Maastricht University

Research project
Final report

Recommender System for Board Games

Authors
Caner Şahinli
Florian Debrauwer
Luca Brugaletta
Taavi Martoja
Vikash Mishra
Yannick Ruppenthal

Supervisor
Cameron Browne

June 24, 2020

Contents

1 Abstract

In this project, several different recommender system models were built based on the downloaded data from BoardGameGeek(BGG), which is a board game database that holds valuable information for this work about both users and board games. Three different sequential recommendation models were implemented and the results compare with each other and general recommendation collaborative filtering models. The sequential models are called BERT4Rec, Caser and DejaVu. To make the predictions for LUDII, a game system that is hosted by Maastricht University, the trained models are exported to the ONNX standard. The generated ONNX models can be then executed with the ONNX Runtime in Java. The implemented models had little success in making recommendations, the results were lower than initially expected. Within the top 5 or 10 recommendations, we usually find only a few relevant new suggestions. The complicated sequential models do not perform significantly better than the simpler general recommendation models.

2 Context

To give a good perspective, this chapter will focus on specifying the domain of the project as well as the state of art methods and our clear motivation detailing what we were aiming for.

2.1 Recommender System

Recommender Systems are software tools and techniques that are used for providing suggestions to users. The suggestions try to influence the user's decisions such as what products to purchase, what music to listen to or which movies to watch. A general term "item" is used to denote the product that is recommended e.g. a movie or a book. The recommendations are personalized, which means that different users receive different suggestions. Often, personalized suggestions are combined with non-personalized recommendations, such as top 10 items to provide more diverse suggestions. Recommender systems have proven to be very useful in online environments, in which there are overwhelming number of alternative items to offer. The recommender systems are beneficial to both service providers and users, as they connect the demand to the supply [15].

Most recommender systems try to predict a user's rating to an item, based on known preferences. The known preferences are often represented as a user ratings matrix, which shows the ratings that users have given for items. An example of the matrix is shown in figure 1. The most common approaches to the problem are Collaborative Filtering and Content-Based Filtering.

Collaborative filtering relies only on historical interactions to predict future ratings. The system does not consider the actual attributes of users or items.

		Items					
		1	*2*	...	*i*	...	*m*
	1	5	3		1	2	
	2		2				4
Users	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		
	a	3	5		?	1	

Figure 1: Example of User Ratings Matrix [10]

The predictions are made only on the basis of previous ratings from a collection of users. This gives the name collaborative. The intuition of the system is that if two users have similar rating history if one of them has rated one item highly, then the other user has a high probability to also give a high rating to the item.

Content-based filtering is based on profile attributes. The system treats each user as an atomic unit. Content-based filtering recommends items that are similar in content to items the user has rated highly in the past. This approach requires profiling of items, e.g. categorizing or valuing some aspects. Most modern approaches use a combination of the previous systems. Additionally, deep learning is can be included in the systems to improve the performance of all of the approaches. [10]

The recommender systems face a couple of problems that need to be taken into consideration. The cold-start problem refers to a situation, where a recommender does not have enough information about a user or an item to make meaningful suggestions. Often occurs when a new item is added or a new user is created. Data sparsity problem occurs when only very few items are rated by users. This leads to a sparse user ratings matrix and finally to weak recommendations. Another issue is scalability. A technique that provides relevant suggestions on a small dataset may fail when the volume of data is increased. [7]

2.2 BoardGameGeek

BoardGameGeek (BGG) is the largest online board game database, it is a collection of data and information on traditional board games. All of the information on the site has been entered by its enormous user-base.[4] The users can provide reviews, ratings, comments, etc, and all of the data is freely accessible via its public API. The website has more than 2 million registered users and over 100,000 games, making it the most comprehensive database for board games. The data that BGG provides can be used for building a recommender system for board games.[2]

2.2.1 Dataset

To make recommendation, we used the ratings that BGG users have given to the boardgames, in a sequential order. BGG offers a XML API [3] to extract information and we had access to an already collected dataset [1]. As the BGG API has quite a few access limitations, we opted to use the uploaded dataset. For example, it is only possible to query a maximum on 100 ratings at a time for a game. As some games have more than 100000 ratings, the amount of queries would be huge already. It is even worse, as we are limited to only about 2 or 3 queries per second. We can query multiple games, at the same time, but then the response time of the query starts increasing. Another option would be to make queries for users, to get ratings for each user. The limitations here are that, there is no easy way to get all usernames, that are required for the queries and we can only query one user at a time. And again only few queries in a second. Due to these limitations we used the existing dataset. The dataset is huge and it caused some difficulties to access it but we still found it a better option. The loaded dataset was roughly 12 GB. It consists of:

- 513.377 unique users
- 104.343 unique games
- 47.573.416 user interactions with games.

First we chunked the interactions and reformatted the data. There were 2 resulting csv files.

- games: game_id(int), games_name(string)
- ratings: user_id(int), game_id(int), rating(int), timestamp(int)

2.3 Ludii

Ludii is a General Game System, that is able to model and play a wide range of board games. The games can be played by a human or AI. The goal of Ludii is to provide a useful tool for researchers in areas including AI, design, history, and education. It is built on a ludeme library. Ludeme is defined as a conceptual unit of game-related information. A ludeme can be used to describe the games equipment, rules, and behaviour through play. The ludemes make up the "DNA" of each game. Through the ludemic model, it is simple to describe new games, while the syntax of ludemes makes the rules of the game clear to people who only access the ludeme. With ludemes, it is, in theory, possible to describe any board game, giving the Ludii System the important generality trait. For that reason, the Ludii System is a good option for researchers in many fields.[14]

2.3.1 Dataset

Ludii's database is very small in comparison with the BGG database. In fact it includes 534 games. The challenge using the dataset from Ludii was matching games from Ludii portal to BGG. After lowering and removing all the irrelevant chars, we had to handle the multiple reference for one single name, We solved the problem using string comparing. This process produced 231 matches. It follows that, the recommendation that we are able to grant are only among those games. Besides, even the input sequence accept only matched games.

2.4 Research questions

Here, the general aim of our project is outlined with the questions that we are going to explore.

- Can we utilize BGG data reliably to determine user preferences?
- Do higher-complexity models perform significantly well than others so that building them is worth the effort?
- Is there a major difference in the evaluation of models that are based on general and sequential recommendation?
- What is the most reliable evaluation metric in recommender systems?

2.5 Social impact

Recommender systems have made a massive contribution to user experience on online public platforms as we see it in the cases of Netflix, YouTube, etc. They are the perfect tool to customize the content of every single individual and thus makes the applications much more enjoyable for users. By comparing different recommender models in terms of various metrics, we look forward to pave the way for future developers that is looking for the best model based on their circumstances so that they create a more satisfying environment for their users, especially in the gaming industry, which still shows a lot of potential despite the enormous growth in the last decade.

3 Concepts

In this section, we will review general concepts such as general recommendations and sequential recommendations.

3.1 General recommendation

As mentioned before, Collaborative filtering or Content-Based Filtering are typical starting points to model users' preferences based on a set of information. Those models use this general information without taking into account the time component. It means that those models are not able to consider the order in the user's behaviour. In 2007, Salakhutdinov et al.[16] proposed to use deep learning to enhance the recommendations, taking advantage of a two-layer Restricted Boltzmann Machine. This breakthrough got to win the Netflix Prize that year. Since then, a lot of different deep learning approaches have been proposed, from Neural collaborative filtering to Auto-encoders.

3.2 Sequential recommendation

Sequential recommendation considers the problem of advising an item to the user, given the sequence of items of this user. Early work on sequential recommendation leverage Markov Chains and Markov Decision Processes. More recently, attention shifted toward deep learning approaches because of their representation and generalization ability for recommender systems. First, a lot of work has been done in implementing networks such as recurrent neural network and their variants such as GRU[6] and LSTM. Another approach consists of taking advantage of the success of convolutional neural networks in image recognition by modelling sequential patterns through the convolution operation[19]. Attention Mechanism[8] has been added to sequential recommender systems as they selectively focus on part of the input sequence.

4 Models

The first task ahead of us was preprocessing the downloaded data from BGG to build the recommender systems mentioned above. Thanks to the transparency of our colleagues who wrote the corresponding papers for the techniques used in this project, we had a solid background to build them and apply in our case. Based on some criteria or metric, we were able to rate and rank these models with respect to one another.

The recommendation algorithms that were used in this project are mostly sequential so that changing user preferences can also be structured. BERT4Rec, Caser and Déjà vu are the sequential models that we will examine whereas Collaborative Filtering is the only general recommendation system in this project.

4.1 BERT4Rec

BERT4Rec stands for Sequential Recommendation with Bidirectional Encoder Representation from Transformer[18]. It is an adaptation of BERT[5], a groundbreaking model in Natural Language Processing. In general, sequential deep learning approaches such as RNN encode user's historical interaction from old items to recent one into the hidden representation to make recommendations. Such unidirectional models are sub-optimal according to Fei Sun et al.[18] as it restricts the power of hidden representation in users' behaviour sequence and they often assume a rigidly ordered sequence which is not always the case in practice. BERT4Rec address these limitations by leveraging bidirectional self-attention mechanism [18].

4.2 Caser

Caser stands for Convolutional Sequence Embedding Recommendation Model[19]. This network aims to capture both general preferences and sequential pattern trough horizontal and vertical convolutions. The embedding matrix is regarded as an image containing n previous items. This paper originally addresses two problems encountered in previous work. On the first hand, some models such as the Markov chain model fail to model union level sequential pattern, meaning that each of the previous items can only influence individually the target item, and not collectively. On the other hand, some existing model fails to allow skip behaviours of sequential pattern where the impact from the past behaviours may skip a few steps and still have strength[19].

4.3 Déjà vu

This recommender algorithm is specifically designed to detect temporal and contextual information rather than the transitional structure in sequential input data[21]. It is proposed that a user's action changes over time so a sequential model should also take how and when a user performed an action into consideration besides the context of the action. With its self-attention mechanism, it is also a great choice in terms of computational efficiency due to reduced number of parameters. In the paper[21], the various advantages of the model over its alternatives are examined.

4.4 Collaborative Filtering

As mentioned in the context section, collaborative filtering relies on the historical interactions between users and games to predict the future rating. The predictions are made on the basis of previous ratings from a collection of users. This model will constitute the baseline to validate the recommendations given by more complex approaches. As this model makes its predictions based on the similarities between user and games. The intuition is that similar people will have similar taste. Collaborative filtering could be item based or user based. This has an impact on the performance of the model. We thus trained the ten most promising algorithms on a subset of BGG dataset and cross-validated their performance. The candidates were SVD, SLopeOne, Normal Predictor, KNN Baseline, KNN Basic, KNN with means, KNN with score, Baseline only and co-clustering. KNN approaches, and especially KNN with means (user based), were the best performing. KNN with means is a basic collaborative filtering algorithm that take into account the mean rating of each user. The major hyperparameter to tune is the similarity measure. We picked cosine similarity between users empirically. The model managed to reach a precision@5 of 79 percent and a recall@5 of 65 percent.

5 Evaluation metrics

In this section, the above-presented models were evaluated and compared to decide which one was the most accurate and efficient. In the first implementation, the metrics used for the evaluation were the well-known measures of precision and recall [17]. In recommender systems, we need to provide different recommended items to the user. Using a value k, recommendations are calculated taking into account only the first k items instead of all of them. Our evaluation metrics were calculated using 1,5 and 10 items.
Precision and recall are defined as follows:

- Precision or true positive accuracy is calculated as the ratio of recommended items that are relevant to the total number of recommended items:

  $$\text{precision} = \text{TP} \ / \ (\text{TP} + \text{FP})$$

  Precision is the fraction of all positive ratings that were correctly classified as such. It measures how good the system is at recognizing positive recommendations

- Recall or true positive rate (also called sensitivity in psychology) is calculated as the ratio of recommended items that are relevant to the total number of relevant items:

  $$\text{recall} = \text{TP}/(\text{TP} + \text{FN})$$

  Recall is the fraction of all positive recommendations that are actually positive. It measures how good the system is at finding positive recommendations [11]

In the last phase of the project and to compare the models we decided to use three different metrics and these are:

- Mean average precision
- Mean average recall
- Normalized discounted cumulative gain

All of these metrics, in the same way as recall and precision, can be calculated at cutoff k. The adopted **k** for this final phase are @3, @5 and @10.

5.1 Mean average precision

Mean average precision (mAP@k) is a popular metric for search engines. It takes each relevant item and calculates the precision of the recommendation set with the size that corresponds to the rank of the relevant item. Then the arithmetic mean of all these precisions is formed. Afterwards, we calculate the arithmetic mean of the average precisions of all users to get the final mean average precision. [17]. Its purpose is to give insight on how relevant is the list of recommended games.

5.2 Mean average recall

Mean average recall (mAR@k) is a metric that considers the order of recommendations, and penalizes correct recommendations if based on the order of the recommendations. mAR as mAP are ideal for evaluating an ordered list of recommendations.[9] It gives insight into how well the recommender is able to recall all the items the user has rated positively in the test set.

5.3 Normalized discounted cumulative gain

Normalized Discounted Cumulative Gain (NDCG@k) is a family of ranking measures widely used in applications. NDCG has two advantages compared to many other measures. First, NDCG allows each retrieved document has graded relevance while most traditional ranking measures only allow binary relevance. That is, each document is viewed as either relevant or not relevant by previous ranking measures, while there can be degrees of relevancy for documents in NDCG. Second, NDCG involves a discount function over the rank while many other measures uniformly weight all positions [20].

6 Ludii implementation

This section explains the steps necessary to get recommendations for Ludii games or BGG games. In regard to the fact that Ludii is implemented in Java and our sequential models are made in PyTorch, we had to export these models to the ONNX standard. The generated ONNX model can then be executed through the inferencing API from ONNX-Runtime.

6.1 ONNX

ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers[12]. ONNX supports a wide variety of frameworks like PyTorch, TensorFlow, Caffe 2, Matlab, MXNet and many more.

6.2 ONNX-Runtime

ONNX-Runtime is a cross-platform inferencing and training accelerator compatible with many popular ML/DNN frameworks, including PyTorch, TensorFlow/Keras, scikit-learn, and more. [13]. It is developed from Microsoft. It works on on programming languages that are a lot faster than python for example C++, C# and Java. The java implementation runs the ONNX models in C++ and manipulates those with JNI Java bindings.

6.3 Approach

To realize the implementation for Ludii we created a Recommender class and an abstract Models class, as can be seen in the UML class diagram below. The Recommender class is responsible for loading the correct abstract Models class and takes care of resolving the different IDs the models used to the standard BGG IDs and Ludii IDs. Furthermore, the IDs can be resolved to the actual name of the game. The matching of Ludii IDs to BGG IDs has been in explained in section 2.3.1. Finally, It also handles the task of calling the correct functions of the desired Model. The abstract models class has an implemented function to load the correct ONNX model and 3 abstract functions that are implemented in the specific model classes. One for each model, BERT, DejaVu, Caser. The first function is prepareInput(sequences) which transforms the sequences in the correct format so that it can be fed into the model. The second function is recommend(input) which performs the actual inference. The third function is prepareOutput(result, ludii only) which takes care of extracting the necessary information from the result. When we talk about the interface of our models, we implemented an interface with three main methods. With the boolean value Ludii, only the resulting recommendation will be chosen only from the Ludii subset that has been matched to the BGG games. Firstly, Result Recommend (input) is the method that runs the actual interface. To get actual recommendation the Recommender class has to be initialized with an enum called "MODEL" which represents one of the 3 recommender models we implemented. Then the recommend() function needs to be called, with the IDs of games played in a list as a list. The results will be a list of 10 recommendations.

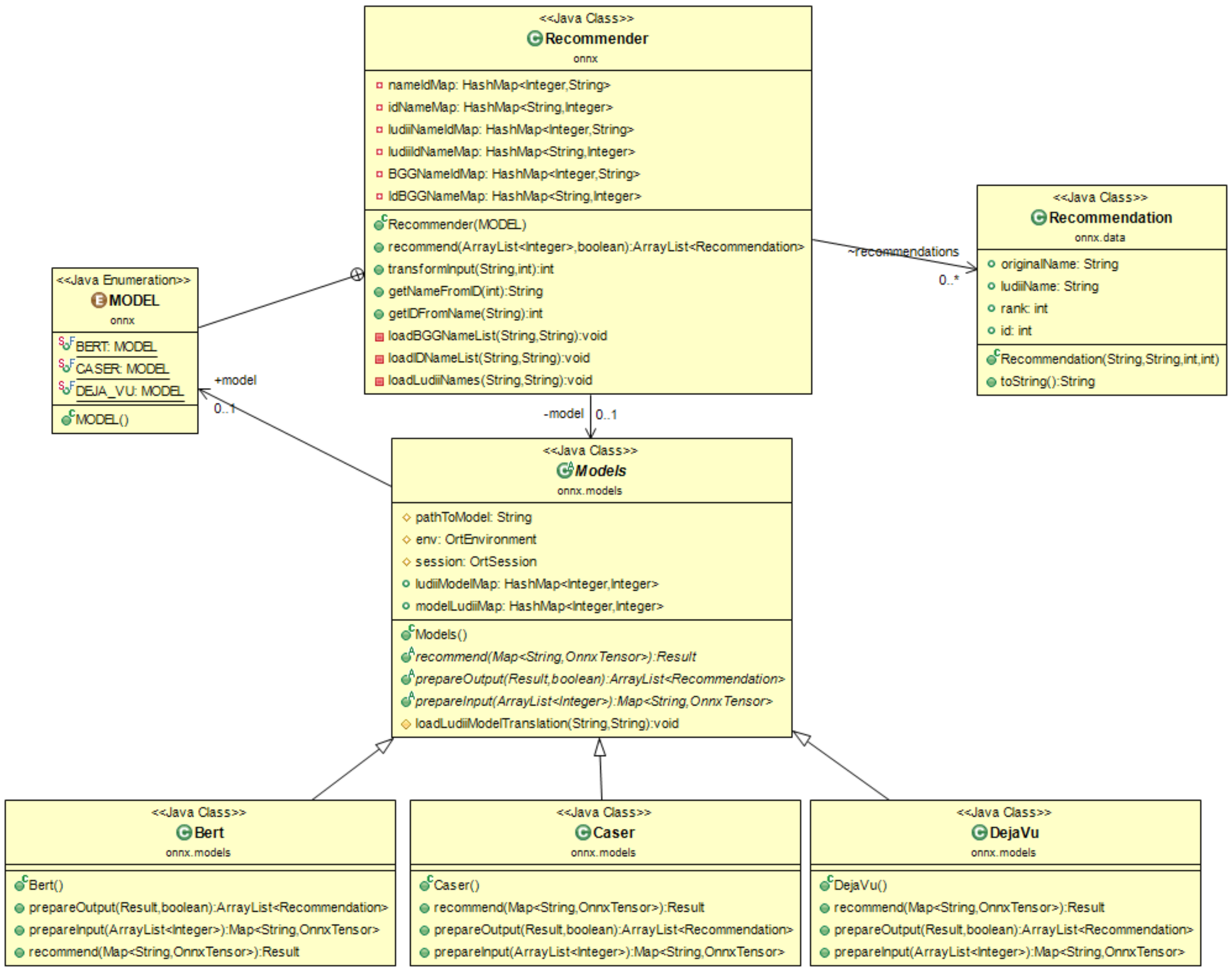
<<Java Class>>
Recommender
onnx
nameIdMap: HashMap<Integer,String>
idNameMap: HashMap<String,Integer>
ludiiNameIdMap: HashMap<Integer,String>
ludiiIdNameMap: HashMap<String,Integer>
BGGNameIdMap: HashMap<Integer,String>
IdBGGNameMap: HashMap<String,Integer>
Recommender(MODEL)
recommend(ArrayList<Integer>,boolean):ArrayList<Recommendation>
transformInput(String,int):int
getNameFromID(int):String
getIDFromName(String):int
loadBGGNameList(String,String):void
loadIDNameList(String,String):void
loadLudiiNames(String,String):void
<<Java Class>>
Recommendation
onnx.data
originalName: String
ludiiName: String
rank: int
id: int
Recommendation(String,String,int,int)
toString():String
~recommendations
0..*
<<Java Enumeration>>
MODEL
onnx
BERT: MODEL
CASER: MODEL
DEJA_VU: MODEL
MODEL()
+model
0..1
-model
0..1
<<Java Class>>
Models
onnx.models
pathToModel: String
env: OrtEnvironment
session: OrtSession
ludiiModelMap: HashMap<Integer,Integer>
modelLudiiMap: HashMap<Integer,Integer>
Models()
recommend(Map<String,OnnxTensor>):Result
prepareOutput(Result,boolean):ArrayList<Recommendation>
prepareInput(ArrayList<Integer>):Map<String,OnnxTensor>
loadLudiiModelTranslation(String,String):void
<<Java Class>>
Bert
onnx.models
Bert()
prepareOutput(Result,boolean):ArrayList<Recommendation>
prepareInput(ArrayList<Integer>):Map<String,OnnxTensor>
recommend(Map<String,OnnxTensor>):Result
<<Java Class>>
Caser
onnx.models
Caser()
recommend(Map<String,OnnxTensor>):Result
prepareOutput(Result,boolean):ArrayList<Recommendation>
prepareInput(ArrayList<Integer>):Map<String,OnnxTensor>
<<Java Class>>
DejaVu
onnx.models
DejaVu()
recommend(Map<String,OnnxTensor>):Result
prepareOutput(Result,boolean):ArrayList<Recommendation>
prepareInput(ArrayList<Integer>):Map<String,OnnxTensor>

Figure 2: UML class diagram

7 Training

7.1 BERT4Rec

The initial batch size of the model was 128, but due to computing power limitations, we had to reduce it to 8. We used Adam optimizer and a learning rate of 0.001 for 30 epochs. A dropout rate of 10 per cent was used. The architecture has 256 hidden units in the fully connected layers and 4 attention heads. To train BERT4Rec we predict 15 per cent of the games in the training set that are randomly masked. Each sequence has a maximum length of 100. All those hyper-parameter were chosen empirically.

7.2 DejaVu

The training hyperparameters for the DejaVu model were kept the same as in the original paper, only the batch size was increased to train the model within a shorter amount of time. The model was trained for 20 epochs with an initial learning rate of 0.001 using the Adam optimizer. The model uses a small dropout rate of 10%. The architecture uses 128 hidden units in fully-connected layers and 2 attention heads for the multi-head attention mechanism. The BGG data was divided into 90% training and 10% testing datasets.

7.3 Caser

After conducting several experiments to find the optimal hyperparameter selection and considering the size of our database, the final model is trained over 30 epochs with a batch size of 1024 and a learning rate of 0.0001 on GPU. The activation functions for both convolutional and fully-connected layers are set to be rectified linear units (ReLU). Also, the dropout rate on the fully-connected layers is chosen as 0.5 to overcome overfitting. Model's regular parameters are optimized based on the objective function to minimize loss whereas hyperparameters are tuned according to the validation data via grid search just like original paper [19]. Also, the optimizer is the same that is the Adam optimizer for faster convergence.

8 Results

We can distinguish two types of results obtained from each model. The first type of results are metrics used for the evaluation and can be seen on the three tables below. Each table refers to a different metric and each row contains a different model. Meanwhile, columns contain the value of k, used to decide the subset length of the recommendations. The second type of results are lists of recommended games utilizing the same game sequences for each model.

8.1 Evaluation metrics results

	MAP@3	**MAP@5**	**MAP@10**
BERT	0.2902	0.1840	0.0960
Dejavu	0.0324	0.0447	0.0593
Caser	0.2700	0.2230	0.1913

Table 1: Evaluation Results for Mean Average Precision

	MAR@3	**MAR@5**	**MAR@10**
BERT	0.5010	0.6110	0.7410
Dejavu	0.0355	0.0490	0.0650
Caser	0.9111	0.7781	0.6300

Table 2: Evaluation Results for Mean Average Recall

	NDCG@3	**NDCG@5**	**NDCG@10**
BERT	0.8020	0.8210	0.834
Dejavu	0.0288	0.0344	0.0396
Caser	0.7140	0.7360	0.7520

Table 3: Evaluation Results Normalized Discounted Cumulative Gain

8.2 Recommended games results

To recommend board games we selected four main categories and we choose four different games per head. The results obtained are lists formed by 3 recommended games.

8.2.1 Categories and Initial Sequences

1. **Math**
   - Leaving Earth - 173064
   - My Little Scythe - 226320
   - Sleeping Queens - 17053
   - Import / Export - 217776
2. **Action / Dexterity**
   - Crokinole - 521
   - KLASK - 165722
   - PitchCar - 150
   - Junk Art - 193042
3. **Fantasy**
   - Gloomhaven - 174430
   - War of the Ring (Second Edition) - 115746
   - Spirit Island - 162886
   - Terra Mystica - 120677

8.2.2 Models and recommendation

Model	**Math** - top three
BERT	Dust Tactics: Allied Fortification – Field Phaser Bunker / Strongpoint Questor Expansion Sherlock Holmes Détective Conseil: L'Homme Sans Visage
Caser	Kirdy Roc Bondegårdsspillet
DejaVu	Import / Export Leaving Earth: Mercury Leaving Earth: Outer Planets

Table 4: Math category recommendations

Model	**Fantasy** - top three
BERT	Draconian Wars Pola Naftowe Ninja All-Stars: Bakusho Mondai
Caser	Robotech CCG Finest Hour Sword & Sorcery: Onamor and Volkor
DejaVu	Miss Lupun…und das Geheimnis der Zahlen KLASK Canadian Trivia Family Edition

Table 5: Fantasy category recommendations

Model	**Action / Dexterity** - top three
BERT	kNOW! Eat Thyself Sherlock Relic Knights: Noh Empire Battle Box
Caser	Abyss: Leviathan Ninja All-Stars: Arashikage Jutland
DejaVu	Asalto Eclipse: Second Dawn for the Galaxy Dungeons & Dragons: Waterdeep – Dungeon of the Mad Mage

Table 6: Action / Dexterity category recommendations

8.3 Collaborative Filtering recommendations

The last model implemented was Collaborative Filtering. We were able to obtain recommendation based only on existing users. Thus, the game played by the users have not been selected by us but are just taking into account the history of existing users. Following there are the users with relative games:

- User1(14532): Rise of Augustus, Reef, Acquire, Shakespeare
- User2(62924): Dungeon Petz, Eminent Domain: Battlecruisers, The Pillars of the Earth, Ogre: Vulcans & Friends Counter Sheets
- User3(54778): Star Wars: X-Wing Miniatures Game, Photo Party, Monopoly

8.3.1 CF results

IdUser	top three recomendations
14542	Hare & Tortoise Gracias The Great Invasion: The Gettysburg Campaign June 24 – July 3, 1863
62924	Shooting the Moon Sackwas Monopoly: UPS
54778	Loaded Questions: Junior Edition Wildscape YoYo

Table 7: CF user recommendations

9 Discussion

BERT4Rec results are lower than expected. The recall of the model is quite high, however, the precision at different k is above simple similarity-based approaches. Our hypothesis is that the training was prematurely stopped. Indeed, BERT4Rec is a relatively complex model that would have required to train roughly a months on the hardware used. Around 30 per cent of the recommended games are considered as relevant. However, the system still manages to recommend a fairly good amount of the ground truth relevant games.

When we take a look at the evaluation of the Caser model with the metrics applied in this project, it can be observed that the values that we have obtained indicate the presence of a reliable model. Especially for the maR@k, all values are above the threshold of 0.5 which means that model is correct more than half of the time. Similar performance also applies to the other metrics. However, when the model is given a sequential list of games with the same categories, we have seen that the model fared relatively poor compared to the other two models. There is a simple explanation for that as Caser is not a regular similarity detection system but it is especially designed to capture advanced features such as skip behaviours, long-term user preferences and union-level sequential patterns [19].

From the evaluation metrics of the DejaVu model, we can see that the model performed fairly badly. One reason for this is the specific problem setting for the model, as each sequence of ratings only has one correct following rating. As the action space for the ratings is huge, the probability of outputting exactly the correct rating for the correct game is very low. In this light, hitting the exact correct output in the top 10 results for 5% of the time might be considered a decent result. From the actual examples, we can see that the model often recommends the inputs that were given, which in a way, make sense, but does not add value to the model. Also, if we look at the top 5 recommendations, we often find some games, which have little relevance to the input games. The model does not perform better than simpler methods either, so using this complicated does not justify itself. In the original paper, the model was tested on datasets that were not based on ratings, which might cause the overall bad performance on BGG data.

10 Conclusions

During this project we tried to train and evaluate a variety of recommender system on the BGG dataset. We encountered several difficulties which could not all be resolved. It started with obtaining the data. Even though BGG provides an API, it is not meant for data mining. We found a dataset that was already collected for other recommender systems. The enormous size of the dataset proved very challenging to prepossess and we were only able to open chunked versions of it. Each of the 3 different models brought their own difficulties, but we ultimately managed to train them on Colab and the Aachen Cluster. As our knowledge in board games was very limited we had to rely on evaluation metrics to judge our models. A big challenge is that errors during the reprocessing and pretraining were only noticeable after a long training period. And it would have been necessary to start the training again. Even though in some aspects we achieved decent scores most of the predictions seemed off. DejaVu output the most logical recommendation, namely, the one that are the closest to the previously played games. As evaluation metrics NDCG@K seem to be the most relient one. With the results we can not conclude that complex models are worth the effort in comparison to a simple baseline model. We Implemented a Java class that can load ONNX models and do inference on them. With this way we can pick up the training of certain models, or create new ones and make them executable in Java.

References

[1] BoardGameGeek Dataset. `https://github.com/John-K92/Recommendation-Systems-for-BoardGame-Platforms`. Accessed: 2020-06-20.

[2] BoardGameGeek. `https://boardgamegeek.com`. Accessed: 2020-03-16.

[3] BoardGameGeek API. `https://boardgamegeek.com/wiki/page/BGG_XML_API2#`. Accessed: 2020-03-16.

[4] BoardGameGeek FAQ. `https://boardgamegeek.com/wiki/page/BoardGameGeek_FAQ`. Accessed: 2020-03-16.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[7] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 11 2015.

[8] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[9] recmetrics. `https://github.com/statisticianinstilettos/recmetrics`. Accessed: 2020-06-20.

[10] P. Melville and V. Sindhwani. *Recommender Systems*, pages 1056–1066. Springer US, Boston, MA, 2017.

[11] Sergey Morozov and Xiaohui Zhong. The evaluation of similarity metrics in collaborative filtering recommenders. In *Hawaii University International Conferences*, 2013.

[12] ONNX. `https://onnx.ai`. Accessed: 2020-06-20.

[13] ONNX Runtime. `https://github.com/microsoft/onnxruntime`. Accessed: 2020-06-20.

[14] Éric Piette, Dennis J. N. J. Soemers, Matthew Stephenson, Chiara F. Sironi, Mark H. M. Winands, and Cameron Browne. Ludii - the ludemic general game system. *European Conference on Artificial Intelligence (ECAI 2020)*, 2020.

[15] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, Boston, MA, 2011.

[16] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007.

[17] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Setting goals and choosing metrics for recommender system evaluations. In *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*, volume 23, page 53, 2011.

[18] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450, 2019.

[19] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018.

[20] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, volume 8, page 6, 2013.

[21] Jibang Wu, Renqin Cai, and Hongning Wang. D\'ej\a vu: A contextualized temporal attention mechanism for sequential recommendation. *arXiv preprint arXiv:2002.00741*, 2020.