UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kristjan Lõhmus

Collaborative filtering recommendation algorithms performance on an implicit feedback dataset

Bachelor's thesis (9 EAP)

Supervisors: Anna Aljanaki, PhD
Hakan Berber, PhD

Tartu 2021

Arvamuspõhiste soovitussüsteemide tomimine kaudsel tagasisidel

Lühikokkuvõte:

Antud töö eesmärgiks on valida ja implementeerida soovitussüsteem USA-s opereerivale võrgumängude platvormile. Süsteemi eripärasid ja olemasolevaid andmeid arvestades valiti mudelipõhine lähenemine süsteemi koostamiseks. Implementeeriti kaks mudelit: Alternating Least Squares (ALS) ja Bayesian Personalized Ranking (BPR), mida treeniti süsteemist saadud andmete põhjal. Mudelite väljundi hindamiseks kasutati AUC-d ja mediaantäpsust. Tulemused näitasid, et mudelid töötasid koguandmetel identse täpsusega, kuid uute mängijate hindamisel sai parema tulemuse ALS.

Võtmesõnad:

Soovitusüsteem, arvamuspõhine filterdamine, maatriksi faktoriseerimine, ALS, BPR

CERCS: P176 Tehisintellekt

Collaborative filtering recommendation algorithms performance on an implicit feedback dataset

Abstract:

This thesis aims at investigating recommender systems based on a prior implicit dataset to enhance customer satisfaction on an online gaming platform operating in the United States (US). Such model-based algorithms as Alternating Least Squares (ALS) and Bayesian Personalized Ranking (BPR) were chosen in addition to baseline algorithms. A dataset including implicit preferences that was received from the platform was used to test and implement the models. The results showed an identical performance on evaluating the whole system, however ALS performed better with players who were new to the system.

Keywords:

Recommendation engine, Collaborative filtering, Matrix factorization, ALS, BPR

CERCS: P176 Artificial intelligence

Contents

1. Introduction

Recommendation algorithms are a major part of modern-day online services. They recommend content based on users interests and thus save users' time. Instead of having to scroll through tens or hundreds of irrelevant items, the user has a higher chance of finding his/her preferable content within the first results. This increases the likelihood of users consuming content of the service. For instance, according to Netflix, 80% of their content streamed is influenced by their recommender system [1].

In online gaming platforms the key for success is player conversion rate - to keep players returning after their first session. Conversion rate can be greatly increased by serving each player a personalized experience. One of the powerful ways to increase conversion rate is via a recommendation engine. Accenture has stated that 91% of consumers will remember a brand and be more likely to return when they have been provided with relevant offers and recommendations to engage with [2].

The aim of this thesis was to investigate recommender systems based on a prior implicit dataset to enhance customer satisfaction on an online gaming platform operating in the United States (US). The platform has been operating since 2018 and has an active player-base of 50 000 users. The novelty of the current thesis is stemming from the investigation of the recommender systems on an online gaming platform.

The "Preliminaries" chapter gives an overview of ways to collect user feedback, general approaches to recommendations and delves deeper into latent factor models. "Methods" chapter describes the approach to extracting user feedback, description and processing of the data extracted, evaluation metrics and model creation. The model training and evaluation results are displayed and explained in "Results". "Summary" chapter concludes everything that has been done, outlines essential findings and ideas for future work.

2. Preliminaries

2.1 User feedback

The purpose of a recommendation system is to provide users with a list of most suitable items, where the first item is most and the last one least suitable for the user [3]. Understanding of prior user preferences play a central role in proposing such a list. Data about user preferences can be gathered in two ways: explicitly and implicitly [4].

Explicit feedback is direct user feedback, i.e., a user is asked to rate a book or a movie on a scale of one to five stars. This means that the user can rate items both positively and negatively, which gives us an accurate presentation of user's interests. Another positive side of explicit feedback is that it is unambiguous, and it does not need further processing or interpretation. However explicit rating requires an extra input from users to rate the item. This additional step is often ignored by the user, and it causes explicit feedback to be rare. When explicit feedback is present, it tends to have a bias for positivity, since users would rather not give feedback to items they do not like [5].

Implicit feedback is indirect feedback, that can be derived from user's behaviour. For example, we can measure user's viewing time of a movie. In contrast to explicit feedback, implicit feedback can be gathered in abundance. We can track different user interactions, and we do not have to rely on direct feedback [6]. However, there are trade-offs for implicit feedback. For example, a user has watched movie A for 10 times, movie B for 2 times, and has not watched movie C. It can be deduced that the user likes movie A better than movie B. Whether the user likes movie B or C cannot be inferred based on this data. This is caused by implicit data being relative rather than absolute [7]. From implicit feedback we also cannot infer that user did not watch movie C because he/she did not like it or if there was another reason. This means that generally implicit feedback cannot be negative [8].

Characteristics of explicit and implicit feedback can be found in Table 1 below.

Table 1. Implicit and explicit feedback properties [7].

	Implicit feedback	**Explicit feedback**
Accuracy	Low	High
Abundance	High	Low
User preference mapping	Positive values	Positive and Negative values
Measurement reference	Relative	Absolute

2.2 Content-based, hybrid and memory-based collaborative filtering

Recommender systems can be designed in a multitude of ways. Most common techniques are content-based filtering, collaborative filtering, and hybrid filtering [9]. Figure 1 illustrates the widely accepted categorization of recommendation techniques.

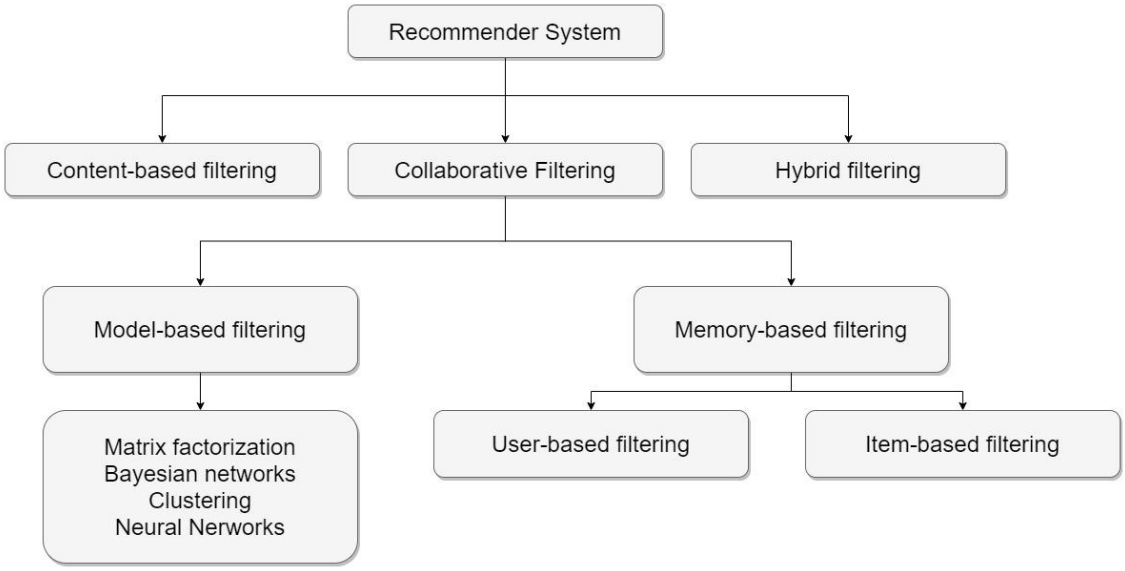

Figure 1. Types of recommendation systems

Content-based filtering (CBF) uses item similarity to recommend new items to the user. Item similarity is calculated on features associated with each item. Similar items are grouped together and if a user interacts with one of the items in the group, other items from the same group are recommended. For instance, if a user has watched many movies with an actor, CBF will recommend other movies with the same actor [3]. The main benefit of using a CBF stems

from addressing the cold-start problem. CBF needs minimal information about the user to make recommendations, and instead requires item features, which often need to be manually defined [10].

Memory-based Collaborative filtering (CF) assumes that similar users like similar items and makes recommendations based on that. Memory-based CF algorithms use memory to find relations between users or items depending on the model type (user-based or item-based) [11]. These models are also called neighbourhood-based methods since they heavily utilize nearest-neighbour algorithms to group similar users [12].

Hybrid filtering techniques are a combination of collaborative and content-based filtering techniques. These are used together to eliminate each other's disadvantages. An example of a hybrid technique would be a memory-based CF system combined with CBF. It is known that CF suffers from a new item problem – if an item is new to the system, it does not have many interactions and therefore is rarely recommended. To solve this, CBF can be integrated with CF to push newer items to the recommendations. This approach eliminates the cold-start problem that CF usually has but increases the complexity of the model substantially [3].

2.3 Model-based collaborative filtering

Model-based CF differs from other systems. In model-based CF, machine learning is used to learn patterns in user behaviour and produce a predictive model [10]. Unlike memory-based CF methods, model-based CF does not calculate similarity between users or items. Instead, model-based CF assumes that user preferences are determined by several different unobserved relations. These are called latent factors [13].

Latent factors are hidden features that the model creates based on the given data. Algorithms then utilise these features to place users and items into a multidimensional space based on their interactions. Figure 2 illustrates a simple two-dimensional latent factor space with given users and items. As can be seen in the figure, the model places users and items in the same space. The closer the user is to an item, the higher the predicted rating is for that user-item interaction. For instance, user A would have high predicted ratings on items 8 and 9, but a low rating for item 1 [14].

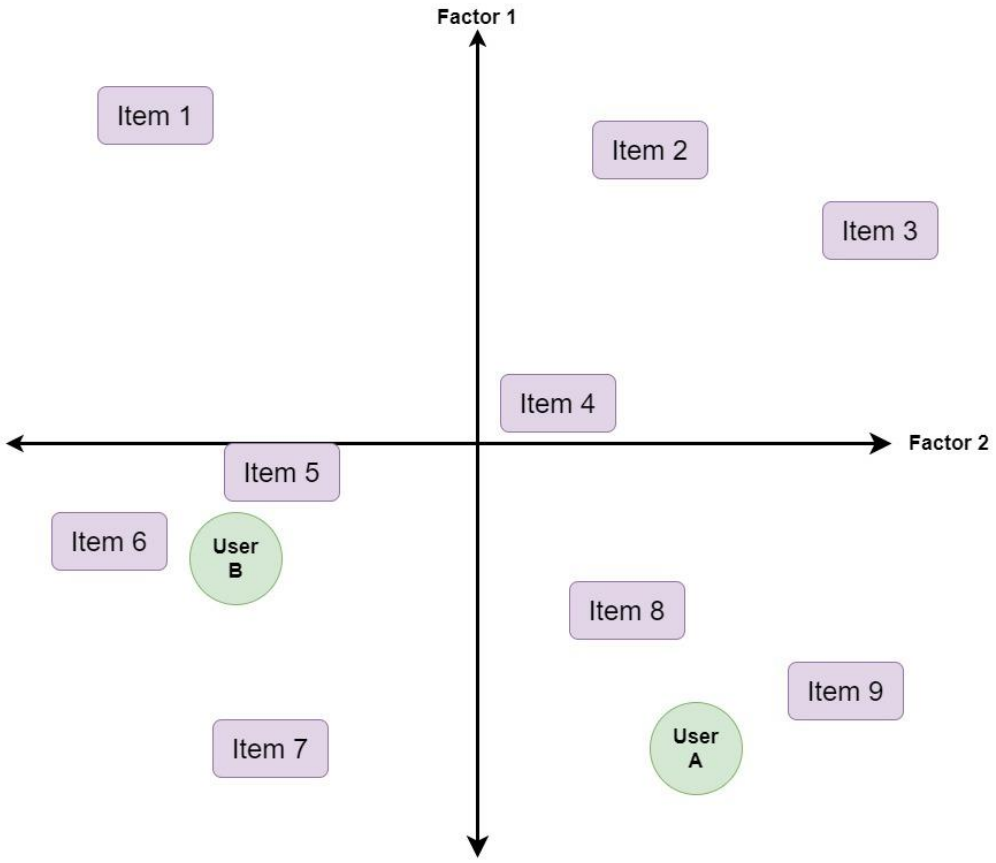

Figure 2. Two-dimensional latent factor space.

Multiple of different approaches have been developed to explore latent factors and find patterns in user behaviour. These include Clustering techniques, Bayesian networks and neural networks [10]. These systems perform well but have been overtaken in popularity by matrix factorization (MF) techniques [15]. MF techniques include singular value decomposition (SVD), non-negative matrix factorization (NMF), Slope one and so on [16].

2.4 Matrix factorization

Matrix factorization (MF) methods operate on a data structure called user-item matrix. A dataset consisting of *m* users and *n* items which gives us matrix *R* with a size of $m \times n$ in which every entry is noted by r_{ui} – user u's rating for item i [17]. An example of a user-item matrix can be seen in Table 2.

Table 2. Example of a user-item matrix

	Item 1	Item 2	…	Item n
User 1	$r_{11} = 0$	$r_{12} = 3$	…	$r_{1n} = 2$
User 2	$r_{21} = 1$	$r_{22} = ?$	…	$r_{2n} = 4$
…	…	…	$r_{ui} = ?$	…
User m	$r_{m1} = 5$	$r_{m2} = ?$	…	$R_{mn} = 0$

General purpose of matrix factorization algorithms is filling the missing values in the user-item matrix *R* by predicting each r_{ui}. To do this MF decomposes the large $m \times n$ dimensional user-item matrix *R* into $k \times m$ user matrix *X* and $k \times$ n item matrix *Y*. In *X*, each user *u* is presented by a vector of latent factors x_u and in Y, each item *i* is presented by a vector y_i. Each interaction can be predicted by getting the dot product of transposed x_u and y_i - $r_{ui} = x_u^T y_i$ [17].

To give the best predictions matrix factorization needs to find the optimal combination of latent factors in *X* and *Y*. To do this, MF tries to minimize square errors of the ratings that are already present using formula:

$$\min_{X,Y} \sum_{known\, r_{ui}} c_{ui}(r_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right)$$

In this formula λ is used for regularizing the model. To indicate preferences of user u on item i a set of binary values p_{ui} are created.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

If user u has interacted with item i then it indicates that user u likes i ($p_{ui} = 1$). If no interaction has been recorded between them, then no preference is extracted from that ($p_{ui} = 0$). This set of binary values does not give information about why the user preferred the item or not. The item might be too expensive or just was not visible for the user. To indicate confidence in

recorded interactions, variable c_{ui} is used to measure confidence in rating r_{ui} Because x_u^T and y_i are unknown means that this approach is non-convex and therefore NP-hard to optimize [17] [8].

2.5 Alternating Least Squares

ALS tries to optimize the problems that classic matrix factorization has by fixing one of the unknown variables.

To converge on the optimal combination of latent factors, ALS fixes user factors X and optimizes item factors Y. Once that is done, item factors Y are fixed and user factors X optimized. This process is repeated until convergence. To serve implicit feedback ratings, confidences in present interactions are gathered into $k \times$ k matrix C_u where $C_{ii}^u = c_{ui}$ and $c_{ui} = 1 + \alpha\, r_{ui}$.

Algorithm 1. Implicit matrix completion with ALS

Initialize X, Y

repeat

for $u \in 1 \ldots m$ **do**

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u\, p(u)$$

end for loop

for $i \in 1 \ldots n$ **do**

$$y_i = \left(X^T C^i X + \lambda I\right)^{-1} X^T C^i\, p(i)$$

end for loop

until convergence

Once X and Y are calculated the system needs to calculate predictions based on these latent factor spaces. To do this, the user-item matrix R must be filled with predictions $r_{ui} = x_u^T y_i$ for each user-item pair. Computational cost of performing this operation is $O(nmk)$. To extract recommendations from R all that has to be done is sorting user u's predicted ratings from highest to lowest values and extracting the first k elements [17] [8] [14].

2.6 Bayesian Personalized Ranking

Bayesian Personalized Ranking (BPR) algorithm does not focus on predicting each personalized score and instead optimizes on correctly ranking items for the user. This is done by taking different item pairs from our user-item matrix and learning patterns from these pairs [15].

Instead of user-item pairs, BPR uses relationship between user u and items i, j. As presented in Figure 3, a smaller item to item matrix is made for each user. In this matrix, each cell in a column represents if the item is preferred over the item in that row. If user u has interacted with item i, but not with item j, then item i is preferred over item j. If the user has interacted with both items i and j, then BPR does not prefer one over the other. Same applies when the user has not interacted with neither of the items [15].

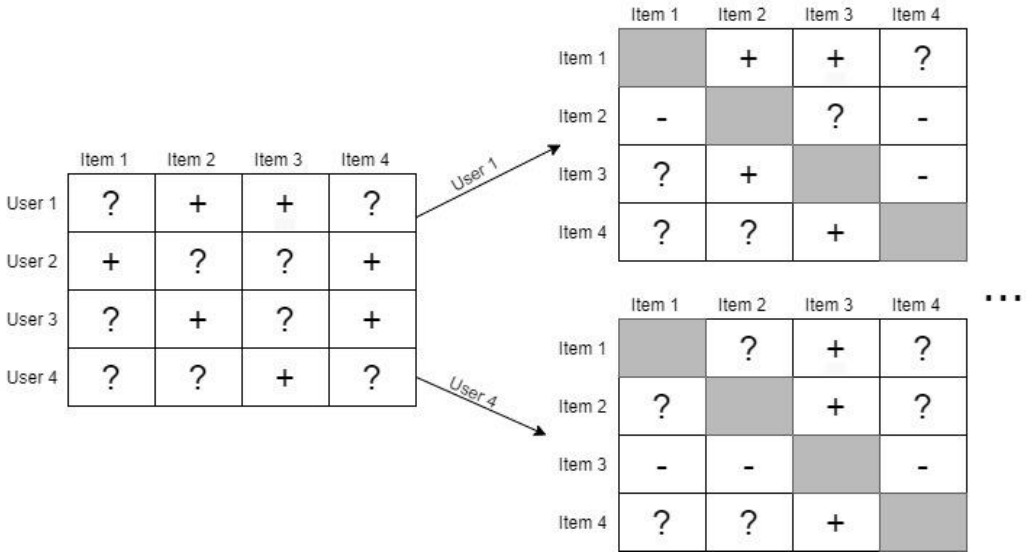

Figure 3. Bayesian Personalized Ranking user-item matrix decomposition.

Probability that user u prefers item i over item j is given as

$$\boldsymbol{p(i >_u j|\Theta) = \sigma(r_{uij}(\Theta))}$$

where σ is the logistic sigmoid

$$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-x}}}$$

and $r_{uij}(\Theta)$ is the underlying model that calculates the difference between assumed ratings r_{ui} and r_{uj}.

BPR calculates the difference between two predictions $r_{ui} - r_{uj}$. But because BPR itself does not generate predictions to get that difference, BPR can use any standard CF technique

such as k-nearest neighbours or matrix factorization. To optimize any collaborative algorithm to classify difference between predictions, the following algorithm can be used:

Algorithm 2. Optimizing models for BPR

Initialize Θ

repeat

draw $(u, i, j)\ from\ R$ **do**

$$\Theta \leftarrow \Theta + \alpha\left(\frac{e^{-r_{uij}}}{1 + e^{-r_{uij}}} \cdot \frac{\partial}{\partial\Theta} r_{uij} + \lambda_\Theta \cdot \Theta\right)$$

until convergence

In this algorithm, α is the learning rate of the algorithm and λ_Θ is model regularization. The way user-item triplets are selected is crucial in the algorithm, because the ordinary 1 to n selection may lead to poor convergence. To avoid this problem, the selection of (u, i, j) from R must be done randomly for each cycle [15].

2.7 Baseline algorithms

To evaluate how well ALS and BPR perform on the provided dataset, two extra models are included. These models are:

- Most popular: Recommends items based on how many interactions it has. The item which is most interacted with by the users is recommended as first.
- Random: Permutations of all the items in the dataset are recommended to the user.

If the models perform worse than the baseline algorithms, then it is a signal that the models are poorly implemented. [18]

3. Methods

3.1 Extracting user preferences

The recommendation system is made for an online gaming platform operating in Pennsylvania in the US. Each user in the system is assigned a unique identification by which all his/her actions can be tracked. When a user starts playing a game, a game session is started. Each game session contains information about each round user plays during the session, how much they invested and how much they gained or lost. Unfortunately, the system does not ask user to rate games which means that explicit feedback is not available for use.

Many different options were considered for what best expresses the user's interest in a game. According to Parra and Matriain in a 2011 study on the relation between implicit and explicit rating, time spent interacting with an item has the most significant correlation on how user rates the item. The more time the user has spent interacting with the item, the more likely he/she is to rate the item highly. The problem here is that from our system, it is impossible to extract if the user was actively interacting with the game or was inactive for the whole time. To extract how much active time user has spent with a game, for each user-item combination, number of rounds played in the game was extracted. In addition to that, data was gathered on how much time each round takes. To get the active time spent on a game, the following formula was used:

$$t_{active\ time\ spent} = i_{amount\ of\ interactions} \times t_{time\ per\ interaction}$$

3.2 Dataset description

Two datasets were extracted from the online gaming platform. First dataset is a list of all games users have played since 01.01.2020 with their round counts. There was a total of 105,680 players, 333 games and 3,644,287 interactions recorded. This dataset has 3 fields:

- Player_id: Unique player identifier.
- Game_id: Unique game identifier.
- Game_category: Type of the game (I.e., slot, table or roulette)
- Rounds: How many rounds the player has in a game.

The second dataset contains all games with how many seconds it takes to complete an average round in the game. This dataset has 2 fields:

- Game_id: Unique game identifier

- Speed: How much time it takes to complete an average round (s).

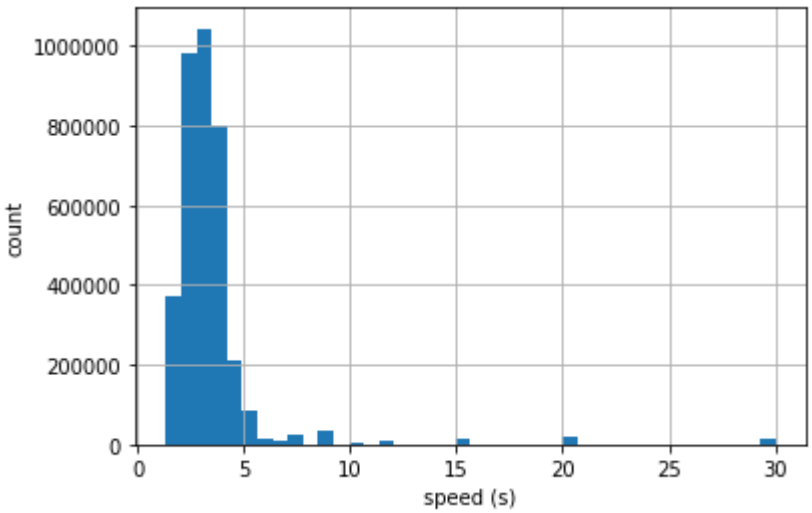

Figure 4. Histogram of how long it takes to complete a round.

3.3 Data pre-processing

To obtain a unified dataset, these datasets were merged by game id, so each row in the rounds dataset was assigned a speed of how long it takes to complete a round in the game. In some cases, game speed was not available for a game in the user-item dataset. To solve this, a mean speed was calculated for each category and added to the games that were in this category and were missing speed.

Next problem was to remove outliers. For this, interactions with 0 rounds were removed since those do not show any preference. Also, interactions from the 0.85 quantile were removed to prevent the data from stretching out too much. This meant excluding interactions where the round count was higher than 633.

Rounds and speed (transformed into minutes) were multiplied together, using the formula mentioned beforehand, to receive the implicit rating for each user-item interaction.

After that, the data was transformed into an actual matrix for matrix factorization. SciPy's Sparse library was used to convert the data into a CSR matrix. CSR matrix is a more memory-optimal way to store a regular matrix, that does not store missing values via zeroes and rather stores each value into memory via indices.

The final matrix had a sparsity of 90.01%. This means that out of all the possible user-item interactions, 9.99% were present.

3.4 Data masking

As is standard practice in machine learning, to get an overview of how well the model is performing, it needs to be trained and tested on two different datasets. For this, a mask was applied to the original dataset, which hid random interactions for each user and added them to the test set. This is also known as the "Temporal user-based splitting" strategy [19]. In total 20% of the user-item interaction was masked including every user that had more than 1 interactions. This process is illustrated in Figure 5.

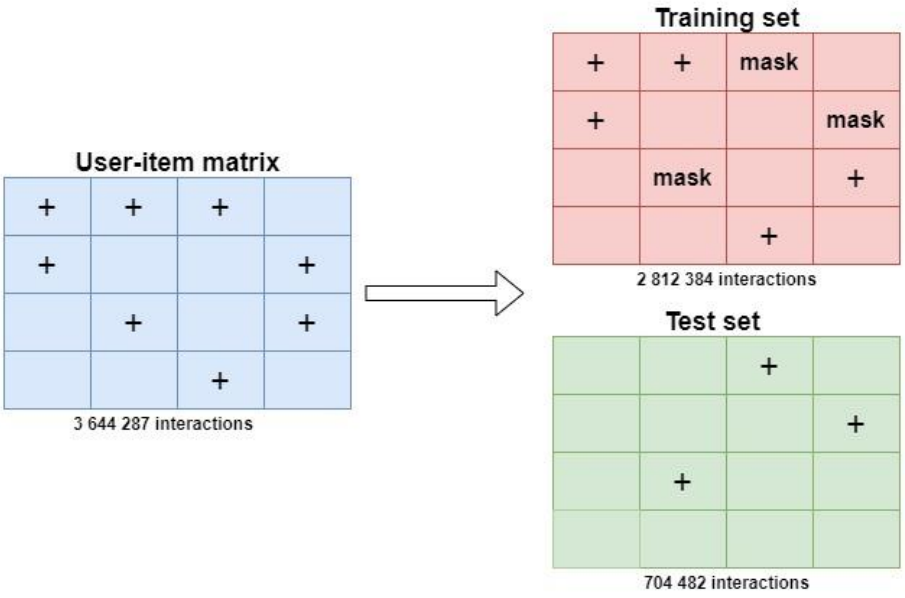

Figure 5. Data masking process.

3.5 Evaluation methods

Recommendation is a problem of ranking items correctly for each user. To test how algorithms used in this paper compare against each other, a unified evaluation system is necessary. The way to evaluate these is to compare how much does the predicted list of items differ from the actual. This can be done via metrics such as precision and Area Under the Curve (AUC) from Receiver Operating Characteristics (ROC).

To understand what these metrics measure, it must be explained first, what successful and unsuccessful predictions are in recommendation systems. Recommendations for a user are in a ranked list, where the first is most relevant and last least relevant to the user. If an item i is more relevant to user u than item j, then it considered to be a positive classification. If it is less relevant, then it considered negative. After training the model with some of the data masked,

the predicted classifications are compared against the known ones in the test set and labelled into four groups: True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) as illustrated in Table 3 [20]. If we define a ranked list of items such as $[1, 2, 3, 4]$ where item number 2 is masked, and our predicted rankings are $[1, 3, 2, 4]$. The evaluation would be done on the position of item 2 and the outcome would be $[TP, FP, 2, TN]$.

Table 3. Classification outcomes in ranked lists.

	Prediction: *Rank(i)* > *Rank(j)*	**Prediction: *Rank(i)* < *Rank(j)***
Actual: *Rank(i)* > *Rank(j)*	True Positive (TP)	False Positive (FP)
Actual: *Rank(i)* < *Rank(j)*	False Negative (FN)	True Negative (TN)

To receive meaningful feedback from classifications, receiver operating characteristic curve (ROC) is used. ROC curve shows the relation between true positive rates (TPR) and false positive rates (FPR). It shows how well a model distinguishes 'noise' from the relevant data on different thresholds [20]. The formulas for calculating TPR and FPR are as follows:

$$\boldsymbol{TPR = \frac{TP}{TP + FN}} \qquad \boldsymbol{FPR = \frac{FP}{FP + TN}}$$

Exemplary ROC curves of a well-performing and a randomly predicting models are illustrated in Figure 6.

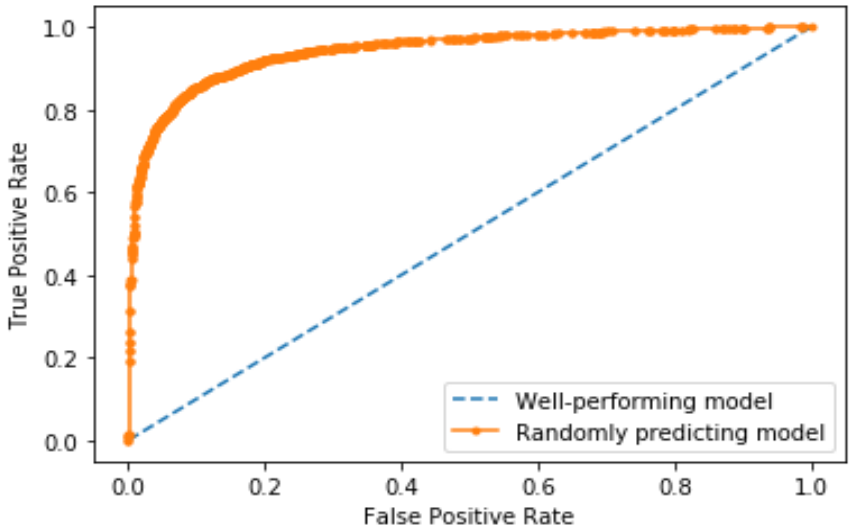

Figure 6. ROC curve for a logistic regression and randomly predicting model.

Area under the ROC curve is a cumulative measure that shows how well the model distinguishes between different classes. As seen from Figure 5, a random classifier would have an AUC of 0.5, which means that it does not distinguish between positive and negative classes. A perfect model, which classifies everything correctly would have an AUC of 1 [21]. AUC is good for assessing recommendation models because it shows how well the predictions are ranked instead of checking how predicted ratings differ from real ones [20].

To show how well the top elements are predicted, precision at k metric is used. Precision at k measures the number of items that are relevant to the user in top k elements. Mean precision at k is the mean of the precision across all users. This metric measures the number of true positives in the top rankings of the list [22].

3.6 Model creation

There are several libraries in Python, which can provide recommendations such as: “Surprise”, “Py-Recommendation” or “LightFM”. Unfortunately, these libraries accommodate for explicit feedback datasets. For creating models based on implicit feedback, “Implicit” library was chosen. “Implicit” implements implicit feedback processing models from their original papers and uses C# together with Python or “Cython” to speed up model training [23].

4. Results

This chapter describes the experimental results. The results are displayed, explained, and analysed.

4.1 Hyperparameter tuning

Implementation of ALS is based on the paper published by Hu, Koren and Volinsky in 2008 with performance optimizations described by Takács, Pilászy and Tikk in 2011. There are three main hyperparameters that determine how well the model performs:

- Regularization (λ): determines regularization for root mean square error calculation.
- Iterations (n): determines how many times the algorithm is performed.
- Latent factors (k): determines how many dimensions are calculated in the latent factor space.

Grid search was used to determine the best parameters available for the model. This meant an exhaustive search through a predetermined hyperparameter space. After finding the optimal hyperparameters, they were fixed and one of the parameters was changed to observe the impact it had on model's AUC. Figure 7 illustrates the impact that number of iterations and regularization had on the AUC score of the model. The best performing ALS model was found with regularization $\lambda = 250$, factor space of $k = 60$ dimensions and the algorithm was iterated through $n = 60$ times.

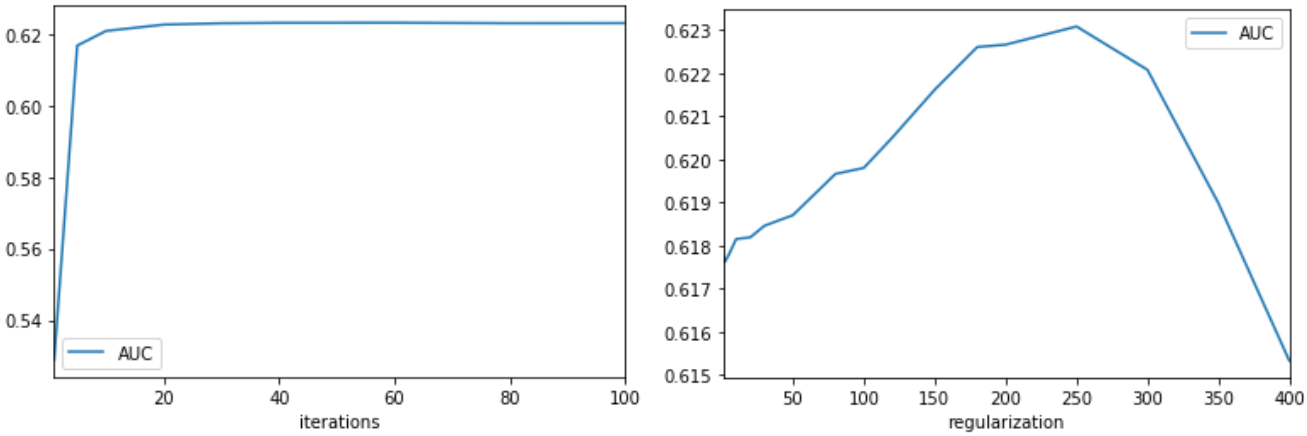

Figure 7. Effect of regularization and number of iterations on ALS model's AUC

BPR was implemented based on the paper published by Rendle et. Al from 2008 called „BPR: Bayesian Personalized Ranking from Implicit Feedback ". The parameters for BPR were the same, but instead of regularization (λ), learning rate (α) was applied to the learning gradient.

The impact of number of iterations and learning rate is illustrated in Figure 8. Best AUC score was received with learning rate $\alpha = 0.02$ factors $k = 60$ and with $n = 400$ iterations.

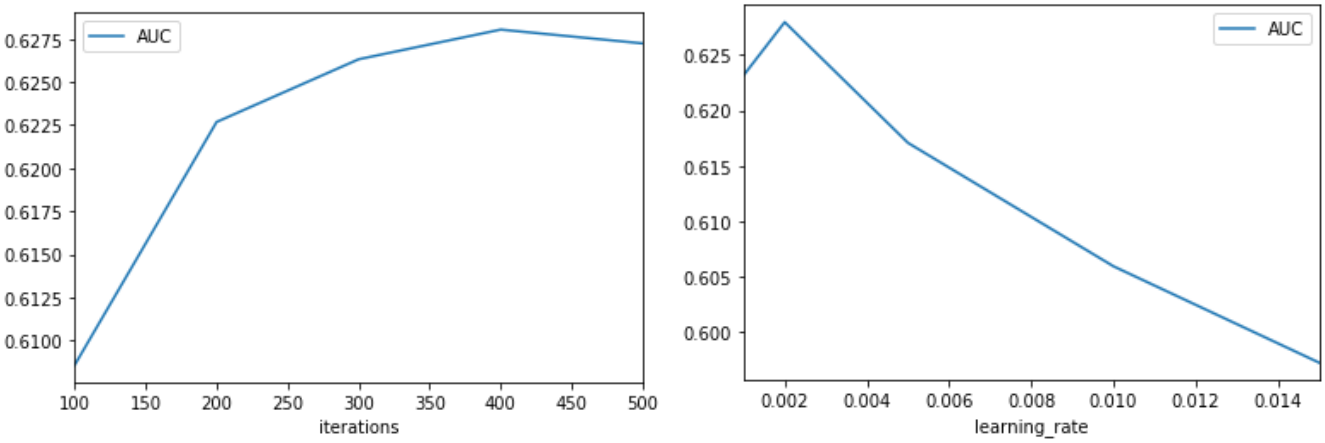

Figure 8. Effect of learning rate and number of iterations on BPR model's AUC

Figure 9 shows the impact that dimensionality had on ALS and BPR. As can be seen from the figure, BPR generally had the better AUC, but ALS achieves similar AUC once the latent space is larger than 60 factors.

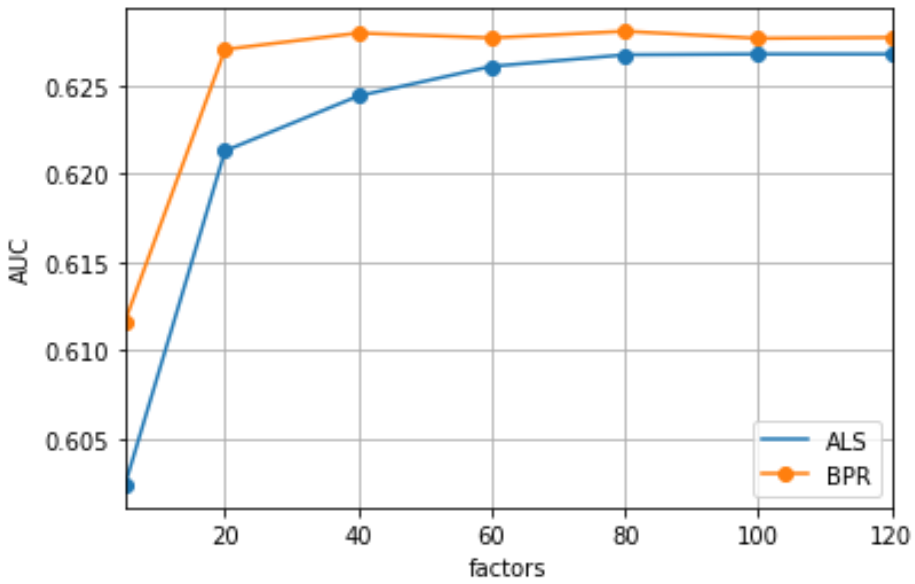

Figure 9. Effect of dimensionality on ALS and BPR AUC.

4.2 Model evaluation

Performance metrics on the whole dataset are displayed in Table 4.

Table 4. Results of model evaluation with optimal hyperparameters.

	AUC	Precision	MAP@5	MAP@10
ALS	.626	**.316**	**.179**	**.171**
BPR	**.628**	.314	.177	.167
Baseline random	.500	.113	.028	.024
Baseline popular	.539	.118	.047	.044

As the table displays, ALS and BPR performed better than baseline algorithms, which means that the models can produce personalized recommendations to some extent. ALS and BPR produce similar results, with ALS gaining a small edge over BPR in precision and BPR having the better AUC.

The evaluation metrics of BPR and ALS are similar and objectively low for latent factor models. In the original paper for BPR by Rendle et Al. (2009), AUC of 0.9 was reached on two datasets. As stated in the original paper, these datasets had been filtered so that every user and item had a minimum of 10 interactions. For dataset presented in this paper, the data was scarce, and such filtering could not be done, resulting in a 0.628 maximum AUC.

Mean average precision (MAP) was evaluated for each user at $k = 5$ and $k = 10$ top items. This evaluation was done to make sure that precision on ranking the first 5 items was not incidental. Generally, MAP should decrease when k gets larger as there are more items to predict. This was also the case with the models evaluated in this paper, but the decrease was marginal and therefore nothing to be alarmed about.

Time measurement of training the algorithm with optimal hyperparameters is illustrated in Figure 9. Training the BPR model takes almost three times as long with every dimensionality. Each iteration takes longer the higher the number of latent factors to calculate. The main cause for this is that ALS requires 100 and BPR 400 iterations. In fact, BPR calculates iterations faster and would take less time if iterated through the same amount.

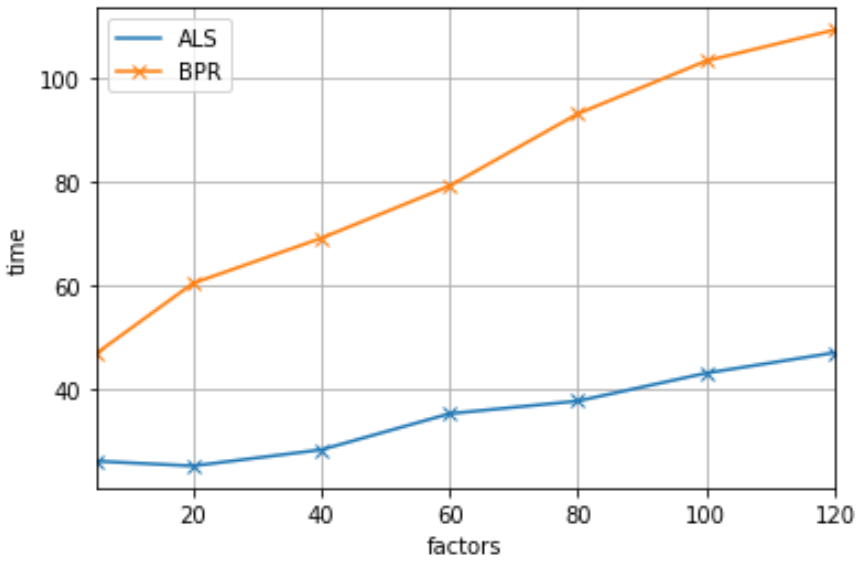

Figure 10. Time for training the models with different dimensionality.

Separate evaluations were done on players whose total time spent on the site was in the first quantile. This was done to examine how well the models can recommend to newer players who have not yet interacted with many items. The results of this evaluation are displayed in Table 5.

Table 5. Results of model evaluation on users with low interaction count.

	AUC	Precision	MAP@5	MAP@10
ALS	**.651**	**.320**	**.200**	**.192**
BPR	.505	.041	.015	.012
Baseline random	.501	.095	.029	.027
Baseline popular	.537	.114	.042	.040

For low interaction users ALS performs significantly better than all other algorithms. ALS has better metrics on low interaction users than it does on the overall userbase. This is an unexpected result, as predictions should be more accurate the more data the model is given. Something similar was observed by Hu, Koren and Volinsky (2008), where their ALS performance remains similar independent of how many items the user had interacted with. Their possible explanation for this was that with increasing user interactions the user profiles become heterogeneous with their profiles being too similar and this makes it difficult for ALS model to differentiate between strong and weak preferences.

5. Conclusion

The purpose of this thesis was to investigate recommender systems based on a prior implicit dataset to enhance customer satisfaction on an online gaming platform operating in the US. The thesis gives an overview of user feedback processing, different recommendation techniques and more specifically matrix factorization.

For recommending items, implicit feedback was gathered from the platform based on how long a user has interacted with an item. This feedback was then used to train two different latent factor models: Alternating Least Squares (ALS) and Bayesian Personalized Ranking (BPR). These models outperformed baseline models that predicted random and most popular items, but still scored objectively low on the AUC metric (.626 and .628 respectively). What was surprising was that the metrics of ALS improved (AUC = .651) when tested on a dataset with players who were relatively new to the system.

No general indicator of which model performed better can be concluded. For future work, both models will be deployed to a live platform and go through A/B testing. From that more decisive results can be drawn.

References

[1] N. Hunt ja C. A. Gomez-Uribe, „The Netflix Recommender System: Algorithms, Business Value,,“ 2015.

[2] Accenture, “https://www.accenture.com/_acnmedia/pdf-77/accenture-pulse-survey.pdf,” 2018. [Online].

[3] F. Ricci, L. Rokach ja B. Shapira, Intoduction to Recommmender Systems Handbook, Springer Science+Business Media LLC, 2011.

[4] J. Iwaniga, N. Nishimura, N. Sukegawa ja Y. Takano, „Improving collaborative filtering recommendations by estimating preferences from clickstream data,“ Elsevier B.V, 2019.

[5] D. Jannach, M. Zanker and L. Lerche, “Recommending based on Implicit Feedback,” Free University of Bozen-Bolzano, Bozen, Italy.

[6] L. Lerche, „Using Implicit Feedback for Recommender Systems: Characteristics, Applications and Challanges,“ der Technischen Universität Dortmund, Dortmund, 2016.

[7] G. Jawaheer, M. Szomszor ja P. Kostkova, „Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service,“ City University London, London, 2010.

[8] Y. Hu, Y. Koren ja C. Volinsky, „Collaborative Filtering for Implicit Feedback Datasets,“ IEEE, Pisa, Italy, 2008.

[9] J. Bobadilla, F. Ortega, A. Hernando ja A. Gutierrez, „Recommender systems survey,“ %1 *Knowledge-Based Systems,* 2013, pp. 109-132.

[10] J. Bauer ja A. Nanopoulos, „Recommender systems based on quantative implicit customer feedback,“ Elsevier B.V, Ingolstadt, 2014.

[11] H. Al-bashiri, M. A. Abdulgabber, H. Kahtan and A. Romli, “An improved memory-based collaborative filtering method based on the TOPSIS,” Education University of Hong Kong, Hong Kong, 2018.

[12] P. Melville ja V. Sindhwani, „Recommender Systems,“ IVM T.J. Watson Research Center, Yorktown Heights, 2009.

[13] R. Salakhutdinov ja A. Mnih, „Probabilistic Matrix Factorization,“ University of Toronto, Toronto, 2007.

[14] Y. Koren, R. Bell ja C. Volinsky, „Matrix factorization tehniques for recommender systems,“ IEEE Computer Society, 2009.

[15] S. Rendle, C. Freudenthaler, Z. Gantner ja L. Schmidt-Thieme, „BPR: Bayesian Personalized Ranking from Implicit Feedback,“ UAI, Hildesheim, Germany, 2009.

[16] M. Hahsler, „recommenderlab: A Framework for Developing and Testing Recommendation Algorithms,“ SMU, 2015.

[17] R. Zadeh, „Matrix Completion via Alternating Least Squares(ALS),“ %1 *CME 323: Distributed Algorithms and Optimization, Spring 2015*, 2015.

[18] M. Viljanen ja T. Pahikkala, „New Recommendation Algorithm for Implicit Data Motivated by the Multivariate Normal Distribution,“ University of Turku, Turku, Finland, 2020.

[19] Z. Meng, R. McCreadie, C. Macdonald ja I. Ounis, „Exploring Data Splitting Strategies for the Evaluation of Recommendation Models,“ ACM, New York, 2020.

[20] Google, „Classification: ROC Curve and AUC,“ 02 05 2021. [Võrgumaterjal]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[21] A. P. Bradley, „The use of the area under the ROC curve in the evaluation of machine learning algorithms,“ %1 *Pattern Recognition, Volume 30, Issue 7*, 1997, pp. 1145-1159.

[22] J. Holländer, „Investigating the performance of matrix factorization tehniques applied on purchase data for recommendation purposes,“ Malmö University, Malmö, 2015.

[23] B. Fredrickson, „Implicit,“ [Võrgumaterjal]. Available: https://implicit.readthedocs.io/en/latest/index.html. [Kasutatud 05 05 2021].

[24] L. R. a. B. S. Francesco Ricci, Introduction to Recommender Systems Handbook, Springer Science+Business Media, 2011.

[25] G. Guo, Z. Tan, Y. Liu, J. Ma ja X. Wang, „Resolving Data Sparsity by Multi-type Auxiliary Implicit Feedback for Recommender Systems,“ %1 *Knowledge-Based Systems*, https://doi.org/10.1016/j.knosys.2017.10.005, 2017, pp. 202-207.

[26] M. Jahrer ja A. Töscher, „Collaborative Filtering Ensemble for Ranking,“ Köflach, Austria, 2012.

[27] C. A. Levinas, „An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals,“ 2014.

[28] P. Lopes ja B. Roy, „Dynamic Recommendation System Using Web Usage Mining for E-commerce Users,“ Department of Computer Engineering, St. Francis Institute of Technology, Mumbai, India, 2015.

[29] F. Ricci, L. Rokach, B. Shapira and P. B. Kantor, Recommender Systems Handbook, New York: Springer, 2011.

Appendix A: Glossary

ALS – Alternating Least Squares

BPR – Bayesian Personalized Ranking

CF – Collaborative Filtering

CBF – Content-based Filtering

MF – Matrix Factorization

NMF/NNMF – Non-negative Matrix Factorization

SVD – Singular Value Decomposition

MAP – Mean Average Precision

ROC – Receiver Operating Characteristic

AUC – Area Under ROC Curve

TPR – True Positive Rate

FPR - False Positive Rate

TNR - True Negative Rate

FNR – False Negative Rate

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Kristjan Lõhmus,

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose "Collaborative filtering recommendation algorithms performance on an implicit feedback dataset" , mille juhendaja on Anna Aljanaki, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kristjan Lõhmus
07.05.2021