User Behavior Set: Medley of User Behaviors to Eliminate Data Spatio-Temporal Sparsity

Yangjun Huang 1, Xuanrui Hu 2, *, Jieyu Lian 3, Xingpeng Jiang 4

1 Sun Yat-Sen University, Guangdong, Guangzhou 510006, China
2 Zhejiang University of Technology, Zhejiang, Hangzhou 310014, China
3 Northeast Normal University, Jilin, Changchun 361022, China
4 Southwest Jiaotong University Sichuan, Chengdu 611700, China

*Corresponding author Email: huangyj93@mail2.sysu.edu.cn

Abstract. A sequential recommendation system is warranted, and several models have been created to meet this need. However, sparsity is a major problem in this field. In order to deal with the space-time sparsity problem in a sequential recommendation system and avoid the unreliability caused by manually generated data, we have established the UBS structure. We treat user behavior sequences in batches. Each batch gets the integrated data set UBS after K iterations through a two-step cycle of clustering and merging. The results are connected as inputs of a sequential neural model. We use three baseline deep learning algorithms –Caser/SASRec/HGN to evaluate the model. We compared the UBS model with the original model in terms of normalized discounted cumulative gain ms(NDCG) and Recall. It is found that when the UBS model is applied to a short sequence of user behaviors and the iteration number K=2, it can improve the recommendation efficiency compared with the original algorithm (represented by K=0).

Keywords: Sequential model, recommender system, data synthesis.

1. Introduction
A recommender system is a subclass of information filtering system in order to recommend items (e.g. music, movies, shopping items) and predict user’s preference for these items. This system could meet customers’ possible interests, therefore, E-commerce companies such as Amazon, Alibaba place these models in high regard. Online businesses have increased over the past decade, and the recommendation systems system has become one of the vital engines of businesses around the world.

During the last few years, Recommender Systems have been a popular application in the e-commerce industry. Nowadays the e-commerce field is confronted with several sticky problems, such as the cold start problem, synonymy, and privacy problem [1]. Meanwhile, in real-time recommender system tasks, the dynamic, precise, and efficient prediction of users’ interests is highly valued. For example, Tom buys a guitar from Amazon and we suppose he will buy something relevant to it (He may likely pay money for a tuner or metronome soon). But as time passes, Tom may lose his enthusiasm for music and put his guitar aside, under which circumstance we can no longer propose musical instrument items to him. Therefore, modeling a sequential recommender system is warranted.
Various attempts have been made to solve such problems. Studies has shown that sequential neural network (e.g. GRU, LSTM, BERT) [2] outperform traditional time-agnostic recommender models. The fundamental idea of such sequential-based models is training with sequenced vectorized features (e.g. user’s click, ratings) as input and the sequential recommended product, candidate product id, or rating as output [3], [4]. Recently, CNN-based and transformer-based models yield strong results. The transformer does a good job in natural language processing tasks (e.g. translation, text generation). The self-attention mechanism is its strength. By implementing such a tool we can adequately apprehend sequential dependency among items and users’ behavior. BST (Behavior Sequence Transformer) model [5] is actually one of the examples of using the transformer model to quickly capture user behavior sequences, improving the recommendation efficiency.

However, due to practical constraints, most of the datasets have the problem of data Spatio-temporal sparsity. If we consider the timeline for each consumer’s behavior, the distance between different actions may be too far, which can be seen in Fig. 2. based on an Amazon musical instruments dataset. Behaviors of five users sketched on a timeline are shown in the figure, in which the long distance between these behaviors can be observed. Fig. 1. shows how these models work in a recommender system. The algorithms we just mentioned however typically implement no specific means to deal with such sparsities. To solve the issue, one way is data augmentation, which has been implemented in NLP problems already. Commonly, adding noise into the corpus (e.g. synonym replacement [6], dropout certain type of words [7]) and Back

Fig. 1. Example of sequential neural network in a recommender system

Fig. 2. The long distance between users’ behaviors
Translation [8]–[10] are utilized. Another way is to reduce the dimensions of data, in other words, lowering the Spatiotemporal distance between each behavior [11]. Nevertheless, one potential problem is that the artificially generated data may lack variety. Another problem is that such data might contain nonexistent features compared with ground truth data, which may lead to limitations in model accuracy.

For tackling the issues above, we propose a data synthesis method. Firstly, to eliminate the negative effect of Spatio-temporal data sparsity, we use clustering algorithms to divide users into different groups, concatenate user behaviors in each group. The synthesized data is defined as a User Behavior Set(UBS). Then, UBS will be the input features of sequential models. Three major benefits to using UBS is listed below:

1. Every single UBS contains more information about items, thus making recommendations for every single person more diverse.
2. All data are derivated from subsistent information, which guarantees the reliability of data resources.
3. Though sequences are getting longer, the total number of sequences is getting smaller, which may accelerate the training process.

The datasets, formulation, and architecture of our model are elaborated in section II. The experiment results on different datasets are elaborated in section III. And we make a conclusion in section IV.

2. Methodology

| Table 1. Notation |
|-------------------|
| **Notation** | **Description** |
| $S^u$ | historical interaction with sequence for user $u$ : $(s^u_1, s^u_2, ..., s^u_{|S^u|})$ |
| $S$ | set of user behavior sequences $S = (S^1, S^2, ..., S^n)$ |
| $s_i^a$ | subsequence between $s_{i-1}^a$ and $s_i^a$ |
| $C$ | indexes of longest common sequence |
| $K$ | number of iterations |
| $S' @ S'$ | merging sequence $S'$ and sequence $S'$ |
| $n$ | number of user behavior sequences |
| $b$ | batch size |

Suppose that $S^u = s^u_1, s^u_2, ..., s^u_{|S^u|}$ denotes the user action sequence in chronological order for user $u$, our goal is to learn a good personalized ranking of top $N$ items out of all items. In the paper, we train a transformer model to make such predictions. To get around the problem of Spatio-temporal sparsity in the training process, we propose a UBS structure, which is based on a clustering algorithm and data synthesis method.

2.1. Datasets

We used three real-world datasets to evaluate our algorithm. These datasets differ significantly in terms of domain and sparsity:

- **Amazon**: A dataset contains Amazon product reviews (ratings, text, useful votes) and metadata (description, category information, price, brand, and image characteristics) of 142.8 million products from Amazon from May 1996 to July 2014. This dataset contains not only multiple products, but also multiple data formats. Here, we use ‘Games’ and ‘Beauty’ as two of our classes.

- **MovieLens 2**: A famous dataset often used to evaluate collaborative filtering algorithms because it contains explicit feedback data, that is, data for rating movies. In addition to the ratings, MovieLens data also contains genre information similar to ”Western” and user application labels, such as ”over the top” and ”Arnold Schwarzenegger”. We consider 1 million ratings of users, MovieLens-1M, as our final dataset.

- **MovieLens-1m**: The dataset has the most active records for each user and item. In contrast, two Amazon datasets are much more sparse. Detailed contrast are shown in II.
2.2. UBS
The generation of UBS can be described as a loop of two-step processing (Clustering and Merging). We use the Hierarchical Clustering method to do a superposition for the items liked by the users with the same preferences and generate a new matrix. The Hierarchical Clustering method can incorporate the most similar pair of objects and create a new matrix with one less element than the previous matrix once [12]. After that, a critical problem with the number of iterations K should be considered and we will discuss it in section III. The cluster merging, specifically, is to find the two clusters with the shortest distance each time, and then merge them into a large cluster, until the merging step is over. Figure 3 shows the process of clustering and merging behavior sequences for four users in 2 iterations (K=2).

To calculate the similarity between each sequence, we introduce the Levenshtein Distance [13]. The Levenshtein Distance, namely the Edit Distance, is designed to count the number of insertions, deletions, and substitutions required to turn one sequence into another. Suppose the Levenshtein Distance between two user behavior sequences $S^a$, $S^b$ is given by $\text{lev}(S^a, S^b)$ and length of sequence $S^a$ is $|S^a|$, we define the similarity between $S^a$ and $S^b$ as:

$$\text{sim}(s^a, s^b) = \frac{|S^a|}{(\text{max}(|S^a|, |S^b|) - \text{lev}(s^a, s^b))}$$

(1)

Fig. 3. Hierarchy Clustering with four behavior sequences

After each clustering step, we will have a merging step. The merging step is to merge two sequential data into one, which can be formulated as:

$$S^o = S^a \oplus S^b$$

(2)

Here $S^a$ and $S^b$ are sequences to be merged. The $S^o$ is the output of merge operation $\oplus$. The merge operation $\oplus$ is performed by a extraction of Longest Common Subsequence (LCS) and combination of other subsequences which is not in the LCS. The LCS problem is to find the longest common subsequence among a set of sequences (usually only two sequences) [14]. We first use classical dynamic program algorithm to get indexes in common subsequence. It can be represented as:
Here $c_{i,j}$ records the LCS indexes of first $i$ elements of $S^a$ and first $j$ elements of $S^b$, and we have:

$$\text{LCS}(s^a, s^b) = C_{[|s^a|, |s^b|]}$$

To save storage space, we implement a traceback approach\(^3\). Trace from all $c_{i-1,j-1}$ or $c_{i,j-1}$ to $c_{i,j}$ is saved so we can rebuild such subsequence backwards after LCS$(s^a, s^b)$ is found. The next step of merge operation is to reorder the non-overlapping subsequence between each overlapping subsequences. We stipulates that if $|s^a| > |s^b|$ or $|s^a| = |s^b|$, $a < b$, every subsequence $s^a_i$ will be ordered precedes $s^b_j$, where $(i, j) \in C$, $s^a_i$ is the subsequence between $s^a_{i-1}$ and $s^a_i$, $s^b_j$ is the subsequence between $s^b_{j-1}$ and $s^b_j$. Suppose that $S = \{s^u | u \in U\}$ denotes the original set of behavior sequences and $S_t$ denotes the behavior sequence set after $t$ clustering-merging iterations, we will have $S_{t+1} = S_t - S^a - S^b + S^o$.

After we finish $K$ iterations, we define UBS as $SK$, and use it as the input of the sequential neural network.

2.3. UBS in batches

A major problem with our original UBS algorithm is time complexity. Given the number of sequences in original sequence set $n = |S|$ and maximum length of a sequence $m = \max_{u \in U}|s^u|$, the time complexity of a distance comparing is $O(n^2m^2)$, and the time complexity of a merging step is $O(2m)$. Thus, the time complexity of our UBS algorithm with $K$ iteration steps is $O(Kn^2m^2) + O(2Km)$, which is unacceptable because $n$ is often big. To resolve the issue, we separate the sequence $S$ in batches with a batch size $b$, implement the UBS algorithm in each batch, and concatenate the results as the input of the sequential neural model. The time complexity of the whole process will be reduced to $O(Kb^2nm^2) + O(2Kbm)$.

3. Experiment

In this section, we will test the efficiency of our structure, UBS, with some state-of-the-art models. The codes are implemented in Pytorch with 4-core Intel i5-7300HQ @ 2.50GHz CPU, 16G RAM and Nvidia GTX 1050 GPU.
3.1. Evaluation Metrics
We evaluated our model according to “Recall@k” and “NDCG@k”. For each user, “Recall@k” refers to the ratio of products in the top-k recommendation list to “all highly rated products”, which measures how many highly rated products appear in the top-k list. NDCG@k is the cumulative benefit of standardized discounts at k. The higher the correlation, the better the recommendation effect. NDCG@k measures the recommendation effect determined by correlation and location.

3.2. Baseline Models and Preprocessing
We include 3 deep-learning algorithms in our evaluations. Caser is an outstanding CNN-based method in sequential recommender system [15]. Inspired by the success of transformer models, SASRec applies self-attention mechanisms in recommendations [16]. Recently, HGN, a model using Bayesian Personalized Ranking to capture long-short term interests, achieves high performance in solving this problem [17].

These baseline models are all open-source. Their pre-processing method have a lot in common and can be summed up in 3 parts: (i) filter all records with low ratings (e.g. fewer than 3) (ii) only can keep items and users of more than 6 related actions. In the case of rating dataset, an action is a user-item mapping (iii) order sequence of actions by timestamp, and for each user u, extract a action sequence S. Then we implement our UBS algorithm with different K to get a merged sequence set S_k. Finally, the S_k is split 75% to be training set and 25% to be test set.

3.3. Result Analysis
Results of the comparison are shown in table III, from which, we can see the strength and weaknesses of our model. To be specific, in the case of Amazon Beauty and Amazon Game dataset, the NDCG and Recall are improved in both

Table 3. Recall@10 and NDCG@10 of different methods on real-world datasets. Compared with baseline model(K=0), better performance is marked in red.

| Dataset Metric | Beauty Recall@10 | Game Recall@10 | ML-1M Recall@10 |
|----------------|------------------|----------------|-----------------|
|                | NDCG@10          | NDCG@10        | NDCG@10         |
| CASER          | 0.516 0.348      | 0.52 0.321     | 0.789 0.554     |
| CASER+UBS(K=1) | 0.531 0.345      | 0.532 0.370    | 0.642 0.381     |
| CASER+UBS(K=5) | 0.517 0.321      | 0.538 0.409    | 0.558 0.371     |
| SASRec         | 0.682 0.417      | 0.742 0.537    | 0.792 0.595     |
| SASRec+UBS(K=1)| 0.673 0.401      | 0.690 0.456    | 0.771 0.550     |
| SASRec+UBS(K=5)| 0.697 0.427      | 0.637 0.442    | 0.723 0.517     |
| HGN            | 0.647 0.455      | 0.716 0.520    | 0.746 0.508     |
| HGN+UBS(K=1)   | 0.660 0.467      | 0.709 0.512    | 0.702 0.442     |
| HGN+UBS(K=5)   | 0.650 0.464      | 0.722 0.527    | 0.653 0.437     |

Fig. 5. Loss in training process
three models when using proper $K$. However, in the case of MovieLens dataset, the performances are all getting worse when implementing UBS.

Upon further analysis, we discover that whether our model works is highly affected by the length of user action sequences in different datasets. The average sequence length for each consumer is 17 in Beauty and Game dataset, while 96 in MovieLens dataset, which can be too long for a sequential prediction. The histogram of sequence length for each dataset is shown in Fig. 4. A long sequence of actions not only includes many noise data but also takes a great deal of time in each merging step. What’s worse, the similarity of every pair of merged sequences is very low in this instance and this would bring down the accuracy of deep learning models. Therefore, it is inappropriate to use UBS when the user action sequence is overlong.

3.4. Impacts on Different $K$

In this part, we take a further look at how $K$ will affect the performance. We specifically choose Amazon Beauty as the dataset and HGN as the baseline model with all hyperparameters unchanged. The number of iterations $K$ is set to be 0, 1, 2, 3, 5, and 10. In Fig. 5, we compare the average loss in the training process among different $K$. Fig. 6 shows NDCG@10 for models with various $K$.

An obvious observation from the figures above is that the number of loops $K$ can effect convergence and model performance. Although a different $K$ will not lead to a faster decrease of loss in the training process, it has an impact on the level of NDCG, in other words, the convergence of the sequential model. In this case, $K = 2$ leads to higher performance while $K = 10$ works a bit worse than the baseline($K = 0$).

4. Conclusion

In this paper, we propose a novel architecture called User Behavior Set to solve Spatio-temporal sparsities in sequential recommender systems. UBS is generated by a hierarchical clustering algorithm and a merging algorithm. We learn the efficiency and defect in UBS by a series of experiments and analyses. It achieves better ranking results in datasets with short behavior sequences but poorer results in the datasets with long behavior sequences. What’s more, we evaluate the impacts of different merging step $K$ on the model performance.

For future work, we will be exploring the length of the longest sequence of user behaviors that UBS can handle and expand the applicable range of users. At the same time, we will probe ways to reduce noise data in very long sequences, for example, classifying commodity types so that sequences are divided into small segments. It is hoped that through continuous efforts, the UBS model can be ameliorated, the accuracy of recommending commodities for more users can be improved, and the value of services can be enhanced.

![Fig. 6. NDCG@10 in training process](image-url)
Acknowledgements
Yangjun Huang and Xuanrui Hu contributed equally to this work

References
[1] I. Ullah, Recommender Systems: Issues, Challenges, and Research Opportunities. Springer Singapore, 2016.
[2] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, “BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer,” arXiv: 1904.06690 [cs], Aug. 2019, arXiv: 1904.06690. [Online]. Available: http://arxiv.org/abs/1904.06690
[3] T. Donkers, B. Loepp, and J. Ziegler, “Sequential User-based Recurrent Neural Network Recommendations,” in Proceedings of the Eleventh ACM Conference on Recommender Systems. Como Italy: ACM, Aug. 2017, pp. 152–160.
[4] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, “Sessionbased Recommendations with Recurrent Neural Networks,” arXiv: 1511.06939 [cs], Mar. 2016, arXiv: 1511.06939. [Online]. Available: http://arxiv.org/abs/1511.06939
[5] Q. Chen, H. Zhao, W. Li, P. Huang, and W. Ou, “Behavior Sequence Transformer for E-commerce Recommendation in Alibaba,” arXiv: 1905. 06874[cs], May 2019, arXiv: 1905. 06874. [Online]. Available: https://arxiv.org/abs/1905.06874
[6] J. Wei and K. Zou, “Eda: Easy data augmentation techniques for boosting performance on text classification tasks,” 2019.
[7] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” 2015.
[8] V. C. D. Hoang, P. Koehn, G. Haffari, and T. Cohn, “Iterative back-translation for neural machine translation,” in Proceed-ings of the 2nd Workshop on Neural Machine Translation and Generation. Melbourne, Australia: Association for Computa-tional Linguistics, Jul. 2018, pp. 18–24. [Online]. Available: https://www.aclweb.org/anthology/W18-2703
[9] R. Sennrich, B. Haddow, and A. Birch, “Improving neural machine translation models with monolingual data,” 2015.
[10] S. Prabhumoye, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black, “Style transfer through back-transla-tion,” 2018.
[11] D. Zhang, “Yading: Fast clustering of large-scale time series data,” in International Conference on Very Large Data Bases, 2015.
[12] M. Steinbach, G. Karypis, and V. Kumar, “A comparison of document clustering techniques,” 2000.
[13] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” Soviet Physics Doklady, vol. 10, p. 707, February 1966.
[14] M. Paterson and V. Dancik, “Longest common sub-se-quences,” in Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science 1994, ser. MFCS ’94. Berlin, Heidelberg: Springer-Verlag, 1994, p. 127–142.
[15] J. Tang and K. Wang, “Personalized top-n sequential recommendation via convolutional sequence embedding,” CoRR, vol. abs/1809.07426, 2018. [Online]. Available: http://arxiv.org/abs/1809.07426
[16] W. Kang and J. J. McAuley, “Self-attentive sequential recommendation,” CoRR, vol. abs/1808.09781, 2018. [Online]. Available: http://arxiv.org/abs/1808.09781
[17] C. Ma, P. Kang, and X. Liu, “Hierarchical gating networks for sequential recommendation,” CoRR, vol. abs/1906.09217, 2019. [Online]. Available: http://arxiv.org/abs/1906.09217