Daniel Ellmeier, BSc

Implementing and Evaluating a Recommendation Framework for Board Games

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Frank Kappe

Institute of Interactive Systems and Data Science

Dipl.-Ing. Lukas Eberhard, BSc

Graz, Oktober 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

Recommender systems are necessary to filter desired items from undesired items. This thesis deals with the domain of board games and addresses the top-k recommendation task by building and evaluating a recommendation framework in Python.

For crawling and preprocessing, the largest board game collaboration website *boardgamegeek.com* is used as a data basis. The framework itself accepts desired and undesired board games as well as optional board-game-specific constraints as input. Four different approaches, of which two are collaborative-based and the others are content-based, create recommendations for board games and form the main components of the framework.

For evaluation, real-user search queries and community-approved recommendations are extracted manually from a board game message board on *reddit.com*. The performance of the framework is evaluated on a test set and compared against a most-popular baseline with the help of classification accuracy metrics such as precision, recall and f1-score.

To further improve the accuracy, the framework is extended by a sequential-ensemble recommendation approach. The top-k recommendations from each approach are reordered by a post-filtering technique which utilizes different board game specific attributes. Each post-filter is provided with an optimized weight from a training phase and the performance gets evaluated and compared on the test set.

Contents

List of Figures

List of Tables

1 Introduction

More and more data gets produced every year. This data is composed of emails, websites, file transfers and more modern accesses such as social media, instant messaging or streaming services.

A study of Berkley (2003) estimated that the amount of data created in 2003 was 530 petabytes. Eight years later, a study from IBM (2011) estimated that around 2.5 quintillion bytes of data were generated every day, meaning 2 500 petabytes only in one day.

This means that the amount of data produced during the whole year 2003, was produced within just five hours in 2011.

Recommender systems nowadays have the goal to filter, select and process desired information from undesired information for a user. As the availability of items steadily increased the need for recommendations evolved, particularly in e-commerce systems.

The problem is also known as the long tail phenomenon (Rajaraman and Ullman, 2011) and expresses the more items are available, the more difficult it is to select proper items for a user. Especially unpopular items, with a low conversion rate, are located in the so-called long tail. A recommender system can help to explore the long tail properly and efficiently.

The steps to get recommendable knowledge from data can be explained on the basis of the Knowledge Discovery Process initially defined by Fayyad, G Piatetsky-Shapiro, and Smyth (1996).

This thesis tries to address the recommendation demand for board games. Especially because the domain of board games is not a frequent subject of scientific research, as opposed to other domains, such as movies.

The following research questions were derived:

1. In the domain of board games, how well are standard recommendation approaches performing when applied to real-user search queries?
2. Can content-based approaches be particularly advantageous over collaborative-based ones?
3. What improvements can be achieved by reordering the top recommendations of the respective approach?

The largest online board game platform *boardgamegeek.com* was crawled, preprocessed and used as data basis. A recommendation framework with four different approaches recommends board games based on user requests from the discussion website *reddit.com*.

The recommended items are evaluated using accuracy metrics towards the community approved items. In addition, a sequential-ensemble approach, called *post-filtering*, is used to further improve the accuracy by re-ranking the recommended items.

2 Background

This chapter gives a theoretical background about recommender systems. First of all, frequently used terms will be defined followed by task and goals, basic models and evaluation techniques of recommender systems. Additionally, related work regarding board games and the data basis, *boardgamegeek.com*, will be presented.

2.1 Recommender Systems Overview

Recommender systems assist in "making choices without sufficient personal experience of the alternatives" (Resnick and Varian, 1997). This means based on preferences of a user, recommendations are made.

A wide diversity of applications for recommender systems exist. Rajaraman and Ullman (2011) classify applications into three major groups which are *product recommendations*, *movie recommendations* and *news articles*.

Nowadays, several other recommendation domains have evolved. To name a few popular ones, *Facebook.com* recommends friends, posts, comments and similar, *Tripadvisor.com* recommends hotels and restaurants and *Netflix.com* recommends movies and series.

2.2 Terminology

A recommender system works with data that can be classified into three groups, *items*, *users* and *transactions* (Ricci et al., 2011).

2.2.1 Items

Items are the objects which get recommended. They have different features and properties which are often domain specific. For instance, a board game is assigned to a certain "boardgamecategory" and usually has a certain description.

2.2.2 Users

Users interact with the recommender system. Different properties of a user may be of interest for different kinds of recommender systems. For a collaborative model (see Subsection 2.4.1), user ratings play a significant role whereas for a demographic system (see Subsection 2.4.4), demographic attributes of a user, such as age and gender, are more important.

2.2.3 Transactions

Transactions are interactions between a user and the recommender system. They can occur as search queries for a certain item or in a more direct form as ratings. A rating represents an association between a user and an item.

Schafer, Frankowski, et al. (2007) define the following types of ratings:

- *Scalar Ratings* consist of a discrete set of ordered numbers or preferences. The set can consist of either numerical ratings such as 1,2,3,4,5, where 5 represents a strong like and 1 a strong dislike, or ordinal ratings presenting the user's level of interest such as Like, Neutral, Disagree. These ratings can also be unbalanced meaning the number of positively classified elements predominates. If a neutral element is missing it is also referred to as a "forced choice rating system" (Charu C. Aggarwal, 2016).
- *Binary Ratings* are ratings with exactly two possibilities, whereas one is positive and the other one is negative. For example, good/bad or like/dislike.

- *Unary Ratings* only give usable information if they are specified. For example, a user purchased an item and therefore a unary rating is set. If there is no rating, no information can be inferred.

Ratings can be collected either implicitly or explicitly (Schafer, Frankowski, et al., 2007). Implicit means that the rating is inferred from the action of a user, for example he added an item to his shopping cart. Where on the contrary, explicit ratings require the user to rate an item intentionally.

2.3 Tasks and Goals

Different definitions exist for the tasks and goals of a recommender system. Ricci et al. (2011) declared that the goals of a recommender system depend on the perspective of the system stakeholders, with the most important ones, being end-user and (service) providers.

Herlocker et al. (2004) identify eleven domain-independent tasks of a recommender system from an end-user perspective.

Ricci et al. (2011) define several goals of a recommender system but from the provider perspective.

Some of them include:

- *Increase the conversion rate:* In order to maximize profit.
- *Provide diversity:* Without a recommender system, it could happen that a user is always stuck at popular items.
- *Satisfy, tie and understand the user:* The user should find the recommendations appropriate and interesting. This, in turn, unconsciously ties the user more and more to the provider. Getting a deeper insight in what the user wants, may also help the provider with strategic decisions, for example adapting its stock and production.

A good example how to understand and tie the user is enforced by *Netflix.com*. They go a step further by showing the user why a certain movie or series gets recommended to them, raising the so-called "Personalization Awareness" (Netflix, 2012). This is also suggested by Pu et al. (2011) in order to raise the *trust* of the user.

According to Charu C. Aggarwal (2016), there are two primary models, how the recommendation goal can be achieved. The first model is to *predict* the rating value for a user-item combination.

The other model does not express any ratings, as it just outputs the *top-k* items. This is also known as the *top-k recommendation task* or *top-N recommendation task* (Cremonesi, Koren, and Turrin, 2010).

Especially in commercial systems, the absolute values of the ratings are more negligible for a consumer and not applied in popular e-commerce applications (Schafer, Konstan, and Riedl, 1999; Linden, Smith, and York, 2003).

2.3.1 Top-K Recommendation Task

In top-k recommendation models, the proper value of k has to be selected. Values for k are not only application dependent, the optimal amount of items further depends on usability considerations. Pu et al. (2011) claimed that the usual list length ranges from 5 to 20 and concluded in one of their guidelines:

"Displaying more products and ranking them in a natural order is likely to increase users' sense of control and confidence."

However, simply increasing the recommendation list length, results in an increased recall and decreased precision (Shani and Gunawardana, 2011). Some popular examples of applications with different top-k values are:

The movie platform *IMDb* presents 12 movies, distributed on two pages, which a user may like. *Amazon.com* displays different amount of items depending on the screen resolution and user agent of the visiting browser. *Google.com* also varies the amount of advertised links depending on the screen resolution.

2.4 Basic Models of Recommender Systems

Models of a recommender system can roughly be categorized by the kind of data used (Charu C. Aggarwal, 2016). A model which uses transactions, such as ratings or behavior patterns, falls into the category of *collaborative filtering* approaches. On the contrary, models which make use of item properties or features can be classified as *content-based* recommender systems.

Ratings are presented by a so-called rating matrix where the columns usually represent the items and the rows represent the users. A content-based recommender system may also use the ratings of the rating matrix but only for a single user (Charu C. Aggarwal, 2016).

Charu C. Aggarwal (2016) names five different models which occur in the literature and real world applications.

2.4.1 Collaborative Filtering

Schafer, Frankowski, et al. (2007) define it as the process of "filtering or evaluating items using the opinions of other people". A rating matrix is usually rather sparse, meaning most users have only rated a small portion of the items available. Such ratings are called specified or observed, whereas entries in a rating matrix which do not have a value are called unspecified or missing. The idea behind collaborative filtering (CF) is, based on the ratings of similar users or items, to infer the missing ratings for a user on an item.

Breese, Heckerman, and Kadie (1998) distinguish the following two variants of collaborative filtering:

2.4.1.1 Memory-based

Memory-based or also often referred as *neighborhood-based* methods consider the neighborhood of users or items. In particular, missing entries are inferred by looking at the neighborhood of most similar users (*user-based CF*) or items

(*item-based CF*). *User-based CF* starts by taking the k most similar users to a target user, for whom missing ratings of items are computed.

Different methods exist to address the disparate rating habits of users. One method used for similarity calculation is the *Pearson correlation coefficient* which was initially mentioned, in the context of recommender systems, by Resnick, Iacovou, et al. (1994) for their famous *GroupLens* recommender project. Other similarity functions, such as the raw cosine similarity, exist but the Pearson correlation has the advantage of taking into account the different levels of generosity of a user (Charu C. Aggarwal, 2016).

Usually, the top-k users with the highest similarity score are used, as a so-called peer group, to predict the missing rating for an item.

Item-based CF uses the k most similar items as peer groups instead of users. Prediction functions such as the *weighted sum* (Sarwar et al., 2001) use the observed ratings of the target user from the most similar item's peer group to estimate the missing rating.

According to Charu C. Aggarwal (2016), item-based CF often provide more relevant recommendations because ratings of the target user are incorporated into the estimated rating. In addition, an application can directly explain why a certain recommendation was selected and thereby raising *Personalization Awareness*. This is difficult to nearly impossible for user-based CF, primarily due to anonymization of user data.

2.4.1.2 Model-based

In model-based CF, a built up model of the data is created in advance, by using supervised or unsupervised learning methods. The model building phase is thereby clearly separated from the prediction phase.

Popular machine learning methods of model-based CF include decision trees, rule-based methods, Bayes Classifiers and latent factor models such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). Matrix factorization approaches such as SVD decompose the original rating matrix and create low-rank approximations which are used for estimating unspecified entries.

2.4.1.3 Cold-Start Problem

One of the challenges of CF is the so-called cold-start problem where the initial amount of ratings is very low. It occurs due to new users as well as new items. Knowledge-based or content-based systems can better cope with such sparsity.

2.4.2 Content-Based

Content-based recommender systems use attributes of items to make recommendations. A user profile is usually built from the user's feedback about several items. This feedback may be either implicit, which may be derived by the user's actions, or explicit which may be defined by the user's ratings.

The steps of a content-based recommender system can be divided into three main processes which are *preprocessing and feature extraction*, *learning of user profiles* and *filtering and recommendation* (Charu C. Aggarwal, 2016).

The first two steps are often executed *offline*, meaning the necessary actions can be pre-calculated and therefore done before a recommendation is requested. This allows the last step, which is executed *online*, with the actual recommendation to be executed much faster.

Depending on the domain, different feature selecting and preprocessing steps may be applied. In the domain of board games, the official description could be a comprehensive source for features. The conversion of textual representation into significant features is achieved by so-called feature selection and feature weighting methods. *Bags of words* are extracted from the textual representation and further extraction steps are done to reduce features which are less descriptive.

One simple form of feature cleaning, called *Stop Words Removal*, is achieved by removing common or not domain-specific words such as articles, conjunctions and so on from the preliminary feature space. *Stop Words Removal* belongs to the unsupervised feature selections methods as the user's ratings

are not taken into account, for defining the significance of a feature (Charu C. Aggarwal, 2016).

Different methods can be applied when learning user profiles. Unlabeled entries, which are unrated items, of a certain user can be estimated by a classification model. One well-known and easy to implement classifier is the *Nearest Neighbor Classifier*, also called *kNN classifier*.

The top-k nearest neighbors are fetched and ordered descending with the nearest one first by using a similarity or distance measure between the query item and an item from the group. *kNN* is a so-called *lazy learner* or *instance-based learner* and does not explicitly build a model beforehand, so classifying unseen items is rather expensive as it will be done on the classification step (Amatriain et al., 2011).

Content-based systems can be superior over collaborative filtering when no rating or too less rating information is available. In particular, new items with few ratings may benefit from this methodology.

However, according to Charu C. Aggarwal (2016), the two quality measures diversity and novelty may suffer as the recommended items may be more obvious to the user. This is due to the nature of similarities between the attributes.

2.4.3 Knowledge-Based

In knowledge-based systems, items get recommended by their "usefulness" for a certain user. There is no relying on *historical rating data* as compared to collaborative or content-based approaches. This can be beneficial in certain cases. For example, a user's high rating on a car model may derive from a combination of specific attributes such as exterior design, engine, price and so on. An updated version of the same car may not necessarily result in the same preference.

Knowledge-based systems can be distinguished into two types, *case-based* and *constraint-based* (Charu C. Aggarwal, 2016). In *Case-based* recommender systems, the user can specify cases containing preferred items or item features. These cases are then matched against item features and ordered

using domain-specific similarity metrics (Burke, 2000). This is also referred as similarity-based retrieval technique (Bridge et al., 2005).

Constraint-based systems use requirements or constraints specified by the user to recommend matching items. They require the explicit definition of *questions*, *product properties* and *constraints* to define a *recommender knowledge base* (Felfernig and Burke, 2008).

In the domain of board games, a user question or requirement could be whether a game is suitable for parties and a filter constraint would connect this requirement with certain board game properties such as amount of players and playing time.

Content-Based and *Knowledge-Based* systems are closely related and both use attributes of the items in order to make recommendations. The following distinction is used by Charu C. Aggarwal (2016):

- *Content-based systems* usually use a learning-based approach based on historical ratings.
- *Knowledge-Based systems* support the explicit specification of user requirements and therefore provide additional interactivity.

2.4.4 Demographic-Based

Demographic-based recommender systems "aim to categorize the user based on personal attributes" (Burke, 2002) in order make recommendations from demographic groups. Examples for personal or demographic attributes are age and gender. Charu C Aggarwal, Sun, and Philip (1998) present the generation of a rule-based classifier which relates a demographic profile to a certain buying behavior.

Standard classifier and regression techniques can also be applied by defining attributes from the demographic profile as features and, for example, the ratings as target values.

2.4.5 Hybrid Systems

Hybrid-recommender systems are a combination of the above mentioned techniques and therefore can overcome certain shortcomings. Charu C. Aggarwal (2016) defines three primary ways of creating such systems.

In *ensemble* systems, different predictions from different approaches are used to create a single result. Ensemble systems can be *sequential*, meaning the output of one approach is the input of another, or *parallel*, meaning the results of several approaches are merged.

An example for a parallel design would be predicted ratings, of a collaborative filtering approach and content-based filtering approach, which get transformed into a single output by creating the weighted average.

Monolithic systems consist of an "integrated recommendation algorithm" which can be a combination of different approaches. For example, the results of stages in a content-based system can be used as the input for a collaborative filtering one.

Mixed systems present results from several recommender systems to the user without any attempts of combining them.

Figure 2.1 shows an abstract classification of hybrid-recommender systems by Charu C. Aggarwal (2016).

2.5 Evaluation

The success of a recommender system and of the underlying approach must be objectively measurable. Especially when multiple algorithms and approaches are available, the most appropriate one must be chosen.

In the literature, three different evaluation kinds are defined which are *offline evaluation*, *user studies* and *online evaluation* (Shani and Gunawardana, 2011).

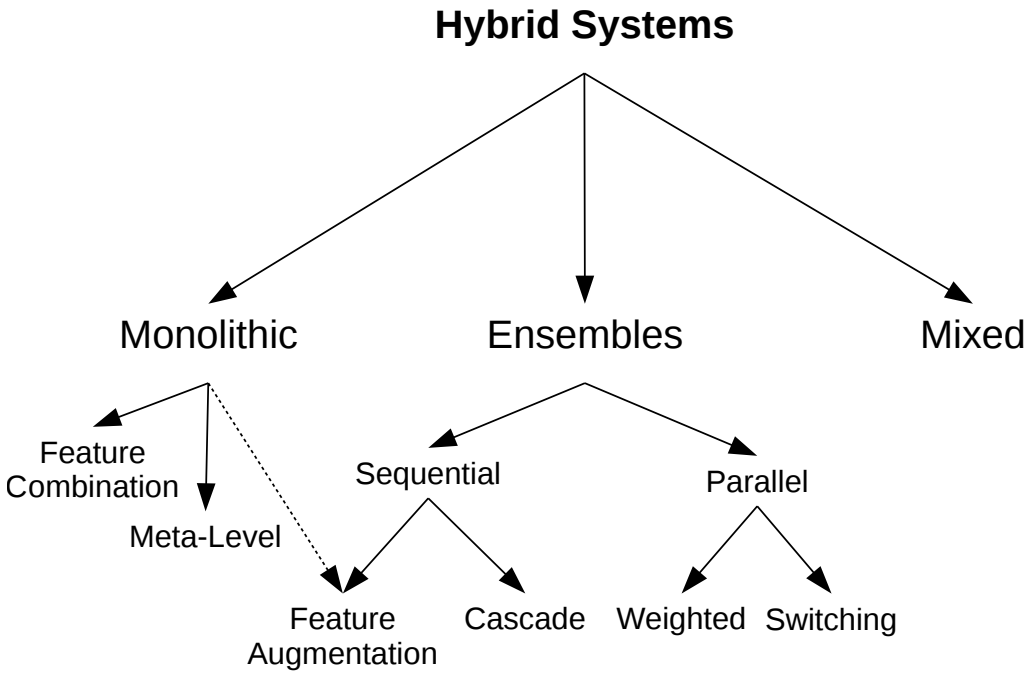

Figure 2.1: Classification of hybrid systems into three primary categories including further subdivisions, defined by Charu C. Aggarwal (2016). Adapted by permission from *Springer Nature*: Charu C. Aggarwal, 2016.

In *online evaluations*, real users interact on an already deployed system. A likely metric for measuring the effectiveness of a recommender system would be the conversion rate.

In order to compare two approaches, *A/B testing* can be executed (Charu C. Aggarwal, 2016). Thereby, users are split into two groups, A and B, and each group gets a different approach. At the end, the conversion rate gets compared. Shani and Gunawardana (2011) identified the risk that if the test users get too many improper recommendations, they may dislike the system as a whole.

In *user studies*, users are actively selected and directed to interact with the system. The most important feedback, after the user interacted with the system, is about the quality of the recommendations. Besides, information about how the user navigates through system may be collected. A risk could be that users may be biased due to the artificial situation (Charu C. Aggarwal, 2016).

Offline evaluations use historical data such as ratings or other forms of feedback. The underlying assumption is that the user behavior of the historical data will be similar to the user behavior when the system is in use (Shani and Gunawardana, 2011). More precisely, because no explicit interaction is needed and different algorithms can be compared with little effort, *offline evaluations* might be more preferable.

As a downside, the assumption about the mentioned user behavior might be inappropriate as data may have evolved. In the above mentioned car example, it could happen that a user's feedback on a certain car model might have changed on the updated version. Popular examples of data sets are the data set from the netflix prize, by Koren (2009), or the *MovieLens* data set from GroupLens (2008).

Besides the primary goal of finding the most *accurate* prediction possible, several secondary goals exist, which contribute to the user experience. One secondary goal, *Scalability* 2.5.2, will be highlighted in more detail as it received special attention in Chapter 3.

2.5.1 Accuracy

Accuracy is one of the most important goals when evaluating recommender systems. The accuracy metric can be defined as the accuracy of an *estimated rating* or the accuracy of a *predicted top-k* ranking.

Breese, Heckerman, and Kadie (1998) made use of both accuracy variants, estimated rating and predicted top-k ranking, on evaluating collaborative filtering approaches on three data sets by using a hold-out method.

In order to estimate the overall *generalization performance*, which is the performance of the built model on *unseen* data, the evaluation data set is usually split into a training and a test segment (Charu C. Aggarwal, 2016). By doing so, a model will be built by using the training set and evaluated on the test set.

The main idea is that the model still performs well on unseen data and therefore *generalizes* well. As shown in Figure 2.2, it is also possible to further split the training set and obtaining a third set called *validation set*. This is usually used for *model selection* and *parameter tuning* (Charu C. Aggarwal, 2016).

Tuning parameters or *hyperparameters* are parameters which have to be defined *a priori* before a model is created and are not directly learned from data. In a *memory-based collaborative filtering* approach such as *k-Nearest Neighbor*, the parameter k, which is the amount of the nearest neighbors, has to be tuned and could be selected by evaluating on the *validation set*.

In practice, the following two methods for segmentation are often used:

2.5.1.1 Hold-out method

The *hold-out method* uses a portion of the original data for training and the remaining data set for testing. A *80/20* split is a frequent choice and means that 80 percent of the whole data set is used for training and the remaining 20 percent is used for testing.

As a general guideline, over $2/3$ should be used for the training set (Amatriain et al., 2011). The segmentation of entries from the original data set can be

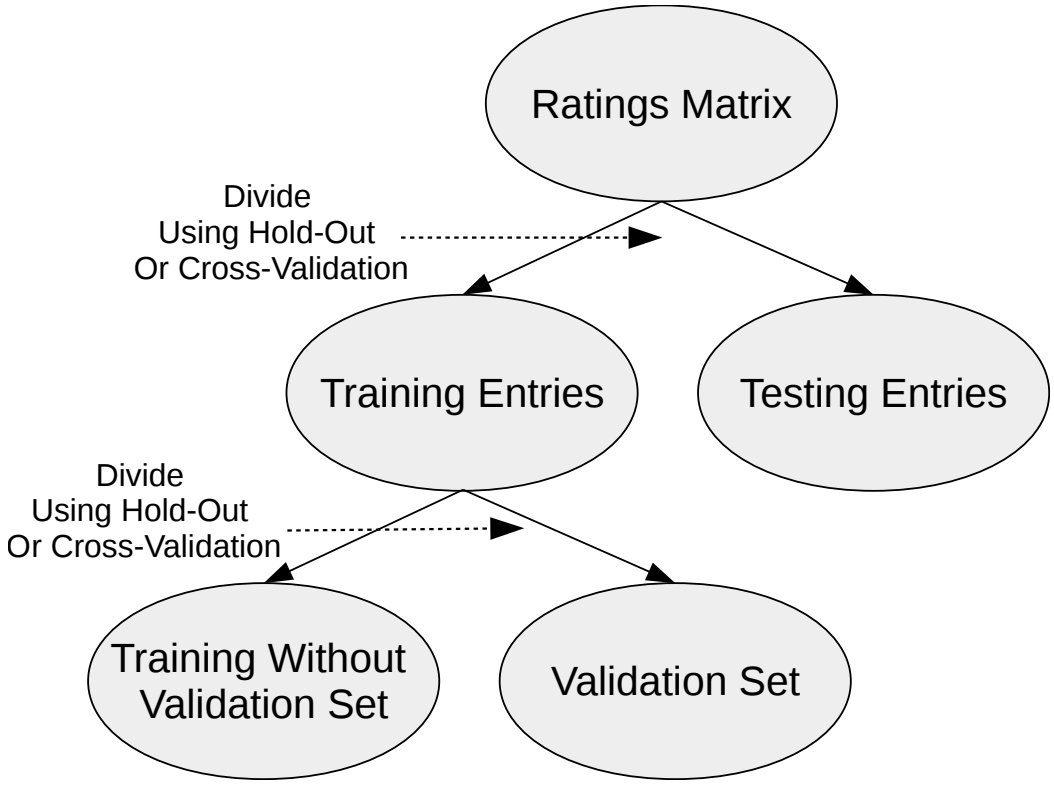

Figure 2.2: Segmentation of a rating matrix into training and test segments by Charu C. Aggarwal (2016). The training segment can be further split into training-without-validation and validation sets. Adapted by permission from *Springer Nature*: Charu C. Aggarwal, 2016.

done in different ways and depend on the properties of the recommendation model.

Agarwal and Chen (2015) define the following four variants, to be applied on a rating matrix:

1. *Random splitting* randomly chooses a certain percentage p from the observed entries of a rating matrix for the training set and $100 - p$ percent for the test set. This could be done either row-wise, for users, or column-wise, for items.
2. *Time-based splitting* divides the overall data set by a certain time point with older entries in the training set and newer ones in the test set.
3. *User-based splitting* divides the data set *user-wise* and considers p percent of users and their ratings as the training set and $100 - p$ percent as the test set.
4. *Item-based splitting* is similar to *user-based splitting* but only makes the split *item-wise*.

2.5.1.2 Cross validation

Cross validation divides a data set of size N into k equal segments. There exist several cross validation techniques such as *k-fold cross validation* and *repeated random sub-sampling* (Amatriain et al., 2011).

In *k-fold cross validation*, $k - 1$ segments are used as training set and the remaining segment is used as test set. This form of segmentation, also known as *sampling without replacement*, is repeated k times and the average *performance metric* across all k folds is then reported.

For example, in a 5-fold cross validation with a data set of $n = 500$ entries, five different models will be built on the training set with 400 entries and the performance will be evaluated on the remaining 100 entries.

There is also a special case of *k-fold cross validation* called *leave-one-out* validation when k equals n. Each single entry is then used exactly once as a test entry with the remaining $n - 1$ entries as training data. The segmentation of sets shown in Figure 2.2 can therefore be also seen as a single phase of the cross validation method.

In *repeated random sub-sampling*, the k segments are not disjoint and not all entries of the whole data set must occur in the segments.

As mentioned before, offline evaluation can be performed by measuring the *accuracy* of estimated rating values or ranked recommendation items.

Herlocker et al. (2004) further describe the term accuracy in more detail and identify three classes of evaluation metrics:

2.5.1.3 Predictive Accuracy Metrics

Predictive Accuracy Metrics measure the accuracy between the estimated rating $\hat{r}_{ij}$, from user i to item j, and the actual rating r_{ij}. In the following equations, T is referred as the test set consisting of (i, j) entries.

The *Root Mean Squared Error* (RMSE), Equation 2.1, measures the average error and is defined as the square root of the average squared differences between the estimated and the actual rating.

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{i,j \in T} \left(\hat{r}_{ij} - r_{ij} \right)^2} \tag{2.1}$$

$\hat{r}_{ij} - r_{ij}$ is often referred to as *entry-specific error* e_{ij}.

Another alternative and popular metric is the *Mean Absolute Error* (MAE) as seen in Equation 2.2. The main difference between MAE and RMSE is that the former penalizes large errors more due to its squared term (Herlocker et al., 2004).

$$MAE = \frac{1}{|T|} \sum_{i,j \in T} |\hat{r}_{ij} - r_{ij}| \tag{2.2}$$

2.5.1.4 Classification Accuracy Metrics

Classification Accuracy Metrics measure how well a system recommends items. They can be used on top-k recommendation algorithms as predicted ratings are not of interest as long as they do not lead to classification errors (Herlocker et al., 2004). From the field of information retrieval systems, two popular metrics, called *precision* and *recall*, are frequently used to measure classification performance.

Table 2.1 defines the relation between relevant and irrelevant items and whether they were selected or not. The following notations will be used in the Equations 2.3, 2.4 and in the Table 2.1. N_{rs} stands for relevant and selected items whereas N_{rn} comprises relevant but not selected items.

Analogous to this, N_{is} is the set of irrelevant and selected items and N_{in} the set of irrelevant and unselected items. Other terminologies for N_{rs}, N_{rn}, N_{is} and N_{in} are *true positives*, *false negatives*, *false positives* and *true negatives*, respectively.

Precision (Equation 2.3) is defined as the fraction of relevant and selected items to selected items in general.

$$Precision = \frac{N_{rs}}{N_s} \tag{2.3}$$

Recall (Equation 2.4) is the fraction of relevant and selected items to relevant items in general.

$$Recall = \frac{N_{rs}}{N_r} \tag{2.4}$$

In an offline evaluation scenario, a user could be selected as a test user, some of his rated items would be hidden and the recommender tries to predict these hidden items as good as possible. Items which are selected but are not relevant, because they did not occur in the feedback from the user, may lead to an overestimation of the number of false positives (Shani and Gunawardana, 2011). For example, a user may like the recommended item

	Selected	**Not Selected**	**Total**
Relevant	N_{rs}	N_{rn}	N_r
Irrelevant	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

Table 2.1: Table by Herlocker et al. (2004) showing the classification of relevant and irrelevant items which are selected or unselected.

but was simply unaware of its existence and therefore it was not considered relevant.

It is important to note that the recommendation list length has a direct impact on precision and recall. If the recommendation list length is too small the amount of false negatives rises and recall decreases. If the recommendation list length is too large, false positives may rise and precision is getting smaller (Shani and Gunawardana, 2011).

According to Gunawardana and Shani (2009), in top-k recommender systems, a higher precision is more preferable than a higher recall. Different methods for summarizing the relation between precision and recall exist.

Precision-recall curves present the trade-off between precision and recall graphically. Other measures that summarizes the precision-recall relation can be used to compare different algorithms (Shani and Gunawardana, 2011).

For example, the *f1-score*, originally introduced by Rijsbergen (1979), transforms precision and recall into a single comparable metric. The metric is defined by the harmonic mean between them, as shown in Equation 2.5. P and R depict the precision and recall, respectively.

$$\text{F1-Score} = \frac{2 * RP}{R + P} \tag{2.5}$$

2.5.1.5 Rank Accuracy Metrics

Rank accuracy metrics measure how well the ordering of the recommendation list matches the ordering from the user. Therefore, they can distinguish between very good and not so good, but still relevant, items.

Herlocker et al. (2004) mention *correlation* metrics, a *half-life utility* metric and the *normalized distance-based performance measure* (NDPM). Correlation metrics, such as the *Spearman rank correlation coefficient* (Spearman, 1904) or *Kendall's Tau* (Kendall, 1938), measure the correlation between two variables. The correlation requires to precompute the ranking of the recommendation list and the ranking of the user, also called the *ground truth*. A higher correlation is to be preferred.

The *half-life utility* metric is applicable for tasks where a user is presented with a ranked recommendation list and it is likely that he will only consider the first results (Breese, Heckerman, and Kadie, 1998).

NDPM, originally introduced by Yao (1995), is similar to Spearman and Kendall's tau coefficient but does not "penalize" the recommender system for tied user ranks, meaning the ground truth contains equally ranked items (Herlocker et al., 2004).

Cremonesi, Koren, and Turrin (2010) show in their empirical study that for top-k recommendation tasks, error metrics such as Root Mean Squared Error (RMSE) (Equation 2.1) and Mean Absolute Error (MAE) (Equation 2.2) may be inferior towards accuracy metrics.

2.5.2 Scalability

As items, users and ratings increasingly grow, a recommender system should still act efficiently. Charu C. Aggarwal (2016) defines *training time*, *prediction time* and *memory requirements* as the three measures for determining *scalability*.

Prediction time, which is the time it takes from requesting to retrieving items, contributes primarily to the user experience. Training time, which

is the time the recommender system takes during the training phase, and memory requirements are more system related.

2.6 Domain Description

Board games are usually played on a table or other flat surfaces with one or several players. They typically consist of one or several categories, however there exist much more subdivisions and attributes. Some attributes are generally applicable to board games and others, such as *complexity* classifications, are board game specific. In this thesis, the term *board game* is continuously used as collective term for card games, dice games and similar.

2.6.1 Boardgamegeek.com

Boardgamegeek.com, also known as *BGG*, is one of the largest online board game databases and was founded in 2000. At the time of data extraction (April 2017) over 88 000 board games were present, including not yet released games. Boardgamegeek.com contains thousands of board games including reviews, ratings, comments, images, videos and user-generated content. Besides, a message board for user interaction and a board game marketplace exist.

The website uses three rating kinds which are *user rating*, *average rating* and *BGG rating* (FAQ, 2018). The BGG rating, also known as *Geek rating*, is basically based on the average rating but with some alternations. It is used to better reflect the popularity for games with few user ratings.

The operators behind BGG also link to *videogamegeek.com* and *rpggeek.com* where video games and role-playing games are listed, see Figure 2.3.

The detail page of the currently top-ranked board game, *Pandemic Legacy: Season 1*, can be seen in Figure 2.4. Figure 2.5 and Figure 2.6 show a profile overview of a user and a user-written review about a board game.

2 Background

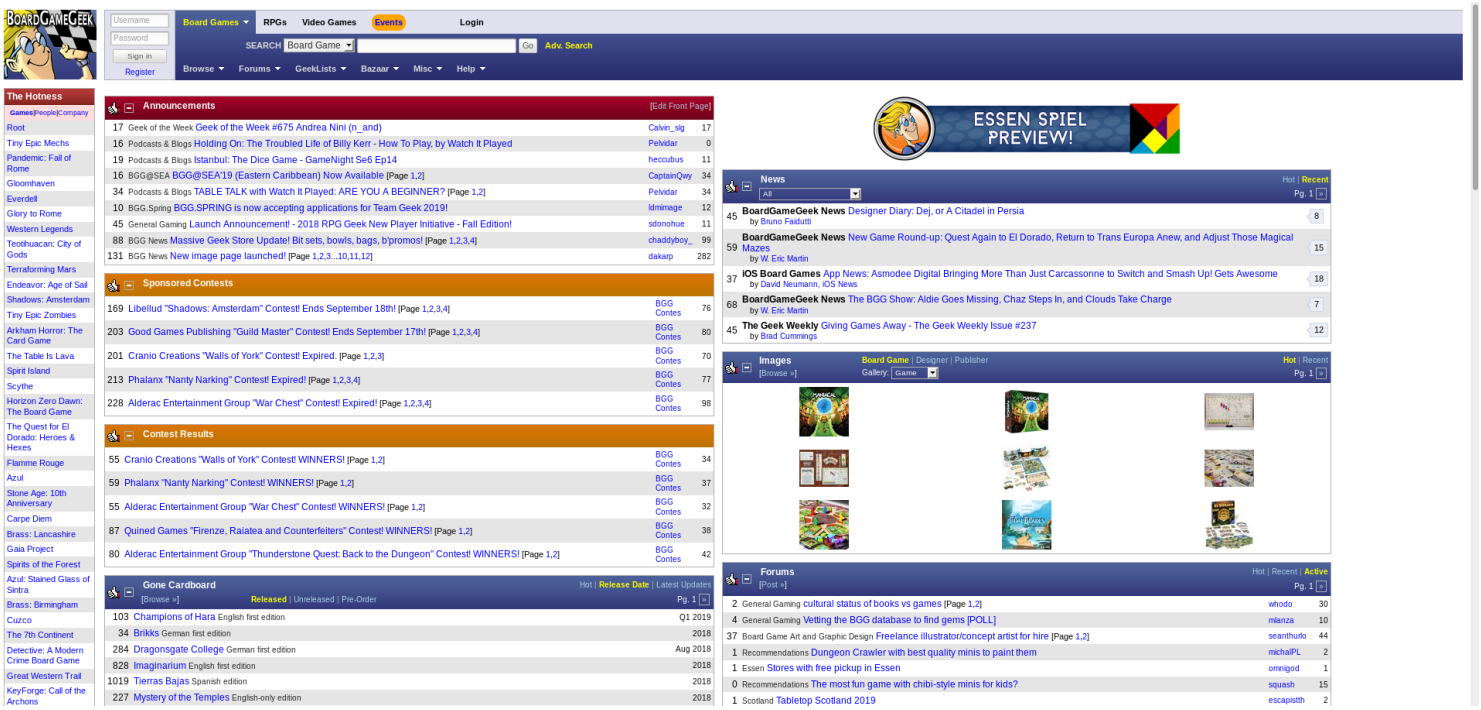

Figure 2.3: Start page of Boardgamegeek.com with the three main components on top, “Board Games”, “RPGs” and “Video Games”.

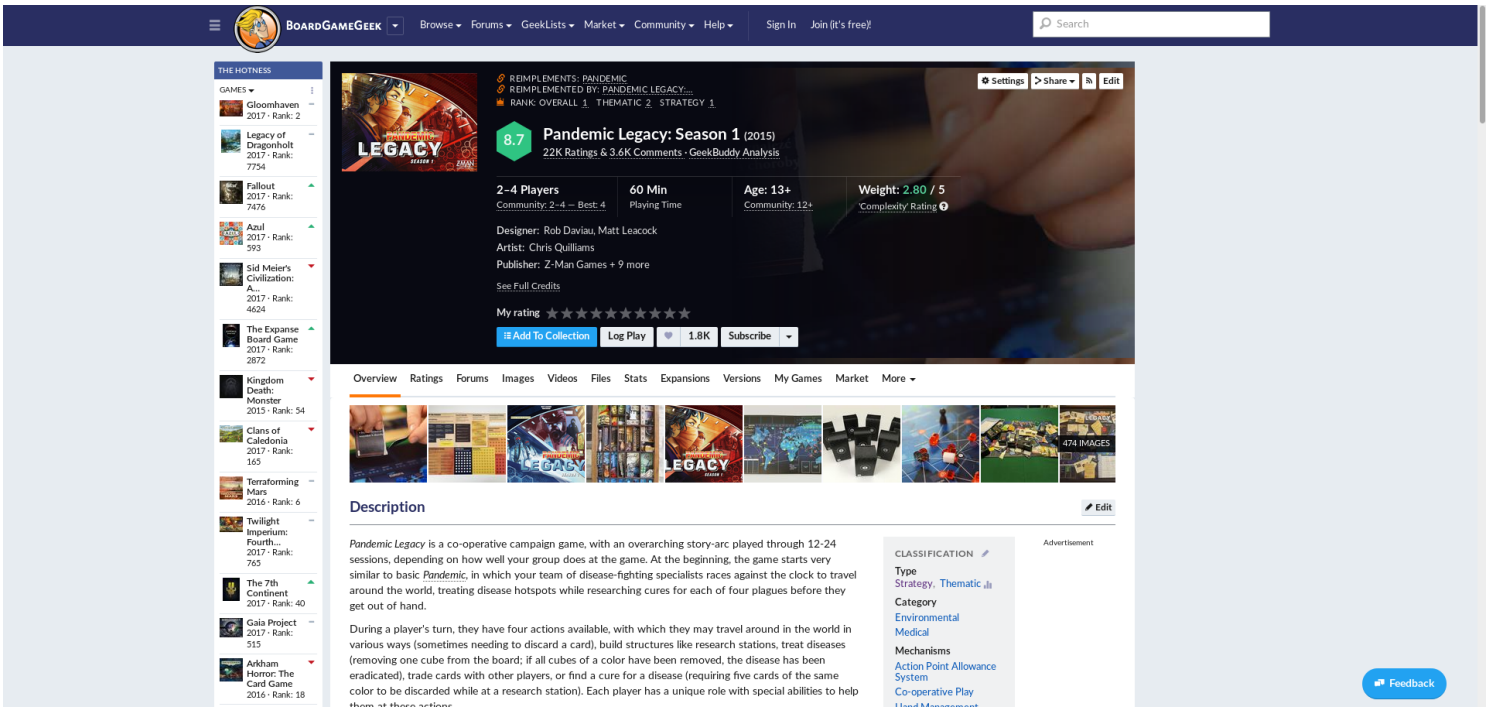

Figure 2.4: Detail Page on boardgamegeek.com showing one of the most popular board games: "Pandemic Legacy: Season 1".

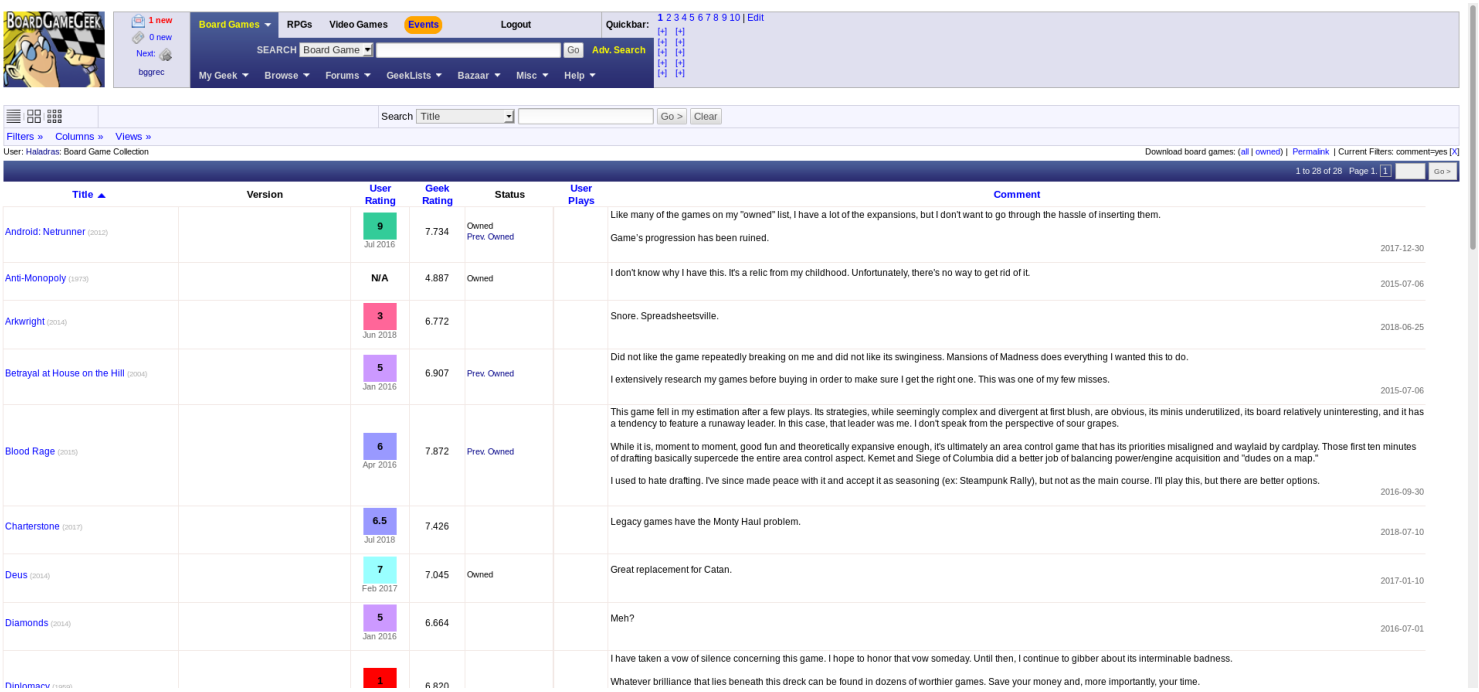

Figure 2.5: Overview of a user profile on boardgamegeek.com. The page shows the ratings and comments of the user as well as collection memberships for different board games.

2 Background

Figure 2.6: A user-written review on boardgamegeek.com about the board game "Pandemic Legacy: Season 1".

2.7 Related Work

The domain of board games is rather rare as a scientific target domain. However, some scientific research has been done in this domain and most of the time, BGG was used as the data basis.

In 2007, Aranda et al. created a *social network-based* recommender system by using relations between users from BGG, called *GeekBuddies*. They incorporated this *friend-relation* information into a probabilistic matrix factorization approach, as a third matrix.

In one of his experimental setups, Mei (2008) constructed a Bayesian network from a subset of the BGG data. He evaluated the model by using *RMSE* and *cross validation* and compared the accuracy against a *linear regression* approach and a *constant prediction baseline*. Mei and Shelton (2012) also built a Bayesian network and used a subset of the BGG data set in their evaluation process.

Faryal et al. (2015) selected BGG as data source as well and used collaborative and content-based filtering approaches to recommend board games. For evaluation, they selected information retrieval metrics, such as *precision*, *recall* and *f1-score*.

In the field of Personalizing Web Augmentation Applications, Wischenbart et al. (2015) demonstrated a novel framework for modifying a website of an end-user based on their personal preferences. They presented a concrete example based on the detail page of a board game on BGG, where a custom rating for a board game was created and presented to the user.

Additional contextual information can be incorporated in a recommendation request. This field is called *Context-aware Recommender Systems* and contextual information could be time, location or social data (Adomavicius and Tuzhilin, 2011). As an example, the location of a user could be used to narrow and improve recommendations.

The evaluation and experimental setup explained in Section 3.4, follows a specific form of context-aware recommender systems called *Narrative-driven Recommender Systems* (NDR). This specific field, introduced by Bogers

and Koolen (2017), was created to meet the requirements of more complex real-user search queries.

Bogers and Koolen coined the term *narrative requests* which consists of information about *user preferences* and a *narrative description* of what is desired by the user. A concrete example for a narrative request could contain desired and undesired board games (user preferences) and a textual description with several sentences about what the user is looking for (narrative description).

2.8 Knowledge Discovery Process

The term Knowledge Discovery in Databases (KDD) or also known as Knowledge Discovery Process (KDP) was first mentioned by Gregory Piatetsky-Shapiro (1991) and further defined by Fayyad, G Piatetsky-Shapiro, and Smyth (1996). It can be described as the process of finding appropriate patterns in data or *knowledge* and was partially used for extracting knowledge from BGG. The exact process definition is as follows:

"Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."

The Process, consisting of nine steps, can be reduced to five major steps. It may be necessary to iterate and make decisions in each step to get the most appropriate results.

1. Selection: Developing an understanding of the application domain and selecting a target data set.
2. Preprocessing: Classifying and removing noise, as well as handling missing data fields.
3. Transformation: Finding appropriate features depending on the goal of the task.
4. Data Mining: Choosing data mining algorithms including models and parameters.
5. Interpretation/Evaluation: Interpreting the mined patterns.

In the following Chapter 3, several steps of the KDD will be applied.

3 Materials and Methods

The following chapter first describes how board games were extracted from *boardgamegeek.com* (BGG) and stored in a MySql database for faster querying. A framework with four approaches was created where, among other options, desired and undesired board games can be specified as input.

Submission requests for board games were taken from a subreddit of *reddit.com* and used for evaluation. To improve accuracy, post-filtering was applied to the returned recommendation list of each algorithm. The submissions were split into training and test sets by using a time-based holdout method.

3.1 Development Platform

In order to execute the necessary tasks, starting by preprocessing until evaluation, a local machine as well as a separate server instance was used. The preprocessing stages, including crawling from BGG, were executed on a local machine. Rating matrix related tasks as well as the evaluation were performed on the more powerful server instance. The development environment used is further described in Table 3.1.

3.2 Selection and Preprocessing

As highlighted in Subsection 2.6.1, BBG was used as the data basis for the knowledge extraction. No data dumps were available so the board games and related information had to be gathered manually.

OS	Ubuntu 14.04.5 LTS
Python	Version 3.4.3
Libraries	Lxml (3.3.3) Matplotlib (2.1.0) Networkx (2.1) NumPy(1.8.2) Pandas (0.19.2) PyQt (4.11.4) SciPy (1.0.0) SkLearn (0.19.1)
MySql DB	Version 5.5.58
Disk Space	3.33TB
CPU	Intel Xeon E5-2620 v3, 24 Cores
RAM	256GB

Table 3.1: Overview of the development environment provided by the server instance.

In the following paragraphs, the term board game also includes board game expansions. A board game expansion is an enhancement to an already existing board game.

BGG provides an *application programming interface* (API) for fetching all board game related data by making HTTP requests with certain parameters. However, as each API request requires a board game ID, the first step, before using it, was crawling the board game overview pages and extracting the board game IDs.

For crawling the overview pages, the python packages *BeautifulSoup* and *urllib* were used. The process was executed sequential with 100 board games at once and a delay of 2 seconds between the requests.

All in all, 88 564 unique board games and board game expansion IDs were extracted. The lowest board game ID was 1 from the board game "Die Macher" and the highest ID 219 708 was from "Professor Evil and The Citadel of Time".

The board game ID list was not numbered consecutively. Reasons for this

were that either the item was removed or it was not a board game but a video or role-playing game (RPG). The operators of BGG seem to use the same database in which all items share a unique and continuous ID.

3.2.1 BGG API

The next step was exploring the available API calls, their differences and their parameters. The BGG API in version 2 (API2, 2018) provides eleven different API calls and returns a custom XML file on success. In error cases an HTTP error (in case of too frequent requests in a short time) or a malformed XML file was returned. *Lxml* was used as the XML parsing library.

3.2.1.1 Board games

In order to fetch the board games, a *thing* request with certain parameters was made. A thing can be a board game or video game. As it was an optional parameter for the API request, it was omitted. The only mandatory parameter was the unique item ID or a list of item IDs.

To not overload the BGG servers, only 50 board games were fetched at once. The following shows a sample request for the board game *Catan* with the unique board game ID 13.

```
https://www.boardgamegeek.com/xmlapi2/thing?id=13&versions=1
&videos=1&stats=1&historical=1&ratingcomments=1}
```

Many properties such as alternate names or user poll results are contained by default, in the returned XML file. Additionally, several options can be passed in the request to further extend the amount of information returned.

For example, the options *comments* and *ratingcomments* return all comments about an item (including ratings if applicable) and all ratings of an item (including comments if applicable). However, it is not supported to specify both options at the same time in a request. As an item can have a rating, or

a comment or both from the same user, two crawling runs were necessary in order to get all rating and comment information. Afterwards, the resulting XML data sets had to be merged accordingly.

3.2.1.2 Reviews

In the BGG forums, users can write their own reviews on board games or comment on existing reviews. The vast majority of the reviews is written in English and different kinds and scopes exist. A review may consist of only text and be very precise or contains a hundred of words, including images, drawings or videos about the game. Usually, more popular or older board games tend to have a higher amount of reviews. For example, for the board game *Catan*, over 150 reviews exist, with many of them having replies from other users.

In order to fetch the reviews, three types of API calls were used. First of all, a *forumlist* call was made for a specific board game. This call returned a list of available forum types, in which one had the title "Reviews" including a unique ID. With this review ID, a *forum* request was done returning all the thread IDs, which were the IDs of the actual reviews. The last API request, of type *thread*, returned the actual review including all its follow-up replies.

3.2.1.3 Collections

A user can also rate or comment an item and add it to its own collections. However, it is not mandatory to specify a collection when rating or commenting an item. The following self-explaining collections exist:

1. Own
2. Previously Owned
3. For Trade
4. Want in Trade
5. Want To Play
6. Want To Buy
7. Preordered

8. Wishlist

The collection *Wishlist* is the only one with further subdivisions starting from 1 “Must have” to 5 “Don’t buy this”.

For retrieving collection data, a *collection* API call was made. The only mandatory parameter was the unique user name.

There was a bug in the initial thing API request which ignored the *stats* parameter which is necessary for retrieving ranking information. The collection requests were therefore also used to fill such missing information.

3.2.1.4 Challenges

Some difficulties have arisen during the request process. Although the BGG “terms of use” for the API did not mention any restrictions, the requests were made with an initial delay of two seconds. However, too frequent calls by the same IP address resulted in HTTP errors with error code 500, “Internal Server Error”, and 503, “Service unavailable”.

The first mitigation technique was an exponentially increasing delay between the requests in case an error occurred and a reset to the initial delay in case the request was successful. Despite this kind of congestion control, the errors still occurred so the assumption was made that there was some kind of “penalization” involved. Therefore, for the board game and review item requests, rotating proxy IP addresses were successfully used.

For time reasons, the user collections scrapping process was implemented multi-threaded. Five threads, each with a random IP address and user agent for every request, fetched the collection data simultaneously. This reduced the estimated scrapping time of two to three weeks to some days. Especially, as each request took five to fifteen seconds.

Other difficulties were more related to the returned XML files. Some requests returned malformed XML, for example, due to improperly escaped characters. In rare cases, the recovery option offered by *Lxml* was not sufficient. As an example, there was a case, where a user comment contained two sequential null bytes which is not allowed according to the *RTF* specification (W3C, 2008).

3.2.2 Database Import

In order to store the board games, a relational database management system (RDBMS) was used. The requirements were fast retrieval of data, making use of a structured query language (SQL) and making use of an object-relational mapping (ORM) framework in Python. The selected RDBMS was MySql, as it is very popular and freely available, and *peewee* as a lightweight ORM framework (Leifer, 2017). Figure 3.2 summarizes the process of getting the BGG data into the relational database.

3.2.2.1 Database Schema

First of all, a database schema was designed. The main requirements were that the schema stores all the data correctly and allows efficient retrieval. Figure 3.1 shows the created schema with eleven relations. For each table, the storage engine *innoDB* and collation *utf8-bin* was used.

Each relation uses an automatically incremented artificial primary key, named *ID*, to store its instances. The reason for this was that it was not known beforehand which IDs and fields of the objects can be considered unique. For example, the field *value id* used in the relation *board_game_link_types* occurred several times in the XML files with different values and types. Another reason was that *peewee* prefers to use artificial keys instead of compound ones.

For performance reasons, several SQL *indexes* were created. To achieve efficient join operations, they were created on the respective foreign keys. Additionally, to handle the initial *board game name matching* faster, indexes were created on the *primary_name* and *bg_id* fields of *board_games* as well as on the *name* field of *board_game_alternate_names*.

The relations *board_game_comments*, *board_game_user_collections*, *board_game_review_replies*, *board_game_to_link_types* and *board_game_version_to_link_type* ensure many-to-many associations between the actual objects by holding the respective foreign keys. After the relations were created, the peewee model generator was used to generate python class models from the relations.

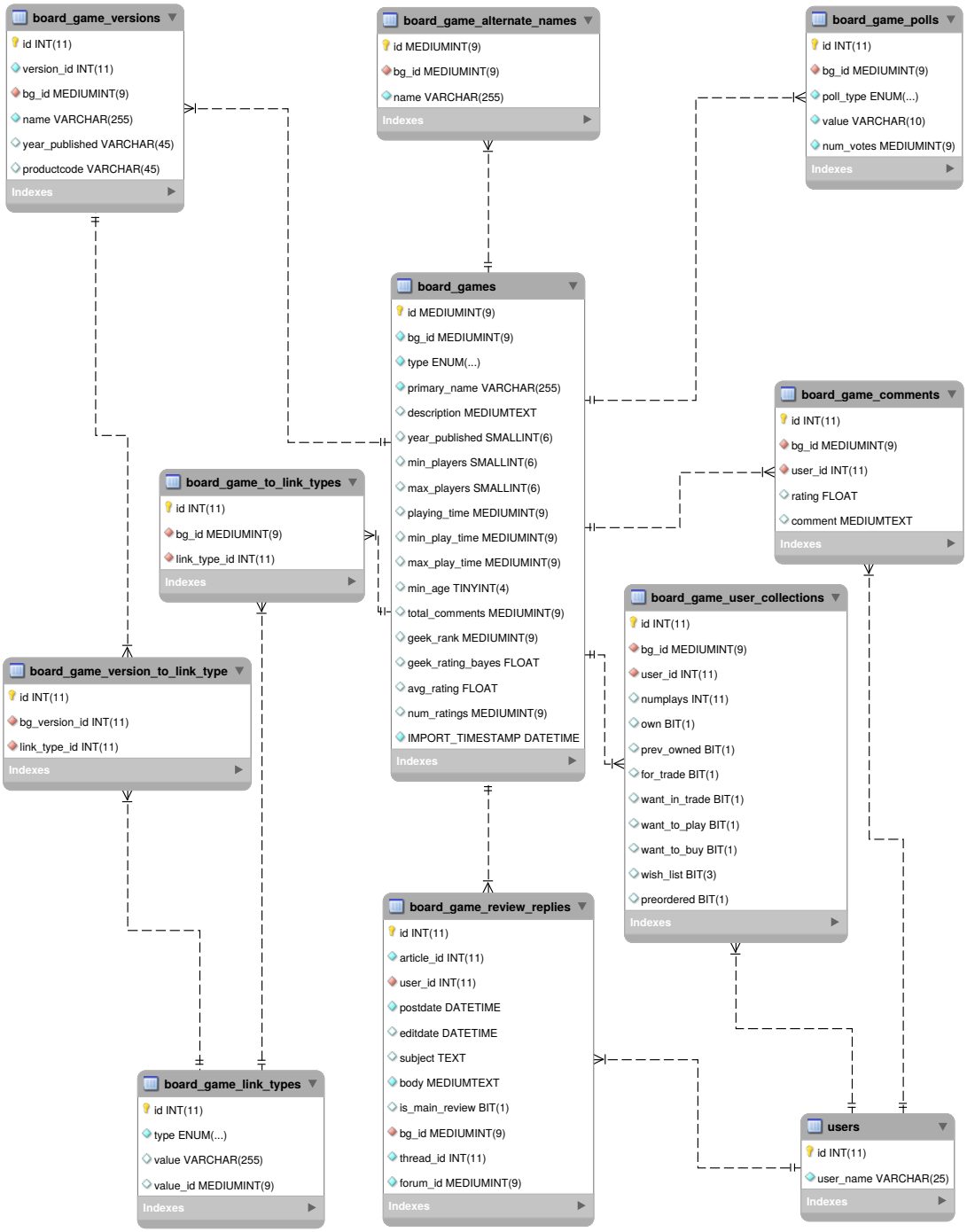

Figure 3.1: Overview of the used MySql database schema with eleven relations, each with an artificial primary key named *id*.

# Total Board Games	88 564
# Board Games Which Are Expansions	13 224
# Board Games (#Ratings >= 1)	64 498
# Alternate Board Game Names	44 088
# Comments Only	782 444
# Ratings Only	8 300 463
# Comments with Ratings	2 411 975
# Users	207 572
# Users (#Ratings >= 1)	197 979
# Reviews	63 195
# Replies including Reviews	459 443

Table 3.2: Selected key data collected after the database import.

The final time stamp of the imported board games was April 2017. Table 3.2 shows selected key data after the import. 64 498 board games have at least one rating and 197 979 users have at least rated one board game. There exist also users who have never rated a board game but commented on it. The amount of 459 443 replies also include the actual reviews, therefore 396 248 comments on main reviews exist.

For comparison, when Aranda et al. crawled data from BGG in 2007, they received about 30 000 board games, with at least one rating, and about 40 000 users, who made at least one rating.

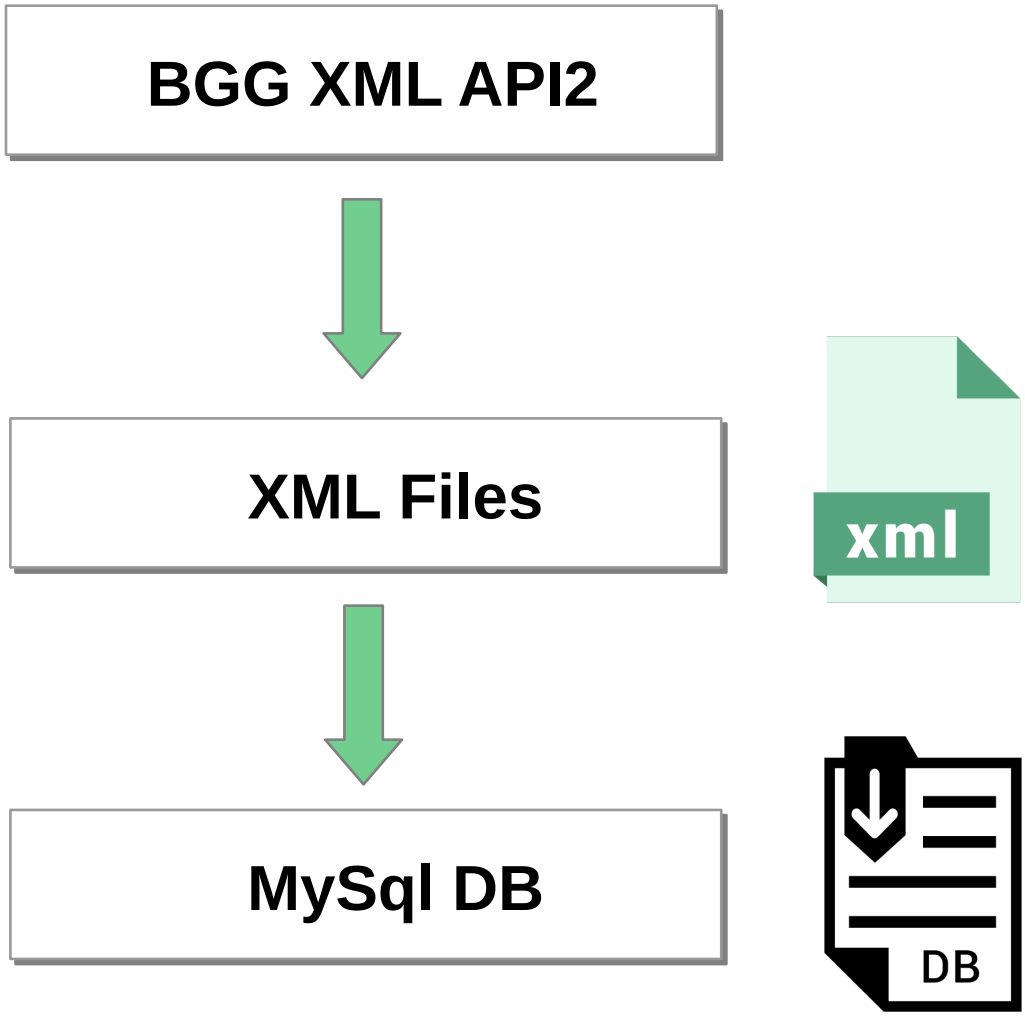

Figure 3.2: Process showing the three steps of fetching data and importing it into the database. HTTP requests are made to the BGG API, the returned XML files are saved locally and are then imported into the MySql database.

3.2.2.2 Challenges

As mentioned above, it was not always clear what the unique fields of the data were. Therefore, importing the data was partly an iterative process, adapting unique constraints on the database schema. Additionally, programmatically assertions in the code as well as database update and insert triggers ensured data integrity. As soon as erroneous states were detected, the import process was aborted and the database transaction was rolled back.

Some data had to be omitted or transformed before they were inserted into the database. For example, usually the ratings for a board game range from one to ten. However, in some cases there existed floating point ratings, which seemed to be a bug as they are not available on BGG. Therefore, such ratings were rounded up or down.

In rare cases, some board games had invalid field values. For example, the XML file of the board game with the ID 12 238 had the invalid published year 636 295. It also occurred that some users had two different ratings for a board game, which should not be possible. In such cases, the latter one was taken.

3.3 Framework and Methods

The goal was to develop a fast and easy extendable recommendation framework which takes user queries in form of desired and undesired board games and return a top-k recommendation list, with k being arbitrarily configurable. Similar to the input of constraint-based systems (see Subsection 2.4.3) additional command line arguments, as seen in Table 3.3, can be added to further constrain and assist the recommendation framework.

3.3.1 Overview

Four different approaches were implemented. Two approaches, defined as *Collaborative Filtering* and *Matrix Factorization* rely on similarities in the

Positive Board Game Name List	`-pbg <NAME> [<NAME> ...]`
Negative Board Game Name List	`-nbg <NAME> [<NAME> ...]`
Positive Board Game ID List	`-pbgids <pbgids> [<pbgids> ...]`
Negative Board Game ID List	`-nbgids <bgids> [<bgids> ...]`
Approach to Use	`-a {<all_available_approaches>}`
Amount of Recommendations	`-l <listlength>`
Minimum Amount of Players	`-minpl <minplayer>`
Maximum Amount of Players	`-maxpl <maxplayer>`
Minimum Playing Time	`-mintime <minutes>`
Maximum Playing Time	`-maxtime <minutes>`
Minimum Playing Age	`-minage <minage>`
Published Year from	`-yf -4000to4000`
Published Year to	`-yt -4000to4000`
Positive Categories	`-cp {<all_available_categories>}`
Negative Categories	`-cn {<all_available_categories>}`

Table 3.3: Selected command line options for the recommendation framework.

historical rating data of BGG, whereas the other two approaches, defined as *Term Frequency – Inverse Document Frequency* and *Two-mode Network*, utilize content attributes.

The expandability of the framework was ensured by calling the respective approach at runtime. In order to create a new approach, only the implementation had to be added as a method.

Getting fast recommendations from the framework, addressing the scalability requirement (see Subsection 2.5.2), was achieved by various techniques. Two stages, *offline* and *online*, can be identified. Whereas the *Collaborative Filtering* and *Two-mode Network* approaches calculate their recommendations solely at runtime (*online*), the *Matrix Factorization* and *Term Frequency – Inverse Document Frequency* approaches use a pre-calculated model (*offline*) which gets loaded at runtime and therefore allows efficient and fast recommendations.

To further improve performance, the *Two-mode Network* approach used a *prefetch* mechanism which preloads peewee's internal object representation

with other fully loaded objects that are specified and related by foreign keys. This technique is very common in ORM frameworks and avoids so-called *n+1 behavior* where otherwise separate SQL queries would be performed on a related object.

Additionally, the rating matrix was always saved and operated in *compressed row storage* (CSR) mode and operations from the *scipy* packages, which are compatible with CSR, were used.

The correctness was ensured by using well-established frameworks, unit tests and conducting manual tests. For evaluation, a global in-memory cache was used to ensure fast reordering of already executed approaches for submissions.

3.3.2 Input Matching

As the framework allows to also input board game names or terms, an input matching strategy was implemented. The following five steps are applied consecutively. When no match is found the next step tries to find a match. At each step, the term is first matched against the primary name of the board game and only if there is no hit, it will be matched against all possible alternate names.

1. Tries an exact match, case-sensitive.
2. Tries an exact match but case-insensitive.
3. Applies a wildcard match to the end of the term, similar to "term*". Operates case-insensitive.
4. Similar to Step 3, but with an additional wildcard match at the beginning of the term, similar to "*term*".
5. In addition to Step 4, all non-alphanumeric characters are replaced by wildcard symbol.

Compared with the search functionality of BGG, this approach differs in the way that it tries a "wildcard end match" first and stops on success, whereas at BGG a "contains" search will be triggered first, even if the exact primary name or alternate name was given.

3.3.3 Collaborative Filtering

An item-based collaborative filtering approach (see Subsection 2.4.1.1) was used for predicting the top-k recommendations. Contrary to the usual off-the-shelf methods, this naive variant simply relies on the similarity between the ratings of items, meaning no rating prediction was done.

The used rating matrix R (shown in Equation 3.1) has 197 979 users and 64 498 board games, denoted m and n respectively. In total, R contains 10 712 438 ratings. Most of the entries $r_{i,j}$ are unobserved which means no rating from user i to board game j exist.

The unobserved-to-observed ratio, referred as sparsity level by Sarwar et al. (2000), of a rating matrix is defined by $SL = 1 - \frac{\text{nonzero entries}}{\text{total entries}}$. For R, the sparsity level is calculated by $SL = 1 - \frac{10\ 712\ 438}{197\ 979 * 64\ 498}$ resulting in 0.99916 which means R is very sparse.

$$R_{m,n} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{pmatrix} \tag{3.1}$$

The *cosine similarity* was used as similarity metric. It presents the cosine of the angle between two board games and ranges from -1 expressing dissimilarity up to 1 for equality. Sarwar et al. (2000) also used that metric when comparing the item-based collaborative filtering performance on the *MovieLens* data set.

The metric is defined by the dot product of the two item vectors divided by their euclidean lengths. Equation 3.2 shows the cosine similarity with b_i being the column vector for board game i and b_j being the column vector for board game j.

$$cosineSim(b_i, b_j) = \frac{\vec{b_i} \cdot \vec{b_j}}{||\vec{b_i}|| \cdot ||\vec{b_j}||} \tag{3.2}$$

Algorithm 1 describes the used recommendation approach in detail. First of all, the rating matrix and the input board games which either express preference, *positive*, or disfavor, *negative*, are passed. The similarities *pCos* between the positive input board games and the non-input board games are calculated and aggregated. The same applies for the negative input board games *nCos*.

In order to avoid a too strong influence of the negative similarities on the positives ones, a weighting procedure was applied. As shown in Procedure 2, the negative similarity score was scaled down by its ranking position, denoted by *rank(item)*.

Finally, the weighted negative similarities were subtracted from the positive similarities. The similarities were sorted in descending order and the *k* most similar board games returned.

Algorithm 1 Item Similarity Calculation

1: **Initialize**:
2: $R \leftarrow$ rating matrix
3: *pbgs* $\leftarrow$ positive input board games
4: *nbgs* $\leftarrow$ negative input board games
5: *pCos* $\leftarrow$ empty positive cosine similarities
6: *nCos* $\leftarrow$ empty negative cosine similarities
7: **For each** *i* of *pbgs*
8: **For each** *j* of *non-input board games* :
9: $pCos[j] \leftarrow pCos[j] +$ ***cosineSim****(R[:, i], R[:, j])*
10: **For each** *i* of *nbgs*
11: **For each** *j* of *non-input board games* :
12: $nCos[j] \leftarrow nCos[j] +$ ***cosineSim****(R[:, i], R[:, j])*
13: *allCos* $\leftarrow$ *pCos* -= ***weightedNegative****(nCos)*
14: Sort *allCos* descending
15: **return** *k* most similar board games based on *allCos*

Algorithm 2 Weighted Negative Calculation

1: **procedure** WEIGHTEDNEGATIVE(NCOS)
2: **For each** j of *non-input board games*
3: $nCos(j) \leftarrow nCos(j) * (1 - \frac{rank(j)}{recommendationListLength})$
4: **return** *nCos*

3.3.3.1 Centralized Version

In order to counteract possible *systematic tendencies*, an additional adjustment was applied to the rating matrix and labeled as separate approach. These tendencies may occur because of users who give higher ratings than others or items which receive higher ratings, for example due to popularity.

Equation 3.3 shows the adjusted rating for user i to item j entry. It gets calculated by subtracting the global average μ as well as the user b_i and item b_j bias. The global average μ is thereby the average over all observed entries and the user and analogously item bias is calculated by averaging over the corresponding entries as well.

This adjustment was inspired by Koren (2008) who also used parts of this *item-user baseline* to additionally adjust his latent factor model.

$$r'_{ij} = r_{ij} - \mu - b_i - b_j \tag{3.3}$$

3.3.4 Matrix Factorization

The idea behind *latent factor models* is that a significant amount of rows and columns of the rating matrix are correlated and therefore missing values can be estimated by a *low-rank* approximation of the original matrix (Charu C. Aggarwal, 2016). This means that a *fully specified* low-rank approximation can be created even though the original matrix has missing entries.

For the matrix factorization approach, a *UV-decomposition* or *unconstrained Matrix Factorization* (Charu C. Aggarwal, 2016) was chosen. This decomposition divides the rating matrix R_{mn} into two matrices P_{mk} and Q_{nk} with k being the amount of factors or concepts.

The ith row p_i of P, called user factors, and the jth row q_j of Q, called item factors, represent the affinity of user i and item j towards the k concepts. Equation 3.4 shows the decomposition $\widehat{R} = PQ^T$, with $\widehat{R}$ being the approximation.

$$\widehat{R}_{m,n} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,k} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,k} \end{pmatrix} * \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{k,1} & q_{k,2} & \cdots & q_{k,n} \end{pmatrix} \tag{3.4}$$

The used objective function is defined as the sum of the squared error which is the difference between the estimated and the specified rating. It is shown in Equation 3.5 with S being the set of specified entries, $S = \{r_{ij} : r_{ij} \text{ is specified}\}$.

$$\text{Minimize } J = \sum_{r_{i,j} \in S} (r_{ij} - p_i^T q_j)^2 \tag{3.5}$$

The objective function was minimized using *stochastic gradient-descent* (SGD). Equation 3.6 shows the iteratively applied update, during SGD, of the respective p_i and q_j factors with learning rate α.

$$
\begin{aligned}
p_i &\leftarrow p_i + \alpha((r_{ij} - q_j^T p_i) * q_j) \\
q_j &\leftarrow q_j + \alpha((r_{ij} - p_i^T q_j) * p_i)
\end{aligned}
\tag{3.6}
$$

For the matrix factorization approach, the learning rate α was chosen to be 0.01 and the factor k was arbitrarily set to 100.

The idea behind the item similarity calculation was the same as in the *Collaborative Filtering* approach, see Subsection 3.3.3. However, the application of the cosine similarity slightly varied in the way that it was calculated between the input board game and the $q*$ factors.

Additional variations of the matrix factorization such as *Pure SVD* and *SVD++* exist. Pure SVD, as denoted by Cremonesi, Koren, and Turrin (2010), uses an additional diagonal matrix of singular values and *SVD++*, introduced by Koren (2008), additionally incorporates implicit feedback.

3.3.5 Term Frequency – Inverse Document Frequency

The following approach combined several information sources of board games to create so-called *document units* and calculate the *Term Frequency - Inverse Document Frequency* (TF-IDF) for each term of a document unit. The resulting *term frequency matrix*, mapping board games (seen as documents) to terms, was then used to find similar board games. As no historical ratings are used, the applied approach can be categorized into content-based recommender systems.

3.3.5.1 Vector Space Model

A *Vector Space Model*, introduced by Salton (1988), is a model in which a set of text documents is represented as vectors of terms. Each dimension of a vector belongs to a different term and each term corresponds to an axis. A term can be a word, keyword or several words combined, for example word n-grams with $n > 1$.

TF-IDF

Different methods exist for calculating term weights. Simple counting of term occurrences in the document, referred as *Term Frequency* (TF), does not reflect the actual significance of each term. For example, when viewing documents about board games, the term board game will probably appear too often and not be equal significant as a term which occurs less often.

Therefore, a so-called inverted document frequency (see Equation 3.7) is used. It will be high if a term occurs less and low if a term occurs too frequently. DF_j denotes the *Document Frequency* of term j, representing in how many documents the term occurs, and N denotes the total number of documents. Additionally, the sub-linear log function has a "smoothing effect" (Management Association, 2012).

$$\text{IDF}_j = log\left(\frac{N}{DF_j}\right) \tag{3.7}$$

Equation (3.8) combines TF and IDF results in the TF-IDF weighting. The TF-IDF score for a given term j in a document i will be highest if the term occurs many times in few documents and lower if it occurs in fewer documents (low TF) or in too many documents (low IDF).

$$\text{TF-IDF}_{i,j} = TF_{i,j} * IDF_j \tag{3.8}$$

Document Similarity

As each document is represented by its document vector with its terms as components, the similarity between them can be measured. When the vector space model is used, the cosine similarity (Equation 3.9) can be preferable to other similarity functions such as *euclidean distance*, as it recognizes similar topics better (Pazzani and Billsus, 2007).

$$\text{cosineSim}(d_i, d_j) = \frac{\vec{d}_i \cdot \vec{d}_j}{||\vec{d}_i|| \cdot ||\vec{d}_j||} \tag{3.9}$$

Van Meteren and Van Someren (2000) also used the vector space model representation and cosine similarity for building their personalized recommender system called *PRES*. However, they built and iteratively updated a user profile vector beforehand and then calculated the cosine similarity between the profile vector and the document vector.

3.3.5.2 Implementation

The idea was that users might prefer items that overlap in their content information. In terms of board games, the content information that was considered suitable and decisive consisted of three parts. The *description*, *reviews* (including all replies) and user *comments* of a board game have been combined into one *document unit*.

As seen in Table 3.4, the length of the descriptions varied from no description at all to the longest one with 15 292 characters. The board game "Carcassonne" had the highest amount of comments with 13 659 and the board game "Dominion" had the most reviews with 219. The vast majority of content was written in English.

In order to create the TF-IDF matrix, the `TfidfVectorizer` from *scikit-learn*, with default settings, was used. It combines several transformation steps such as tokenization, occurrence counting and TF-IDF weighting. As the result is by default L2 normalized, it was sufficient to multiply the result matrix by its transpose for calculating the cosine similarities.

Additionally, a default stop list, containing over 1 000 stop words, was used to remove words with low information gain such as conjunctions, prepositions and similar. No explicit tokenizer was used and the default extraction mode was word by word. The resulting TF-IDF matrix had 88 564 board games and 1 049 787 terms.

The similarity calculation for positive and negative board games was similar to the calculation in the *Collaborative Filtering* approach, see Algorithm 1.

Max. Description Length (characters)	15 292
Avg. Description Length (characters)	801
# Board Games with $[1-9]$ comments	37 553
# Board Games with $[10-99]$ comments	16 444
# Board Games with $[100-999]$ comments	4499
# Board Games with $>= 1000$ comments	565
# Board Games < 5 reviews	13 488
# Board Games $[5-9]$ reviews	1696
# Board Games $[10-99]$ reviews	1290
# Board Games $>= 100$ reviews	19

Table 3.4: Selected descriptive statistics of the three data sources used for the TF-IDF calculation.

The similarities between the positive input board games and the non-input board games were accumulated. The same was applied for the negative board games and then the result was subtracted from the similarities of the positive ones.

Contrary to the *Collaborative Filtering* approach, no negative weighting was done. The final similarities were ordered descending and the k most similar board games returned.

3.3.6 Two-mode Network

For the following recommendation approach a so-called two-mode network of board games and persons was created. The terms *two-node network* and *one-mode* were initially introduced by Wasserman and Faust (1994) in the context of social network analysis. They defined a social network as "a set or set of actors and the relation or relations defined on them".

The difference between a one-mode and two-mode network is that the former require that all nodes are of one "type" or group whereas the latter consists of two different types and edges are only possible among them. In graph theory, two-mode networks are also known as *bipartite graph* and the different type sets are also referred as *top nodes* $\top$ and *bottom nodes* $\bot$.

In order to use traditional graph analysis notions, a two-mode network can be also transformed into two one-mode networks called *top projection* and *bottom projection*.

3.3.6.1 Implementation

First of all, a two-mode network, from board games and persons/organizations connected by unweighted, undirected edges, was constructed. Persons were defined to be of type *boardgamedesigner*, *boardgamepublisher* or *boardgameartist* and meant to be uniquely identifiable by their name and type.

A person can also occur as a boardgamedesigner and boardgameartist or even as boardgamepublisher at the same time. However, for the so-called person weight calculation described below, the name of the person was the only identification. For example, "Rob Fisher" is the designer, artist and publisher of the board game "Monkey Dash" but the person weight calculation would be only for the name "Rob Fisher".

The implementation behind the approach is described in Algorithm 3. The approach gets called with the input board game list (positive and negative board games) and tries to find similar ones.

Each, from the input list different, board game gets assigned a board game score *bgs* which increases, the more matching and relevant persons it has

with the positive input(s) and decreases, the more matches it has with the negative input(s). The notion of matching and relevant persons is denoted by the term *person weight pw*. Of course, they can also overlap within the input board games.

Finally, the k most similar board games are returned. Board games with persons that did not match with the input at all, will not be recommended. This means the resulting board game list can be less than the desired recommendation list length.

Algorithm 3 Network Weight Calculation

Initialize:
  $pw \leftarrow$ person weights
  $bgs \leftarrow$ board game scores
  $pbgs \leftarrow$ positive input board games
  $nbgs \leftarrow$ negative input board games
For each *pbg* of *pbgs*
  For each person name *pname* of *pbg* :
    $pw[pname] \leftarrow pw[pname] +$ ***personCount****(pbg, pname)*
For each *nbg* of *nbgs*
  For each person name *pname* of *nbg* :
    $pw[pname] \leftarrow pw[pname] -$ ***personCount****(nbg, pname)*
For each *bg* of *relevant non-input board games*
  For each person name *pname* of *bg* :
    $bgs[bg] \leftarrow bgs[bg] + pw[pname]$
Sort *bgs* descending
return k most similar board games based on *bgs* with *bgs score* > 0.0

Algorithm 4 Person Count Calculation

procedure personCount(bg, pname)
    $count \leftarrow 0.0$
    For each *person* in *bg*
      if $\leftarrow$ *pname* == *person.name*
        $count \leftarrow count + 1.0$
    return *count*

Latapy, Magnien, and Del Vecchio (2008) elaborated and extended basic notions for analyzing large two mode networks, as an alternative to analyzing one-mode projections of two-mode networks.

Table 3.5 gives an overview of the created two-mode network including the largest connected component (LCC). The bipartite density is defined by the fraction of existing links m to possible links $n_{\top} * n_{\perp}$ and is 0.00007624289 for the whole network. Introduced by Latapy, Magnien, and Del Vecchio (2008), the bipartite clustering coefficient $cc(u)$ (Equation 3.10) of a node u measures the local density in the bipartite context.

The bipartite clustering coefficient makes use of a "neighborhood overlap function" $cc(u, v)$ defined between two nodes u and v to be the fraction of actual neighbors to possible neighbors, $\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$, with N as neighborhood function.

$$cc(u) = \frac{\sum v \in N(N(u)) cc(u, v)}{N(N(u))} \tag{3.10}$$

If u and v share the exact same neighborhood, the clustering coefficient would be 1, contrary if they share fewer neighbors it moves towards 0. The average clustering coefficient of the LCC is 0.2784314722566027.

Figure 3.3 shows the second largest connected component of the two-mode network of BGG with 22 board games on the left side and 18 persons on the right side.

# Connected Components	4 711
# Connected Components with #nodes < 5	3 712
# Connected Components with #nodes $[5 - 14]$	555
# Connected Components with #nodes $>= 15$	16
#Nodes of Largest Connected Component (LCC)	122 521
#Edges of LCC	263 366
#Nodes of Board Games (top nodes)	43 982
#Nodes of Persons (bottom nodes)	78 539
Bipartite Density of LCC	0.0000762
Avg. Bipartite Clustering Coefficient of LCC	0.278
Highest Degree regarding Board Games (LCC)	"Magic: The Gathering" degree 510
Highest Degree regarding Persons (LCC)	Designer "(Uncredited)" degree 18448

Table 3.5: Overview of the generated two-mode network.

3.3.7 Post-Filtering

To further improve the results, a re-ranking of the recommended board games for each approach was done. This process was named *post-filtering* as it was applied on the returned board game recommendation list for each approach[1]. Post-filtering can be classified as a *sequential-ensemble recommender system* which is a particular type of *hybrid recommender systems*, see Subsection 2.4.5.

The underlying idea was based on the fact that each board game has additional identifying attributes such as, to which *category* it belongs, which game *mechanic* (*mechanism*) it uses or of which *family* it is a part of.

Such attributes were used to find similar board games between the positive input board games and the board games from the recommendation list. A

[1]Co-supervisor Lukas Eberhard made the suggestion to look into post-filtering techniques.

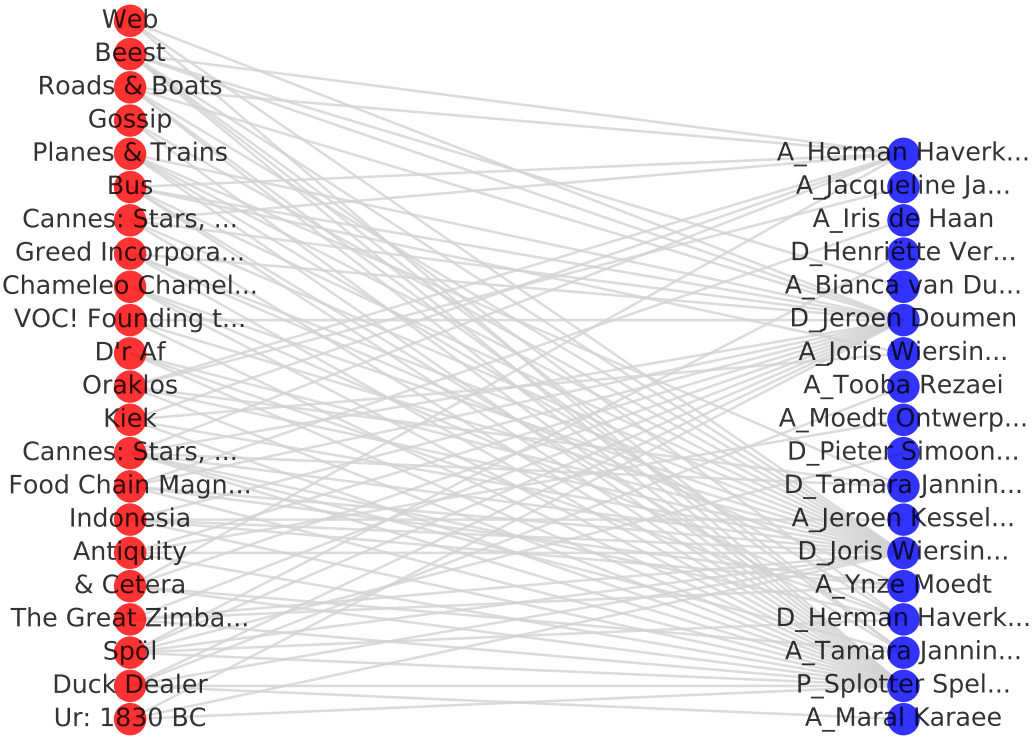

Figure 3.3: Visual representation of the second largest component of the two-mode network of BGG with 22 board games on the left side and 18 persons on the right side. The prefixes *A_*, *D_* and *P_* represent *Artist*, *Designer* and *Publisher* respectively. The graph is connected through 97 edges. The publisher "Splotter Spellen" has the highest degree of 22.

Board game category score (*bgcs*), a *Board game mechanic score* (*bgms*) and a *Board game family score* (*bgfs*) were defined to reflect the similarity between two board games.

In particular, the more overlapping attributes an input board game has, the higher its similarity gets. The respective *bg*scores* between all positive input board games and the result board game (from the recommendation list of the approach) are normalized to 1.

All *bg*scores*, including the normalized recommendation score *rcs*, are then weighted and accumulated, as seen in Equation 3.12. The final scores of each result board game get resorted in descending order for the updated recommendation list.

The optimal weight factors $*Weight$, representing the influence of a certain score, are determined in the evaluation step. For similarity calculation between attributes of two board games, the *Jaccard similarity*, as shown in Equation 3.11, was used.

$$jaccardSim(BG_a, BG_b) = \frac{|BG_a \cap BG_b|}{|BG_a \cup BG_b|} \tag{3.11}$$

$$\begin{aligned} updatedScore(resultBg, PIBG) = {} & rcs(resultBg) * recWeight + \\ & bgcs(resultBg, PIBG) * catWeight + \\ & bgms(resultBg, PIBG) * mecWeight + \\ & bgfs(resultBg, PIBG) * famWeight \end{aligned} \tag{3.12}$$

Example

Considering “Catan” and “1989: Dawn of Freedom” to be the positive input board games $PIBG$ and “Twilight Struggle” to be the first board game of the recommendation result list, denoted $resultBg$. Table 3.6 shows the attributes in more detail.

Board game	Categories	Mechanics	Families
Catan	Negotiation	Dice Rolling Hand Management Modular Board Route/Network Trading	Catan Promotional Board Games
1989: Dawn of Freedom	Political	Area Control/Influence Campaign Dice Rolling Hand Management	Country: Czech Republic Country: Germany Country: Hungary Country: Poland Country: Romania Country: Slovakia Theme: Cold War
Twilight Struggle	Modern Warfare Political Wargame	Area Control/Influence Campaign Dice Rolling Hand Management Simult. Action Selection	Country: Soviet Union Country: USA Historical Figures: Fidel Castro Theme: Cold War

Table 3.6: Attributes of board games "Catan", "1989: Dawn of Freedom" and "Twilight Struggle".

Then the $bgcs(\{TwilightStruggle\}, \{Catan, 1989 : DawnofFreedom\})$ is $\frac{1}{3} = 0.33$ because only "Twilight Struggle" and "1989: Dawn of Freedom" have one category in common.

The $bgms(\{TwilightStruggle\}, \{Catan, 1989 : DawnofFreedom\})$ is $\frac{4}{5} + \frac{2}{8} = 1.08$ and the $bgfs(\{TwilightStruggle\}, \{Catan, 1989 : DawnofFreedom\})$ is $\frac{1}{10} = 0.1$.

Assuming full weights are applied to the scores, the updated final score for the top-ranked game is then given by $1.0 * 1.0 + 0.33 * 1.0 + 1.08 * 1.0 + 0.1 * 1.0 = 2.51$.

When re-ranking the results, the amount of board games which should be considered from the result recommendation list, denoted as *reorder list length*, must be defined. Properties such as *reorder list length* as well as the respective weights $*Weight$ must be set before the recommendation run.

Those properties were considered as *hyperparameters* and extracted and evaluated using a time-based holdout method, see Subsection 3.4.2.

Ziegler et al. (2005) also re-ranged recommendations in their offline experiment. They used the results from item-based CF and user-based CF and applied a diversification algorithm to both cases, altering a diversification factor from 10% to 90%.

3.4 Evaluation

For evaluating the respective recommendation approaches, real-user board game requests including community-approved recommendations were used as *ground truth*. The board game requests with the recommendations were taken from *reddit.com* where a separate forum, a subreddit[2], exists for board games.

3.4.1 Reddit.com Submissions

On the board games subreddit, specially prefixed threads exist where users can request recommendations for board games based on ones they liked or did not like. Such threads have titles with "What should I Get" or "[WSIG]" prepended. They consist of the actual request and comments from other users discussing or recommending possibly matching board games.

In general, requests and comments consist of an author, a community voted score (which is initially 1.0) and the actual content. The author or community members can express like or dislike by "up voting" or "down voting" them.

Besides desired or undesired board games, a submission request may contain additional constraints, similar to a constraint-based recommender (see Subsection 2.4.3) such as minimum amount of players or maximum playing time. Such constraints have also been extracted and passed as additional

[2] www.reddit.com/r/boardgames/

input to the recommendation framework. As mentioned in Section 2.7, the board game request can therefore be considered as a *narrative request*.

Based on a reddit.com data dump with entries from 2011 to 1016, 150 WSIG submissions, from September 2016 to the end of December 2016, have been manually extracted. The only requirements for a request to be accepted as ground truth were that each submission request had at least ten *approved* recommendations and the request itself had a positive score.

A comment (with recommendations) was considered *approved* when it had a score greater than 1 (community approved) or the author responded on it and expressed favors (author approved).

Figure 3.4 shows the, with *pyQt4* developed, GUI and parts of an extracted submission with its comments. The submission request was created by the user "ApolloNoir" who tries to find further board games to play by him and his wife. He specifies ten board games he liked, stated that two players and more are preferable and that his wife does not like violent games.

Therefore, when extracting this request, the ten board games, the minimum amount of players and the negative category "wargame" were set. The given constraints only act as a prefilter for the recommendation framework. This means that the approaches skip board games which would not match the desired restrictions.

Table 3.7 shows selected key data from the extracted submissions and recommendations.

3.4.1.1 Accuracy Calculation

As classification accuracy metrics (see Paragraph 2.5.1.4), precision, recall and f1-score was selected to be appropriate. Rank accuracy metrics have not been used as the recommendations from the comments did not imply a certain ordering.

The f1-score between the recommendations of the approach and the recommendations from the submission were calculated, as shown exemplary in 3.8.

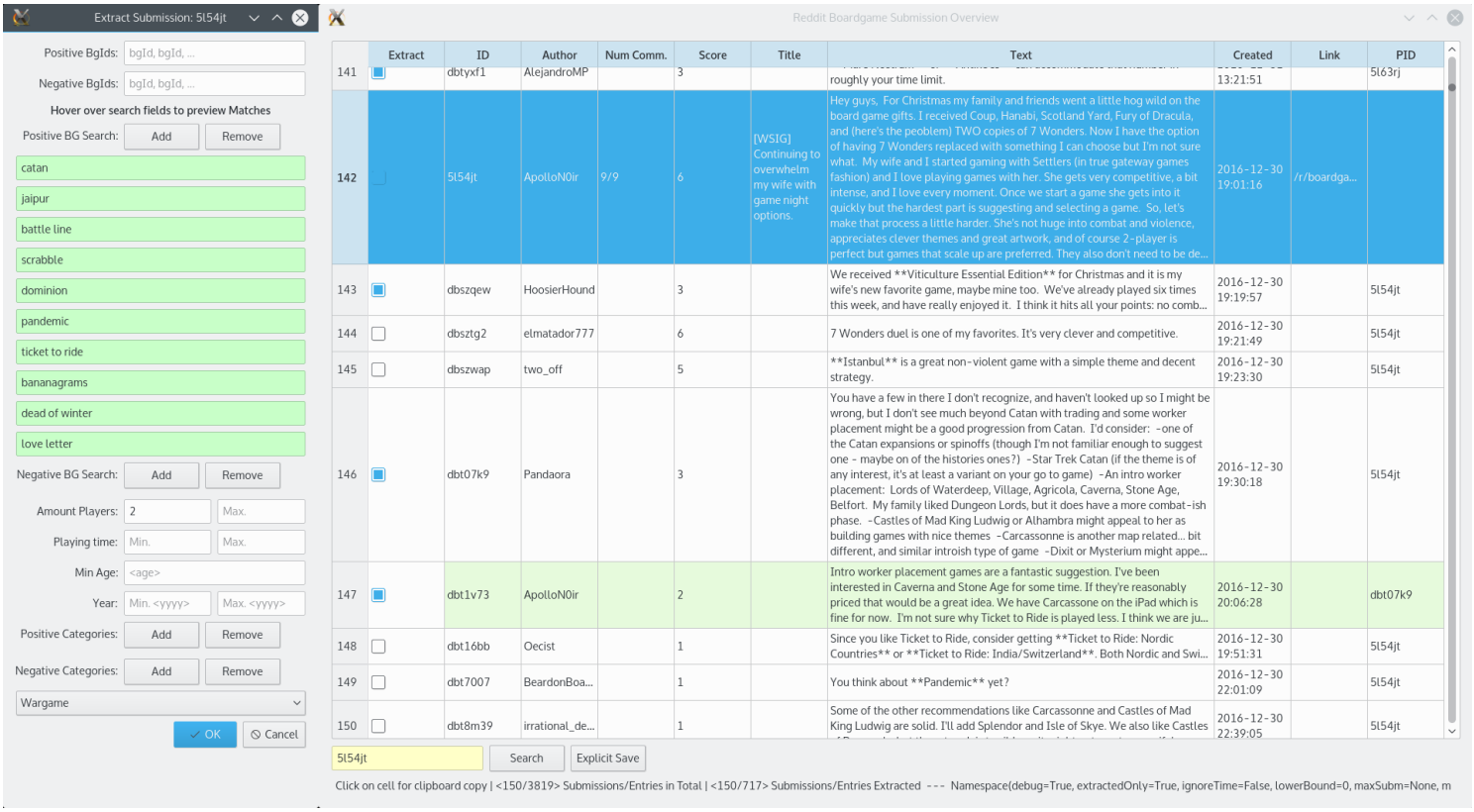

Figure 3.4: Overview of a submission (highlighted) and its comments. On double-click, an additional pop-up window opens to specify additional constraints (on submissions) or the actual recommendations (on comments).

#Submissions:	150
Avg. Submission Score:	8.22
Avg. #Positive BGs (per Submission)	4.28
Avg. #Negative BGs (per Submission)	0.29
#Submissions with no filter:	40
#Submissions with amount players:	96
#Submissions with playing time:	47
#Submissions with minimum age:	1
#Submissions with year:	0
#Submissions $\#categories >= 1$:	26
#Recommendations:	2 628
#Unique Recommendations (UR):	838
#Recommendations which are expansions:	55
Avg. #Recommendations (per Submission):	17.52
#Recommendations in Top 10	85
#Recommendations in Top 50	408
Most frequent board game (test set)	"Codenames"(8)
Most frequent board game (training set)	"Carcassonne"(25)

Table 3.7: Selected key data of the extracted submissions.

BGIDs from the result recommendation (per approach):	$1, 4, 6, 9$
BGIDs from the reddit submissions (ground truth):	$4, 9, 10, 20, 21$
Precision:	$\frac{2}{4} = 0.5$
Recall:	$\frac{2}{5} = 0.4$
F1-Score:	0.44

Table 3.8: Example calculation of the accuracy for evaluation.

3.4.2 Experimental Setup

The performance of each approach was reported as the average over all f1-scores of each submission. The returned amount of recommendations from the framework was fixed by list length 10 as it seemed to be most appropriate and is a typical choice, see Subsection 2.3.1. This procedure was partially similar to the experimental setup of Sarwar et al. (2000) who also used a fixed list length of 10 and reported an average f1-score as performance metric.

For the f1-score calculation of each approach, a time-based split (see Paragraph 2.5.1.1) was done. The latest 20 percent of the 150 submissions were used for reporting the f1-score accuracy. A train or test split was omitted as the approaches operated on unseen data and had no model parameters to learn.

In contrast, the *weights* of the post-filtering scores as well as the optimal *reorder list length* at which re-ranking should be done, had to be selected. Therefore, a manual grid search with alternating score weights from $\{0, 20, 40, 60, 80, 100\}$ percent had been executed on the oldest 120 submissions, considered as *training set*, and validated on the latest 30 submissions (*test set*). Additionally, the reorder list length was varied between 50, 100 and 200 for all approaches and the reorder list length yielding the best results was selected.

1. Pandemic Legacy: Season 1	8.47
2. Through the Ages: A New Story of Civilization	8.31
3. Twilight Struggle	8.22
4. Terra Mystica	8.15
5. Star Wars: Rebellion	8.14
6. Scythe	8.08
7. 7 Wonders Duel	8.03
8. Caverna: The Cave Farmers	8.02
9. The Castles of Burgundy	7.99
10. Puerto Rico	7.99

Table 3.9: Top ten board games with their geek rating on the right side.

3.4.3 Baseline

For relative performance comparison, a naive baseline of *most-popular* board games has been chosen. The board games are ordered descending according to their geek rating. The first top ten games are shown in Table 3.9 with "Pandemic Legacy: Season 1" being on top.

4 Results

This chapter presents the results of the applied recommendation approaches and post-filtering method described in Chapter 3. First of all, the recommendation approaches are used to recommend 10 board games and are applied and evaluated on the latest 30 submissions including their recommendations. As described in Subsection 3.4.2, these 30 submissions represent the *test set* for reporting the f1-scores.

Improvements are achieved by further restricting the recommendation results to not include board game expansions which are related to the input board games. Finally, the results on optimizing the post-filtering *weights* on the training set and the accuracy results on the test set are shown.

For more clarity, the following notations were introduced:

- *NW* $\longrightarrow$ Network approach
- *TF-IDF* $\longrightarrow$ Term Frequency – Inverse Document Frequency approach
- *MF* $\longrightarrow$ Matrix Factorization approach
- *CF* $\longrightarrow$ Collaborative Filtering approach
- CF_{centr} $\longrightarrow$ Centralized Version of Collaborative Filtering approach
- *Baseline* $\longrightarrow$ Baseline approach

4.1 Default Run

Figure 4.1 and Table 4.1 show the average f1-score of the test set. The run has been referred to as *Default Run* as no additional restrictions were made.

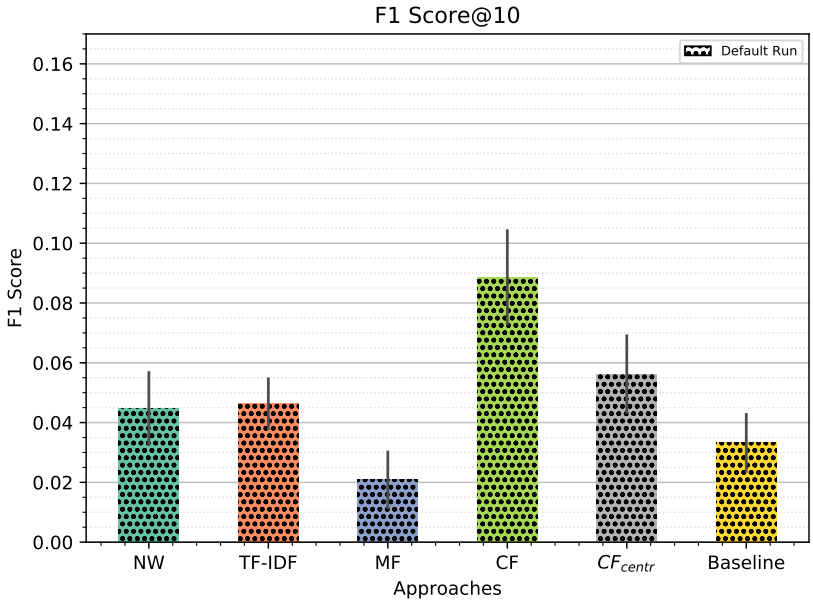

Figure 4.1: Barplot of the average f1-score for each approach on recommending 10 board games.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
F1-Score @10	0.045	0.046	0.021	0.089	0.056	0.033
Max F1-Score @ LL	0.053@22	0.051@24	0.043@34	0.119@44	0.079@21	0.079@73
Std. Error $\pm$	0.012	0.009	0.01	0.016	0.013	0.01

Table 4.1: Table showing the average f1-score at list length 10, the highest f1-score and the standard error of the mean. *LL* denotes the *list length* and shows at which length the highest f1-score occurred.

4.2 Ignore Expansions Run

4.2.1 F1-Score per Approach at List Length 10

Figure 4.2 and Table 4.2 show the average f1-score of the test set. The run has been referred to as *Ignore Expansions Run* as board games which are expansions of the input board games have been skipped.

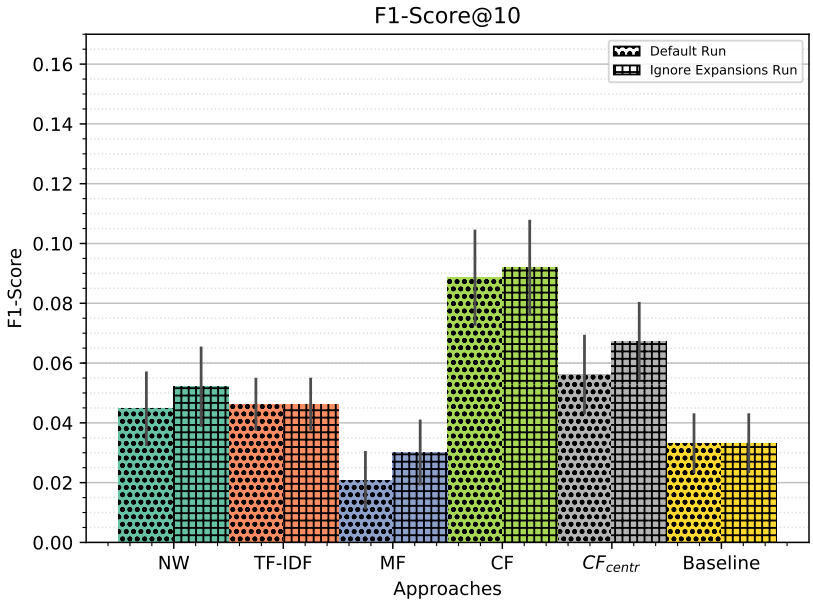

Figure 4.2: Barplot of the average f1-score for each approach on recommending 10 board games.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
F1-Score @10	0.052	0.046	0.03	0.092	0.067	0.033
Std. Error ±	0.013	0.009	0.011	0.016	0.013	0.01

Table 4.2: Table showing the average f1-score at list length 10 and the standard error of the mean.

4.2.2 F1-Score per Approach

Figure 4.3 and Table 4.3 show the average f1-score of the test set for different recommendation list lengths varying from 1 to 100.

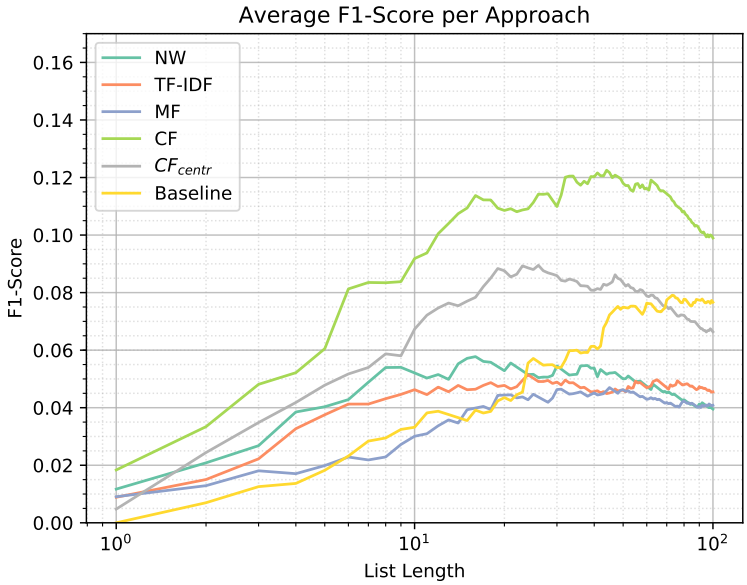

Figure 4.3: The plot shows the average f1-score for each approach for different recommendation list lengths.

	NW	TF-IDF	MF	CF	CF_{centr}	Baseline
Max F1-Score @ LL	0.058@16	0.051@24	0.047@45	0.123@44	0.089@26	0.079@73

Table 4.3: Table showing the maximum f1-score at different list lengths (LL).

4.2.3 Precision per Approach

Figure 4.4 and Table 4.4 show the average precision of the test set for different recommendation list lengths varying from 1 to 100.

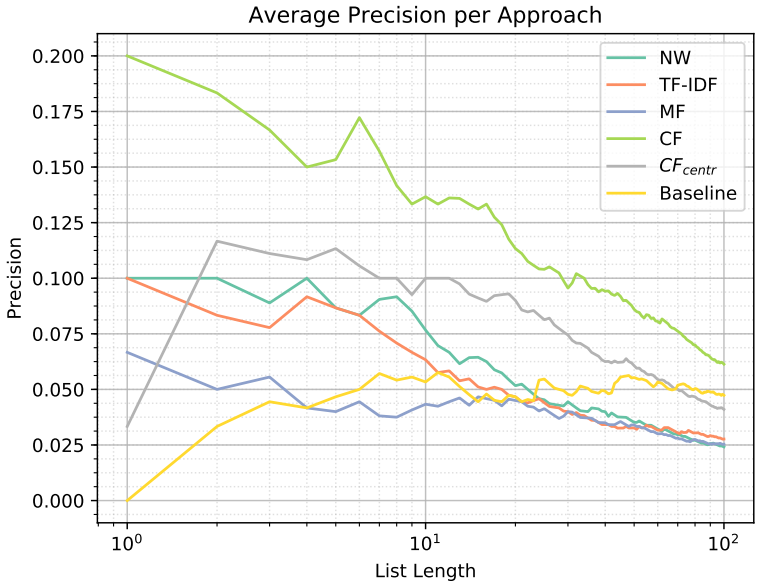

Figure 4.4: The plot shows the average precision for each approach for different recommendation list lengths.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
Max Precision @ LL	0.1@1	0.1@1	0.067@1	0.2@1	0.116@2	0.058@11

Table 4.4: Table showing the maximum precision at different list lengths (LL).

4.2.4 Recall per Approach

Figure 4.5 and Table 4.5 show the average recall of the test set for different recommendation list lengths varying from 1 to 100.

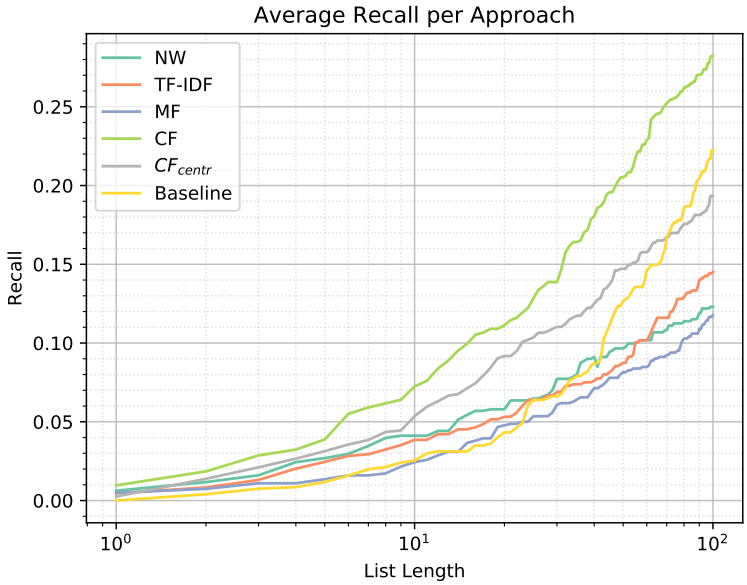

Figure 4.5: The plot shows the average recall for each approach for different recommendation list lengths.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
Max Recall @ LL	0.123@98	0.15@100	0.118@100	0.282@99	0.193@98	0.222@99

Table 4.5: Table showing the maximum recall at different list lengths (LL).

4.2.5 Amount of Submissions Used

Figure 4.6 shows how many submissions were taken into account to achieve the desired requested recommendation list length.

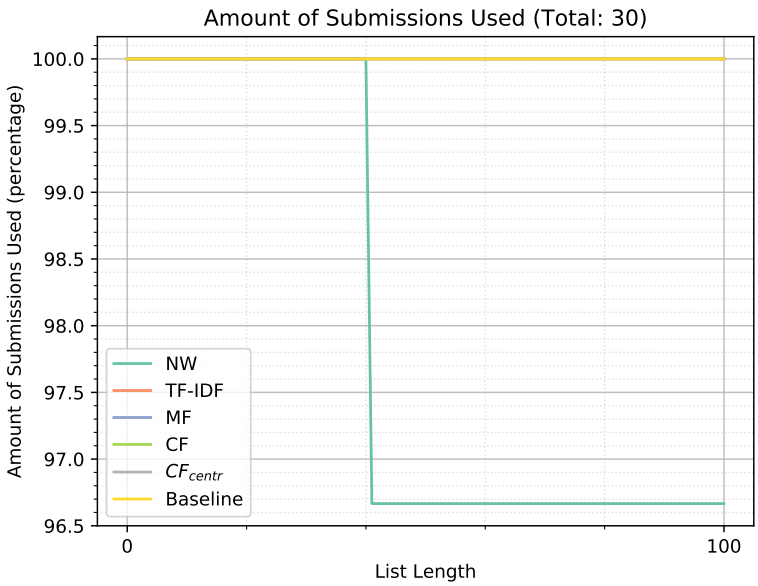

Figure 4.6: Overview showing how many submissions were taken into account for evaluation at different list lengths. All approaches except the network approach used all 30 submissions at all list lengths. At list length 41 onwards, the network approach skipped 1 submission as it was not able to provide more than 41 recommendations for this submission.

4.3 Post-Filtering

4.3.1 F1-Score per Approach at List Length 10

Figure 4.7 and Table 4.6 show the average f1-score of the test set after post-filtering has been applied. The run has been referred to as *Grid Search 200 Run* as the reorder list length was set to 200.

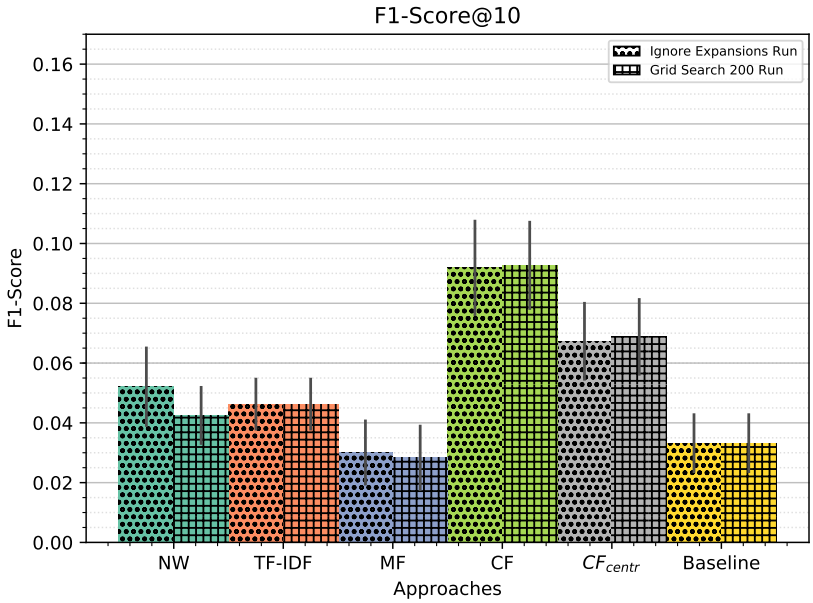

Figure 4.7: The plot shows two bars each representing a run with the average f1-score at list length 10. The left bar shows the scores without post-filtering and the right bar with post-filtering applied.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
F1-Score @10 (left)	0.052	0.046	0.03	0.092	0.067	0.033
F1-Score @10 (right)	0.042	0.046	0.028	0.093	0.069	0.033
Std. Error $\pm$	0.01	0.009	0.011	0.015	0.013	0.01

Table 4.6: Table showing the average f1-score at list length 10 and the standard error of the mean.

4.3.2 Grid Search Weights

Figure 4.8 shows the obtained grid search *weights* from the grid search on the training set.

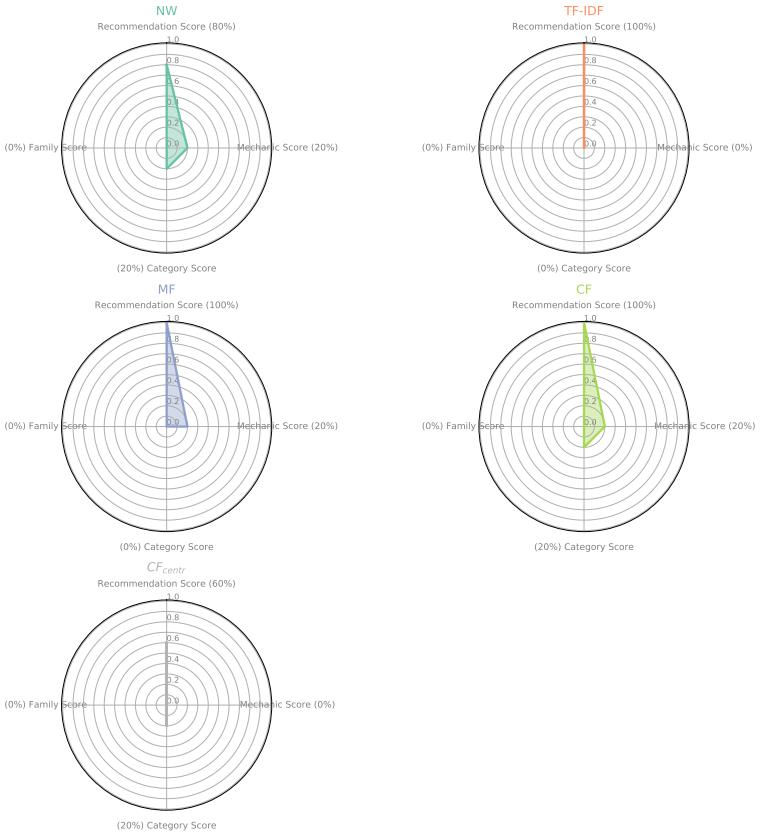

Figure 4.8: Overview of the obtained grid search weights for each approach.

NW	TF-IDF	MF	CF	CF_{centr}
0.8|0.2|0.2|0	1.0|0|0|0	1.0|0.2|0|0	1.0|0.2|0.2|0	0.6|0|0.2|0

Table 4.7: The obtained grid search *weights* presented in the form <recommendation weight> | <mechanic weight> | <category weight> | <family weight> .

4.3.3 F1-Score per Approach

Figure 4.9 and Table 4.8 show the average f1-score of the test set for different recommendation list lengths varying from 1 to 100.

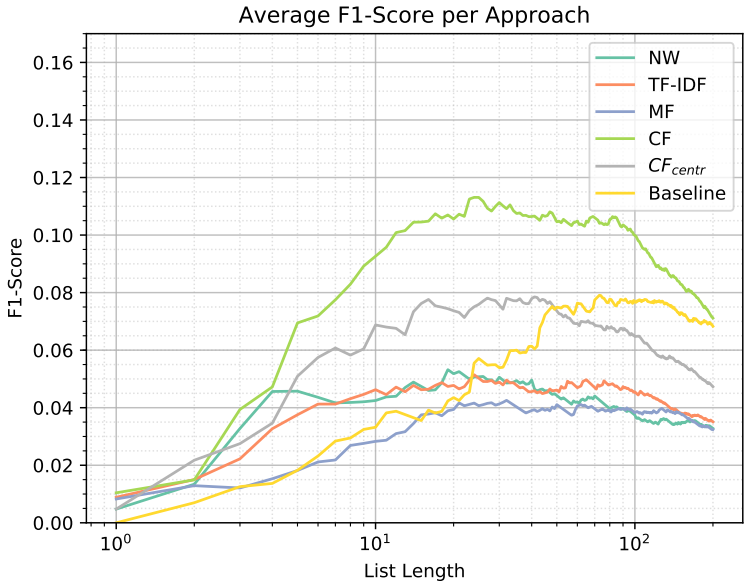

Figure 4.9: The plot shows the average f1-score for each approach for different recommendation list lengths.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
Max F1-Score @ LL	0.053@19	0.051@24	0.043@32	0.113@24	0.079@41	0.079@73

Table 4.8: Table showing the maximum f1-score at different list lengths (LL).

4.3.4 Precision per Approach

Figure 4.10 and Table 4.9 show the average precision of the test set for different recommendation list lengths varying from 1 to 100.

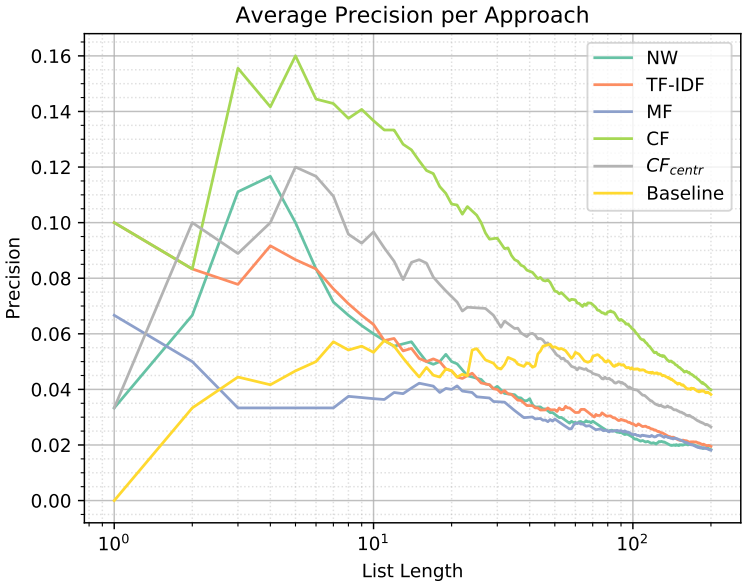

Figure 4.10: The plot shows the average precision for each approach for different recommendation list lengths.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
Max Precision @ LL	0.12@4	0.1@1	0.067@1	0.16@5	0.12@5	0.058@11

Table 4.9: Table showing the maximum precision at different list lengths (LL).

4.3.5 Recall per Approach

Figure 4.11 and Table 4.10 shows the Figure and Table shows the Figure and Table shows the

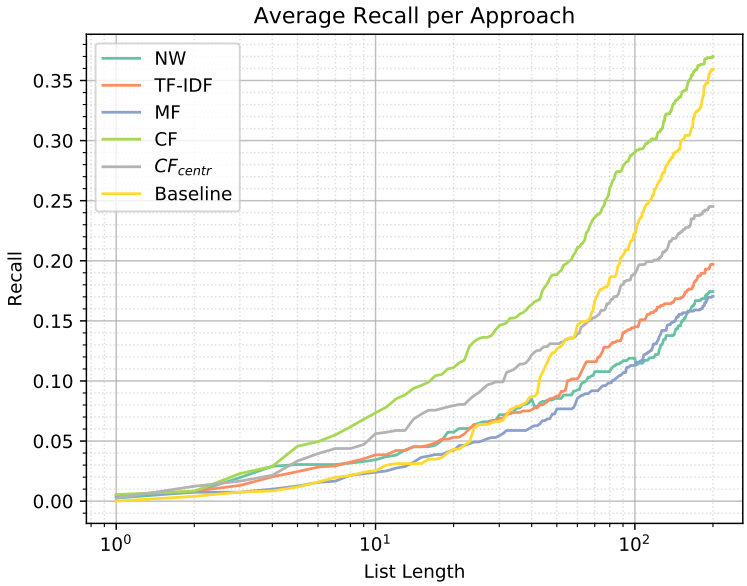

Figure 4.11: The plot shows the average recall for each approach for different recommendation list lengths.

	NW	**TF-IDF**	**MF**	**CF**	**CF_{centr}**	**Baseline**
Max Recall @ LL	0.174@194	0.197@196	0.171@199	0.37@199	0.245@194	0.359@198

Table 4.10: Table showing the maximum recall at different list lengths (LL).

5 Discussion

This chapter discusses the evaluation results presented in Chapter 4 answering the research questions defined in Chapter 1.

1. *In the domain of board games, how well are standard recommendation approaches performing when applied to real-user search queries?*

Figure 4.1 and Table 4.1 show that the average f1-score of almost all approaches perform better than the *baseline*. The average f1-score of the *CF* approach is almost three times as high as the one of the *baseline*. However, the *MF* approach has a lower f1-score than the *baseline*. One of the reasons for this might be that the *MF* approach has no regularization term which could reduce overfitting towards the training set.

The centralized version of the collaborative filtering approach did not result in the expected accuracy gain. The suggested integration of user and item biases as well as the global average might be too much correction.

An improved run on almost all approaches was achieved by skipping board games from the recommendation list which are expansions of any input board game. This run was denoted as *Ignore Expansions Run*. Figure 4.2 and Table 4.2 show the accuracy gain with the CF_{centr} approach having the highest improvement by 19 percent.

Figure 4.3 and Table 4.3 show that a higher accuracy score could be achieved by taking a longer recommendation list length. The respective precision (Figure 4.4) and recall (Figure 4.5) graphs also confirm the relationship between the recommendation list length and precision/recall which is also mentioned by Shani and Gunawardana (2011).

2. *Can content-based approaches be particularly advantageous over collaborative-based ones?*

As described in Chapter 3, the approaches *MF*, *CF* and CF_{centr} are considered collaborative-based whereas the approaches *TF-IDF* and *NW* are considered content-based.

With the exception of the *MF* approach, the *CF* and CF_{centr} and therefore collaborative-based approaches performed better than the *TF-IDF* and *NW* approach. This assumption also stays valid at higher recommendation list lengths as Figure 4.3 and Table 4.3 show.

However, a general advantage of the content-based approaches could be in an item cold-start scenario. When a new item with few ratings occurs as input, the *TF-IDF* and *NW* approaches could have an advantage.

Figure 4.6 also shows the limitation of the network approach which was not able to provide more than 41 recommendations for a specific submission. This restriction is not as much present in the other approaches due to the underlying workings of the *NW* approach.

3. *What improvements can be achieved by reordering the top recommendations of the respective approach?*

The grid search reorder list length was varied between 50, 100 and 200 during the grid search on the training set. However, the grid search reorder list length at 200, as seen in Figure 4.7, was selected and presented as it was the only one with minor improvements.

The *CF* approach as well as the CF_{centr} had minor improvements compared to the run without post-filtering. The *NW* and *MF* approaches performed worse and the *TF-IDF* approach had the best results with no *bg*Scores* meaning no post-filtering was done.

As seen in Table 4.7, the general influence of the obtained *bg*Scores* is very low. The *bgFamilyScore* did not play any role at all meaning the highest f1-scores were achieved by omitting it. The *NW*, *MF* and *CF* approaches had the highest *mechanicWeight* (*mecWeight*) with 20 percent. Additionally, *NW* and *CF* had a *categoryWeight* (*catWeight*) of also 20 percent. The CF_{centr}

approach was the only one where the original recommendation score from the approach was only incorporated with 60 percent.

Although all of the selected *weights* achieved an enhancement on the training set, this result could not be fully reproduced on the test set. One reason for this might be that the *weights* are too much tailored to the training set, meaning *overfitting* occurred. As the post-filtering method prioritizes board games with similar attributes, it could also be that the approved recommendations from the submissions of the test set are more diverse than the ones of the training set.

For future recommendation requests, the highest performing *CF* approach with 20 percent of the *bgMechanicScore* and *bgCategoryScore* would be selected.

6 Conclusion

This thesis presented different recommendation approaches applied in the domain of board games and evaluated on real-user search queries.

Chapter 1 showed the need for a recommender system in the domain of board games. The three research questions

1. In the domain of board games, how well are standard recommendation approaches performing when applied to real-user search queries?
2. Can content-based approaches be particularly advantageous over collaborative-based ones?
3. What improvements can be achieved by reordering the top recommendations of the respective approach?

were introduced as well as methods to answer them.

The theoretical background for the approaches and evaluation methods was provided in Chapter 2. First of all, an introduction to recommender systems including their terminology, tasks and goals was given. Basic models of recommender systems as well as evaluation methods were presented. The domain description and the data basis *boardgamegeek.com* (BGG) was highlighted, as well as related work particularly in the domain of board games. Finally, the Knowledge Discovery Process was shown as steps of it were used for the extraction process.

Chapter 3 gave an overview about the development platform and explained how board games have been extracted from BGG and stored in the relational database management system. It has been shown how the four recommendation approaches, of which two are collaborative-based and two are content-based, were incorporated into the created recommendation framework.

Additionally, a post-filtering method was implemented as sequential-ensemble recommendation approach. Finally, the steps of the evaluation which included the manual submission extraction from *reddit.com* and the experimental setup were presented.

The evaluation results of the approaches were shown in Chapter 4. Accuracy scores of a *Default Run*, without any additional configurations, and an improved *Ignore Expansions Run* were presented. Finally, accuracy results as well as optimal weights of the post-filtering method, obtained from the training set, were shown.

The discussion about the obtained evaluation results was covered in Chapter 5. Almost all recommendation approaches could surpass the baseline, except the *Matrix Factorization* approach. The results for the two best performing approaches *CF* and CF_{centr} have been further improved by using post-filtering, even if only minimally. The highest performing *CF* approach with 20 percent of the *bgMechanicScore* and *bgCategoryScore* would be preset for unseen recommendation requests.

Future work can be done in optimizing the hyperparameters of the respective approaches and adding new approaches. An additional matrix factorization approach which incorporates implicit feedback could be implemented, similar to *SVD++* (Koren, 2008). Therefore, the collections of the BGG users could be used as a source of implicit feedback. Furthermore, additional scores for the post-filtering approach could be examined.

Generally, a larger data basis for evaluation would be beneficial. This could be achieved by automating the reddit submission extracting process.

One possibility would be to enforce predefined templates for board game requests on reddit, similar to an online evaluation. Another possibility, with its own research field called *Natural Language Processing*, would be to automatically detect the semantics behind the submissions and therefore being able to process unstructured requests. This would also allow to use historical submission data.

Appendix

Bibliography

Adomavicius, Gediminas and Alexander Tuzhilin (2011). "Context-Aware Recommender Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 217–253. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_7. URL: https://doi.org/10.1007/978-0-387-85820-3_7 (cit. on p. 27).

Agarwal, Deepak K and Bee-Chung Chen (2015). *Statistical methods for recommender systems*. Cambridge University Press (cit. on p. 17).

Aggarwal, Charu C. (2016). *Recommender Systems: The Textbook*. Cham: Springer. ISBN: 978-3-319-29657-9. DOI: 10.1007/978-3-319-29659-3 (cit. on pp. 4, 6–16, 21, 45).

Aggarwal, Charu C, Zheng Sun, and S Yu Philip (1998). "Online Generation of Profile Association Rules." In: *KDD*, pp. 129–133 (cit. on p. 11).

Amatriain, Xavier et al. (2011). "Data mining methods for recommender systems." In: *Recommender systems handbook*. Springer, pp. 39–71 (cit. on pp. 10, 15, 17).

API2, Boardgamegeek.com (2018). *BGG XML API2*. https://boardgamegeek.com/wiki/page/BGG_XML_API2. [Online; accessed 20-April-2018] (cit. on p. 32).

Aranda, Jorge et al. (2007). "An online social network-based recommendation system." In: *Toronto, Ontario, Canada* (cit. on pp. 27, 37).

Berkley (2003). *How Much Information 2003?* http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/internet.htm. [Online; accessed 25-November-2017] (cit. on p. 1).

Bogers, Toine and Marijn Koolen (2017). "Defining and Supporting Narrative-driven Recommendation." In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: ACM, pp. 238–242. ISBN: 978-1-4503-4652-8. DOI: 10.1145/3109859.3109893. URL: http://doi.acm.org/10.1145/3109859.3109893 (cit. on pp. 27, 28).

Breese, John S, David Heckerman, and Carl Kadie (1998). "Empirical analysis of predictive algorithms for collaborative filtering." In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 43–52 (cit. on pp. 7, 15, 21).

Bridge, Derek et al. (2005). "Case-based recommender systems." In: *The Knowledge Engineering Review* 20.3, pp. 315–320 (cit. on p. 11).

Burke, Robin (2000). "Knowledge-Based Recommender Systems." In: 69 (cit. on p. 11).

Burke, Robin (2002). "Hybrid recommender systems: Survey and experiments." In: *User modeling and user-adapted interaction* 12.4, pp. 331–370 (cit. on p. 11).

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin (2010). "Performance of Recommender Algorithms on Top-n Recommendation Tasks." In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: ACM, pp. 39–46. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864721. URL: http://doi.acm.org/10.1145/1864708.1864721 (cit. on pp. 6, 21, 46).

FAQ, Boardgamegeek.com (2018). *Boardgamegeek.com FAQ*. https://boardgamegeek.com/wiki/page/BoardGameGeek_FAQ. [Online; accessed 27-April-2018] (cit. on p. 22).

Faryal, Ali et al. (2015). "Data mining based recommendation system using social websites." In: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2015 IEEE/WIC/ACM International Conference on*. Vol. 1. IEEE, pp. 365–368 (cit. on p. 27).

Fayyad, UM, G Piatetsky-Shapiro, and P Smyth (1996). "Knowledge Discovery and Data Mining: Towards a Unifying Framework." In: *Int Conf on Knowledge Discovery and Data Mining*, pp. 82–88. DOI: 10.1.1.27.363. URL: http://www.aaai.org/Papers/KDD/1996/KDD96-014 (cit. on pp. 1, 29).

Felfernig, Alexander and Robin Burke (2008). "Constraint-based recommender systems: technologies and research issues." In: *Proceedings of the 10th international conference on Electronic commerce*. ACM, p. 3 (cit. on p. 11).

GroupLens, Research (2008). *MovieLens Dataset*. https://grouplens.org/datasets/movielens/. [Online; accessed 10-June-2018] (cit. on p. 14).

Gunawardana, Asela and Guy Shani (2009). "A survey of accuracy evaluation metrics of recommendation tasks." In: *Journal of Machine Learning Research* 10.Dec, pp. 2935–2962 (cit. on p. 20).

Herlocker, Jonathan L et al. (2004). "Evaluating collaborative filtering recommender systems." In: *ACM Transactions on Information Systems (TOIS)* 22.1, pp. 5–53 (cit. on pp. 5, 18–21).

IBM (2011). *IBM Study: Digital Era Transforming CMO's Agenda, Revealing Gap In Readiness*. http://www-03.ibm.com/press/us/en/pressrelease/35633.wss. [Online; accessed 19-November-2017] (cit. on p. 1).

Kendall, M. G. (1938). "A new measure of rank correlation." In: *Biometrika* 30.1-2, pp. 81–93. DOI: 10.1093/biomet/30.1-2.81 (cit. on p. 21).

Koren, Yehuda (2008). "Factorization meets the neighborhood: a multifaceted collaborative filtering model." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 426–434 (cit. on pp. 44, 46, 79).

Koren, Yehuda (2009). "The bellkor solution to the netflix grand prize." In: *Netflix prize documentation* 81, pp. 1–10 (cit. on p. 14).

Latapy, Matthieu, Clémence Magnien, and Nathalie Del Vecchio (2008). "Basic notions for the analysis of large two-mode networks." In: *Social networks* 30.1, pp. 31–48 (cit. on p. 52).

Leifer, Charles (2017). *Peewee ORM*. http://docs.peewee-orm.coms. [Online; accessed 20-April-2017] (cit. on p. 35).

Linden, G., B. Smith, and J. York (2003). "Amazon.com recommendations: item-to-item collaborative filtering." In: *IEEE Internet Computing* 7.1, pp. 76–80. ISSN: 1089-7801. DOI: 10.1109/MIC.2003.1167344 (cit. on p. 6).

Management Association, I.R. (2012). *Data Mining: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*. Contemporary research in information science and technology Bd. 1. Information Science Reference. ISBN: 9781466624566. URL: https://books.google.at/books?id=oLqeBQAAQBAJ (cit. on p. 47).

Mei, Guobiao (2008). *Dimensionality reduction algorithms with applications to collaborative data and images*. University of California, Riverside (cit. on p. 27).

Mei, Guobiao and Christian R Shelton (2012). "Visualization of collaborative data." In: *arXiv preprint arXiv:1206.6850* (cit. on p. 27).

Netflix (2012). *Netflix Recommendations: Beyond the 5 stars (Part 1)*. https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429. [Online; accessed 10-November-2017] (cit. on p. 5).

Pazzani, Michael J and Daniel Billsus (2007). "Content-based recommendation systems." In: *The adaptive web*. Springer, pp. 325–341 (cit. on p. 47).

Piatetsky-Shapiro, Gregory (1991). "Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop." In: *AI Mag.* 11.5, pp. 68–70. ISSN: 0738-4602. URL: http://dl.acm.org/citation.cfm?id=124898.124915 (cit. on p. 29).

Pu, Pearl et al. (2011). *Usability Guidelines for Product Recommenders Based on Example Critiquing Research.* (Cit. on pp. 5, 6).

Rajaraman, Anand and Jeffrey David Ullman (2011). *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press. ISBN: 1107015359, 9781107015357 (cit. on pp. 1, 3).

Resnick, Paul, Neophytos Iacovou, et al. (1994). "GroupLens: an open architecture for collaborative filtering of netnews." In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, pp. 175–186 (cit. on p. 8).

Resnick, Paul and Hal R. Varian (1997). "Recommender Systems." In: *Commun. ACM* 40.3, pp. 56–58. ISSN: 0001-0782. DOI: 10.1145/245108.245121. URL: http://doi.acm.org/10.1145/245108.245121 (cit. on p. 3).

Ricci, Francesco et al. (2011). *Recommender Systems Handbook*. Vol. 532, pp. 9–16. ISBN: 978-0-387-85819-7. DOI: 10.1007/978-0-387-85820-3. arXiv: arXiv:1011.1669v3. URL: http://link.springer.com/10.1007/978-0-387-85820-3 (cit. on pp. 3, 5).

Rijsbergen, C. J. Van (1979). *Information Retrieval*. 2nd. Newton, MA, USA: Butterworth-Heinemann. ISBN: 0408709294 (cit. on p. 20).

Salton, Gerald, ed. (1988). *Automatic Text Processing*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-2:1-1227-8 (cit. on p. 46).

Sarwar, Badrul et al. (2000). "Analysis of recommendation algorithms for e-commerce." In: *Proceedings of the 2nd ACM conference on Electronic commerce*. ACM, pp. 158–167 (cit. on pp. 42, 61).

Sarwar, Badrul et al. (2001). "Item-based Collaborative Filtering Recommendation Algorithms." In: *Proceedings of the 10th International Confer-*

ence on World Wide Web. WWW '01. Hong Kong, Hong Kong: ACM, pp. 285–295. ISBN: 1-58113-348-0. DOI: 10.1145/371920.372071. URL: http://doi.acm.org/10.1145/371920.372071 (cit. on p. 8).

Schafer, J Ben, Dan Frankowski, et al. (2007). "Collaborative filtering recommender systems." In: *The adaptive web*. Springer, pp. 291–324 (cit. on pp. 4, 5, 7).

Schafer, J Ben, Joseph Konstan, and John Riedl (1999). "Recommender systems in e-commerce." In: *Proceedings of the 1st ACM conference on Electronic commerce*. ACM, pp. 158–166 (cit. on p. 6).

Shani, Guy and Asela Gunawardana (2011). "Evaluating recommendation systems." In: *Recommender systems handbook*. Springer, pp. 257–297 (cit. on pp. 6, 12, 14, 19, 20, 75).

Spearman, C. (1904). "The Proof and Measurement of Association Between Two Things." In: *American Journal of Psychology* 15, pp. 88–103 (cit. on p. 21).

Van Meteren, Robin and Maarten Van Someren (2000). "Using content-based filtering for recommendation." In: *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pp. 47–56 (cit. on p. 48).

W3C (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/xml/#charsets. [Online; accessed 20-April-2017] (cit. on p. 34).

Wasserman, S. and K. Faust (1994). *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press. ISBN: 9780521387071. URL: https://books.google.at/books?id=CAm2DpIqRUIC (cit. on p. 50).

Wischenbart, Martin et al. (2015). "Recommender Systems for the People-Enhancing Personalization in Web Augmentation." In: *IntRS@ RecSys*, pp. 53–60 (cit. on p. 27).

Yao, YY (1995). "Measuring retrieval effectiveness based on user preference of documents." In: *Journal of the American Society for Information Science* 46.2, pp. 133–145 (cit. on p. 21).

Ziegler, Cai-Nicolas et al. (2005). "Improving recommendation lists through topic diversification." In: *Proceedings of the 14th international conference on World Wide Web*. ACM, pp. 22–32 (cit. on p. 57).