 1: collaborative filtering

The second method is the model-based approach. This approach can use different machine learning algorithms such as bayesian networks, neural networks, clustering models and latent factor models to recommend items. With these algorithms, a model can be constructed, based on what it has learnt from the user-item interaction matrix. Recently, the latent factor models in particular have been gaining more popularity, due to their scalability and attractive accuracy (Yehuda Koren, Robert Bell, and Chris Volinsky 2009). The first time these latent factor models really stood out and outperformed the classic memory-based recommender models, was during the Netflix recommender system competition in 2009 (Yehuda Koren, Robert Bell, and Chris Volinsky 2009).

Collaborative models have the advantage that they can work on basically any item, without understanding the items themselves. Additionally, a collaborative model can also identify new interests for the users, if a lot of similar people also like a certain item.

One of the biggest problems collaborative models currently face, is the cold start problem (Lika, Kolomvatsos, and Hadjiefthymiades 2014). For new users, it is very hard for the model to generate meaningful recommendations, as it is impossible to calculate similarities between the new user and the existing users, as there is no data on the interests of the new user. This also applies to new items, since there is no feedback on these items.

2.2.3 Content-based filtering. Content-based recommender systems focus on the similarities between items, instead of similarities between users. It tries to recommend items, that are similar to the items that were previously liked by the active user. These similarities are based on the characteristics of an item, or the contents of an item. Content-based filtering therefore needs to two types of data, the user-item interaction matrix and a dataset that contains information about the characteristics and contents of the items.

First, the textual characteristics or contents have to be vectorized to a numerical vector representation, which is most commonly done with the term frequency-inverse document frequency (TF-IDF) function (Wang et al. 2018). This TF-IDF function outputs a vector where each word is represented with a measures the relevancy for that specific word. Then the most relevant words on average are selected from this output, and used to build a user and item profile. The list with the most relevant words for a user basically becomes the user profile, as they indicate what the users' interest are (Pazzani and Billsus 2007). Lastly, the cosine similarity between the user profile of the active user and the item profiles is calculated, and the most similar items are recommended to the active user. Figure 2 illustrates how a content based model works.

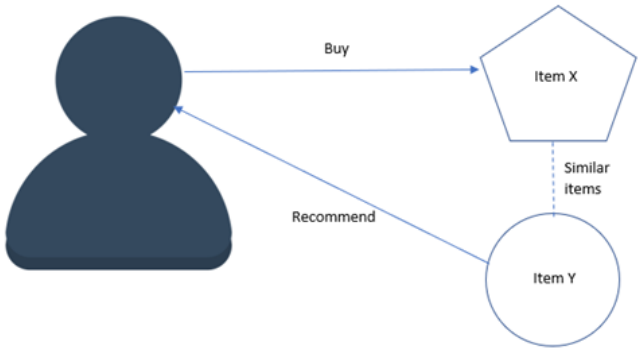

Figure 2: content based filtering

The big advantage of the content-based filtering method over the collaborative filtering method is that it does not suffer from the cold start problem. If a user has interacted with one item, the content based model can predict meaningful recommendations, while the collaborative model would struggle to find similar users.

The biggest downside of the content-based model, is that it only looks at similar items, and does not include the opinions of other users. This means that it can't look beyond the scope of the previous liked items of the active user, which means that all recommendations will be similar to the previous liked items.

2.3 Matrix factorization

For the collaborative model in this study, the choice was made to go with the model-based approach, and more specifically, to build a Singular value decomposition (SVD) model. This model was popularized by Simon Funk in 2009 when Funk ranked high in the Netflix recommendation competition, and even shared his code with the world before the end of the competition, so that others could use it (Piatetsky). There are more Matrix factorization-based models, but for the sake of this study, the scope is limited to the SVD algorithm.

SVD breaks the user-item interaction matrix down into three separate, smaller matrices. The size of these smaller matrices are determined by the amount of latent factors the SVD had, and can impact the performance of a model significantly. When the original matrix is decomposed, the models puts out separate user matrix, a separate item matrix and a diagonal matrix, containing the singular values, which can be seen as weights. The dot product of these three matrices will result in a matrix that contains the recommendations for all users (Becker 2016).

2.4 TF-IDF

To vectorize the characteristics of the games, the term frequency-inverse document frequency(TF-IDF) vectorizer is used. TF-IDF is a vectorizer that can calculate the relevancy of words in a document by analyzing their occurrences. This vectorizer is the most common one for vectorizing characteristics of items for content based models (Beel et al. 2016), because it can vectorize the metadata on the items in a more meaningful way, than a traditional count vectorizer. Some other advantages of the TF-IDF vectorizer include filtering out stop words, and not penalizing different lengths of the documents that include the characteristics . After the characteristics are vectorized, the similarity between items can bet calculated using the cosine similarity, as stated in section 2.2.3.

3. Experimental Setup

This section explains what datasets have been used for this study, how these datasets were preprocessed, what models were made and how these models can be evaluated.

3.1 Users dataset

The first dataset is a popular dataset about steam users and their playtime per game, called the steam-200k dataset. It includes 12393 unique users, 5155 unique game titles and has a 199999 x 5 shape. The dataset was downloaded from kaggle, and provided by the Tamber Team (Tamber Team 2016).

The dataset was first cleaned, so that it was easier to work with the data. First, all titles were removed that polluted the dataset, such as DLC's, season passes and special editions, as they contained very little information, and aren't really main game titles. Another reason for deleting these types of titles, is that they are really easy recommendations. If for instance, game X has three DLC's and the recommender system knows a user owns game X, the three DLC's will probably be recommended, and thus pollute the recommendations.

Secondly, the cold start problem had to be taken into account. Cases where the user owned less than five games, or where a game is played for less than 25 hours in total (over the entire dataset), were removed. These numbers are arbitrarily chosen. If a user owns less than five games, it is hard to find similar people, which leads to meaningless recommendations. This also applies to games which haven´t been played much, as there are too little players (sometimes even none at all) who liked that game, and it will end up not being recommended at all.

Lastly, the decision was made to transform the playtime data, into explicit ratings, as this works better for the SVD algorithm. This was inspired by (Parra and Amatriain 2011), as they showed that there is a strong relation between implicit feedback and explicit ratings. They also describe that adding a 'recentness' component to this transformation has a significant positive effect on the transformation. Unfortunately, this dataset does not contain any information about when a certain game is played for the last time, which makes adding a recentness component to the transformation impossible. Thus the transformation of the implicit data to explicit ratings was fairly basic. For each game the global average of playtime was calculated, which was then used to map the individual playtimes to a rating between one and five. For each rating there were the following thresholds: if a user played more than average, a rating of 5 was given, if a user played 80% or more of the average playtime, a rating of 4 was given, if a user played 50% or more of the average playtime, a rating of 3 was given, if a user played 10% or more of the average playtime, a rating of 2 was given and lastly if a user played 10% or less of the average playtime, a rating of 1 was given.

Now that the dataset was ready to be used, the user-item interaction matrix was constructed. As stated in section 2.2.2, the user-item interaction matrix is a MxN matrix, where M is equal to the unique number of users, and N is equal to the unique number of items. First a MxN zero matrix was created, to then be filled with the calculated ratings for each user. Appendix A shows a screenshot of what the matrix looks like.

3.2 Games dataset

The second dataset contains information about the characteristics and descriptions of 27075 unique titles that are available on steam. There are 21 unique features in this

dataset, which gives the dataset a shape of 27075x21. This dataset was downloaded from kaggle, and provided by Nik Davis (Davis 2019).

As this dataset is only used for the content based model, a selection of relevant features had to be made. The following features were included for the model: developer, categories, genre, tags and description. All other irrelevant features were dropped, and to make it simpler for the modelling later on, all the features were merged into one big feature, called the metadata feature. The metadata feature first needed to be tokenized, so that it could be vectorized later on. Accordingly, all punctuation and inconvenient spaces between names were removed, so that the feature could be vectorized. As stated in section 2.4, the most commonly used method for vectorization is the TF-IDF vectorizer, which was also used for this dataset.

3.3 Combining the datasets

For the content based model, the user-item interaction matrix needed to be combined with the games dataset from the previous section. Initially, the idea was to combine the models based on the game titles, as the assumption was that the game titles would match. Unfortunately, not all game titles of the user-item interaction matrix could be found in the games dataset, as a lot of titles weren't a one on one match. Therefore a large proportion of the steam-200k dataset had to be removed, since a part of the titles could not be found in the games dataset. To find as much titles as possible, the titles were first stripped from all odd symbols and punctuation, and then all titles were lower cased. Removing the games that could not be found eventually resulted in the size of the steam-200k dataset being reduced from 199999x5 to 57579x5. Lastly, the steam app ID was also added to the steam-200k dataset, so that each game had an unique ID, which referred to the same game in both dataset and the user-item interaction matrix.

After the datasets were matched, the new user-item interaction matrix could be constructed following the same procedure as described section 3.1. An user-item interaction matrix with the size of 2814x943 was the result.

3.4 Method / Models

This subsection will discuss the different models, and how they were constructed.

3.4.1 Baseline model. A commonly used baseline model for recommender systems, is the popularity model. This model recommends items from the top 10 most popular items, which the user hasn't interacted with yet. For this study, the popularity model is set to the baseline model.

3.4.2 Collaborative model. For the collaborative model, the matrix factorization model called Singular Value Decomposition(SVD) from scipy is used. As stated in the relevant works section, SVD decomposes the original matrix into three smaller matrices. In this case, the first two decomposed matrices U and V, will represent the users matrix and the items matrix respectively. The third matrix S is a diagonal matrix filled with singular values, which can be seen as weights. If we call our user-item interaction matrix R, we can formulate the following equation (Becker 2016):

$$R = USV^t$$

The most important part of the SVD are the number of latent factors. Latent factors are basically the most important features the SVD can extract from the original dataset, and they decide the size of the decomposed matrices (Becker 2016). In general, the more latent factors there are, the better the decomposed matrices can reconstruct the original matrix. Although more latent factors might seem like a good thing, this downside of having more latent factors is that the model is prone to over fit. To test how much latent factors were optimal for this study, two models were tested, one with 10 latent factors and one with 50 latent factors. These values were initially chosen to determine whether our dataset could benefit from a more complex decomposition of the dataset, or a more basic one. Later the model was tuned to the optimal amount of latent factors.

Another important aspect for the collaborative model is the input. To test if changing the input would impact the performance of the collaborative model, a model with a function that scaled all the ratings down was included as well. The function was a simple log transformation, which was already provided by the notebook of Gabriel Moreira (Moreira 2019). If R is our new rating and r is our old rating, the function can be written as follows:

$$R = \log_2(r)$$

The last thing that needed to be done was actually constructing the matrix that contained all the recommendations for each user. This matrix was constructed by first calculating the dot product of matrix U and S, and then multiplying this outcome with matrix V. The result is the approximation of the original matrix at K latent factors, and at the same time contains no zero values anymore, as can be seen in appendix 2.

Then for each user, the values within the matrix are sorted and the highest ranked ones can be recommended to the user. Figure 3 illustrates the process for the collaborative model.

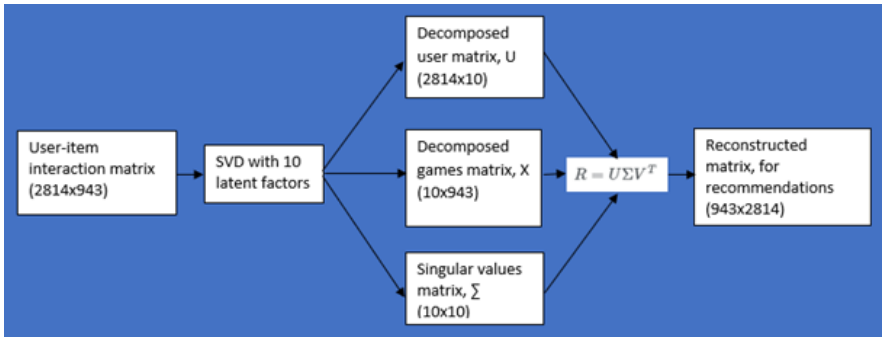

Figure 3: SVD

3.4.3 content based model. The content based model makes use of the different characteristics that each game has, and looks for similarities within those characteristics. As stated in section 2.2.3, the content based model constructs item and user profiles, which it uses to recommend items. Since the recommendations of the content based model are heavily influenced by the metadata feature, there were three variations of metadata, to see which one was performing best. The first and the most basic model included only the relevant characteristics of a game. The second variation of metadata combined

the characteristics with a detailed description of the game, and the last one only included the detailed description. When all the selected characteristics were merged into the metadata feature, the feature could be tokenized and vectorized, as explained in section 3.2. The TD-IDF vectorizer assigns values to each word in the metadata, which represent the relevancy of that word in the metadata. This results in each game having a vector, filled with values for all words within its metadata, where the highest values correspond with the most relevant words for that game. The vectors that store these values, represent the item profiles (Pazzani and Billsus 2007). An example of an item profile can be found in appendix B.

To build the user profiles, the user-item interaction matrix was needed again to find the games each user interacted with. Additionally, the ratings calculated for each interaction were also taken into account, as they served as a weight. It makes sense to weigh the relevant words from liked games heavier than the relevant words of games that the user didn't like as much. First, all item profiles of the items a user interacted with, were extracted from the matrix that stored these profiles. Then, all extracted item profiles get multiplied by the individual ratings that were given for each item. This makes it so that the most liked items profiles contain higher values than the less liked items. After the multiplication, the average relevancy per word was calculated and sorted. The result is a user profile, where the most relevant words for that specific user are at the top. For this study, a function was used that iterated over all unique users to build all the user profiles. An example of a user profile can be found in appendix C.

After these profiles were constructed, the cosine similarity between the user profiles and the item profiles were computed, and the item profiles that are the most similar to the user profile, got recommended to the user.

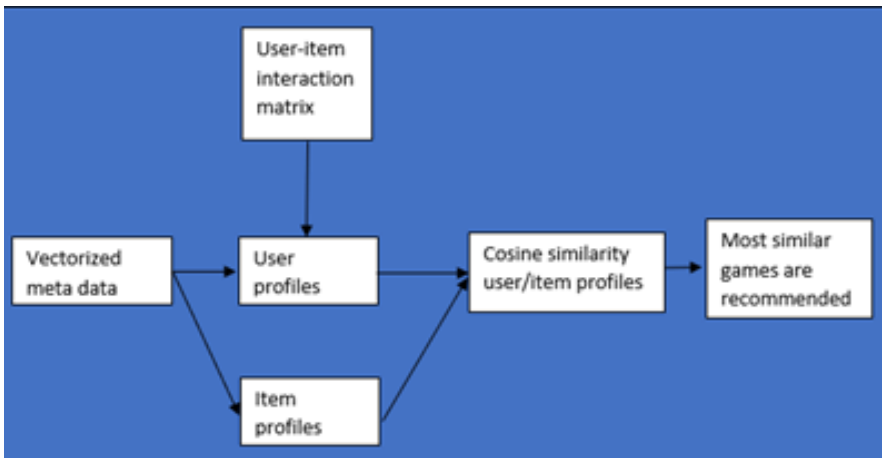

Figure 4: Content based RS

3.5 Evaluation

In order to measure the performance of the different models in an objective manner, the data was first splitted in a training set and a testing set. This allowed hyper parameter tuning, as the change in performance can be seen when the model gets tested on the test set. The split was a 80/20 split, and both the training and test set, were randomly selected.

For the evaluation metric, the top-n accuracy was used, as this is an easy metric to compare across different methods (Cremonesi, Koren, and Turrin 2010). The top-n accuracy works as follows:

For each game that's in the test set, another 100 random games were selected, and it is assumed these games are not relevant for the user. After the selection, the models were used to compute the recommendation values for all of the selected games (100 random ones and 1 relevant one). All these 101 recommendation values were then sorted from high to low and the top N recommendations of that list are selected. If the test game is included within this top N, we have a hit, otherwise a miss. With these hit or miss rates we can calculate the recall metric at a certain N, which is called the Recall@N metric.

With this evaluation metric two collaborative models were tested with different amounts of latent vectors, to get an idea of where the optimal amount would be, one with 10 latent factors and one with 50 latent factors. The model with 10 latent factors performed significantly better. After running several tests with models that had less than 10 latent factors, the optimal number seemed to be 7.

4. Results

This section discusses the results that were obtained per different model. It will be split up in three parts, one for the popularity recommender, one for the collaborative model and one for the content based model.

4.1 Popularity

The popularity model is a very strong baseline model, which makes sense, since it recommends items that are liked by the majority of the users. For this study, the popularity scored a recall@5 of 0.422, and a recall@10 of 0.558, which are fairly decent scores for a baseline model.

4.2 Collaborative

As stated in section 3.5, the models were trained on the training set, and tested on the test set. For the collaborative model, two models were tested, one with 10 latent factors and one with 7 latent factors. For each of the models there was one other variation of the model added where the ratings, basically the input, was smoothed out. This leaves a total of 4 models being tested for the collaborative models.

The difference in performance between the models with 10 latent factors and 7 latent factors was very little. However, smoothing out the ratings did improve the performance of both models by quite a bit. The vanilla SVD@10 LF, scored a recall@5 of 0.483 and a recall@10 of 0.613. If the ratings are then smoothed out, the recall@5 was pushed to 0.505 and the recall@10 to 0.638. For the vanilla SVD@7 LF, a recall@5 of 0.489 and a recall@10 of 0.619 was obtained and with the added smoothing for the ratings the model scored a recall@5 of 0.512 and a recall@10 of 0.643. A clear overview of the collaborative models and their performance can be seen in figure 5.

MODEL	Recall@5	Recall@10
SVD @10 LF	0,483	0,613
SVD @10 LF + smooth ratings	0,505	0,638
SVD @7 LF	0,489	0,619
SVD @7 LF + smooth ratings	0,512	0,643

Figure 5: Collaborative models

4.3 Content based

For the content based three models were built, as stated in 3.4.3. The first model only looks at the more general characteristics of a game, such as category, genre and game tags. The second model combines the same characteristics of the first model, with a detailed description of the game, to see if adding a description improves performance of the model. The last model only takes the description of a game into account. The first model, which is the most basic model, scored a recall@5 of 0.312 and a recall@10 of 0.436. The second model, which combined the characteristics and the description of a game, scored a recall@5 of 0.328 and a recall@10 of 0.479. Removing the characteristics of a game from the game, hurt the performance significantly. The recall@5 for the last model was 0.244 and the recall@10 0.379. A clear overview of the content based models and their performance can be seen in figure 6.

MODEL	Recall@5	Recall@10
Content	0,312	0,436
Content + description	0,328	0,479
Content + description only	0,244	0,379

Figure 6: Content based models

4.4 Comparing models

In order to objectively compare the models, only the best performing model for each method were selected. As the results show, the collaborative model works best for our dataset on steam games. The recall@5 score for the best collaborative model was 0.184 higher than the best content based model. For the recall@10, the difference between the collaborative model and the content based model slightly decreased to 0.164, obviously still in favor of the collaborative model. Figure 7 shows all the models that were tested, and their performance.

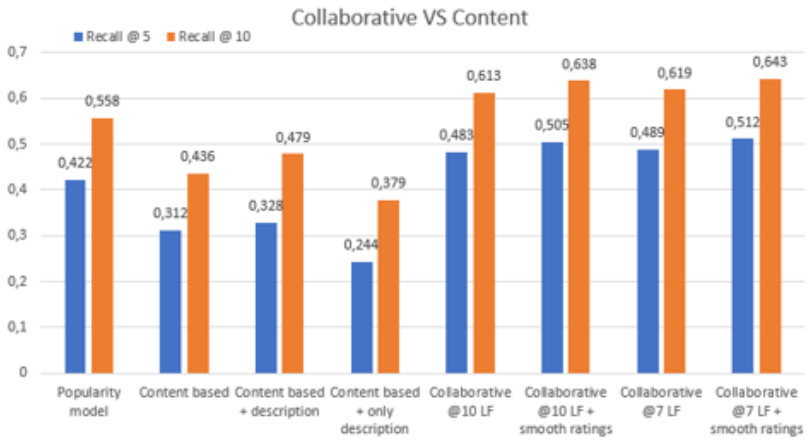

Figure 7: All models

5. Discussion

In this section the results will be interpreted, and the setup of this study will be evaluated. Furthermore, the limitations of this study and some suggestions for further research will be discussed.

5.1 Results

For the collaborative models I expected the model with 7 latent factors to perform best, considering that the dataset for this study was not that big. Although it was somewhat surprising to see the model perform slightly better when the ratings were scaled down. However, to see the content based model with the added description outperform the other models was surprising. One could argue that because you add a lot more data to the model, some of the valuable characteristics of the games could suffer from this added data, as you are polluting the metadata with a lot more words that aren't necessarily relevant. I expected that the plain simple genres, categories and tags would be the best model for the content based model, as they practically describe the game in less words. My guess is that the TF-IDF does a really good job at calculating the relevancy of each word, and that the pollution in the metadata feature doesn't really hurt the TF-IDF vectorizer.

Now the overall conclusion we can derive from comparing the models, is that the collaborative model is the superior model in terms of predicting meaningful recommendations, which is in line with the findings of other studies like (Yehuda Koren, Robert Bell, and Chris Volinsky 2009). Although the comparison within the study was easy, it is unfortunate that we can't really compare absolute performance of the models between different studies, as the top N accuracy metric is influenced by so many factors. Just the fact that no dataset is the same, and that the quality of different datasets varies wildly, makes it almost impossible to objectively compare my best model to someone else's model.

5.2 Data

First, the datasets seemed very promising, but after working with the data for a while, there were some unfortunate issues with the datasets. First of all, a large proportion of the users didn't play enough games to be valuable for the recommender system. These players had to be deleted from the dataset, which resulted in a dataset that included only 2814 unique users instead of the original 12393 unique users. Secondly, combining both datasets gave some issues, as the titles of the games didn't match one on one. In an attempt to counter this issue, the datasets were matched based on 'most' similar titles, which was calculated using the Levenshtein distance. Although this method found 20 titles more, it did have its flaws. Some titles ended up being matched to titles which weren't the right ones. Since the Levenshtein method only found 20 titles more and was not reliable, the decision was made to stick with just deleting the games that weren't a one on one match. This leaves the dataset with only 2814 unique users and 943 unique games, which is relatively small compared to some of the commonly used datasets for testing recommender systems like the MovieLens dataset (MovieLens). Although the dataset this ended up in being a limiting factor for this study, the cleaned dataset might be of use for other research purposes.

For further research and testing, there is one publicly available dataset on steam users and games, which includes information on 109 million users, provided by Marc

O'Neil, and his colleagues at Brigham Young University (O'Neil 2016). The only downside of this dataset was the fact that it was a SQL dataset, which I'm not familiar with. Due to the limited time there was for this study, I could not explore the possibilities with this dataset.

5.3 Models

For the collaborative model there several ways of improving the performance of the model. Currently the SVD algorithm was used to build the collaborative model, but there are a lot of different matrix factorization models. One model in particular that seems promising, is the SVD++ model. In most research papers where the SVD++ model is tested, it performed better than the SVD model, and it seems like it could potentially work on the datasets for this study (Xian et al. 2017).

Another technique that was not covered in this study are hybrid models, which is a combination of the collaborative model and the content based model. This technique could also potentially be tested on this dataset, although I don't expect a big performance boost. Boon did research to the hybrid models on a similar dataset, where the hybrid recommender only performed 3,7 percent better than his traditional collaborative model (Boon 2019) . The content based model was performing quite well for this dataset, and so I wouldn't change much about the model itself.

5.4 Summary

The real limitations for this study were the time in which the study had to be completed, and the quality of the dataset. If there was enough time to either match both the datasets in a better way, or to explore the possibilities of the alternative dataset mentioned in section 5.3, the results and findings of this study could have been more decisive.

Exploring the different techniques for the collaborative model, could have been very interesting, but testing so many variations of the collaborative model, could also become overwhelming. In terms of performance of the collaborative models, there also was less to be gained here, since the basic but solid SVD technique for the collaborative model performed quite well.

Finally the conclusion of this study is that collaborative models work better for recommending items than the content based models, which is in line with the findings of other studies.

6. Conclusion

Finally, in this section the research questions will be answered.

RQ: What type of recommender system performs best on the implicit data we have on the users?

For the dataset that was used for this study, the collaborative model scored a higher recall@5 and recall@10 percentage than the content based model, thus we can conclude that the collaborative model is superior to the content based model. However, this does not mean that content based models are useless, as they can recommend more niche items than the collaborative model, which actually might be preferred in some situations.

SRQ1 : What is the optimal number of latent factors for the collaborative model?

First the collaborative model was tested with 10 latent factors and 50 to determine if a more complex SVD model would be perform better on this dataset. Luckily, the more basic model performed better and after finetuning the model for some time, 7 latent vectors seemed to be the best number of latent vectors.

RQ2: Does adding descriptions of games to the metadata impact the performance of the content based model?

The model which only contained the characteristics of a game, performed slightly worse than the model where the description of games were added. For this dataset, adding more relevant information to a content based model, improved the quality of the recommendations.

References

Aggarwal, Charu C. and Charu C. Aggarwal. 2016. An Introduction to Recommender Systems. In *Recommender Systems*. Springer International Publishing, pages 1–28.

Becker, Nick. 2016. Matrix Factorization for Movie Recommendations in Python - nick becker.

Beel, Joeran, Bela Gipp, Stefan Langer, and Corinna Breitinger. 2016. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338.

Bolding, Jonathan. 2019. Steam now has 30,000 games | PC Gamer.

Boon, Justin V. 2019. DESIGNING A HYBRID RECOMMENDER SYSTEM FOR STEAM GAMES. Technical report.

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin. 2010. *Performance of Recommender Algorithms on Top-N Recommendation Tasks General Terms*.

Davis, Nik. 2019. Steam Store Games (Clean dataset) | Kaggle.

Gough, Christina. 2020. Number of Steam users 2020.

Herlocker, Jonathan L., Joseph A. Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pages 230–237, Association for Computing Machinery, Inc, New York, New York, USA.

Hu, Yifan, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. Technical report.

Karlgren, Jussi. 1990. An Algebra for Recommendations An Algebra for Recommendations Using Reader Data as a Basis for Measuring Document Proximity. Technical report.

Karlgren, Jussi. 1994. Newsgroup Clustering Based On User Behavior - A Recommendation Algebra.

Lika, Blerina, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. 2014. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4 PART 2):2065–2073.

Moreira, Gabriel. 2019. Recommender Systems in Python 101.

MovieLens. MovieLens | GroupLens.

O'Neil, Justin; Vaziripour Elham; Zappala Daniel, Mark ; Wu. 2016. Steam Dataset.

Parra, Denis and Xavier Amatriain. 2011. Walk the Talk Analyzing the relation between implicit and explicit feedback for preference elicitation. Technical report.

Pazzani, Michael J. and Daniel Billsus. 2007. Content-based recommendation systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4321 LNCS, pages 325–341, Springer Verlag.

Piatetsky, Gregory. Interview with Simon Funk. Technical report, KDnuggets.

Ricci, Francesco, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook.

Tamber Team. 2016. Steam Video Games | Kaggle.

Wang, Donghui, Yanchun Liang, Dong Xu, Xiaoyue Feng, and Renchu Guan. 2018. A content-based recommender system for computer science publications. *Knowledge-Based Systems*, 157:1–9.

Xian, Zhengzheng, Qiliang Li, Gai Li, and Lei Li. 2017. New Collaborative Filtering Algorithms Based on SVD++ and Differential Privacy. *Mathematical Problems in Engineering*, 2017.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. Technical report.

Appendix A: User-item interaction matrix

contentId	10	30	50	70	80	130	220	240	280	300	...	391540	392470	393420	402840	403190	405980	407120	417860	608580	890400
personId																					
5250	0.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
76767	5.0	0.0	1.0	3.0	0.0	1.0	0.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
86540	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
103360	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
144736	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
181212	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
229911	0.0	2.0	1.0	2.0	5.0	1.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
298950	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	...	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
299153	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
381543	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 943 columns

Appendix B: Item profile of Skyrim

```
array([0.11039685, 0.11301022, 0.10805911, 0.08521391, 0.10059346,
       0.10805911, 0.11166508, 0.11166508, 0.10223777, 0.09906265,
       0.08685822, 0.09056482, 0.10594437, 0.10919721, 0.08058206,
       0.10401376, 0.11301022, 0.14087357, 0.0802628 , 0.10697656,
       0.10919721, 0.07621745, 0.07963774, 0.10223777, 0.20991615,
       0.10805911, 0.08331634, 0.0856137 , 0.21907417, 0.1017869 ,
       0.07814806, 0.1586634 , 0.0950173 , 0.17755246, 0.10385931,
       0.04990349, 0.04934577, 0.07348699, 0.17287115, 0.08190711,
       0.07621745, 0.11039685, 0.12611709, 0.07994807, 0.08772845,
       0.11039685, 0.25196474, 0.21821631, 0.09213168, 0.06070621,
       0.12141243, 0.06057531, 0.06552642, 0.07279715, 0.11241208,
       0.07595487, 0.0603158 , 0.07372179, 0.13587358, 0.10140039,
       0.24230125, 0.06410464, 0.04880173, 0.09628553, 0.05918415,
       0.05348574, 0.04585814, 0.03805883, 0.05578138, 0.05006544,
       0.04661034, 0.04819588, 0.10603326, 0.06365026, 0.08331634,
       0.13773059, 0.04559081, 0.06018717, 0.09884918, 0.03796441,
       0.05567647, 0.16609407, 0.0400368 , 0.067576  , 0.04500084,
       0.03967768, 0.0793317 , 0.0836831 , 0.12611709])
```

Appendix C: User profile

	token	relevance
0	strong	0.310708
1	isaac	0.204231
2	strong br	0.143215
3	play	0.137881
4	rogue	0.123555
5	magic	0.122033
6	items	0.116970
7	support	0.116300
8	rpg	0.109794
9	roguelike	0.108278