Translation-based Sequential Recommendation for Video Game Reviews

Ishit Mehta
University of California San Diego
ibmehta@eng.ucsd.edu

Hertz He
University of California San Diego
z1he@ucsd.edu

Chengsheng Wu
University of California San Diego
c7wu@ucsd.edu

ABSTRACT

Recommending items to users based on the feedback of their previous interactions is a highly explored problem. Most conventional methods model the user-item interactions and item-item sequential interactions separately. This article, as an assignment report for the course *CSE 258* at UC San Diego, explores translation-based recommendation [4]. It models both type of interactions in a unified way, as third-order relationships. It projects items to a low-dimensional latent space, where the next item is recommended by translating the previous item by a user vector. On the Steam Video Game dataset, it improves upon a few previous methods according to several metrics.

1 INTRODUCTION

We all need suggestions from other people to make our lives easier, especially on things that we do not know much about or have difficulty in making a decision on. Benefited from today's fast advance in artificial intelligence, people are getting used to taking advice from computing machines. When we listen to music on Apple Music, , watch videos on YouTube, or make a purchase on Amazon.com, it is fairly common to see recommended items, be it music, video or product, that closely meet our needs. Behind all recommendations is a recommender system that learns your preferences by eating big data and utilising advanced learning algorithms.

A recommender system is to model and predict the interactions between users and items, sometimes also including the relationships among the items themselves. Conventional recommendation systems focus on the two-way interactions between users and items. This is also called pairwise interactions. Successful techniques include Matrix Factorization techniques (MF) [8], Markov Chain (MC) models [15]. MF technique uses a vector to describe a user/ item and uses their inner product to describe their compatibility. MC models focus on capturing the pair-wise relationship between adjacent items. Often, it factorizes the transition matrix so that it can well generalise.

To take the sequence of the interactions into account, third-order relationships between a user and two items have to be modeled. Factorized Personalized Markov Chains (FPMC) [13] is one such method. It combines a term generated from Matrix Factorization with an item-item transition term from a Markov Chain. So FPMC has the benefits from both MF and MC. It is able to capture both long-term trend and short-term transition properties at the same time. After the birth of FPMC, there are also many improments made to it. For example, people tried to replace the inner product with some distance metric, e.g. Euclidean distance [2]. Other people used deep learning methods to apply convolutional operations to extract transitions [16]. The former method still models user preference and item sequence separately, while the deep learning method only performs better than traditional methods when the dataset is dense.

In this assignment, we implement a model called Translation-based Recommendation (TransRec) [4] for doing sequential recommendation. This model absorbs the strong points from previously successful models. It models the items as embeddings in a 'transition space' and models each use as a 'translation vector' in the same space. The transition from an item i to another item j is captured by a third-order interaction. The compatibility between i, j with the same user is computed in a distance metric. More details on the Model section. We also implement a Bayesian Personalized Ranking algorithm as a baseline for comparison.

2 RELATED WORK

In this section we explore the landscape of general recommendation systems, systems which employ temporal recommendations and finally sequential recommendations.

General Recommendation Systems These systems try to model user and item relationships using recorded interactions between a sparse set of user-item pairs. These interactions could be *explicit*, like movies rated by users, or *implicit*, like movies titles clicked on by users. Modelling negative interactions in these cases is hard, since the interactions are not observed. Typically, these recommendation systems are related to Collaborative Filtering and Matrix Factorization (MF) [14]. These MF methods aim to project user-item interactions to a latent space with reduced number of dimensions, such that the reprojection error back to the input space is minimized. Most systems can be categorized into point-wise [6, 10] and pair-wise

methods [11, 12]. The point-wise methods assume that all non-observed interactions are negative interactions, for instance a user dislikes a movie if they have not rated/liked it. The pair-wise methods make a preferential assumption that a user prefers movies with which they have interacted over movies that they haven't interacted with. These methods however do not exploit the timing of the interactions. As learnt from the Netflix Prize Challenge [1], the time at which a user rated a movie is crucial.

Temporal Recommendation Systems One of the earliest methods which exploit the temporal dimension, TimeSVD++ [7] segments the interactions according the time period and then use matrix factorization for each of the time segments. As observed in the Netflix Challenge [1], the movie ratings drift temporally, *i.e.* the average rating of movie increases as it gets older. TransRec [4] and other sequential recommendation systems differ from these methods in terms of how they incorporate time. Temporal systems try to model the ratings v/s time patterns, in contrast to sequential recommenders which exploit simply the order in which a user interacts with the items.

Sequential Recommendation Systems These systems usually model the user-item iteractions as Markov Chains (MC) [2, 17]. Factorized Personalized Markov Chains [13] model item-item transitions as first-order Markov Chains, implicitly assuming that the next recommednation for a user highly depends on their last interaction. Higher-order MC based methods have also been proposed [3, 5]. We use TransRec [4] in this assignment, which models the third-order interactions between a user, item and next item. We discuss the methodology in detail in section 5.

3 DATA-SET ANALYSIS

For this assignment, we are interested in the Steam Video Game "Version 2: Review Data" [9] dataset extracted from Steam, one of the most popular online video game distribution platforms.

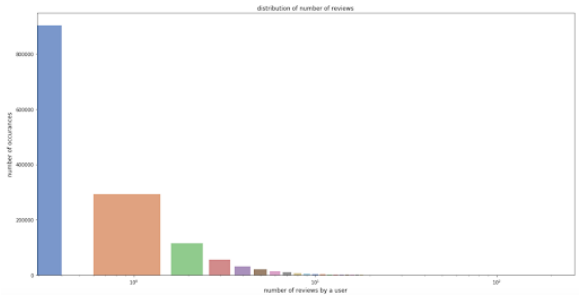
The dataset is fairly large, comprising of nearly 3 million text reviews. The reviews spans from 2010-10-15 to 2018-01-05, and also detailed information on user-item and user-platform interactions per review (*e.g.* whether the user early accessed the game, found it funny, play hours before review and number of products purchased). There are 342791 duplicate reviews in the dataset. A peek into the dataset after the cleaning and processing is shown in figure 1. The dataset contains 28255528 reviews, with 148323 unique users and 14448 unique games. On average, approximately one game has 219 reviews; one user rated 2 games, owned 155 products and played 111 hours before the review. 100% of the users recommended the game in the review (late found to be a mistake made during data crawling), 14.6 percent chose to early access the game while 14 percent found the game funny.

	username	user_id	product_id	date	hours	text	recommended	products	early_access	found_funny	year	month
1103439	user651233	76561198016995995	350740	2018-01-05	0.5	Nice Pang alternative.\nComing from Hollow Kni...	True	171	False	None	2018	1
2917995	pigmypanda	76561198196457970	730	2018-01-05	1253.4	This game is utterly fun, but toxic. If you ar...	True	13	False	None	2018	1
1180725	Mr_Tietje	76561198022080000	620	2018-01-05	4.1	great game, great story.	True	30	False	None	2018	1
2713867	[BONES] Knight	76561198006537081	383870	2018-01-05	4.3	Really well written story game.	True	59	False	None	2018	1
2713868	-{XMAS}- thebiggerboat3	76561198094023474	383870	2018-01-05	5.8	Found a turtle, made a friend out of him. good...	True	67	False	None	2018	1

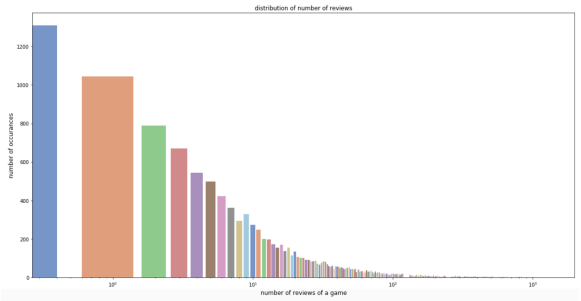
Figure 1: Sample set of records in the dataset

#actions	#users	#items	avg. actions/item	avg. actions/user	avg. products	avg. play hours	% recommended	% early access	% found funny
2825528	148323	14448	219.291182	2.136091	155.294396	111.3512	1.0	0.145987	0.141129

Figure 2: Various statistics



(a) Histogram of users



(b) Histogram of games

Figure 3: Sparsity of the dataset

The overall statistics are shown in figure 2.

Apart from the above basic statistical analysis, we also analyze the sparsity of the dataset. Figure 3 shows the histogram of the number of reviews by a user and the number of reviews of a game.

For the number of reviews by a user, it is surprising to see that nearly half of all users only review one game. The majority of the users review number of games ranging from 1 to 50. Users' devotion to the platform is quite astonishing

in the most extreme case. An individual user wrote 1253 reviews, played for 32146 hours and owned 12832 products. Most users wrote one review, played 0.2 hours on an average, owned 5 games and most games had one review. The number of reviews of a game is much higher compared with that of number of reviews per user. The most popular game has 76085 reviews. Comparing with other commonly used datasets, such as MovieLens, Amazon, the Steam dataset is denser than Amazon dataset but sparser than MovieLens dataset.

Given these analysis results, to do sequential recommendation, first we expect that a user has reviewed at least some number of games. Considering the distribution of number of reviews per user/game, we select the threshold to be 5, which means that we only care about users with more than 5 reviews or games with more than 5 users having reviewed it. For partitioning, we split the game sequence that a user has reviewed into three parts. The last item in the sequence for testing. The last but one in the sequence for validating. All remaining items are used for training.

4 PREDICTIVE TASK

As discussed earlier, the predictive task for this assignment is to do sequential recommendation given users-item interactions in history. Not only user-item interaction is included, but also we have to take item-item interactions to capture sequential trend. To be clearer, given a series of users actions$S_1^u, S_2^u, ..., S_{|S^u|}^u$ (S^u is the historical interaction sequence for user u), we are trying to predict the next item for this sequence. This could be very interesting in a sense that now we are taking time into consideration. People's behavior usually changes as time goes on. By capturing this dimension, we are making our recommendation time sensitive.

For this task, we will use three different types of metrics for evaluating our model performance, which are Hit@10, NDCG@10 and AUC. Hit@10 is abbreviation for Hit Rate at position 10. It counts the faction of times that the ground truth item is within the top 10 ranked items. NDCG is short for Normalized Discounted Cumulative Gain. It differs slightly from Hit@10 in that it is position aware. NDCG@10 would assign larger weights for items that ranked higher. AUC is short for Area Under the ROC Curve. ROC curve shows the relationship between TPR vs FPR under different threshold conditions. AUC is just the area under this curve. It represents the probability that a model would rank a random positive sample higher than a random negative sample. Ways to compute the three metric:

$$AUC = \frac{1}{|\mu|} \sum_{u \in \mu} \frac{1}{|I \ not \ in \ S^u|} \sum_{j' \in |I \ not \ in \ S^u|} 1(R_{u,g_u} < R_{u,j'})$$

$$Hit@10 = \frac{1}{|\mu|} \sum_{u \in \mu} 1(R_{u,g_u} < 10)$$

$$NDCG@10 = \frac{1}{|\mu|} \sum_{u \in \mu} \frac{1}{log_2(R_{u,g_u} + 2)} (R_{u,g_u} < 10)$$

where g_u is the ground truth item associated with user u. $R_{u,i}$ is the rank of item i for user u. 1() is an indicator function that returns 1 of the argument in the parenthesis is true or 0 otherwise. S^u is the item sequence for user u. μ is the set of all users. I is the set of all items.

Two baseline models are used here for comparison. They are Bayesian Personalized Ranking and Latent Factor Model. These two models could serve as baseline since they are not designed for modeling sequential information but prove to be strong in general recommendation. However, as denoted above, tranRec is able to model item-item transition, this kind of property makes it suitable for sequential recommendation task.

For BPR and LFM, the features we use are just user-item interactions, ignoring item sequence information. We use latent factors for describing each user and item. For TransRec, besides using just user-item interactions, we also extract the item sequence for each user and sort the items increasingly with timestamp. For detailed information on how we implement the user and item features, please refer to the next section.

5 MODEL

In this section we describe our solution which is based on [4]. The sequence prediction task can be formulated as follows: given a user u and their sequential history of interactions with items i, predict their next interaction. The users belong to a pre-defined set $\mathcal{U}$ and similarly the items belong to a set $\mathcal{I}$. The sequence of interactions can be defined as an ordered set: $S_u = (S_u^1, S_u^2, S_u^3, \ldots, S_u^k)$ for a given user u with k interactions.

Unlike other models (FPMC [13] and MF [14]), TransRec jointly models the third-order interactions between the user, items and next item. Inspired by [4], we project the item and user feature vectors on a high-dimensional Euclidean space where the triangle inequality is preserved. In this latent space, every user and their interactions are defined on a vector. The direction of this vector is defined uniquely for every user and the elements in the sequence of interactions define the magnitude. The magnitude of these interactions grow monotonically and are equidistant. If a user u represented using t_u, interacts with an item represented using γ_j after γ_i, then $||\gamma_i|| < ||\gamma_j||$ and:

$$\gamma_i + t_u \approx \gamma_j$$

Note the approximation in the above equation. At inference time, $\gamma_k + t_u$ is estimated and the next prediction is obtained

using a nearest neighbours algorithm in the latent space. The distance metric used in this space hence requires that triangle inequality holds. We use $L1$ as the distance function d.

To train the model's parameters γ and t, we wish to optimize the following objective:

$$\max_{\Theta} \sum_{u} \sum_{i} \sum_{j'} ||(\beta_j - d(\gamma_i + t_u, \gamma_j)) - (\beta_{j'} - d(\gamma_i + t_u, \gamma_{j'}))||$$

Here, item i precedes item j in the sequence of interactions for user u and item j' is a negative item which has not interactions with the user. We capture the popularity of items using bias terms β_i for every item. Additionally, we use scalar T which acts as the bias for users. This especially is useful for users with minimal to no interactions. We tweak the objective with these changes, add regularization for the parameters and maximize the log-likelihood for the given parameter set $\Theta = \{\gamma, t, T, \beta\}$:

$$\Theta^* = argmax_{\Theta} \sum_{u} \sum_{i} \sum_{j'} \log \sigma(d_{uij} - d_{uij'}) - \lambda \sum_{\theta \in \Theta} ||\theta||_2^2,$$

where $d_{uij} = \beta_j - d(\gamma_i + T + t_u, \gamma_j)$.

We train the model using stochastic gradient descent. At every step, a sample is generated by randomly sampling users and their positive and negative interactions. The parameters of the the model Θ, are updated with a learning rate ϵ. By using a L2 regularization and carefully tune the regularization term λ_θ, our model scales well with increasing size of traning data and does not exibit overfitting on the validation set.

$$\theta = \theta + \epsilon.(\sigma(d_{uij'} - d_{uij}) \frac{\partial(d_{uij'} - d_{uij})}{\partial\theta} - \lambda_\theta.\theta)$$

For our baseline models BPR and LFM, we use latent factors to model user/ item. We discuss the two models together.

We use γ_u to denote user vector and γ_i and $\gamma_j$to denote item vector. The primary difference between BPR and LFM is their objective. The objective for BPR is:

$$max\ ln\sigma(\gamma_u.\gamma_i - \gamma_u.\gamma_j)$$

where i represent an item that the user interacts with and j is not. However, for LFM, the objective is usually:

$$min \sum_{u,i} (\gamma_u.\gamma_i - R_{u,i})^2$$

For simplicity, here we don't write the regularization term. For parameter estimation, here we use stochastic gradient descent method for BPR and alternating least squares for LFM.

6 EVALUATION

After doing grid search using different parameters on the validation set, we find the best parameter combinations for each model. But to make the comparison fair, we use the same vector dimension for the latent factors, which is set to be 40. The best parameter combination for learning rate, regularization term is as below:

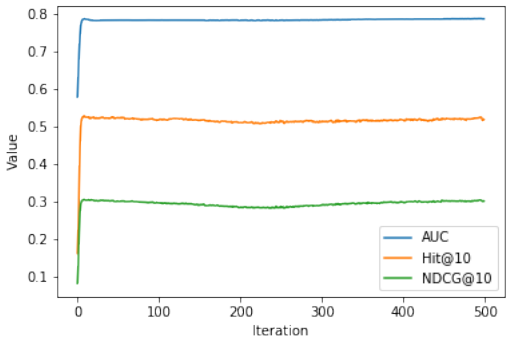
	Learning Rate	Regularization Term
transRec	0.001	0.01
BPR	0.1	0.1
LFM	–(ALS)	0.01

The learning curve of all three models are shown in figure 4. All three models learn the optimal parameter quite fast. After 10 iterations, the metrics do not improve much. For transRec and BPR, they do not overfit when the metrics is steady. This is probably due to the fact that in out implementation, during training we randomly sample negative items from all negative ones except the ground truth. By adding this kind of randomness, the model won't overfit on the dataset.

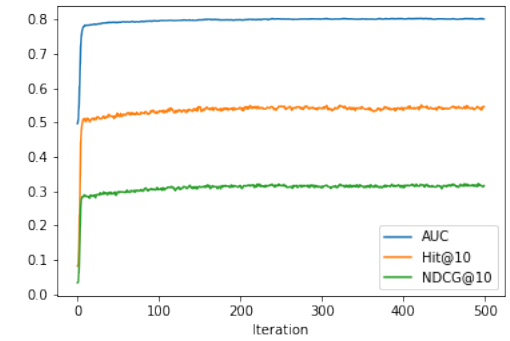
We also show the optimal performance of each model using a bar chart (Figure 5). We can see that BPR performs the best across all three metrics. TransRec is slightly worse than BPR and LFM is the worst. This is a bit surprising to us. After digging into the dataset, we find that the dataset is very dense in terms of reviews for a game/user. Since the timestamp only tells us the date of a review, it is very likely that for a particular game, many reviews occur on the same day thus we are unable to differentiate between them in the order of time. Further analysis on the dataset to prove this can be done in the future. And other options include using a less dense dataset or try to down sample the Steam Reviews dataset to see how the performance will change. The result could provide insights for us to decide what's the reason behind this phenomenon. Other possible ways to improve the performance of transRec include trying another distance metric, e.g. L1 distance, Manhattan distance.

We can also observe that all three metrics: Hit@10, NDCG@10 and AUC are consistent across the three models. A high Hit@10 means that the model is good at ranking the ground truth item among the top ten. A high NDCG@10 means that the ground truth item is ranked even at the top within the top ten. On contrast, AUC measures on average the number of times a positive item is ranked above a negative item. So it does not care about how high your ranking is. So typically for the same model, AUC is the highest, Hit@10 is higher than NDCG@10. But to make our recommendation really accurate, we want to have a really high value of NDCG@10. From the consistency of the three metrics in our analysis, we may conclude that the models are able to improve the hit rate, normalized hit rate and average accuracy for ranking at the same time.

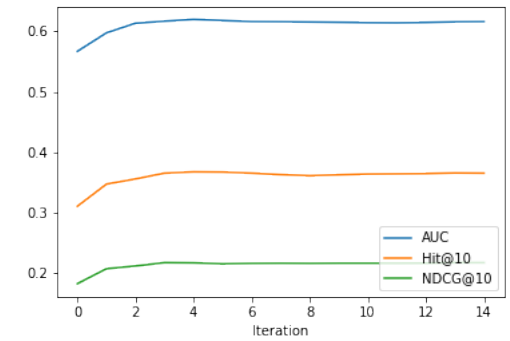
The interpretation of the model parameters of transRec can be referred to in the model section. For BPR and LFM, the model parameters are embedding vectors of user/item.



(a) TransRec



(b) BPR



(c) LFM

Figure 4: Learning curve for transRec, BPR and LFM

It is not very easy to make direct interpretation on these latent vectors. But usually by using these latent factors, we are able to capture important features such as user's preference where each dimension could represent some kind of user taste, also item's properties where each dimension of

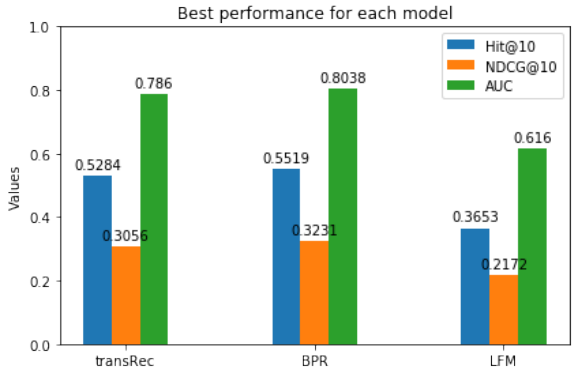


Figure 5: Optimal Performance for transRec, BPR and LFM

the latent factor could have something to do with item's genres, platform etc. As a result, when using latent factors for recommendation, we do not have to do feature extraction by ourselves since latent factors are able to capture this information for us.

7 CONCLUSION

In all, for this assignment, we implemented three different types of recommender system models: Rranslational Recommender, Bayesian Personalized Ranking and Latent Factor Model. We did a thorough analysis of the models and conducted grid search for best parameters. We used some metrics which are widely adopted in both academia and industry such as Hit@10, NDCG@10 and AUC. We compared the model performance under these metrics, which provides us with useful insights for future works.

REFERENCES

[1] Robert M Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *SiGKDD Explorations* 9, 2 (2007), 75–79.

[2] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized ranking metric embedding for next new POI recommendation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[3] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 309–316.

[4] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 161–169.

[5] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

[7] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 447–456.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[9] Julian McAuley. 2019. Recommender Systems Datasets. https://cseweb.ucsd.edu/~jmcauley/datasets.html

[10] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 502–511.

[11] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 273–282.

[12] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[13] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[14] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[15] Richard Serfozo. 2009. *Basics of applied stochastic processes*. Springer Science & Business Media.

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

[17] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 403–412.