Sequence Aware Recommenders: Sensitivity perturbations of the sequence

Federico Chung: fchung12, Charles Reinertson: crein1, Ernesto Cediel: eced

March 2023

1 Introduction

A very strong assumption of any usage of data is that the data has no errors. However, due to many issues/edge cases in the data collection process, there can be violations of this assumption. This is no different for movie recommender systems which in some cases collect their data through explicit feedback. Explicit feedback sequences are collected through user feedback mechanisms such as user ratings. However, the point in time when they consumed or purchased an item might be quite different from the time they rated an item [3].

Another potential way movie data can be unintentionally perturbed is due to the rating of random movies within a sequence. This might be due to many causes such as account/password sharing. A recent study estimates that 22.6 % of people engage in password sharing in the US [2]. This means that we cant ensure that movie lists are from one user but of multiple users.

This paper tries to explore the potential effects of different levels of perturbations on recommender systems. By using different levels of imputation and order perturbations we try to understand the limitations of recommender systems when we find perturbed datasets. We want to better understand how these sequence-aware recommender systems perform in these edge-case scenarios, and how well-equipped they are to deal with them.

2 Data Pre-Processing

We use the MovieLens dataset with one-million rows for our project. We consider this dataset to be large enough for the scope of our project. We consider three tables from the dataset: users, ratings, and movies. A UML diagram was created in Figure 1 to understand the structure of our dataset and make the necessary joins.

Given that we are interested in sequences, it is very important to maintain the timestamps of our data. The idea is to exploit the sequential signal underlying the user's behavior sequences. Hence, a function is used to create every possible sequence of length L from a user's viewing history. In Figure 2 we show an example of this with $L = 3$ and $StepSize = 1$, this user has five movies in their viewing history.

Our dataset now has sequence granularity, if we have a set of users U, and on average a user has user history H, then we will expect to have approximately $|U|(H - L + 1)$ sequences.

To better explain the whole data processing and transformation, it is best to first give a brief explanation of the goal of our project, so that this context allows for a better understanding of the structures that we create. The goal consists in evaluating the robustness of sequence-aware

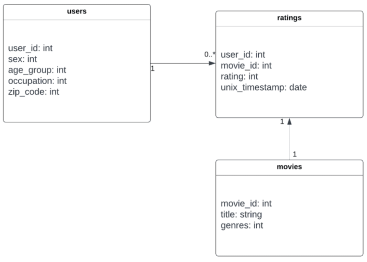

Figure 1: Data Description

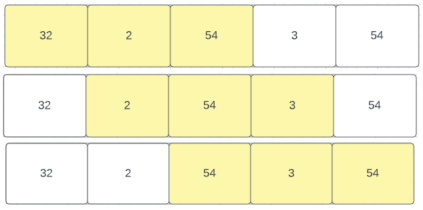

Figure 2: Window Creation

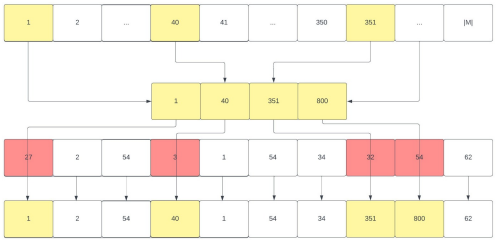

Figure 3: Imputation Procedure

Figure 4: Sequence Order Permutation

recommender systems by perturbing the sequence arrangement of the entire dataset. This evaluation will be done using two main perturbations and will be described in section 3.

3 Simulation Procedure

For the simulations in this paper, we are going to perturb the whole dataset, both train and test, and the perturbations are going to be incurred in the full list of movies watched per user. This is to emulate the potential perturbations we might see in the real world, as in the real world perturbations do not only happen to the training or test datasets but to the whole dataset. Given that usually we just get a list of movies a user watched, we do not necessarily know where in the list of movies a user might have perturbed data. It can be common to have sequences that are slightly perturbed and others fully perturbed. We will explore two kinds of perturbations to the list of movies by a user. The two perturbations that we will explore are sequence order randomization and sequence item imputation.

For sequence item imputation, as seen in Figure 3, we select a proportion of indices at random and replace movies from a set of movies the user has not watched. But as we will mention later in the paper, this does not mean that all the sequences will have the same amount of perturbations, this is because we first perturb the whole list of movies and then we create sequences based on that perturbed movie list. Their movie ratings would be randomized from a value of 1 through 5.

For sequence order randomization, we will choose a percentage or a number of indices that will be perturbed, and from that set of indices, we will shuffle the values using a randomizer. This procedure is outlined in Figure 4 above, where we see what a 50% perturbation might look like. We choose four indices at random and then shuffle those indices and impute them back into their new order.

Perturbation	Average.Corruption
Inputed 10%	9.751234
Inputed 20%	19.773976
Inputed 30%	29.740220
Inputed 40%	39.765296
Inputed 50%	49.867462
Order 10%	9.170873
Order 20%	19.208164
Order 30%	29.192588
Order 40%	39.204658
Order 50%	49.294998

Table 1: Average Sequence Corruption by Perturbation Scheme

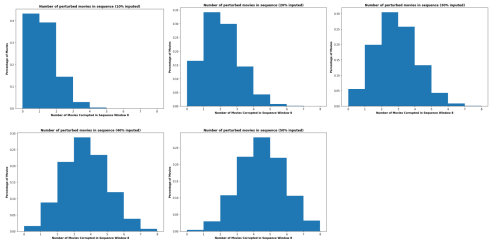

Figure 5: Sequence Imputation

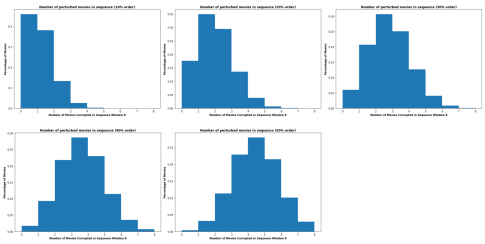

Figure 6: Sequence Order Perturbation

This will potentially change the order of the original sequence. The percentage of randomization will always be an upper bound on the perturbation of the sequence as it is possible that items get mapped to their original index. If items are mapped to the original index, it is also possible that the sequence is not perturbed at all, that probability decreases as we increase the percentage of movies perturbed.

For each user movie list with a window size of 8, we will create contiguous sequences (step size = 1) to incur the most data that we can from the list of movies for each user. The BST model (subsection 4.2) uses a window size of 8 where it uses 7 movies to predict the last movie in the sequence, while the CASER (CNN) model (subsection 4.1) uses a window size of 8 where it uses 5 movies to predict the last 3 movies in the sequence. The number of perturbed movies in each sequence will range from 0-8, however, the mean value of the number of perturbed movies will increase the higher the percentage of the list of movies we watch per user increases.

Evaluation of Perturbation Procedures

Table 1 shows the true perturbation metrics for all of our different perturbation schemes. And as we can see from Table 1, the average corruption for all the sequences matches the perturbation on the whole list per user. However, it is important to note that the perturbations will be an upper bound of the total perturbation.

As we can see from Figure 5 and Figure 6 above as the perturbation percentage increases there is a shift in the histograms in the number of movies corrupted in the sequence. From the histograms, we can see that even in the cases for 10 % imputation (or order randomization) there are sequences with 5 perturbed movies within the sequence window. And for cases for 50 % movie list imputation (or order randomization), there are still some sequences that are not perturbed. But it is important that on average most of the sequences will be perturbed on average close to the overall perturbation

per user as seen in Table 1. But it is important to note that even in lower-level perturbations we will see some sequences with a high level of perturbations.

4 Literature Review

Depending on the specifics of the application or the scenario the order of sequences can be considered relevant. For example, [3] states that the order of the past events may or may not be relevant for the recommendation task and that depends on the context and dataset. Order might be important due to implicit feedback constraints, where you can recommend Season 2 of a series after Season 1 but not vice versa. While multiple time/sequence-aware models have proved to have much better performances in these kinds of scenarios, there is no consensus on ideal ways to prove that these recommender systems are leveraging the sequential nature of the data as an advantage to improve performance.

Sequential recommender systems matter only when the dataset includes sequential patterns. Which can be measured using the sequential intensity of a dataset Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding [4]. Perturbation of sequences has been a recent way of evaluating the sensitivity of recommender systems. The literature on sequence perturbation evaluations of recommender systems have used mainly a LOO (leave one out) perturbation frameworks [4], [1] to evaluate the effect of perturbation on recommender systems. However [1] has additionally added replacement, and imputation to further evaluate the effect of multiple different perturbations on the sensitivity of these models.

A paper such as Rank List Sensitivity of Recommender Systems to Interaction Perturbations [Oh et al., 2022], explores how small changes in the training data for one user drastically alter predictions in other users' interactions. This is dangerous when recommender systems are used in more sensitive areas, as data perturbations in one user can alter recommendations in other users, which is why we must explore how sensitive these sequential-based recommender systems are.

Different approaches have been used to capture the sequential nature of these events. One of these sequential aware-recommender systems can be seen in the paper Behavior Sequence Transformer for E-commerce Recommendation in Alibaba [4], which shows that current recommender systems in industry often embed features into low dimensional vectors and then feed those features through a MLP for final recommendations, ignoring the time-series nature of many recommendation data-sources. Later, a transformer can leverage this time-series data to outperform non-sequence-aware recommender systems. Other models such as CASERs have also been used as a sequence-aware model. [4], hints that CASER (CNN) models that learn these sequential features will perform better on tasks where random products are introduced in user sequences.

4.1 Convolutional Sequence Embedding Recommendation Model (CASER-CNN)

A CNN is a feed-forward neural network that processes data with grid-like topology. For this specific CNN, the authors create a state-of-the-art performance alternative to RNN, claiming that it solves some of its limitations by using horizontal and vertical convolutional filters to capture sequential patterns at point-level, union-level, and of skip behaviors.

As shown in the previous figure, the model consists of three components: Embedding Look-up, Convolutional Layers, and Fully-connected Layers.

For the Embedding Lookup section, each user u extracts every L successive items as input and their next T items as the targets from the user's sequence S_u, shown on the left side. Each item

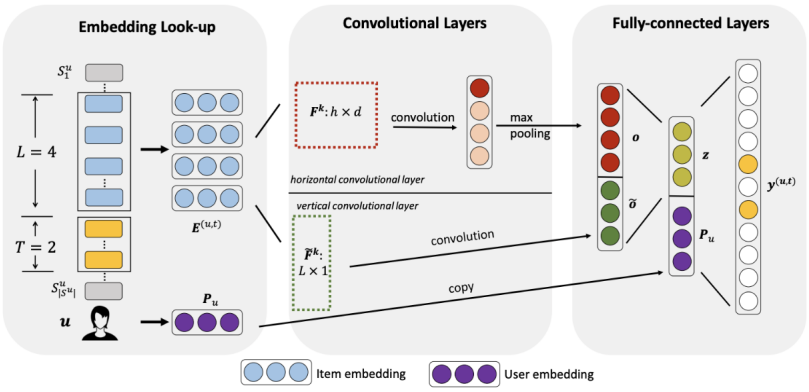

Figure 7: CASER architechture

is embedded using latent factoring with dimension d. Each of these latent representations for each item is stacked together in a single matrix E(u,t). The specific user information is also embedded in a latent vector called P_u.

For each user sequence we have the L by d matrix E(u,t) that will be used as input to the Convolutional Layers. The diagram shows two horizontal filters that capture two sequential patterns. These h by d filters pick up signals for sequential patterns by sliding over rows of E. For example, the first filter picks up sequential (union-level) patterns such as (Star Wars I, Star Wars II) $- >$ Star Wars III by having larger values in latent dimensions where Star Wars I and II have larger values. Similarly, the L by 1 vertical filter will slide over columns of W. These filters are capturing point-level sequential patterns through weighted sums over previous items' latent representations.

The results of these two convolutional layers are concatenated and fed along with the user embedding into the fully connected layers to get more high-level and abstract features. The output of the last layer is probabilities (using a sigmoid function) of a user watching each and every movie given its previous sequence.

4.2 Behavior Sequence Transformer Recommender System (BST)

The Behavior Sequence Transformer Recommender System or BST algorithm is directly implemented from the paper [3]. In the era of deep learning, embeddings and MLP have been the standard paradigm for industrial recommender systems. However, these recommender systems do not take into account the sequential signal underlying the users' behavior sequences. Recurrent neural networks and their successor, Transformers, can take advantage of this signal in the learning process. Recurrent neural networks fall short in remembering long-term dependencies and can be slow to train as they use recursion and cannot be parallelized.

Transformers avoid recursion allowing parallel computation and reducing drops in performance due to long dependencies. In our example, user watch history is processed as a whole rather than movie by movie to avoid long-term dependency issues. There is no risk to lose past information. Transformers utilize positional embeddings to replace recurrence. The idea is to use fixed or learned weights that encode information related to a specific position of a token in a sequence. Furthermore, transformers implement self-attention to compute similarity scores between movies in a sequence (movie IDs in a user's watch history). In addition, attention and positional embeddings provide information about the relationship between different words.

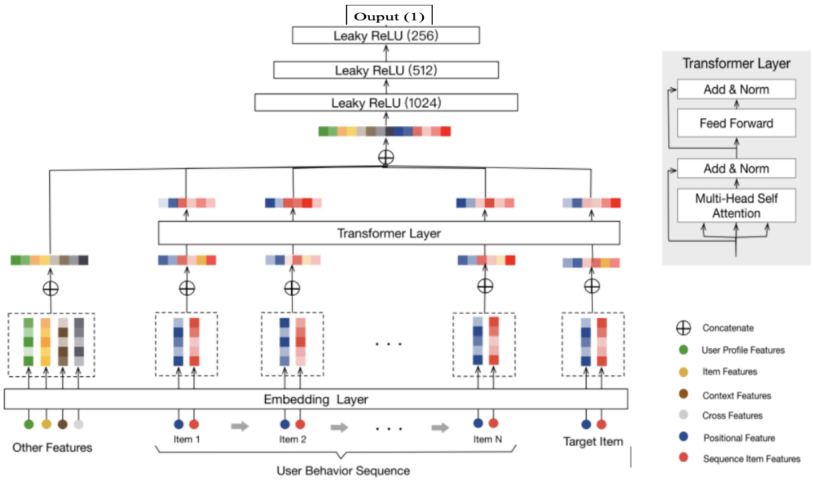

Figure 8: BST architechture

Our BST architecture takes as input the user's watch history, including the target movie rating, and "other features". It first embeds these input features as low-dimensional vectors. To better capture the relations among the items in the behavior sequence, the transformer layer is used to learn a deeper representation of each item in the sequence. Then, by concatenating the embeddings of "other features" and the output of the transformer layer, the three-layer MLPs are used to learn the interactions of the hidden features, and finally learn to predict the ratings of unseen movies through Mean Squared Error Loss between the predicted movie rating and the actual movie rating.

4.3 Metric Evaluation

Note: For the following evaluation metrics we cut off the prediction and ground truth recommendations at $k = [5, 10, 15]$ because, in many recommender system problems, often users will stop looking at recommendations after skimming the first k in the list. We calculate the metrics@k per user and then average over all users. We do this because we have one sequence per user so in order to have a generalized metric we average over users.

Average per User Precision@k is calculated by dividing the true positives over anything that was predicted as positive. Precision asks the question, how many retrieved items are relevant?

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Average per User Recall@k is calculated by dividing the true positives over anything that should have been predicted as positive. Recall asks the question, how many relevant items are retrieved?

$$Recall = \frac{TruePositive}{TruePositive + FalseNegatives}$$

For evaluating model recommendations on both the perturbed and non-perturbed test set we use Average per User NDCG@k. NDCG is created from several from several components. The first component cumulative gain is the sum of graded relevance values of all results in a recommendation

list.

cumulative gain = $\sum_{j=1}^{n} relevance_i$

The next component, DCG, penalizes highly relevant documents that appear lower in the recommendation by reducing the graded relevance value logarithmically proportional to the position of the result.

Discounted cumulative gain = $\sum_{j=1}^{n} \frac{-2^{relevance}-1}{log_2(i+1)}$

An issue arises with DCG when we want to compare the recommendation performance from one query to the next. We can normalize DCG to get NDCG which is the metric we perform for our evaluation. We perform this by taking our ground truth recommendations and calculating DCG for this set. Then we divide the predicted DCG by the ground truth DCG to get normalized DCG.

We take NDCG@k to be the final metric. The discount function is set as $D(r) = 0$ for all $r > k$ meaning that if our predicted recommendation is not in the first k ground truth recommendations, the discount function is zero. Precision and Recall do not take into account ordering like NDCG which sheds more light on the performance of our system.

The paper [3] describes in detail sequence-aware recommender systems and their benefit in scenarios with short-term user interests and longer-term sequential patterns. This paper describes the metrics most applicable to measuring the success of recommender systems. Precision and Recall are great when we do not care about the order of recommendations. However, Mean Average Precision (MAP) or NDCG is preferred over classical metrics when order matters. For our problem, want the reader to learn from our results whether or not order is important for their specific task. Therefore, we chose to use Precision and Recall to gain insight into the situation when order does not matter, and NDCG to gain insight into the situation when order matters.

5 Results

We have divided our results into 12 graphs, figures 9 through 20, where each graph shows the change in per-user average Precision@k, Recall@k, and NDCG@k for both models and both types of perturbations, namely imputation and randomization. The change is observed as we increase the percentage of perturbation for each of the user's sequences as we explained in section 3.

From the results, on Figures 9-14, it is clear that the CASER (CNN) is more robust to randomization perturbations of sequences compared to imputation perturbations. In the case of imputation, for all metrics and all k's we observe that as we increase the perturbation there is a strong drop in performance, this change in performance is less as there we keep increasing perturbance, showing a non-linear decrease. On the other hand, when the model has randomization perturbances, we observe a less aggressive drop in performance that becomes stagnant at around 30 to 40 percent perturbance.

As we observe, on average, we are obtaining precision scores between 0.1 and 0.2, which means that in expectation, for $k = 10$, the model is correctly recommending 1 to 2 movies per user sequence. This intuition helps us understand why we observe that as we increase k we obtain larger scores given that there is a higher probability for at least one or two of the model's initial recommendations to be in a larger list of movies.

The BST model is severely flawed for this recommendation task. This explanation can be found

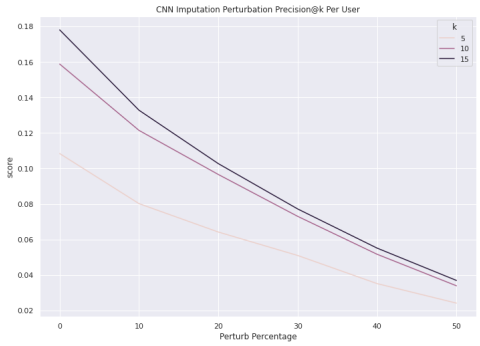

Figure 9: CASER Inputed Data Precision

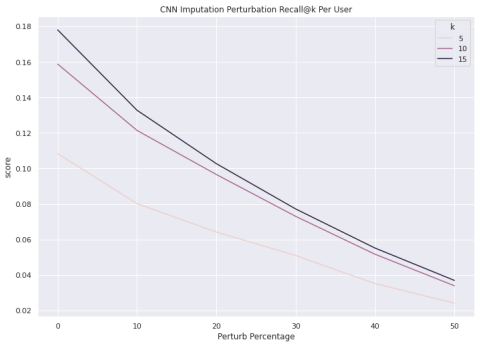

Figure 10: CASER Inputed Data Recall

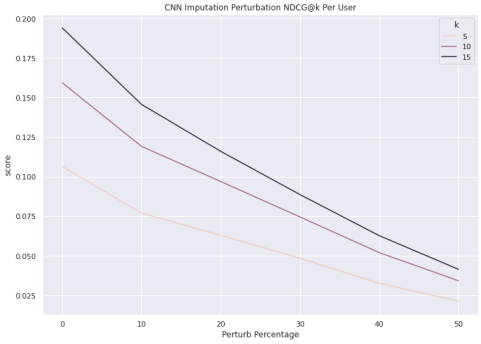

Figure 11: CASER Inputed Data NDCG

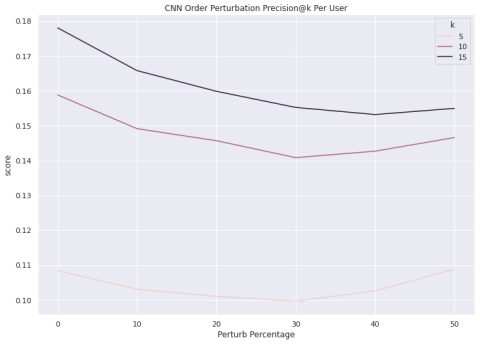

Figure 12: CASER Random Data Precision

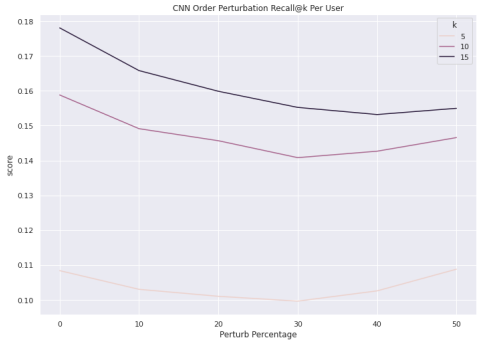

Figure 13: CASER Random Data Recall

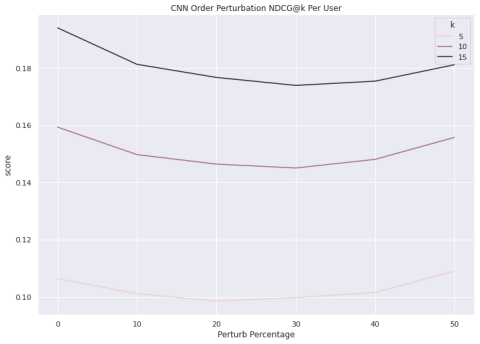

Figure 14: CASER Random Data NDCG

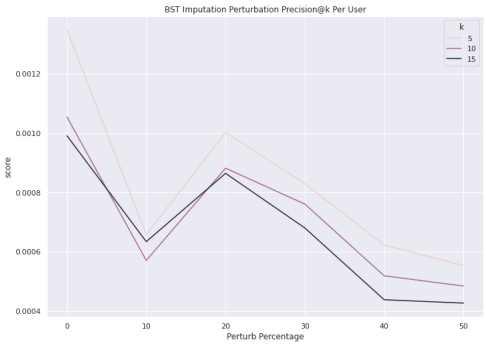

Figure 15: BST Inputed Data Precision

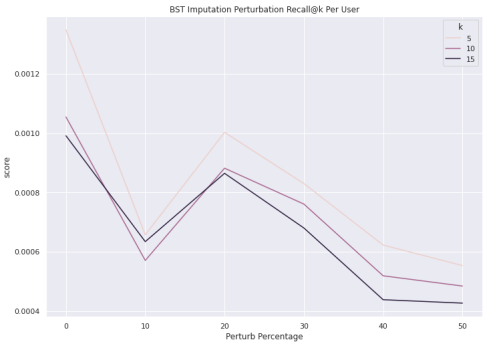

Figure 16: BST Inputed Data Recall

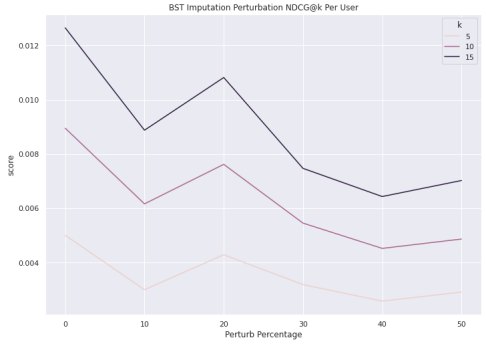

Figure 17: BST Inputed Data NDCG

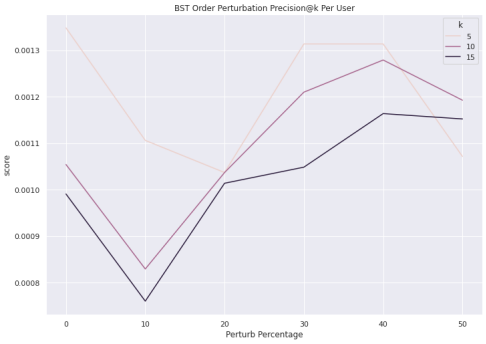

Figure 18: BST Random Data Precision

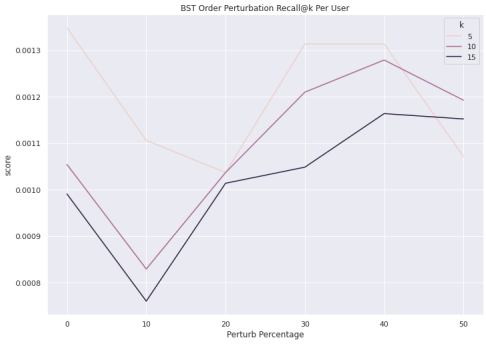

Figure 19: BST Random Data Recall

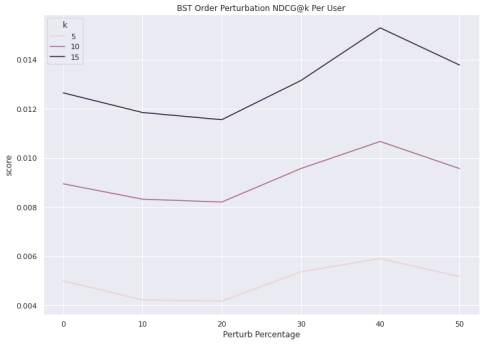

Figure 20: BST Random Data NDCG

in the conclusion. For the imputation perturbation (Figures 15, 16 and 17), the BST has a maximum score of 0.0013 for precision, 0.0013 for recall, and 0.013 for NDCG. The Order Perturbation maximum scores are very similar. For comparison, the CASER (CNN) evaluation metrics are anywhere between 10 and 100 times higher. These scores are so low that over thousands of users, minimal improvement in recommendations will drastically increase this score. That is why these graphs do not show a discernible pattern as a new model is trained for each data perturbation. This creates small amounts of randomness in the recommendations. If recommendations are good, this randomness should have a minimal effect. However, with such poor recommendations, randomness gives us nonsensical patterns in these graphs.

The relationship between the k's is not linear, the difference between $k = 5$ and $k = 10$ is larger than the difference between $k = 10$ and $k = 15$ for precision, recall, and NDCG. However, each value of k substantiates the results above. We plotted these values of k to see if there were different findings at different intervals, but the findings stayed consistent.

6 Conclusion

The CASER (CNN) greatly outperforms the BST in all of our evaluation metrics. This is because the CASER (CNN) embeds a target sequence of T items from the user watch history. The BST, however, only embeds the next item as the target item. The BST is therefore learning to predict the very next item's rating the best it can, whereas the CASER (CNN) learns to predict a target sequence that aligns more favorably with the evaluation metrics Precision@k, Recall@k, and NDCG@k. The BST has a final feed-forward network with the last layer having length one. This is because it learns to predict the ratings of unseen movies through Mean Squared Error Loss between the predicted movie rating (the final network layer) and the actual movie rating. This is not directly learning an ordered sequence recommendation task that we are evaluating. The CASER's loss function is much more favorable to this task. The output of the last layer of the CASER (CNN) are scores of a user watching each and every movie. Binary cross-entropy loss is then used to learn the probabilistic sequence of these scores.

Because of the above limitations of the BST, we chose to limit our conclusions to findings only related to the CASER (CNN). The CASER (CNN) is robust against random shuffling of data but performs poorly with data imputations. This is because random shuffling does not change the most important signal, the movies that a user watches. Although the sequence of the movies is important and shuffling this sequence causes performance degradation, this is not the most important model signal.

The CASER (CNN), according to the paper [4], exhibits skip behavior that allows the model to be robust enough to make sensible recommendations despite imputed user-watch history sequences. However, the benefits of skip behavior are not readily seen when we perform data imputations. The more we impute random movies in the sequence, the worse the CASER (CNN) performs in this recommendation task. We see a negative, exponential decay to linear correlation between the percentage of random movies imputed and model performance over recall, precision, and NDCG.

In general, the benefits of sequence-aware recommender models compared to their sequence-agnostic counterparts are small. The correct, non-imputed information is much more important than the signal from the order of the sequence. More research must be done to harness the benefits of sequential patterns for recommendation systems.

References

[1] Sejoon Oh, Berk Ustun, Julian McAuley, and Srijan Kumar. Rank list sensitivity of recommender systems to interaction perturbations. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1584–1594, 2022.

[2] Niko Pajkovic. Algorithms and taste-making: Exposing the netflix recommender system's operational logics. *Convergence*, 28(1):214–235, 2022.

[3] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

[4] 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.