Recommending Systems for Board Games

(Systemy Rekomendujące dla Gier Planszowych)

Adrian Urbański Maria Wyrzykowska

Praca inżynierska

Promotor: dr Piotr Lipiński

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

February 2022

Abstract

The goal of recommender systems, finding their use in more and more domains and services, is suggesting items relevant to users. These models should be able to learn users' preferences based on their previous activities and use that knowledge in order to create personalized recommendations. In comparison to previous applications of recommender systems in the domain of board gaming, we decided to put more focus on features of recommended games and their match to users' likings. In order to do so, we devised a metric evaluating that match. Then, we modified the training process of an existing model, so that it optimizes that metric. We compared the obtained results with those produced by other algorithms. While the recommendations of our model satisfy our expectations, it would require conducting more extensive tests to evaluate their practical relevance.

Zadaniem systemów rekomendacyjnych, wykorzystywanych w coraz większej liczbie dziedzin i serwisów, jest proponowanie użytkownikom przedmiotów mogących wzbudzić ich zainteresowanie. Takie modele powinny potrafić nauczyć się preferencji użytkowników na podstawie ich dotychczasowych zachowań i używać tej wiedzy do tworzenia personalizowanych rekomendacji. W porównaniu do dotychczasowych aplikacji systemów rekomendacyjnych w dziedzinie gier planszowych, w naszej pracy chcieliśmy zwrócić szczególną uwagę na cechy polecanych gier i ich dopasowanie do upodobań użytkowników. W tym celu stworzyliśmy metrykę oceniającą to dopasowanie. Następnie dokonaliśmy modyfikacji w procesie trenowania istniejącego modelu, tak aby optymalizował on wartość tej metryki. Otrzymane wyniki porównaliśmy z rezultatami innych algorytmów. Zestawy rekomendacji oferowane przez nasz model spełniają nasze oczekiwania, ale sprawdzenie czy szersza grupa użytkowników byłaby nimi usatysfakcjonowana wymaga dodatkowych testów.

Contents

Chapter 1

Introduction

1.1 Recommender systems

For the last few decades, due to rapid growth of e-commerce and other web services, the need for personalized content has been steadily increasing. In order to meet that demand, a lot of research was conducted on the concept of recommender systems. They are a subclass of information filtering systems that try to find items which may be interesting to a certain user, based on their previous preferences. The systems can and are used in variety of areas: generating playlists at music services such as Spotify, personalizing items presented for purchase at Amazon [19] suggesting new movies or series to check out at streaming services like Netflix [13] or even potential partners at dating sites [20]. Produced recommendations can be advantageous both for the business, increasing income, and consumers, saving time to find what they look for or resulting in pleasant new discoveries.

Traditional methods used in recommender systems often are based on either collaborative or content-based approach. Collaborative filtering [21] is based on an assumption that if we find two similar users, an item liked by one of them is more probable to be liked by the other. Simple collaborative filtering approaches can be based on kNN [27], while more sophisticated ones can employ learning of users and items' latent representations in order to produce viable recommendations. Content-based recommendations focus on the attributes of the items that might be relevant to users, such as e.g. category or color, in order to recommend items with similar properties.

Current state-of-the-art models are often sophisticated extensions of these basic methods. New methods often employ neural networks due to their overwhelming successes in other fields. Amongst such approaches are EASE [30] or Graph Neural Networks [12]. Other ideas include optimally choosing one of multiple pretrained models for the task using Multi-Armed Bandit method [11].

1.2 Board games

Board gaming as a hobby has been experiencing a rapid surge in popularity in the recent years. The enormous success of The Settlers of Catan, published in 1995, resulted in many new people picking up board gaming. Nowadays, thanks to recent developments, such as the rise of funding platforms (e.g. Kickstarter), the hobby is doing better than ever and many people consider it to be experiencing a new Golden Age.

Board games can feature multiple different genres, levels of randomness or competition between players, highly-developed themes, and varying complexity. This great diversity makes board gaming a perfect domain for employing a recommender system.

Given that, it should not come as a surprise that there have already been a few attempts. First of them is a basic collaborative filtering approach [1]. Another explored using clustering of games' data and calculating distances between the clusters and users [34], but unfortunately used only a very small dataset. Lastly, the most advanced approach [29] uses the recommender implemented by Turi Create [2].

1.3 Our approach

We believe that features of recommended games, such as mechanics or complexity, have significant influence on users' satisfaction with them. Commonly used functions evaluating quality of recommender system do not take that into consideration and have other drawbacks, that we discuss in Chapter 2. There, we also introduce the algorithms that we are going to use as a base.

Data we are using in the rest of our work comes from BoardGameGeek, which is the most popular website dedicated to board games. It is described in Chapter 3.

The problems with model evaluation is the reason why we decided to focus on devising a better method for the task. The biggest challenge we encountered was coming up with a proper metric for evaluating the match between games' features and users' profiles. We had to devise ways to reasonably discretize continuous data without losing its semantics, as well as ensure that the values in the profiles were independent of the number of games in users' collections or number of features each game has. We describe these challenges and the solutions we came up with in Chapter 4. This chapter also introduces the metric itself and the modifications in the algorithm that result in its optimization.

In Chapter 5 we present and analyze detailed results of our custom model in comparison with other approaches. We wanted to check whether our assumptions about users' preferences are justified and if our modifications successfully serve their purpose.

Lastly we summarize our findings, discuss their significance and provide possible directions of further development in Chapter 5.

Chapter 2

Recommender Systems and Methods of Their Evaluation

Recommender systems use the set of users U and a set of items I, as well as a matrix R in order to produce recommendations. R is a matrix where each row represents a user from U, and each column represents an item from I. Values of R represent interactions between users and items. As such, R is often very sparse, because most users only interact with a very small subset of items.

One of the approaches to creating recommender systems is collaborative filtering [21], based on an assumption that if we find two similar users, an item liked by one of them is more probable to be liked by the other. There are different approaches to collaborative filtering, based on the way the system interacts with R. Memory-based ones operate directly on R matrix to compute similarity between users and items [27]. Model-based systems often try to compress user-item matrix into matrices U, I of latent representations of users and items, usually employing some method of matrix factorization [15]. The latent embeddings when multiplied should approximate ratings present in matrix R but also provide predictions of missing ratings. The extension of this approach, known as Factorization Machines [24], allows taking into consideration additional features of users and items.

Recommendation algorithms can also be divided based on the type of data they work with. Explicit ones, where R contains ratings given to items by users, try to predict the value of rating that each user will assign to each item in the dataset. An example of such method is singular value decomposition (SVD) from family of collaborative filtering approaches. This is a very basic approach, so we will not describe it: for explanation of this method see [32]. Another type of models is one working with implicit data, where R matrix only contains binary indicators whether a user interacted with an item. Instead of trying to predict exact rating, they use data about interactions between users and items in order to model each user's preferences. Implicit models often assume that lack of interaction indicates that a user does not even want to try a game or watch a movie, because they suspect

that they will not enjoy it [31]. One clear advantage of this type of data is that it can often be easier to collect in real-life application, as we can simply track user's behaviour.

One example of a model working with implicit feedback data is LightFM [17]. In this chapter we will describe it in greater detail, because we later propose it's modified version tailored to our specific task. Afterwards, we will present common methods of evaluating recommender systems and discuss their drawbacks that led to our proposing of a custom metric in Chapter 4.

2.1 LightFM

LightFM [18] is a hybrid matrix factorization model created by Maciej Kula. It represents users and items as linear combinations of their features and combines collaborative and content-based approaches to produce recommendations relevant both in cold-start and warm-start scenarios.

2.1.1 Model overview

LightFM has been designed with two major considerations in mind:

First of all, the model must be able to learn user and item representations from interactions. Consequently, items consistently liked by the same users should have similar representations. This allows for transfer learning to occur, since now a user can be recommended items that he has not yet interacted with, but that have similar representations to the ones that he likes.

Secondly, the model should be able to generalise to new users and items, which is achieved by representing users and items as linear combinations of their content features. This means that as long as we have some initial information about an item, such as category or designer in case of games, we can calculate its representation.

2.1.2 Latent representations

LightFM's users and items latent representations are calculated by summing embeddings of their features. Each feature is also described by a scalar bias term. As such, users and items are also described by their biases, constructed analogically to their latent representations.

To allow items with same features to have different latent representations each user and item has its own unique indicator variable as a feature. This means that if no user or item features are provided, LightFM reduces to Matrix Factorization[15] algorithm.

2.1.3 Scoring

Let q_u and b_u be the latent representation and bias term of user u, p_i the latent representation of item i, and b_i it's bias term. Given that, the model's predicted score for user u and item i is calculated as follows:

$$\hat{r_{ui}} = f(q_u \cdot p_i + b_u + b_i) \tag{2.1}$$

where $f(\cdot)$ is the sigmoid function.

2.1.4 Training process

The goal of LightFM model's training is to learn embeddings and biases that produce the best predictions according to a given metric.

The LightFM implementation supports training using numerous fitting functions. We will only describe WARP (Weighted Approximate-Rank Pairwise) function here, more details can be found in LightFM model's documentation[16].

WARP's goal is to maximise the rank of positive interactions, which it tries to achieve by sampling negative interactions until it finds one that violates the ranking. Once such pair of a positive and a negative interactions is found, the embeddings are updated so that the rank of positive interaction is higher than the negative one. It directly optimises precision@k metric, which will be described in Section 2.2.

```
def fit_warp(user):
    for pos_it in positive_interactions(user):
        sampled = 0
        while sampled < max_sampled:
            neg_it = get_random_negative_interaction(user)
            if rank(neg_it) > rank(pos_it):
                update_weights()
                break
```

Where:

- *positive_interactions(user)* returns an iterator over all items that user interacted with.
- *get_random_negative_interaction(user)* returns a random item that user did not interact with.
- *update_weights()* performs gradient descent in order to update embeddings so that positive interaction ranks better than negative one.

2.2 Evaluating recommender systems

Many metrics have been constructed for the task of evaluating the performance of recommender systems. Despite this, it is still an ongoing research effort to invent better ways to measure their performance. That is mostly because the task is quite hard, if not downright impossible for most of the researchers.

One approach, favored by industry practitioners, is online experiments on working systems. For example, websites with huge traffic can evaluate recommendations of their models through A/B testing. The model which persuaded the biggest number of users to act on the recommendations (watch the movie, click the ad or buy the product) is obviously the best for the profit. Tradeoff for this certainty of correct evaluation is that it requires a lot of data from real people, which in most cases is impossible to gather in academic setting.

Other option is offline model evaluation on past data, where most of the commonly-used metrics are reliant on hiding part of user's interactions and then comparing them with the recommendations. While this approach is much easier to conduct, it has some drawbacks which often cause it to fail to reflect how well the model will do in a real-world application [22].

First, it is not obvious how the data should be split into training in testing set. To better reflect real life scenario of recommender system application, we could use data up to certain date to train the model and evaluate it on information gathered from that point onward. Users' preferences change in time and a quality recommender system should take that into consideration. However, using this approach, we can encounter situations where certain user or item is not included in the training set, therefore model is unable to produce any informed predictions regarding them. Such items should be removed from the test dataset. Another option is to choose a subset of users for which we will hide a certain fraction of their interactions. In this case, the problem of unknown items appearing in the testing set is still possible, albeit it should be much rarer. Additionally, the aspect of temporality of recommendations is lost here. We decided upon the second approach and ignore its drawbacks in our work. Modeling temporal evolution of users' preferences is a major task in itself and is usually addressed by using Recurrent Neural Networks [9].

Secondly, while there exists a wide range of metrics created in order to assess quality of the system in offline setting [25], none of them is perfect. One group of them focuses on measuring the accuracy of ratings prediction, for example using Absolute Error, Mean Squared Error (MSE) or Root Mean Squared Error (RMSE). Another evaluates quality of the sets of recommendations by calculating relations between number of true and false positives and negatives. Precision and recall are examples of this type of metrics. Finally, statistics such as diversity of recommended items or coverage of item space can be taken into consideration. We will discuss the problems with such metrics using the example of RMSE (Root Mean Square Error)

and precision@k. They motivated us to introduce our own in Chapter 4.

2.2.1 RMSE

One of offline methods used to evaluate models working with explicit data is calculating root mean squared error between predicted and real values of ratings included in test dataset: RMSE. It is calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T}(P_{(u,i)} - R_{(u,i)})^2}{|T|}} \tag{2.2}$$

where:

- T - test set consisting of observed ratings
- $R_{(u,i)}$ - observed rating of item i by user u
- $P_{(u,i)}$ - predicted rating of item i by user u

This metric works with explicit data and, as a consequence, assumes that user rated all the items known to them and all the model should focus on is predicting these scores correctly. That hypothesis rarely holds, because users tend to rate items that they enjoyed or hated but often forget to rate items that they did not have strong feelings for. Secondly, practical goal of recommending systems is to produce a small set of sound recommendations that can be presented to a user. RMSE pays as much attention to accuracy of both high and low ratings, while in reality the importance of accurately predicting high ratings is greater as it directly influences the produced recommendations.

2.2.2 Precision@k

Popular metric used to offline evaluation of systems working with implicit data is mean precision@k. It represents the fraction of top k recommended items that are among user's test interactions, averaged over all the users.

$$\textit{precision@k}(u) = \frac{|\textit{top-k-recommendations}(u) \cap T_u|}{k} \tag{2.3}$$

$$\textit{mean-precision@k} = \frac{\sum_{u\in U} \textit{precision@k}(u)}{|U|} \tag{2.4}$$

where:

- u - certain user
- k - parameter of this metric, specifying how many of top recommendations should be taken into consideration

- *top-k-recommendations*(u) - set of k items ranked the highest by the model for user u
- T_u - set of items that the user u has interacted with in test dataset
- U - set of users in test dataset

Here, the assumption is that items that the users interacted with in test dataset are the ones the system should recommend to them. However, it is not obvious how a user would react to different recommendations. We believe that we should not expect that just because the user did not interact with a certain item they would not do so being given the chance. It is likely that they are simply not aware of its existence. In addition to that, optimizing precision often leads to popularity bias - frequent suggestions of the most popular items and neglecting the more niche ones. It could be argued that while the popular items make up majority of test interactions, sometimes they do not need to be recommended, because user will end up interacting with them either way.

Chapter 3

Description of Available Data

Before we are able to explain our approach, description of data that was available to us is required.

The source of our data is Board Game Geek (BGG) [7], which is the largest and most popular website dedicated to board and card games. It features content such as reviews or ratings as well as live discussion forums. Dataset we are using is a historical collection of games, users and their interactions scraped using BGG's API by Markus Shepherd [28]. While the dataset is being updated daily, we settled for using snapshot taken as of 8th of October 2021 to avoid compute-intensive data processing.

3.1 Games

Games dataset includes information about each game accessible on the website. There are 104268 games in the dataset, and the Table 3.2 describes some out of their 46 attributes that we believe are the most important to users.

3.2 Users

Users dataset includes information about each registered user. There are 396800 users in the dataset, and a few of their 12 possible attributes are described in Table 3.3. We believe that the ones included here may serve as additional context useful to improve recommendations' quality.

Table 3.1: Table of selected attributes available in games dataset, their descriptions and examples of their possible values. We believe that characteristic that we show here are the most important to users.

Name	Description	Examples of possible values
name	the name of a game	”Monopoly”, ”7 Wonders”, ”Spirit Island”
bgg_id	unique identifier of the game	integer values
category	used to describe game characteristics such as theme, activity players will be participating in, components used [4] etc.	Card Game, Wargame, Party Game, Dice, Fantasy, Fighting
mechanic	used to describe gameplay mechanisms that can be encountered during each play [6]	Dice Rolling, Hand Management, Tile Placement, Cooperative Game
complexity	difficulty/heaviness of the game calculated as average of results of community poll	floating points between 1.0 and 5.0
designer	name of designer(s) of the game	Lloyd Krassner, Reiner Knizia, Uncredited
min/max_players_rec	recommended minimum/maximum number of players as voted by the community	integer values, usually between 1–10
min/max_players_best	best minimum/maximum number of players as voted by the community	integer values, usually between 1–10

3.3 Interactions

This dataset is a collection of interactions between users and games. There are more than 41 mln rows in the dataset. Around 15 attributes are available, but most of them are very sparse, so in the Table 3.3 we describe only the most common ones.

Table 3.2: Table of some attributes available in users dataset, their descriptions and examples of their possible values. We believe that they may serve as additional context useful to improve recommendations' quality.

name	description	examples of possible values
bgg_user_name	username	beastvol, fu_koios, mycroft
country	country that the user provided as his origin	United States, Poland, Spain
region	region of the country that the user provided as his origin	mostly states of the United States
registered	date of registration of that user	date
last_login	date of last login of that user	date

Table 3.3: Table of the most common attributes available in interactions dataset, their descriptions and examples of their possible values.

name	description
bgg_user_name	username of the player
bgg_id	identifier of the rated game
bgg_user_rating	integer grade between 1 and 10 given by the user to the game
bgg_user_owned	whether the user owns the game
bgg_user_want_to_play	whether the user want to play the game
updated_at	date of the last update of interaction

Almost 18 mln (43%) of rows include ratings. In more than 22 mln (54%) rows, users marked that they own the game. Only 2 mln (5%) times users mentioned that they would like to play the game. While there are a lot of different games and users in the dataset, many of them are not very popular or active. Only 2762 (2%) games have been rated more than 1000 times and 205710 (52%) users rated more than 10 games. Number of ratings added each year is rising. Interesting insight is noticeable increase of number of new ratings each winter season. Figures 3.1 and 3.2 depict ratings count and the number of updated (added or modified) interactions through time.

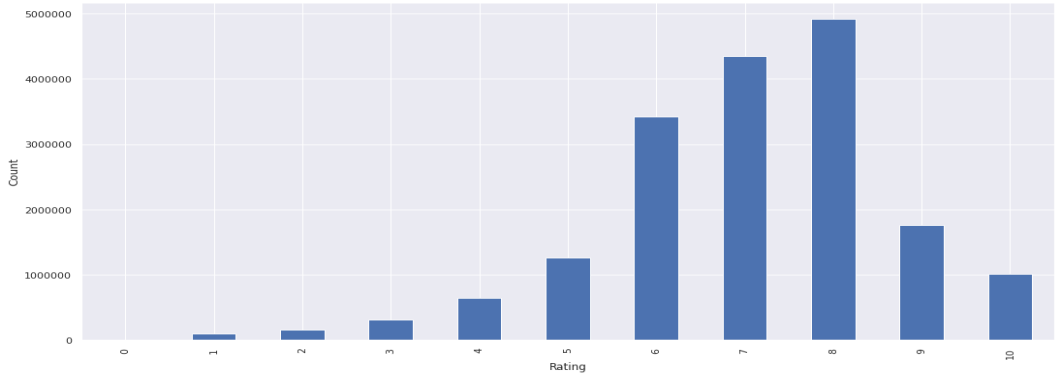


Figure 3.1: Plot of ratings count grouped by their values. The most popular rating is 8.0, with 7.0 and 6.0 following up. Values higher or lower than that are much less common.

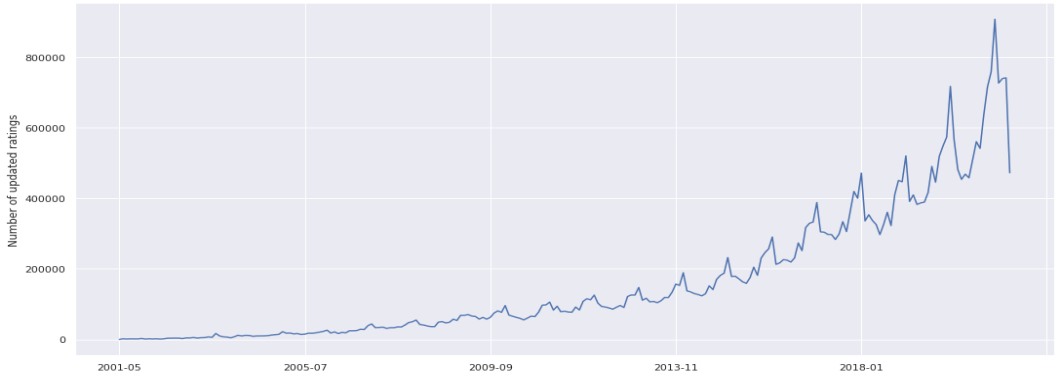


Figure 3.2: Plot of count of added or updated rating each month. It can be observed that each year this number is rising due to increase in BGG's popularity. Periodical peaks each winter are probably caused by holiday season.

Chapter 4

Our Approach to Board Game Recommendations

Because of the problems with common methods of model evaluation described in Chapter 2, we propose a domain-specific metric for evaluating board games recommendations. Our method is purely content-based, making use of the games' categories, mechanics, the best player count and complexity. We believe that in case of board game recommendations, focusing on these features will yield more diverse and accurate recommendations than these produced by using RMSE or precision.

First, we prepare users' profiles representing their preferences. We also transform the games' features into normalised representation similar to users' profiles. When calculating the metric, we check how well the set of recommended games' features matches the profile of the user for whom the recommendation was made.

Afterwards, we propose a modification to the LightFM model that allows it to optimize our metric.

4.1 Motivation behind features chosen for our metric

In order to create our metric, we first had to decide which of the game features have the biggest impact on user's preferences. In this section, we describe the reasoning that led to our choices.

4.1.1 Categories

These features describe themes present in the game. We believe that choosing the game with the right categories might help pique user's attention. Certain users might immediately become interested if presented with game about mythology or animals, even if the game itself is mediocre at best. On the flip side, some users might be so biased against e.g. fantasy or sports, that they will not pick up a game

featuring these themes no matter how much we think that they might enjoy the gameplay.

4.1.2 Mechanics

Mechanics features correlate directly with the gameplay. They represent common ways that users can interact with the game and convey challenges that players must deal with. Some players might enjoy games that require in-depth understanding of their mechanics and present complex logical puzzles, while others could prefer ones that are lighter, but incorporate elements of social interaction (such as bluffing or making alliances).

4.1.3 Player count

We also decided to incorporate data about recommended player counts into our metric. The reasoning behind this is that most players have a certain friend group that they play with. While most games are best played in three or four people, there are certain party games that can be played with over eight participants. Moreover, there is a significant number of players who enjoy playing boardgames by themselves[23].

4.1.4 Complexity

The last component of our metric is complexity of a board game. This describes how mechanically heavy a given game is. We believe that this also constitutes a non-negligible piece of user's preferences, as e.g. young players might get easily discouraged when presented with a game with complex rules.

4.2 Preparing features vector

After choosing the right features, we need to process them so that we can use them to calculate a score. In this section, we describe the transformations we perform on the features.

Let:

- $U = \{u_1, u_2, \ldots, u_m\}$ be the set of users
- $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items
- $T_u = \{i_{t_1}, i_{t_2}, \ldots, i_{t_k}\} \subseteq I$ be the set of items that user u interacted with
- $R_u = \{i_{r_1}, i_{r_2}, \ldots, i_{r_l}\} \subseteq I$ be the set of items recommended to user u

4.2.1 Categories

Let:

- $C = \{c_1, c_2, \ldots, c_n\}$ be the set of categories
- c_i, $c \in C \wedge i \in I$ be equal to 1 if item i belongs to category c and 0 otherwise

We construct the vector $\boldsymbol{ci}_i$ of item i's categories as follows:

$$\boldsymbol{ci}_i = \Big[c_i | c \in C\Big] \tag{4.1}$$

Vector $\boldsymbol{cu}_u$ of user u's category preferences is a sum of category vectors of items he interacted with. Then, the vector is normalised by the sum of values, so that the maximum score of categories' component of our metric in Equation 4.21 is equal to w_c.

$$\boldsymbol{cu}_u = \sum_{i_t \in T_u} \boldsymbol{ci}_{i_t} \tag{4.2}$$

$$\hat{\boldsymbol{cu}}_u = \frac{\boldsymbol{cu}_u}{\sum \boldsymbol{cu}_u} \tag{4.3}$$

Vector $\boldsymbol{cr}_u$ of category scores for recommended games is calculated similarly, except for being normalised by the number of recommendations.

$$\boldsymbol{cr}_u = \sum_{i_r \in R_u} \boldsymbol{ci}_{i_r} \tag{4.4}$$

$$\hat{\boldsymbol{cr}}_u = \frac{\boldsymbol{cr}_u}{|R_u|} \tag{4.5}$$

We also considered making $\boldsymbol{cr}_u$ vector binary, with values being equal to one if there exists at least one game in the recommendations set with this feature. We decided to use sum instead, because doing so rewards models for recommending multiple games with desirable features.

The exact same reasoning led to us normalising by the number of recommendations, as opposed to normalising by maximum value. Doing so would award the same score to a set of recommendations containing exactly one game with each feature the user likes as to a set of recommendations where each game contains every feature liked by the user.

Another approach we thought about was normalising by the sum of vector, however we decided against it, as we want to reward games featuring bigger set of categories.

4.2.2 Mechanics

The construction of mechanic vectors is strictly analogous to construction of category vectors. As the reasoning behind final form of vectors is the same, we will not repeat it here.

Let:

- $M = \{m_1, m_2, \ldots, m_n\}$ be the set of mechanics
- m_i, $m \in M \wedge i \in I$ be equal to 1 if item i incorporates mechanic m and 0 otherwise

We construct the vector $\boldsymbol{mi}_i$ of item i's mechanics as follows:

$$\boldsymbol{mi}_i = \Big[m_i | m \in M\Big] \tag{4.6}$$

Vector $\boldsymbol{mu}_u$, representing mechanic preferences of user u, is a sum of mechanic vectors of items he interacted with. It is normalised by the sum of values.

$$\boldsymbol{mu}_u = \sum_{i_t \in T_u} \boldsymbol{mi}_{i_t} \tag{4.7}$$

$$\hat{\boldsymbol{mu}}_u = \frac{\boldsymbol{mu}_u}{\sum \boldsymbol{mu}_u} \tag{4.8}$$

Vector $\boldsymbol{mr}_u$ of mechanic scores for recommended games is calculated similarly, except for being normalised by the number of recommendations.

$$\boldsymbol{mr}_u = \sum_{i_r \in R_u} \boldsymbol{mi}_{i_r} \tag{4.9}$$

$$\hat{\boldsymbol{mr}}_u = \frac{\boldsymbol{mr}_u}{|R_u|} \tag{4.10}$$

4.2.3 Player counts

Let:

- $P = \{p^1, p^2, \ldots, p^{n-1}, p^{n+}\}$ be the set of available player counts, where p^k corresponds to a k player game and p^{n+} corresponds to a game for at least n players
- p_i^k, $p^k \in P \wedge i \in I$ be equal to 1 if game i is best played with k players and 0 otherwise

In our data, we have information about minimum and maximum recommended number of players. A game can have minimum recommended number of players equal to i and maximum equal to j, and then $p^i, p^{i+1}, \ldots, p^j$ will be equal to 1, while the rest of the variables will be equal to 0.

We construct the vector $\boldsymbol{pi}_i$ of item i's possible player counts as follows:

$$\boldsymbol{pi}_i = \left[p_i^k | p^k \in P\right] \tag{4.11}$$

Vector $\boldsymbol{pu}_u$, representing player counts preferences of user u, is a sum of player counts vectors of items he interacted with. It is normalised by the sum of values.

$$\boldsymbol{pu}_u = \sum_{i_t \in T_u} \boldsymbol{pi}_{i_t} \tag{4.12}$$

$$\hat{\boldsymbol{pu}}_u = \frac{\boldsymbol{pu}_u}{\sum \boldsymbol{pu}_u} \tag{4.13}$$

Vector $\boldsymbol{pr}_u$ of player count values for recommended games is calculated similarly, except for being normalised by the number of recommendations.

$$\boldsymbol{pr}_u = \sum_{i_r \in R_u} \boldsymbol{pi}_{i_r} \tag{4.14}$$

$$\hat{\boldsymbol{pr}}_u = \frac{\boldsymbol{pr}_u}{|R_u|} \tag{4.15}$$

4.2.4 Complexities

In order to use complexities in our metric, we needed to devise a way to transform continuous complexities to discrete representation.

Let:

- $X = \{x^1, x^2, \ldots, x^n\}$ be the set of discrete, evenly spaced game complexities, where x^1 corresponds to complexity 0 and x^n - complexity 5
- $cpx_i \in [0, 5]$ be the complexity of game i

The vector $\boldsymbol{xi}_i$ of item i's complexity is a one-hot encoding of its discretized value.

$$\boldsymbol{xi}_i = \textit{one-hot}(\textit{round-to-nearest}(cpx_i, X)) \tag{4.16}$$

We believe that users who prefer games of complexity i will also enjoy playing games of complexity $i \pm \epsilon$. For that reason, if a game has a complexity i, we not only assign the complexity preference score to the discretized value of i, but also values in its vicinity (with scores linearly depending on the distance). Formula for calculating vector $\boldsymbol{xu}_u$, representing complexity preferences of user u, can be found

in Equation 4.17. We normalise this vector by its maximum value, as we want the highest achievable score of complexities' component of our metric in Equation 4.21 to be equal to w_x. Contrary to previous cases, normalising by sum would not yield the same effect here, due to our utilization of one-hot encoding in items' complexity vectors.

$$\boldsymbol{xu}_u = \sum_{i_t \in T_u} \Big[max(0, cpx_i - k) | x^k \in X \Big] \tag{4.17}$$

$$\hat{\boldsymbol{xu}}_u = \frac{\boldsymbol{xu}_u}{max\ \boldsymbol{xu}_u} \tag{4.18}$$

Vector $\boldsymbol{xr}_u$ of complexity values for recommended games is, once again, simply sum of item vectors. It is normalised by the number of recommendations.

$$\boldsymbol{xr}_u = \sum_{i_r \in R_u} \boldsymbol{xi}_{i_r} \tag{4.19}$$

$$\hat{\boldsymbol{xr}}_u = \frac{\boldsymbol{xr}_u}{|R_u|} \tag{4.20}$$

4.3 Calculating the metric

Once the features vectors are prepared, we can calculate the score of each metric component as the dot product of its recommended games' feature vector and user's feature preference vector.

For example, the mechanic score of a set of recommendations for user u is the dot product of $\hat{\boldsymbol{mr}}_u$ (mechanic scores of games recommended to user u) and $\hat{\boldsymbol{mu}}_u$ (user u's mechanic preferences).

The final score of our metric is a weighted sum of the feature category scores. As such, it can be calculated in the following manner:

$$score(u) = w_c(\hat{\boldsymbol{cu}}_u \cdot \hat{\boldsymbol{cr}}_u) + w_m(\hat{\boldsymbol{mu}}_u \cdot \hat{\boldsymbol{mr}}_u) + w_p(\hat{\boldsymbol{pu}}_u \cdot \hat{\boldsymbol{pr}}_u) + w_x(\hat{\boldsymbol{xu}}_u \cdot \hat{\boldsymbol{xr}}_u) \tag{4.21}$$

w_c, w_m, w_p, w_x in the above equation are custom weights for respectively categories, mechanics, player counts and complexities.

4.3.1 Motivation behind using dot product

We decided to use the dot product in Equation 4.21 for two reasons.

First, as opposed to vector similarity, it does not penalize the recommendations for fully fitting to user profiles. For example, if user's fantasy preference score is 0.75, then by using similarity we would expect 3 out of every 4 games in our recommendation set to incorporate fantasy themes. However, we believe that there

is no reason to penalize a model if every one of the recommended games features this theme, as long as it does not compromise other features' scores.

Secondly, this operation is great from practical point of view, as it generalises well into one-item scenario - a property we make use of in our modified version of LightFM model. On top of that, dot product is very frequently used, which led to a great optimization of this operation in modern computer architecture.

4.4 Profiles

First, let us define profile as a concatenation of feature vectors

- $\boldsymbol{vu}_u = concat(w_c\ \hat{\boldsymbol{cu}}_u, w_m\ \hat{\boldsymbol{mu}}_u, w_p\ \hat{\boldsymbol{pu}}_u, w_x\ \hat{\boldsymbol{xu}}_u)$ - user profile
- $\boldsymbol{vr}_u = concat(\hat{\boldsymbol{cr}}_u, \hat{\boldsymbol{mr}}_u, \hat{\boldsymbol{pr}}_u, \hat{\boldsymbol{xr}}_u)$ - recommendations profile
- $\boldsymbol{vi}_i = concat(\hat{\boldsymbol{cr}}_i, \hat{\boldsymbol{mr}}_i, \hat{\boldsymbol{pr}}_i, \hat{\boldsymbol{xr}}_i)$ - item profile

Notice how we incorporated the weights into user profiles. It is done to simplify later computations, so that we do not have to keep track of the length of different feature type vector after concatenation.

Let us observe that

$$\begin{aligned} score(u) = & \\ & = w_c(\hat{\boldsymbol{cu}}_u \cdot \hat{\boldsymbol{cr}}_u) + w_m(\hat{\boldsymbol{mu}}_u \cdot \hat{\boldsymbol{mr}}_u) + w_p(\hat{\boldsymbol{pu}}_u \cdot \hat{\boldsymbol{pr}}_u) + w_x(\hat{\boldsymbol{xu}}_u \cdot \hat{\boldsymbol{xr}}_u) \\ & = concat(w_c\ \hat{\boldsymbol{cu}}_u, w_m\ \hat{\boldsymbol{mu}}_u, w_p\ \hat{\boldsymbol{pu}}_u, w_x\ \hat{\boldsymbol{xu}}_u) \cdot concat(\hat{\boldsymbol{cr}}_u, \hat{\boldsymbol{mr}}_u, \hat{\boldsymbol{pr}}_u, \hat{\boldsymbol{xr}}_u) \\ & = \boldsymbol{vu}_u \cdot \boldsymbol{vr}_r \end{aligned} \tag{4.22}$$

In practice, when it comes to computing metric we use the whole profiles for convenience. Similarly, we will use the profiles later in this paper where there is no need for distinction between different types of features.

4.5 Testing profiles

In order to more truthfully evaluate trained models, we split the interactions into two sets: T_{train} - training interactions, and T_{test} - testing interactions.

Then, we construct training user profiles from T_{train}, and testing user profiles from $T_{train} \bigcup T_{test}$. Later, we will evaluate recommendations on both training and testing profiles.

4.6 Our modifications to the LightFM model

In order to adjust the LightFM model to optimize our metric, we made the following modifications. First, we passed the user and item profiles to the model, in order to be able to calculate our metric score during training. Then, we made the fitting function compare games based on how well they match the user profile.

4.6.1 Matrices of users and games' profiles

Let us take $\boldsymbol{vu}_u$ - user profiles, and $\boldsymbol{vi}_i$ - item profiles. Let us construct matrices:

- $\boldsymbol{VU} = \left[\boldsymbol{vu}_u | u \in U\right]$ - matrix of user profiles, where each row is a user profile vector
- $\boldsymbol{VI} = \left[\boldsymbol{vi}_i | i \in I\right]$ - matrix of item profiles, where each row is an item profile vector

We pass these matrices as arguments to function fitting LightFM model, which allows it to access our user and item representations.

4.6.2 Custom fitting function

We modified *fit_warp* function so that it optimizes our metric. In order to do so, we redefined positive and negative item pairs so that positive item is the one that achieves higher score using our metric.

```
def score(u, i):
    return vu_u · vi_i

def fit_custom(user):
    sampled = 0
    for _ in positive_interactions(user):
        while sampled < max_sampled:
            pos_it = get_random_item()
            neg_it = get_random_item()
            if score(user, neg_it) > score(user, pos_it):
                swap(pos_it, neg_it)

            if rank(neg_it) > rank(pos_it):
                update_weights()
                break
```

Where:

- *positive_interactions(user)* returns an iterator over all items that user interacted with.
- *update_weights()* performs gradient descent in order to update embeddings so that positive interaction ranks better than negative one.

Regular *fit_warp* function tried to find such pairs where a positive interaction ranked worse than a negative one and update the embeddings so that this is no longer the case. After the change the function encourages the model to rank the items that better match the user's profile higher, which directly optimizes our metric.

Chapter 5

Results

This chapter presents the results of our work. Details about processing we conducted to prepare data for training and evaluating the models are described in Appendix A.

First, we show that out method of creating users' profiles and calculating scores of recommendations is able to successfully model users' preferences. Then, we prove that our adjustment in WARP function leads to the desired increase in custom metric scores of LightFM model's recommendations. We compare results achieved by SVD and LightFM model using either vanilla WARP or our modified version. Lastly, we analyze models' recommendations for various users.

5.1 Validity of users' profiles

Based on our own experiences with board gaming, we believe that using categories, mechanics, recommended players' count and complexities of games liked by certain a user is a sensible method to represent their preferences. Nevertheless, we conducted a sanity check to prove that it is indeed the case.

First, we chose random 5000 users from the ones who had at least 10 interactions in train dataset and 5 in test dataset. We did not include users with fewer interactions, because their profiles would be more vulnerable to noise. For each of them, we calculated values of our metric based on their train profiles for:

- their test games
- 5 random games

The mean result for test games was 1.48, while for random games it was much worse: 0.31.

Additionally, we created 5 strongly defined artificial users. Each of them has 3–5 train games and 3 test games that we believe represent a preference for certain type of items. For them, we calculated values of our metric based on profiles created from their train games for their test and 3 random games. Descriptions of the users and scores of metric can be found in the Table 5.1.

Table 5.1: Table of artificial users with strongly-defined preferences that we used to "sanity check" our approach to modeling users' tastes. It describes what type of games each of them like, proposes some examples of such games and provides scores of our custom metric for selected set of items that we believe suit the user and a set of random items. Higher scores of test than of random games suggest that during the process of creation of the profiles, we are able to correctly encapsulate some aspects of players' likings.

Name	Description	Examples of games	Random games' score	Test games' score
USER_1	Simple family games	Catan, Monopoly, UNO	0.42	1.71
USER_2	Complex games	Game of Thrones, War of The Ring, Twilight Imperium	0.90	1.92
USER_3	Worker placement [8]	Agricola, Viticulture, Feast for Odin	0.52	1.61
USER_4	Ameritrash [3]	Runewars, Arkham Horror, Talisman	0.06	1.69
USER_5	Eurogames [5]	El Grande, Puerto Rico, Carcassone	0.64	1.25

The results were much better for the test games. Almost always, every component we take into consideration (mechanics, categories, recommended number of players, complexity) scored better than for random games. Only noticeable exception was USER_5, for which both test and random games received 0.0 score for categories and very low score for mechanics. This may result from the fact that the user preferred Eurogames, which are designed to be a challenge for the players. They often use wide variety of innovative mechanics and diverse themes, therefore it can be difficult to find games with closely matching characteristics. Despite the latter, test games still received better overall score, thanks to similar recommended number of players and complexity. This shows that our metric can be robust to situations where a user does not have a strong preference concerning one or few of the features we take into consideration.

5.2 Training

We trained three models: SVD (using implementation from Surprise [14] scikit), LightFM model using WARP and LightFM model using our modified version of WARP (which we will be referring to as "custom").

To train SVD model, we used train explicit dataset. LightFM models were trained using train implicit dataset. Custom model was additionally using train users profiles (i.e. created based on train implicit dataset). To evaluate our metric's scores, we used test users profiles (i.e. created based on combined train and test implicit dataset for users present in test dataset). The ratio between the number of games in train and test datasets for users included in the test set was around 7:3. Nonetheless, the train and test profiles ended up very similar: average cosine distance between them is only equal to 0.011. Therefore, values of metric's scores calculated using train and test profiles are highly correlated.

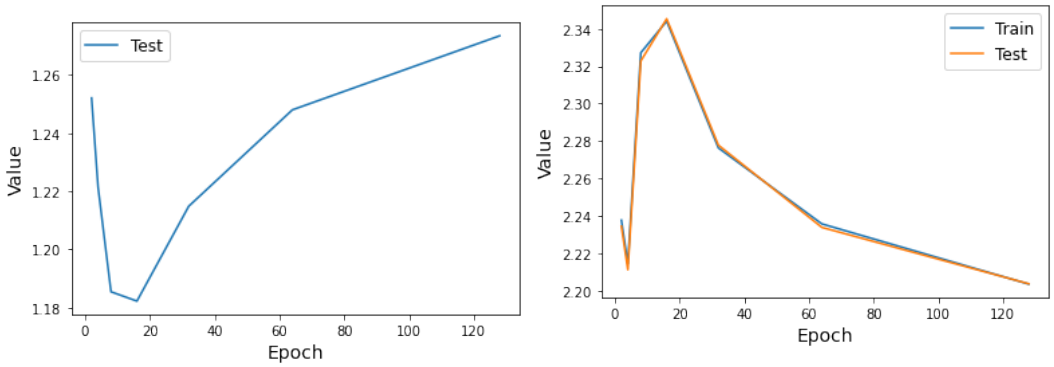
Additional plots, visualizing training progress of the models and broken down by custom metrics' components, can be found in the appendix.

5.2.1 SVD training

During the training of SVD with n_factors hyperparameter set to 70 and the rest left as default, we calculated its RMSE (Root Mean Square Error) on test explicit dataset and the scores of our metric based on 2000 random train and test users' profiles after [2, 4, 8, 16, 32, 64, 128] epochs. The results are visualized on Figure 5.1. It can be observed that until the 16^{th} epoch, both values of RMSE and metrics' scores were continuously improving. After reaching that point, the model started to overfit, which led to the increase of RMSE and the decrease in metrics' scores. This may suggest that the values of these two metrics are related - improving one of them should lead to better scores of another. In spite of the correlation, the best achieved results were not satisfactory, as at 16^{th} epoch RMSE was equal to 1.182 and score of our metric reached 2.345 for test profiles.

5.2.2 LightFM fitted using WARP

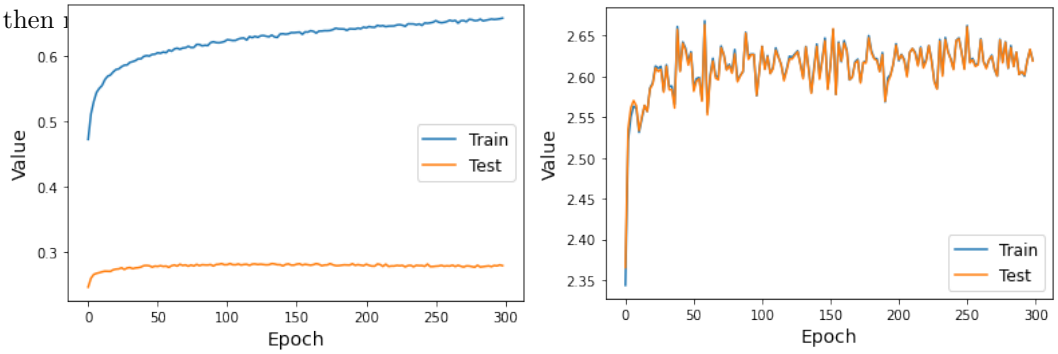
LightFM model was trained with the following hyperparameters: no_components: 70, learning_schedule: adadelta, loss: warp, item_alpha: 3e-10 and rest left as default. During the training for 300 epochs, after each 2, we calculated model's precision@5 on both training and testing data and scores of our metric based on 2000 random train and test users' profiles. It can be observed that scores of test precision were improving until around 50^{th} epoch and then stayed at a similar level. The best result achieved was equal to 0.281. Similarly, test custom metric values were rising visibly at the beginning, achieved the highest value of 2.65 after 64 epochs, and then



(a) Test RMSE values during training

(b) Test and train custom metric values during training

Figure 5.1: Plot of changes of values of RMSE and custom metric scores during training of the SVD model. It can be observed that the best scores were achieved at 16th epoch and then



(a) Test and train precision@5 values

(b) Test and train custom metric values

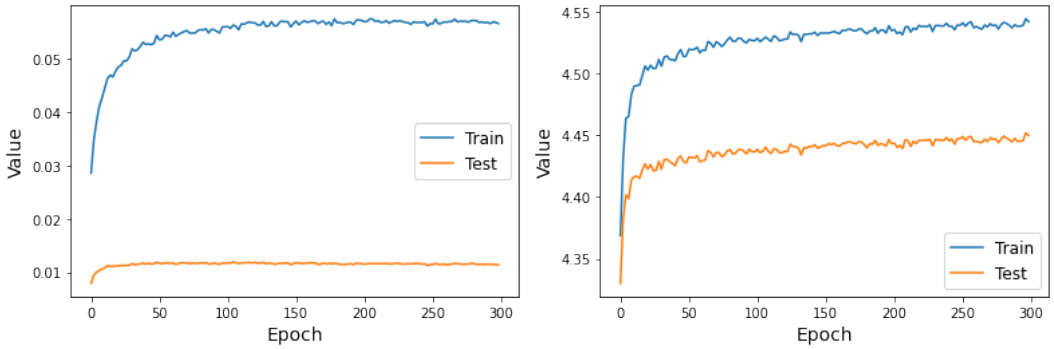
Figure 5.2: Plot of changes of values of precision@5 and custom metric scores during training of the WARP LightFM model. It can be observed that scores of test precision were improving until around 50 epoch and then stayed at similar level. Similarly, test custom metric values were rising visibly at the beginning, achieved the highest value of 2.65 after 64 epochs, and then stayed between range 2.55 – 2.65.

stayed between the range of 2.55 – 2.65. This score is better than for SVD, but still is not very impressive.

5.2.3 LightFM fitted using custom function

The process of training custom LightFM model was identical to the one described in 5.2.2, except for the fact that the ”loss” parameter has been changed to our modified version of WARP. Since first epochs, custom metric score was almost 2 times bigger than for previous models, which proves that our modifications in fitting function led to desired optimization of metric's scores. The increase in test score slowed down after 50^{th} epoch, but never stopped. The reason for the lack of overfitting is previously mentioned similarity between test and train users profiles. Best result achieved was 4.452 at 298^{th} epoch. Precision scores were rising at the beginning, achieved best test value of 0.0118 and then ranged between 0.0115–0.0117.

While this model satisfied our goal of high custom metric values, precision scores were much lower than for the model using WARP. The reason behind this is that the model did not learn to rank games included in training interactions higher than the rest when using our custom function. A game unseen by a user may be ranked much higher than his training games if there is a better profile match. As a result, precision values suffer because model tends to recommend less popular games.



(a) Test and train precision@5 values

(b) Test and train custom metric values

Figure 5.3: Plot of changes of values of precision@5 and custom metric scores during training of the custom LightFM model. Test custom metric values were rising steadily during first 50 epochs. Then the increase slowed down, but the model never started to overfit because of similarity of train and test users' profiles. Precision scores were rising at the beginning, achieved best test value of 0.0118 and then ranged between 0.0115–0.0117.

5.3 Analysis of trained models

In this section we analyze results, predictions and internal users' embeddings of fitted models. We use SVD trained for 16 epochs and LightFM model using WARP trained for 104 epochs, as they achieved the best values of metrics they intend to optimize (see training section). LightFM model using our custom fitting function did not overfit during training because of similarity of train and test users' profiles. We have decided to use version trained for 104 epochs here, to match the duration of the training of the LightFM model trained using WARP.

5.3.1 Custom metric's scores

First, we evaluated the models on full test users' profiles. The results are similar to the scores calculated during training on sample of test data and can be found in the table 5.3.1.

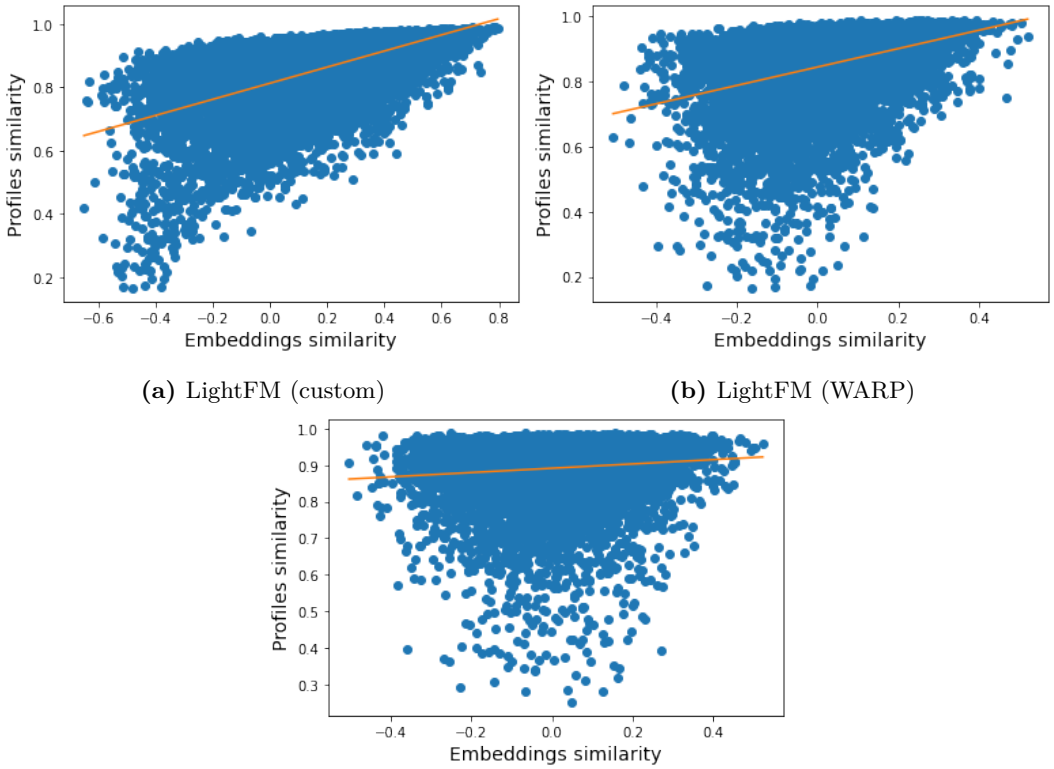
Table 5.2: Table of custom metric scores, broken down by their components, achieved by trained models. It can be observed that, as expected, LightFM trained using custom fitting function achieved the best results. Full scores of other models are similar, but some differences in scores of individual components can be noticed.

Model	Categories' score	Mechanics' score	Players' count score	Complexities' score	Full score
SVD	0.573	1.023	0.358	0.389	2.343
LightFM (WARP)	0.668	0.851	0.314	0.781	2.615
LightFM (custom)	1.510	1.464	0.701	0.765	4.440

As expected, custom LightFM model achieved best results with score of 4.44. It significantly exceeded SVD's and WARP LightFM scores in every component, except for WARP models' complexity score. It may suggest that taking the difficulty of games into consideration may be important when optimizing precision of the model. Visible improvement, despite the decrease in the value of this component during training (see B.1), was our model's players' count score. The lowest-hanging fruits and main portion of the increase in our model's full score, both due to their higher weight A.1 and information included in item features, were categories' and mechanics' components. We also examined custom LightFM model's scores for users grouped based on their activity levels. Results can be found in section B.2.

5.3.2 Users' embeddings

We also compared similarity of embeddings representing users' with similar profiles in trained models. From users with 10 or more training interactions, we randomly chose user u_0 and a group of 10000 unique users $u_1, \ldots, u_{10000}$. That group is the same for both LightFM models, but different for SVD, as it was trained using other dataset. Then, for each pair (u_0, u_i),$1 \leq i \leq 10000$, we calculated cosine similarity between these users' train profiles and their embeddings in the model. Similarities between embeddings sorted by corresponding similarities of users' profiles are shown on figures 5.4. Additionally, we used linear regression to better visualize the relationship between these variables. Surprisingly, in both LightFM models the achieved relation meet our goal: users with similar profiles have similar embeddings. In SVD, it seems that similarity of embeddings and profiles is not correlated.



(a) LightFM (custom)

(b) LightFM (WARP)

(c) SVD

Figure 5.4: Plots illustrating similarity between users' embeddings and profiles in trained models. We chose one user random user. Then, we calculated cosine similarity between their profile and embeddings from trained models to profiles and embeddings of 10000 other random users. Axes X correspond to similarity between the embeddings, axes Y - the profiles. It can be observed that in both LightFM models, embeddings and profiles are correlated. This relation is not noticeable in SVD.

5.3.3 Predictions' analysis

Table 5.3: Table of examples of recommendations produced by SVD for users with strong preferences.

Name	Description	Examples of recommended games
USER_1	Simple family games	Catan 3D, Pandemic Legacy, Cards Against Humanity, Brigde, Exploding Kittens: NSFW Deck
USER_2	Complex games	Pandemic Legacy, Twilight Imperium (4th Edition), Star Wars: Rebellion, Gloomhaven
USER_3	Worker placement	Eclipse, Anachrony, Kanban EV, Scythe
USER_4	Ameritrash [3]	Twilight Imperium (3rd Edition), Scythe, Gloomhaven, The 7th Continent
USER_5	Eurogames [5]	El Grande Big Box, Agricola, Anachrony

Table 5.4: Table of examples of recommendations produced by LightFM (WARP)for users with strong preferences.

Name	Description	Examples of recommended games
USER_1	Simple family games	Carcassone, 7 Wonders, Pandemic, Risk, Scrabble
USER_2	Complex games	Twilight Struggle, Carcassone, Catan, Terraforming Mars, Pandemic
USER_3	Worker placement	Terraforming Mars, Wingspan, 7 Wonders, Scythe, Carcassone
USER_4	Ameritrash [3]	Twilight Imperium (3rd Edition), Twilight Struggle, Betrayal at House on the Hill, Catan
USER_5	Eurogames [5]	7 Wonders, Pandemic, Ticket to Ride, Ticket to Ride: Europe, Agricola

Table 5.5: Table of examples of recommendations produced by LigtFM (custom) for users with strong preferences.

Name	Description	Examples of recommended games
USER_1	Simple family games	Variations of Monopoly, Byzanz, Attack!
USER_2	Complex games	Eclipse, Star Wars: Imperial Assault, BattleTech, Star Wars: Rebellion
USER_3	Worker placement	New Dawn, Agricola (Revised Edition), Letter Tycoon, Terraforming Mars
USER_4	Ameritrash [3]	Mordheim: City of the Damned, Krosmaster: Quest, Dungeon Saga: Dwarf King's Quest
USER_5	Eurogames [5]	504, The Gnomes of Zavandor, Caverna: The Cave Farmers, Archipelago

In the Tables 5.3.3, 5.3.3 and 5.3.3 we provide examples of games recommended by each model for artificial users with strong preferences. Recommendations of each model are somewhat sensible, although a few problems can be noticed.

SVD model suggested user interested in family games two items with explicit language (Cards Against Humanity, Exploding Kittens: NSFW Deck), which are definitely not suitable for younger players.

LightFM model using WARP tended to recommend very popular and highly rated items, which led to visibly lower diversity and weaker match to user preferences. For example, suggestions for USER_2, interested in complex games, included Carcassone, perceived by BGG users as light/medium light game.

As expected, predictions of LightFM model with custom fitting function tended to place games matching the user profile the highest. It did not take into consideration items' popularity, which sometimes led to suggestions of rather niche and not very highly rated games, such as 504. On the one hand, if we analyze the set of unique recommended games, this model provided the most diverse recommendations. On the other hand, each of the tested models tended to suggest reimplementations or new versions of games already played by users. In the custom LightFM model this behaviour is the most prominent: for USER_1, who had Monopoly in his training interactions, all of the top5 recommendations are different flavours of this game. It is difficult to judge whether this phenomenon is desired or not - some users may be interested in playing slightly modified releases of their favourite products, others might be looking for new experiences.

Chapter 6

Conclusions and Future Work

In our thesis we introduced new, purely content-based metric for evaluating board games recommendations. We tackled the challenge of making it independent of the number of games in users' collections or number of features each game has. We also found a method of transforming the data to be usable in our metric so that it does not lose its semantics. Then we analyzed the metrics behaviour, modified the LightFM model in order to improve its scores and examined produced predictions.

Recommendations of traditional models we considered tended to focus on popular or highly-ranked games, which did not always fit preferences that could be inferred from users' interactions. We strived to model these preferences by constructing users' profiles and a metric evaluating how much a set of games matches the profiles. For that reason, we took into consideration features of games we believe are most important from players' perspective, such as mechanics or complexity.

Then, to achieve desired improvement of the designed metric's scores, we altered WARP fitting function. Our modifications, while simple, successfully lead to prioritizing ranking high the games which match the best the profile of each user. While similar results may be achievable by simply sorting the games based on their scores, we believe that randomness introduced by the training helps to regularize the model and diversify the predictions. Also, by using modified WARP, it would be easier to incorporate optimizing another metric into fitting process, as described in 6.1.

Finally, we evaluated and compared the predictions of trained models. While most of the time they make some sense, each model has its own drawbacks which we explained in sections 5.3.1 and 5.3.3. We could not run many tests and experiments due to the nature of our work. While there are many metrics trying to evaluate recommender systems, online A/B testing remains the best way to assess quality of the models. Unfortunately, it requires access to resources unavailable to us, such as permissions to present recommendations produced by our system on BGG and analyze users' reactions to them.

6.1 Future work

As previously described, online testing is the best method to evaluate quality of recommendations. Were we able to conduct it, we would like to compare results of our model to the suggestions of traditional solutions optimizing precision or RMSE.

Additionally, it would be sensible to consider creating and evaluating hybrid recommender system, which would try to optimize both precision and our metric. To create it, we could modify WARP function to alternate between improving our metric and trying to rank positive interactions higher than negative. We hope that model fitted using that function may have advantages of both approaches and would recommend games which are popular and closely suited to users' profiles.

Another modification worth testing is filtering out reimplementations of games from recommendations. It would be fairly easy, as the information necessary to do that is already included in raw games' dataset. As a result, we would not encounter situation described in 5.3.3, where all the top5 games recommended for one user were different flavours of Monopoly. Without online testing, it is difficult to decide whether it would be beneficial or not. Moreover, it is not obvious which versions of games we should exclude: should we always recommend the newest reimplementation? Or maybe the user is a Star Wars fan - would he prefer playing vanilla version of Monopoly or is it safe to assume that he is more interested in Monopoly: Star Wars edition?

Last but not least, if we were to release our recommender system, there remains a lot o engineering work to be done. Server with deployed model should be created and integrated with frontend attractively presenting predictions to users. Some additional features such as filtering recommendations based on users input may be added. New data should be automatically, periodically downloaded and used to update the model. Some kind of CI/CD pipeline may be needed in order to ensure stability of the system during modifications.

Bibliography

Print Resources

[9] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. "Sequential User-based Recurrent Neural Network Recommendations". In: *Proceedings of the eleventh ACM conference on recommender systems*. 2017, pp. 152–160.

[11] Crıcia Z Felıcio et al. "A Multi-Armed Bandit Model Selection for Cold-Start User Recommendation". In: *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*. 2017, pp. 32–40.

[12] Chen Gao et al. *Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions*. 2021. arXiv: 2109.12843 [cs.IR].

[13] Carlos A Gomez-Uribe and Neil Hunt. "The Netflix Recommender System: Algorithms, Business Value, and Innovation". In: *ACM Transactions on Management Information Systems (TMIS)* 6.4 (2015), pp. 1–19.

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37.

[18] Maciej Kula. *Metadata Embeddings for User and Item Cold-start Recommendations*. 2015. arXiv: 1507.08439 [cs.IR].

[19] G. Linden, B. Smith, and J. York. "Amazon.com Recommendations: Item-to-Item Collaborative Filtering". In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80. DOI: 10.1109/MIC.2003.1167344.

[21] Mehrbakhsh Nilashi et al. "Collaborative Filtering Recommender Systems". In: *Research Journal of Applied Sciences, Engineering and Technology* 5 (Apr. 2013), pp. 4168–4182. DOI: 10.19026/rjaset.5.4644.

[22] Ladislav Peska and Peter Vojtas. "Off-line vs. On-line Evaluation of Recommender Systems in Small E-commerce". In: *Proceedings of the 31st ACM Conference on Hypertext and Social Media* (2020). DOI: 10.1145/3372923.3404781. URL: http://dx.doi.org/10.1145/3372923.3404781.

[24] Steffen Rendle. "Factorization Machines". In: *2010 IEEE International conference on data mining*. IEEE. 2010, pp. 995–1000.

[25] Francesco Ricci et al. *Recommender Systems Handbook*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 0387858199.

[26] Badrul Sarwar et al. *Application of Dimensionality Reduction in Recommender System – A Case Study*. Tech. rep. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[27] Badrul Sarwar et al. "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 285–295.

[30] Harald Steck. "Embarrassingly Shallow Autoencoders for Sparse Data". In: *The World Wide Web Conference on - WWW '19* (2019). DOI: 10.1145/3308558.3313710. URL: http://dx.doi.org/10.1145/3308558.3313710.

[31] Harald Steck. "Training and Testing of Recommender Systems on Data Missing not at Random". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (2010).

[34] Jan Zalewski, Maria Ganzha, and Marcin Paprzycki. "Recommender System for Board Games". In: *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*. 2019, pp. 249–254. DOI: 10.1109/ICSTCC.2019.8885455.

Online Resources

[1] Scott Alden and Toby Mao. *Announcing New Feature - Game Recommendation System*. URL: https://boardgamegeek.com/thread/2140686/announcing-new-feature-game-recommendation-system.

[2] Apple. *Turi Create*. URL: https://github.com/apple/turicreate.

[3] BoardgameGeek. *Amerithrash*. URL: https://boardgamegeek.com/wiki/page/Ameritrash.

[4] BoardgameGeek. *Category*. URL: https://boardgamegeek.com/wiki/page/Category.

[5] BoardgameGeek. *Eurogame*. URL: https://boardgamegeek.com/wiki/page/Eurogame.

[6] BoardgameGeek. *Mechanic*. URL: https://boardgamegeek.com/wiki/page/Category.

[7] BoardgameGeek. *Title Page*. URL: https://boardgamegeek.com/.

[8] BoardgameGeek. *Worker Placement*. URL: https://boardgamegeek.com/boardgamemechanic/2082/worker-placement.

[10] *DVC*. URL: https://dvc.org/.

[14] Nicolas Hug. *Surprise - A Python scikit for recommender systems*. URL: https://github.com/NicolasHug/Surprise.

[16] Maciej Kula. *LightFM Documentation*. URL: https://making.lyst.com/lightfm/docs/home.html.

[17] Maciej Kula. *LightFM Github*. URL: https://github.com/lyst/lightfm/tree/master/lightfm.

[20] Steve Liu. *Personalized User Recommendations at Tinder*. 2017. URL: https://mlconf.com/sessions/personalized-user-recommendations-at-tinder-the-t/.

[23] Reddit. *Soloboardgaming subreddit*. URL: https://www.reddit.com/r/soloboardgaming/.

[28] Markus Shepherd. *BoardGameGeek games and ratings datasets*. URL: https://www.boardgamegeek.com/thread/2287371/boardgamegeek-games-and-ratings-datasets.

[29] Markus Shepherd. *Recommend Games*. URL: https://recommend.games/.

[32] Adrian Tam. *Using Singular Value Decomposition to Build a Recommender System*. URL: https://machinelearningmastery.com/using-singular-value-decomposition-to-build-a-recommender-system/.

[33] Wikipedia. *Cold start (recommender systems)*. URL: https://en.wikipedia.org/wiki/Cold_start_(recommender_systems).

Appendix A

Data Processing

Before feeding the data to recommendation algorithms, a lot of preprocessing was needed. The goal was to reduce the size of datasets, get rid of irrelevant information and turn data into format required by the algorithms.
Data integrity, reproducibility and ease of collaboration was assured by using DVC [10]. DVC is an open-source version control system build specifically to suit the needs of machine learning projects, designed to handle large files and create lightweight data pipelines.
The steps of the data processing pipeline are shown on the Figure A.1.

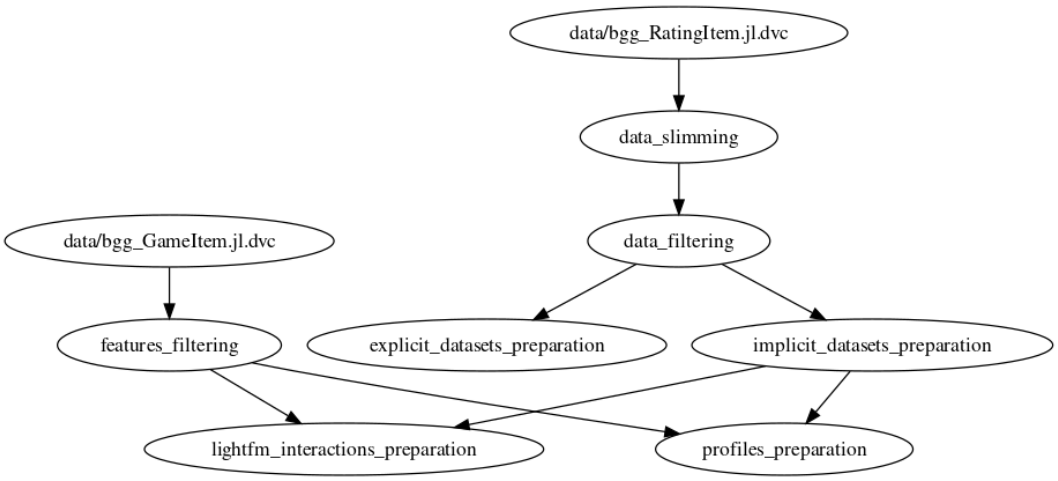


Figure A.1: An illustration of data processing pipeline. Arrows correspond to dependencies between files/steps of processing. What is the function of each of the nodes is described in A.1 and A.2.

A.1 Interactions dataset processing

Pipeline for interactions dataset (bgg_RatingItem.jl) works as follows:

1. We drop all the columns except bgg_user_name, bgg_id, bgg_user_rating, and bgg_user_owned in order to reduce size of the dataset. Then we drop all the rows, where user neither rated nor owned the game (data_slimming).
2. We filter out games which now appear in less than 1000 rows and users who are present in less than 10 rows of dataset (data_filtering). It would be difficult to produce meaningful predictions for them and dealing with cold-start problem [33] was not the goal of our thesis.
3. Then, for explicit and implicit models, data is handled differently:
   - For explicit dataset we drop rows which do not have the rating field filled. They are the explicit feedback we want to use, therefore data not including them is not useful.
     Then, in order to split the data into training and testing set, we decided to divide users into test (30% of users) and train (70% of users) groups. The users from train group have all their interactions added to training set. The users from test group have 70% of their interactions added to training set, while the remaining 30% are used to create testing set. (explicit_datasets_preparation)
   - For implicit dataset, we drop rows which were not a positive interaction. Our assumption is that positive interactions are those, where the user rated the game higher than 6.0 or did not rate the game at all, but marked it as owned.
     Then the data is split into training and testing set like in explicit_datasets_preparation stage (implicit_datasets_preparation).
     It is worth noting that we use the name "implicit" as simplification here - this dataset is in fact hybrid, as it includes both implicit (owning the game) and explicit (rating) information.

After the processing, the full explicit dataset contains 15154652 interactions of 250629 unique users and 4484 unique games. It is then used to train and evaluate the SVD [26] model.

After the processing, the full implicit dataset contains 19691743 interactions of 255150 unique users and 4484 unique games. It is then used to train and evaluate the models based on LightFM library [17]. In order to do so, it is transformed into LightFM-specific data structures (lightfm_interactions_preparation).

Another use of implicit data is to create games and users' profiles (profiles_preparation), which was described in greater detail in chapter 4. Granularity of complexities' discretization was 0.25, resulting in 21 possible discrete counterparts. Recommended

players' counts that we distinguished were $1, 2, \ldots, 10+$. The weights we decided to use for creation of users' profiles were $w_c = 3$, $w_m = 3$, $w_p = 1$ and $w_x = 1$. Therefore, the maximum theoretical score of our custom metric, that a set of recommendations can achieve, is 8.

A.2 Games features preprocessing

LightFM model uses features to construct users and items' embeddings. We decided to only inform it about games' 20 most popular categories, mechanics and designers (excluding designer "Uncredited"). features_filtering stage in the pipeline removes other game features. We have only chosen the most popular features, as LightFM is unable to learn their significance. Including too many can lead to worse results, especially if they are noisy and sparse.

Complexity and recommended number of players were not initially included, because LightFM only handles discrete features. Afterwards, we discretized them for the purpose of using them in our metric. Nonetheless, we did not update the features, because we wanted to see whether the model will be able to optimize components of our metric, for which it does not have specific features.

Figures A.2 and A.3 show distribution of the most popular mechanics and categories in games' dataset (bgg_GameItem.jl).

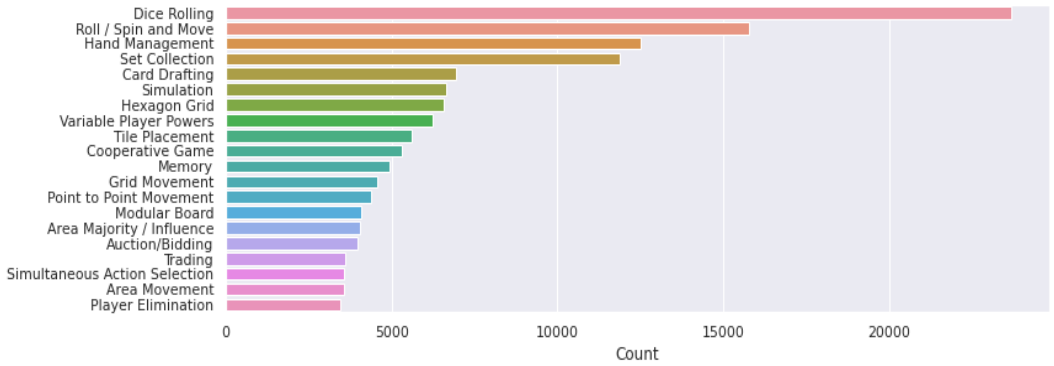


Figure A.2: Histogram illustrating number of games featuring each of the 20 most popular mechanics.

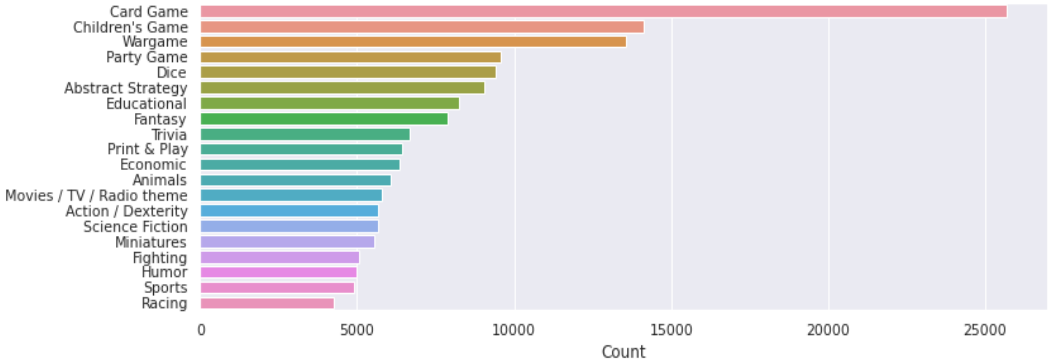


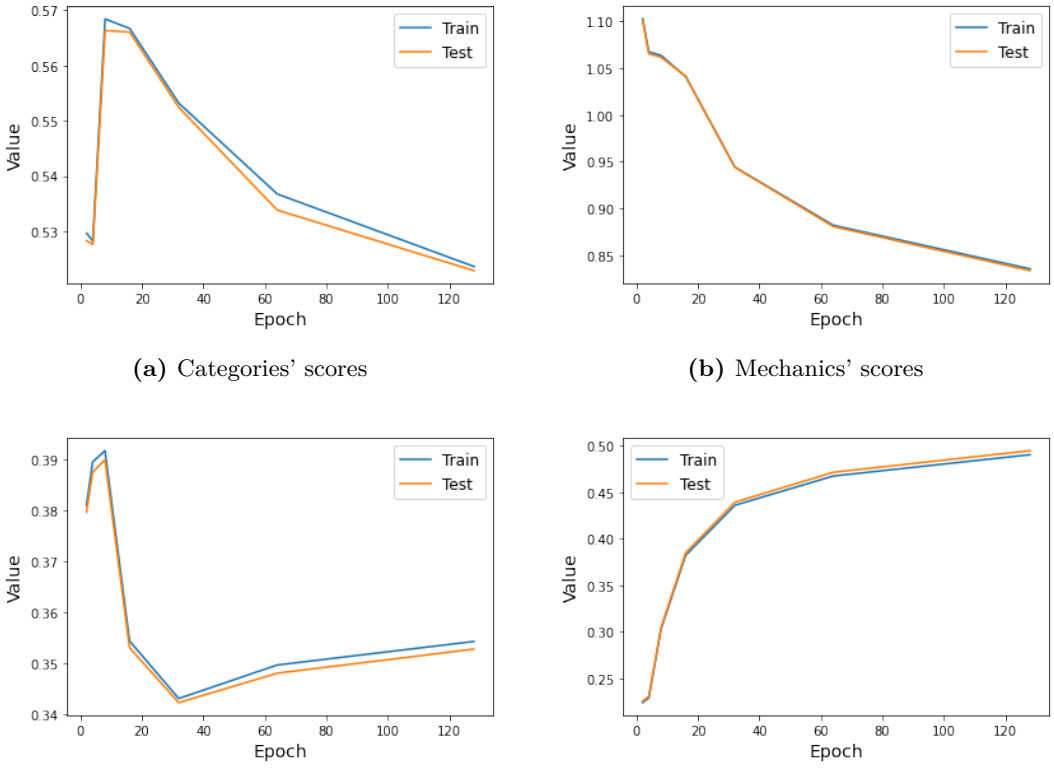
Figure A.3: Histogram illustrating number of games featuring each of the 20 most popular categories.

Appendix B

Additional Results

B.1 Additional plots of models' training progress

B.1.1 SVD



(a) Categories' scores

(b) Mechanics' scores

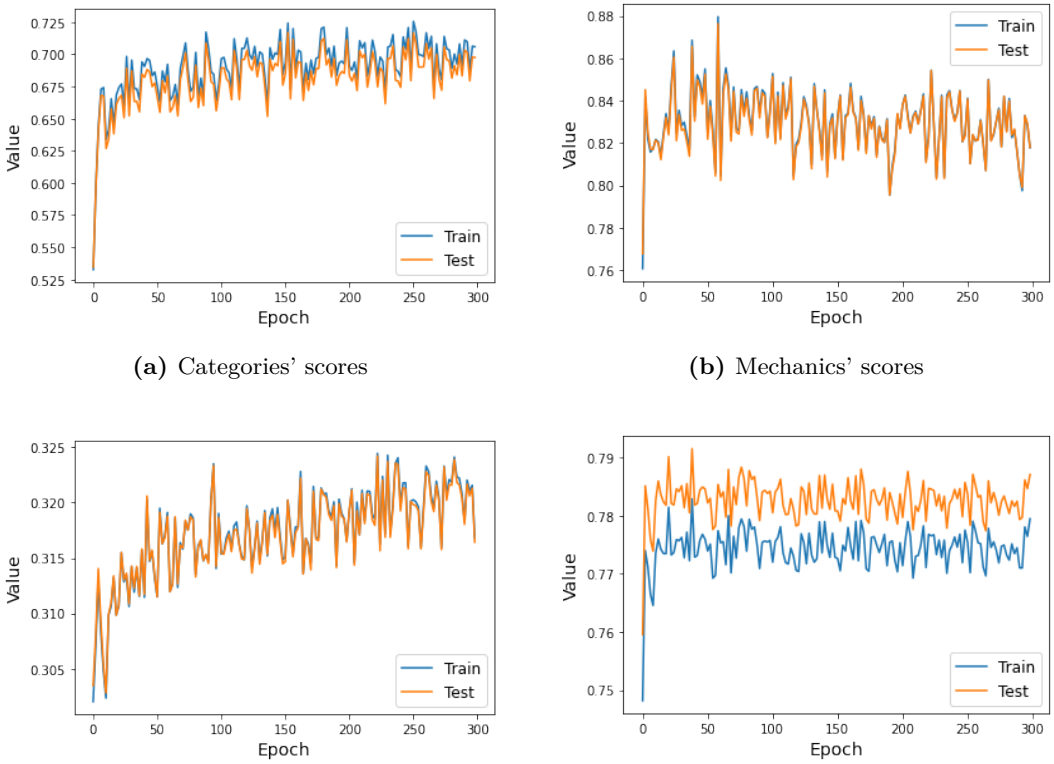
(c) Players' counts' scores

(d) Complexities' scores

Figure B.1: Plots of scores of custom metrics' components achieved during training of SVD model. Most of them were decreasing with time, with exception of complexities' scores.

Trends in categories' and players counts' scores are similar to the ones observed in RMSE values plots from section 5.2.1. Complexities' scores were improving during the whole of training, in contrast to mechanics' scores.

B.1.2 LightFM with WARP fitting function



(a) Categories' scores

(b) Mechanics' scores

(c) Players' counts' scores

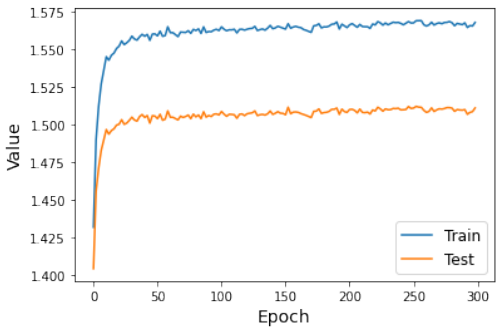
(d) Complexities' scores

Figure B.2: Plots of scores of custom metrics' components achieved during training of LightFM (WARP) model. It can be observed that after initial increase, scores of every component except for players' counts remained in similar ranges for the rest of training

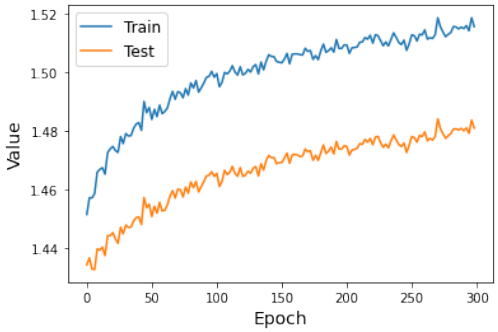
After initial increase, scores of every component except for players' counts remained in similar ranges for the rest of training. They were oscillating rapidly from epoch to epoch, which suggests that their values are only loosely connected to precision of model. Players' counts' scores exhibited a slight upward trend.

B.1.3 LightFM with custom fitting function

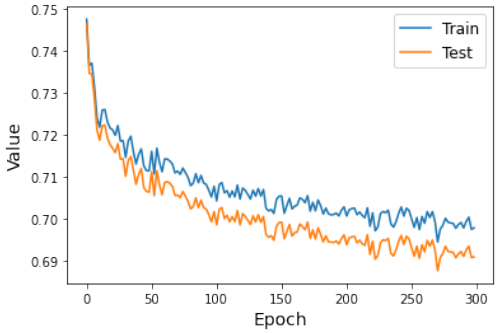
Categories' scores seemed to be the easiest to increase - they quickly achieved high values and were slowly increasing for the rest of epochs. Model was also able to steadily improve mechanics' results. These two components were the least challenging to accommodate for, probably because information about them were a part of item features passed to model. Their improvement in comparison with other models was the biggest. Both trend and achieved values of complexities' scores were similar to respective results of LightFM model trained using WARP loss. Players' counts' information ended up being the most difficult or unprofitable to model compared to the rest of components. Their scores, while bigger than for previous models, were decreasing with time.



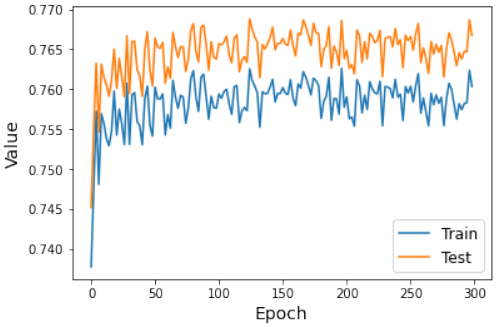
(a) Categories' scores



(b) Mechanics' scores



(a) Players' counts' scores



(b) Complexities' scores

Figure B.4: Plots of scores of custom metrics' components achieved during training of LightFM (custom) model. It can be observed that score of mechanics' component was steadily improving. The one corresponding to players' count was decreasing. Complexities' and categories' components quickly improved at the beginning and then their increase slowed down.

B.2 Custom LightFM model's scores for users with different activity levels

We analyzed results of LightFM model trained using custom fitting function for users with different activity levels. Users with at least 5 items in test interactions were chosen (so that achieving 1.0 precision@5 is theoretically possible) and divided into groups based on number of their train interactions. Then, precision@5 and custom metric score based on their test profiles was calculated. Number of users and achieved results in each group are included in Table B.2. Precision score is rising with number of train interactions. Because of the way we split our data into train and test sets, bigger number of known liked items leads to bigger number of test interactions. Therefore, it is more probable that model will recommend a game included in users' test set and achieving higher precision is easier. On the other hand, the more train interactions user have, the lower the custom metric score gets. One reason for that may be that profiles created based on larger number of games are more diversified and may represent preferences for wider variety of features with lower weights. As a result, it is more difficult to cover them with a limited set of recommendations.

Table B.1: Table of scores of precision@5 and custom metric of recommendations produced by LightFM (custom) model for users with various level of activity. It can be observed that the more training interactions a user has, the lower the score of custom metric is. On the other hand, precision is increasing with the number of train interactions.

Minimum number of training interactions	Maximum number of training interactions	Number of users in group	Precision@5	Custom metric's score
0	15	8387	0.008	4.543
15	25	11874	0.009	4.467
25	50	17057	0.010	4.400
50	100	13102	0.014	4.351
100	250	8515	0.023	4.295
250	500	1656	0.038	4.237
500	–	307	0.076	4.184

Appendix C

Technical Documentation

In this appendix we provide short instructions how to set up the project in order to be able to reproduce our results. We also go over the project structure.

C.1 Setup

1. Create virtual environment:

   ```
   python -m venv venv && source venv/bin/activate
   ```

2. Install required dependencies:

   ```
   pip install -r requirements.txt
   ```

3. Install our modified version of LightFM:

   ```
   cd lightfm-custom && python setup.py cythonize && pip install -e .
   ```

4. To acquire raw data contact either us or Markus Shepherd [28]. Then to preprocess it, run the pipeline manually in order described in Appendix A or use DVC:

   ```
   dvc repro
   ```

5. Finally, run following command to properly set up PYTHONPATH environmental variable:

   ```
   export PYTHONPATH=$PYTHONPATH:.
   ```

C.2 Project structure

The project consists of following directories:

- `data` - place for storing all the data
- `data-pipeline` - includes scripts making up the data processing pipeline
- `lightfm-custom` - includes modified version of LightFM package; needs to be installed as described in C.1
- `recommender` - includes utilities for calculating our metric and evaluating the models
- `notebooks` - includes all the Jupyter notebooks we used to produce our results, grouped into subdirectories; references provided in the brackets link to the descriptions of the results obtained using these notebooks:
  - `data_analysis` - the notebooks for analyzing the data (Chapter 3)
  - `custom_metric_analysis` - the notebooks for experiments regarding our custom metric (Section 5.1, Section B.2)
  - `model_fitting` - the notebooks for training the models
  - `training_progress_analysis` - the notebooks for analysing progress in training the model (Section 5.2, Appendix B)
  - `model_evaluation` - the notebooks for evaluating the models (Subsection 5.3.1)
  - `embeddings_analysis` - the notebooks for analyzing the users' embeddings (Subsection 5.3.2)
  - `predictions_analysis` - the notebooks for analyzing recommendations produced by the models (Subsection 5.3.3)