Replication of a Knowledge Graph Recommendation System

Joey Cook-Gallardo, Weijia Ma, Sam Terwilliger, Rosa Zhou

March 2020

1 Introduction

Whether it be Google's targeted advertising or Youtube's recommended videos, recommendation systems play a fundamental role in today's digital landscape. Recommendation systems, as the name suggests, are simply systems that make recommendations to users based typically on user or item data. For example, Spotify has millions of songs and users. The goal of a recommendation system in this case would be to determine what yet unseen song a given user would most likely want to listen to. Collaborative and content based filtering are two common approaches used to solve this problem.

Collaborative filtering is an approach where the preferences of similar users are used to make recommendations. In the Spotify example, similar users could denote two users that like similar songs. A collaborative recommendation would be recommend a user a song based on the information that a similar user liked that song. The assumption behind this approach is that if two users like a similar song/s, those users will have similar preferences for other songs.

Another common type of recommendation systems are ones which use content based filtering. As opposed to collaborative filtering, where recommendations are based on other users' information, content based filtering makes recommendations solely based on a single user's preferences. For Spotify, a user's preference might be that they like songs from the artist Beyoncé. A content based recommendation system would recommend that user another song by Beyoncé.

Combining content based and collaborative filtering has the potential to improve recommendation systems, as demonstrated in "Explainable Reasoning over Knowledge Graphs for Recommendation" [1]. This paper used a knowledge graph and a mixture of both types of filtering to make song and movie recommendations, proposing a model called the Knowledge Path Recurrent Network(KPRN). For our project we replicated this model, experimented on varying parameters, and investigated our own baseline model. Before outlining the KPRN model and our replication, its important to understand why replication of this paper would be useful. As previously established, recommendation systems are pervasive online. Furthermore, efficient and effective ones are in high demand. As the replication paper proposes a new and seemingly more effective model than many standard models, we believe the accuracy of this study is worth verifying. Replication helps with this task since it may reveal errors that a study could have such as improper implementation or unclear documentation.

2 Original Paper

The original paper both proposes the new KPRN model, and uses other baseline models for comparision. Thus before we discuss our replication, we will first discuss how the KPRN and MF baseline models work.

2.1 KPRN Model

The goal of the KPRN is to calculate the plausibility or likelihood that a user listens to a song. To find the plausibility, we start with a type of graph known as a knowledge graph. Then, we find paths from users to songs in the knowledge graph. Next, the paths are inputted into a neural network model which calculates the plausibility score. A knowledge graph is a graph where each of the vertices and edges contain a variety of information. This information can be used to learn about complex relationships within the graph. The knowledge graph in this paper contains vertices that represent users, songs, and artists, and edges that represent the relationships between those vertices.

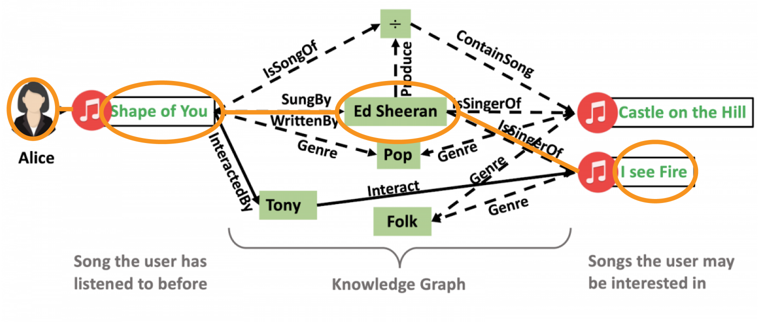


Figure 1: Knowledge Graph path example 1 [1].

Figure 1 represents a path for the user-song pair (Alice, I See Fire). As we can see, Alice listened to *Shape of you*, which was sung by Ed Sheeran, who is also the singer of *I see Fire*. This path suggests that Alice is likely to listen to *I see Fire* because she has listened to another song by the same artist.

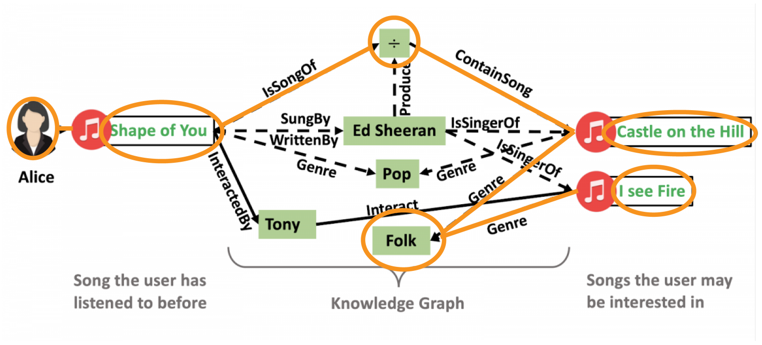


Figure 2: Knowledge Graph path example 2 [1].

Figure 2 represents another path for the same song user pair. Again, Alice listened to *Shape of* *you*, which is in the album division. This album contains *Castle on the hill*, which is in the folk song genre, and *I see Fire* is also in that genre.

Now between these two paths, how do we know whether one is more plausible than the other? Well if we consider Figure 1's path, we can see there is a short direct relationship. Alice likes an artist, and therefore it is likely she would like a song from that artist. Figure 2's path on the other hand isn't quite as clear. For example, there can be multiple genres in an album and it could be that Alice does not listen to folk songs at all. Therefore, this path could be less useful in predicting the preference of Alice. A neural network will allow us to predict the usefulness of paths.

For each user song pair, the KPRN's neural network takes a set of paths from that user to that song. As the paper's knowledge graph contains song and artist vertices in addition to user vertices, each path can be expressed as a sequence of entities (vertices) and relations (edges). The neural network is composed of the embedding, LSTM, and weighted pooling layers.

The embedding layer converts categorical data such as entities and relations into embedding vectors. The benefit of embedding layers is that as opposed to methods such as one-hot encoding which stores categorical data in sparse matrices (ie. wasting a lot of space), embedding layers store categorical data using indices which can be used to find vectors. This makes embedding layers much more memory efficient with larger datasets [2].

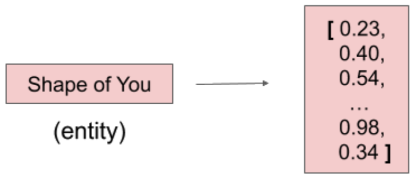


Figure 3: Entity embedding example.

Figure 3 shows how the song *Shape of you* can be turned into a vector of numerical values. As the neural network is trained, the weights in the embeddings will update and learn the representations of the song.

Since the paths in the paper's knowledge graph can be expressed as a sequence of (entity, entity type, relation) triplets, different embeddings are concatenated together to get an overall embedding vector (see Figure 4). Figure 5 demonstrates what the resulting structure of an embedding of a path looks likes.

Next, the LSTM layer outputs a score for each path embedding. An LSTM is a type of of recurrent neural network (RNN). An RNN is a neural network that takes in its previous output as part of its input. Therefore, each step within a network will contain information from all its preceding steps. These types of networks are useful for analyzing data whose components rely on previous information. Although RNNs can be helpful for constructing graphs that contain information about the dependencies and influence of sequences, they are ineffective when dealing with long sequences of information. This is because RNNs rely on using backpropagation (a recursive algorithm used to determine the prediction accuracy of a graph) to train and determine the dependencies of sequences. Therefore as you go back through the sequences within your network, the sigmoid functions used

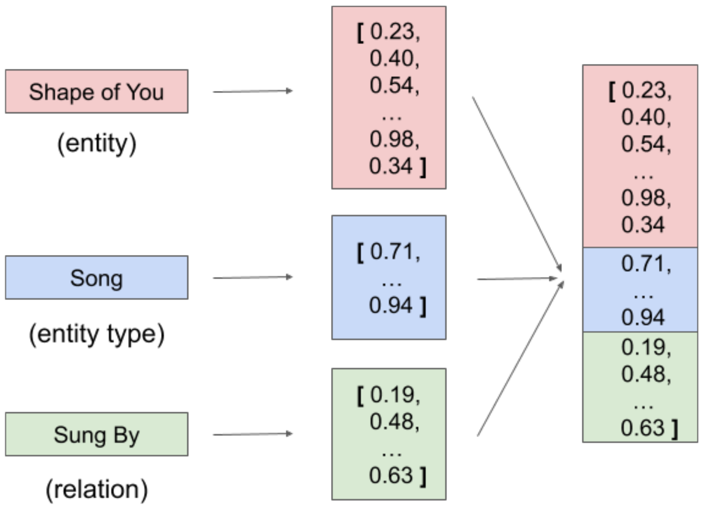


Figure 4: Triplet embedding example.

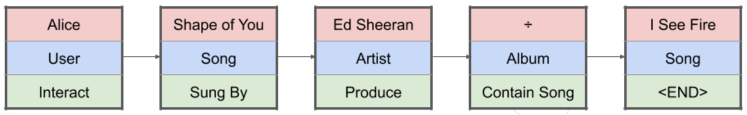


Figure 5: Path embedding example.

to determine the error signals for preceding sequences during back-propagation cause error signals to decrease exponentially (ie. the error signals stop providing useful information). LSTM's are one common way that is used to solve this issue. To simplify, whereas a standard RNN must remember every single sequence of information within a network, an LSTM gives an RNN the functionality of remembering and forgetting information by incorporating input, output, and forget gates throughout the network, thereby avoiding the problem with standard RNNs [3].

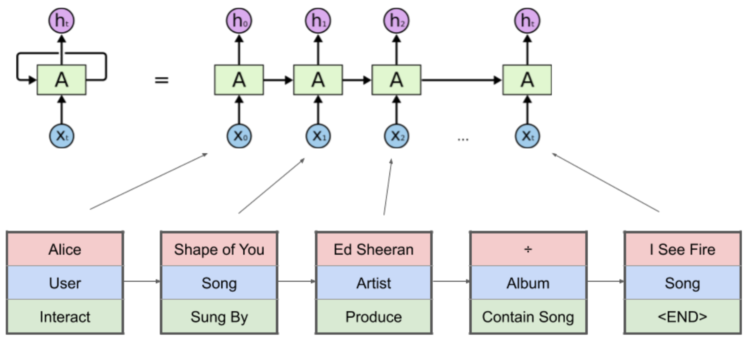


Figure 6: KPRN LSTM example (LSTM diagram from [3])

For each step of the path within this paper's knowledge graph, a triplet is taken as an input for the LSTM and at the end of the path, the LSTM will output a score for the entire path representing things learned so far (see Figure 6).

Then, between each user-song pair, there can be many paths, so the model needs to somehow combine their scores together. This is done via the weighted pooling layer which aggregates the LSTM output scores for each of the paths from u to s and get a single score indicating the plausibility of user u listening to song s.

$$g(s_1, s_2, ..., s_n) = log[\sum_{i=1}^{n} exp(\frac{s_i}{\gamma})]$$

The aggregation is expressed as the equation above, where the denominator γ is a hyperparameter. This weighted pooling aggregation is potentially more effective than a simple average. As [1] describes, the pooling is capable of giving more weight to more important paths, rather than treating all paths equally. Also note how different values of gamma result in different aggregation scores in Figure 7.

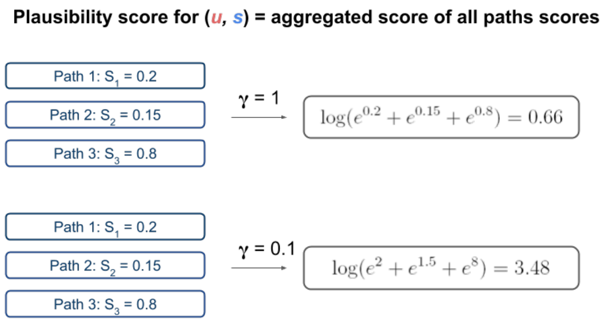

Plausibility score for (u, s) = aggregated score of all paths scores
Path 1: S1 = 0.2
Path 2: S2 = 0.15
Path 3: S3 = 0.8
γ = 1
$\log(e^{0.2} + e^{0.15} + e^{0.8}) = 0.66$
Path 1: S1 = 0.2
Path 2: S2 = 0.15
Path 3: S3 = 0.8
γ = 0.1
$\log(e^{2} + e^{1.5} + e^{8}) = 3.48$

Figure 7: Plausibility score examples

2.2 MF Baseline

The authors compared the KPRN model with an existing recommendation model MF. MF is matrix factorization with Bayesian personalized ranking (BPR) loss [4]. Matrix factorization is a common collaborative filtering approach for modeling recommendation. It works by decomposing a user-items interaction matrix into a product of two lower-dimensional matrices. One of the matrix would represent hidden features of the users and the other matrix would contain hidden features of the items. Matrix factorization with BPR loss is designed specifically to optimize for personalized ranking [4]. More specifically, BPR loss is defined as:

$$\text{BPR-OPT} := \sum_{(u,i,j)\in D_s} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta)$$

Θ stands for all the parameters in the model, and $p(\Theta)$ is a general prior density following a normal distribution with zero mean and variance-covariance matrix Σ_Θ. In each training step, the model trains via a gradient descent based on the loss function. As a result, different learning rate can affect how much the model parameters change.

$(u, i, j) \in D_s$ are (user, item, item) triplets that are in the training set. Furthermore, the user u interacts with item i but did not interact with item j or we are missing the interaction data between user u and item j in the training data. The MF model assumes that u enjoys i over j in both of the cases. They also defined $\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$, where $\hat{x}_{ui}$ and $\hat{x}_{ui}$ are entries in the product matrix of the two lower-dimensional matrices. According to the authors of the MF model, this design means that their criterion does not try to regress a single predictor for each entries in the product matrix but instead tries to classify the difference of two predictions $\hat{x}_{ui} - \hat{x}_{uj}$.

Since their training data takes into account every user's preference among items and they classifies the differences among each user's preferences, they are able to generate a score for how likely a user will enjoy one item relatively to other items. This way, the model can also output a ranking list of a user's preference.

Besides the methodological differences between matrix factorization and KPRN, the MF baseline solely uses user-item interaction whereas the KPRN model also uses auxiliary information of the items.

3 Our Implementation

In this section we go over details of our replication of the KPRN model and our work with the MF baseline. Source code for our project can be found at: `https://github.com/terwilligers/knowledge-graph-recommender`.

3.1 Dataset Choice

The paper conducted experiments using two sets of data, a combination of MovieLens-1M and IMDb (MI) for movie recommendation, and KKBox for music recommendation. The knowledge graph constructed from the song dataset is less dense compared to the knowledge graph constructed from the movie dataset. We decided to work with the KKBox music recommendation dataset on Kaggle [5] as it is more readily available. This dataset has two components: song characteristics and song interactions. For each song the song characteristics include information such as the song name, artist name, lyricist, and composer, while for each user the song interaction portion lists songs the user has listened to. These 2 components make up our knowledge graph.

When constructing the knowledge graph from this dataset, we attempted to replicate the approach of the original paper. The paper did not specify the exact information they used in the graph, so we constructed a tripartite knowledge graph using 3 kinds of nodes and 2 kinds of edges. The nodes represent the 3 entity types: user, song and artist, where artist name, lyricist, and composer are all grouped into the artist category. The edges represent the 2 bidirection relations: user-song and song-artist, which are the item characteristics and the user interactions respectively. These relations are stored using an adjacency list format. While we could have used more specific nodes and relations, we decided this would be enough for our replication project.

Statistics	**Sparse Subnetwork**	**Dense Subnetwork**	**KKBox**	**MI**
# of users	2754	3116	34K	6K
# of item	53K	144K	2M	4K
# of user-item interactions	81K	2.8M	3.7M	998K

Table 1: Statistics on the subnetworks and the original knowledge graphs.

3.2 Subnetworks

Due to limited computing power, we decided to train and test our model on a subset of the data by choosing subnetworks within the KKBox knowledge graph. The original paper compared KPRN's result on the MI dataset and the KKBox dataset, and MI is a dense subnetwork than KKBox. Therefore, we decided to find both a standard and a dense subnetwork, where we try to simulate the KKBox knowledge graph with the standard subnetwork and the MI dataset with the dense subnetwork. To find the standard subnetwork, we randomly selected 10% of each type of nodes and kept the edges connecting them. As for the dense subnetwork, we selected top 10% of each type of nodes by their degree and kept the edges connecting them. The vertices not connecting to other vertices in the subnetwork are removed, so the number of vertices remaining in the subnetwork is not exactly 10% of the original number of vertices. More specifically, both the standard and the dense subnetwork contain around the same number of vertices, but the standard subnetwork has 81 thousand edges and the dense subnetwork has 2.8 million edges. Due to the way we constructed the subnetwork, the average degree per node in the standard subnetwork ended up being smaller than that in the original KKBox knowledge graph.

3.3 Train Test Split

For training and testing, we performed the same procedure as the original paper. That is, we split the user-song interaction history into two sets: a training set and a testing set. For each user, we put a random 80% of their interaction history in the training set, and the other 20% of their interactions in the test set. Note, the artist-song connections were kept in full in both sets, since we did not explicitly evaluate on these.

Then when training, for each positive user-item interaction in the training set we sampled 4 items not interacted with by the user, adding these to our training data. This ensured we were learning from both positive and negative interactions. For testing on the other hand, for each positive user-item interaction we sampled 100 items not interacted with by the user. This allowed us to evaluate on each grouping of 101 items. When sampling these negative interactions, we only sampled from the subset of interactions that we found paths for in the path finding step. This was necessary since if we included negative interactions with no paths, problems would occur both with training and testing since the input to the model is the paths themselves. If there are no paths then the model has no information about an interaction, giving it no way to compute a score. Note, that if we just assigned a score of 0, our evaluation results would be artificially inflated.

3.4 KPRN Replication

After the dataset construction and train test split, the two major pieces needed to replicate the KPRN model were path finding and the neural network creation.

3.4.1 Path Finding and Negative Interaction Sampling

The original paper's KPRN model takes as input a sequence of paths between each user-item interaction. The path finding stage of the algorithm finds these paths from the Knowledge graph, and in our case from the subnetworks we created.

Just like the original paper we only found paths of length less than or equal to 6, since longer paths would introduce noisy input. There would be far more longer paths than shorter paths connecting two nodes, and a longer path would indicate less about listening behavior than shorter paths. Because of the tripartite structure of the graph this resulted in only paths of length 4 and 6. This allowed us to implement the path finding by using a two depth-first searches starting from each user in the graph: one to find paths of length 4 and one to find paths of length 6.

In addition, we made a few modifications to ensure timely computation. First, we performed a single depth-first search from each user rather than a search for each user-song interaction. Then, when this was still not fast enough, we added a branching factor of k at each layer of the depth first search. This meant that from a node in the graph, we would randomly sample k edges from that node to use as edges in our search. This was necessary since it was infeasible to find all paths of a certain length within the graph. We used a smaller branching factor for paths of length 6 than paths of length 4 since we would find up to k^4 paths of length 4 and k^6 paths of length 6. The branching factor was set to $k = 50$ for paths of length 4 when both training and testing, $k = 6$ for paths of length 6 when training, and $k = 10$ for paths of length 6 when testing. An increase of the length 6 branching factor was necessary when testing since for each positive interaction we had 100 negative interactions to find paths for compared to 4 when training.

3.4.2 Neural Network Construction and Training

For implementing the Embedding, LSTM, and Weighted Pooling components of the KPRN model we used version 1.2.0 of the Pytorch machine learning library, and trained and evaluated our model on Google Colab GPUs.

Our parameter values for training the KPRN model are shown in Table 2. We used the same interaction batch size and dimensions as the original paper. The model was optimized with Adaptive Moment Estimation (Adam). Hyper-parameters were found from searching combinations of the optimization learning rate in $\{.01, .02, .001, .002\}$ and L_2 regularization coefficient in $\{.01, .001, .0001, .00001\}$. Models were trained for 10 epochs for the standard subnetwork and 3 epochs for the dense subnetwork which had more interaction data. Further epochs appeared to have little effect on evaluation results.

3.5 MF Baseline

To compare against the MF baseline, we found an existing replication implementation of the MF model at `https://github.com/gamboviol/bpr`. We trained and tested the MF model using the

Parameter	**Value**
Batch size	256
Entity embedding dimension	64
Type embedding dimension	32
Relation embedding dimension	32
LSTM hidden dimension	256
Learning rate	.002
L_2 regularization coefficient	.0001

Table 2: KPRN parameters in our trained model.

same train/test data as our KPRN model. However, we included songs that were not listened to by any user in our training data even if they may not provide any useful information. They were included because these songs compose of some paths and are included in the KPRN model. More specifically, we trained on both the standard and the dense subnetwork with a learning rate of 0.05, and set the dimension of the two lower-dimensional matrices to be 64. We trained the MF model for 200 epochs on the dense subnetwork and for 600 epochs on the sparse/standard subnetwork. We stopped training once the loss converges, and it took more epochs for convergence when training on the sparse subnetwork. This is due to the fact that there are less training data in the sparse subnetwork than in the dense one, and loss can only go down when we traing for enough number of instances.

4 Experiments

In the following sections we explain the process of evaluating our model, present results from different KPRN variations, and discuss a baseline we constructed based on some interesting observations of paths within the graph.

4.1 Evaluation

To evaluate the performance of our implementations on the test dataset we used two different metrics: *hit*@*K* and *ndcg*@*K*. Both of these attempt to evaluate the accuracy of a model's top-K recommendations by comparing how well a model recommends a positive interaction among a group of negative interactions. Intuitively, we want our model to rank the positive interaction above the negative interactions, since that would indicate better recommendations.

As described above, when evaluating, each positive interaction for a user is paired with 100 negative interactions. Then for each K from 1 to 15, *hit*@*K* and *ndcg*@*K* are computed for each group of 101 interactions. For our use case these metrics are defined as:

$$hit@K = \begin{cases} 1 & \text{if positive interaction in top } K \text{ interactions} \\ 0 & \text{if positive interaction not in top } K \text{ interactions} \end{cases}$$

$$ndcg@K = \begin{cases} \frac{\log(2)}{\log(i+1)} & \text{if positive interaction in position } i \text{ of top } K \text{ interactions} \\ 0 & \text{if positive interaction not in top } K \text{ interactions} \end{cases}$$

$ndcg@K$ is similar to $hit@K$, but it takes into account the relative order of the interactions. So when computing $ndcg@15$, the positive interaction ranked first would result in a higher score than the positive interaction ranked tenth. A derivation of the formula can be found at [6].

For each K, the $hit@K$ and $ndcg@K$ results from all lists are averaged together to compute a final score. Each list is 1 positive interaction corresponding to a song a user listened to that is paired with 100 negative interactions.

For both these metrics assigning scores of 0 to interactions without paths would not suffice. We evaluated on a group of 101 interactions, and if some negative interactions had no paths then a score of 0 would place them at the bottom of the ranked list. This means no matter what score our model gave to the positive interaction, it would still be ranked above negative interactions with scores of 0. This would bias our results toward higher scores, so we decided to choose our 100 negative interactions from a set that we found paths for.

4.2 KPRN Results

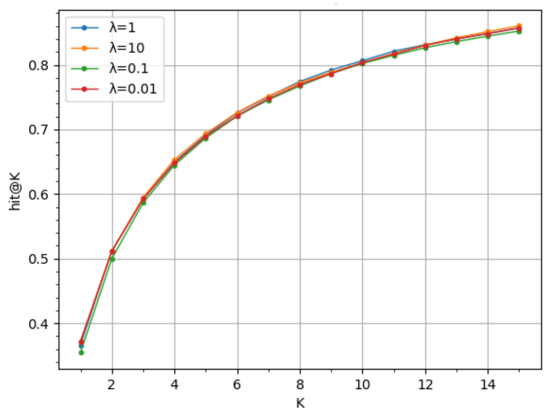
Here we present results from experiments we ran on our implementation of the KKPRN model. Note that unless specified otherwise we are using the standard subnetwork of the KKBox dataset.

4.2.1 KPRN

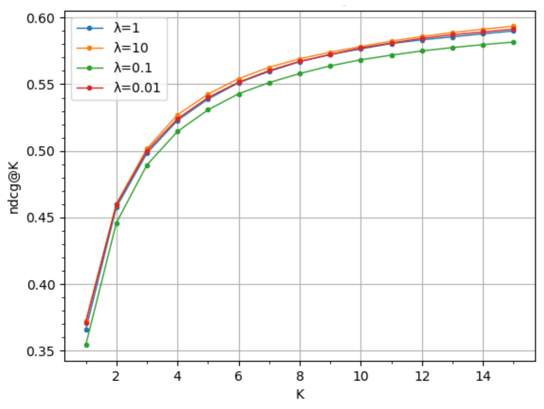
Figure 8 shows the $hit@K$ and $ndcg@K$ scores from our implementation of the KPRN model. Our results followed the same general trend as the paper's results, with slightly lower scores as seen in Figure 9. The lower scores can be attributed to our smaller subnetwork dataset. Like the original paper we experimented on $\gamma \in \{.01, .1, 1, 10\}$, but varying γ hardly affected our results. Thus results presented later are all for weighted pooling $\gamma = 1$. Moreover, throughout all our results $ndcg@K$ followed the same trend as $hit@K$, with slightly lower scores as expected.

4.2.2 KPRN vs MF

To investigate our KPRN model results further, we used an existing implementation of an MF baseline as described in Our Implementation section. A comparison of the paper's KPRN and MF models is presented in Figure 10a, and a comparison of our results is in Figure 10b. While our MF baseline was able to rank positive interactions high in the training dataset, it was unable to generalize to the test set. We tried training on 10 users, and got $hit@15 = 0.9$ when testing on the training set. However, when we tested on the test set, $hit@15$ was only 0.0003. Due to the poor generalization, the MF model created very low $hit@K$ results on the subnetworks, making comparisons difficult.



(a) Hit@K scores for our implementation



(b) Ndcg@K scores for our implementation

Figure 8: KPRN results from our implementation.

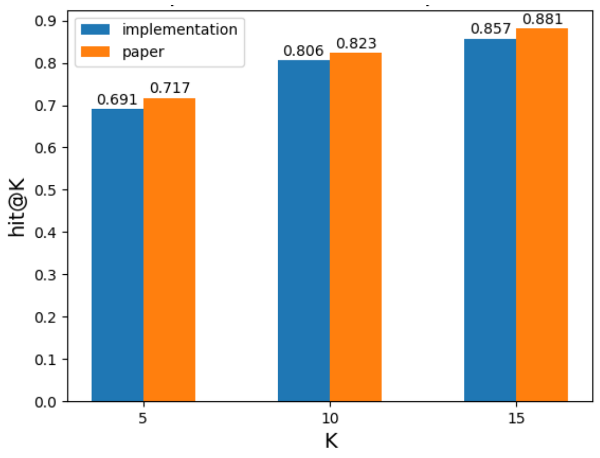


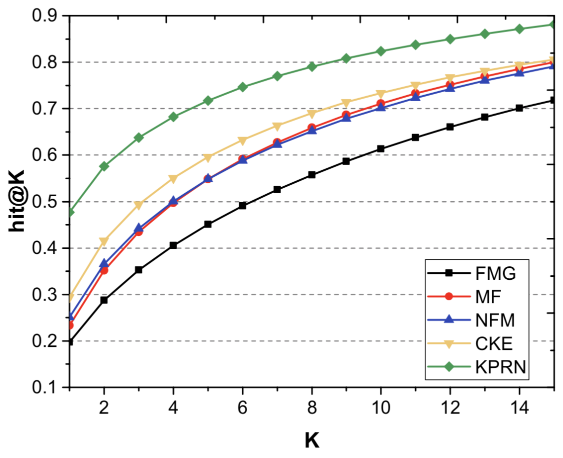
Figure 9: Comparison of our KPRN implementation vs paper's hit@K scores.

4.2.3 Without Relation

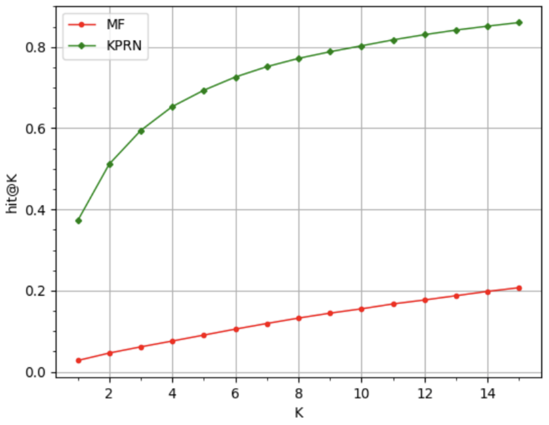
The paper also created and analyzed a variant of the KPRN model called KPRN-r. This was identical to the original model, but with the relation information removed from the LSTM input, resulting in only entity and type embeddings. The authors had found that this variant performed slightly worse than the original KPRN model, but the difference was almost negligible, especially for the KKBox dataset. We also implemented KPRN-r, and found no noticeable differences compared to the results of the original model as seen in Table 3.

Model	**hit@5**	**hit@10**	**hit@15**
KPRN	0.677	0.800	0.850
KPRN-r	0.672	0.798	0.851

Table 3: KPRN (with relation) vs KPRN-r (without relation



(a) Paper's KPRN and MF results



(b) Our KPRN and MF results

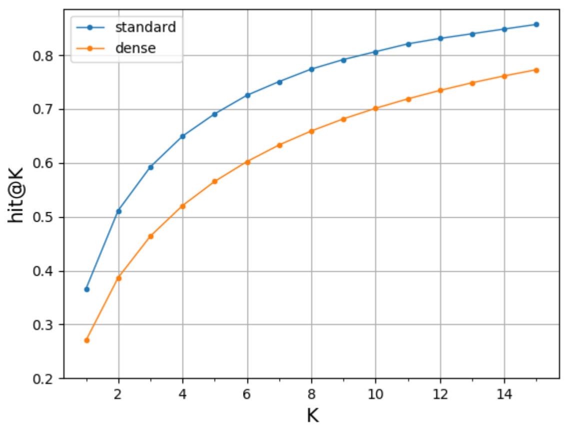
Figure 10: Matrix Factorization baseline model hit@K scores.

This lack of difference makes sense intuitively, since in our knowledge graph the information of a relation in a path would be entirely represented by the type embedding. For example, a path step from a user entity to a song entity inherently includes the underlining user-song relation. Thus the relation portion of the path contributes no extra information indicating that upon its removal scores would likely not decrease. Moreover, it is unclear whether the original authors used the same relations as we did. The KKBox dataset has information on the artist name, composer, and lyricist of a song, which we grouped together to form the song-artist connections. The original paper did not specify the exact relations they used for this dataset, and splitting artist into these three components may have the potential to increase the effectiveness of KPRN over KPRN-r.

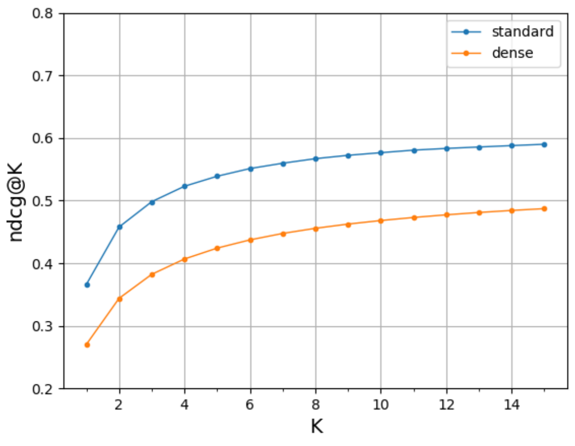
4.2.4 Dense Subnetwork

As mentioned previously, all the results above were for the standard subnetwork, constructed by randomly selecting 10 % of each type of node and keeping the edges connecting them. As described in the subnetworks section, we also constructed a dense subnetwork by selecting the top 10 % of each type of node by their degree and keeping connecting edges.

Figure 11 compares the two subnetworks. Across all K the results for the dense subnetwork were slightly lower than the results for the standard version. This follows the patterns seen in the original paper, since the KKBox dataset (denser) had higher results than the MI dataset (sparser) as seen in figure 12. We believe the larger number of edges in a denser graph may result in paths for positive interactions that are less detectable due to the larger amount of noise in the surrounding area. Detecting paths is important since our KPRN model's task is to figure out which paths are useful and what those paths tell us about an interaction occurring. If paths for positive interactions are more similar to those of negative interactions in the denser datasets, this could cause a decrease in scores.



(a) Hit@K scores



(b) Ndcg@K scores

Figure 11: Our dense vs standard subnetworks Results

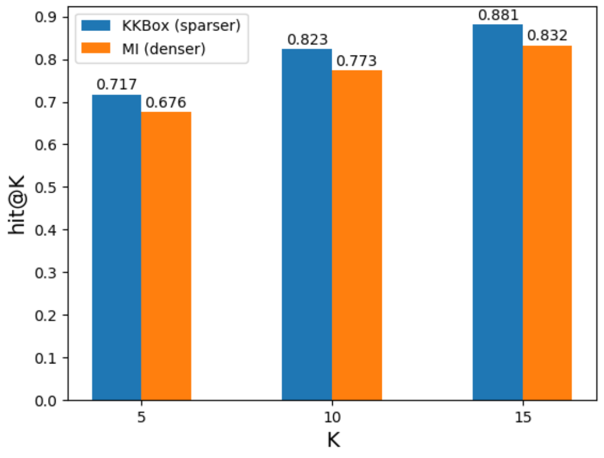


Figure 12: Paper's KKBox vs MI hit@K scores

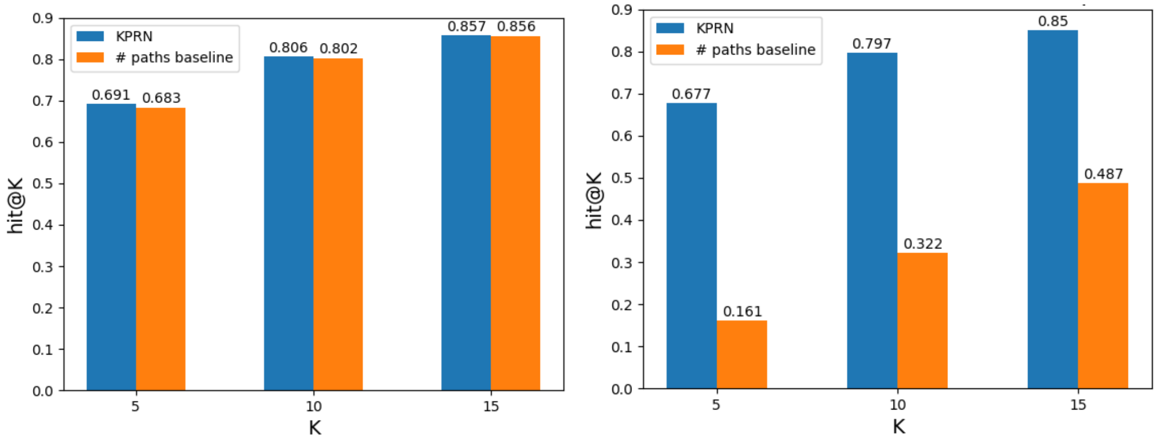
4.3 Number of Paths Baseline Extension

For the final piece of our project, we constructed and examined the results of a baseline we created that utilizes the number of paths between interactions. Since our Matrix Factorization baseline had trouble generalizing to the test set, we wanted to come up with an alternate baseline for comparing results. The inspiration for this model developed from some path-finding observations.

When path-finding we often struggled to find paths for negative interactions, leading us to sample negative ones from the set we found paths for. After looking at our path interaction data a bit closer we realized even for interactions with paths, on average we were finding far more paths for positive interactions than negative interactions. For example in the training set of the standard subnetwork we found on average 117.3 paths per positive interaction, and only 6.3 paths per negative interaction.

From these observations we decided to test out our own baseline which simply ranked each group of 101 interactions by merely the number of paths we found. The top ranked interaction would be the one with the most paths. As shown in Figure 13a the results of the baseline were quite surprising since they were nearly identical to the complicated KPRN model!

After seeing these results we wondered if the paper's KPRN model was simply learning the number of paths connecting an interaction, even though this parameter was not an input to the model. To investigate this further we changed the path-finding step to only sample 5 paths for each interaction as input into the model. While this would not fully remove the gap in numbers of paths (some negative interactions had less than 5), it would reduce any major differences. If the model was only learning things from the number of paths inputted we expected the results to decrease, but as shown in Figure 13b, the KPRN scores were similar to using all paths. As one would expect, the scores of the number of paths baseline were greatly reduced due to far fewer differences in the path count.



(a) Using all paths as input into the model **(b)** Using a sampling of 5 paths as input into the model

Figure 13: Number of paths baseline comparisons.

These results indicate that the KPRN model is at least able to learn something non-trivial from its input. Somehow, when ranking, the information it learns from a small subset of paths is roughly equivalent to the information the total number of paths gives the number of paths baseline. Whether this is more useful than the total number of paths remains uncertain.

5 Discussion

Overall, our replication of the KPRN model was mostly consistent with the results of the original paper. While our MF baseline performed poorly, our scores from a subnetwork of the KKBox dataset were almost as high as the paper's scores on the full dataset. Moreover, differences between sparser and denser datasets were similar between our implementation and the original paper, and varying parameters produced similar trends.

Still, there do appear to be limitations to the paper's KPRN model. First, the high results of our number of paths baseline indicate that the actually information learned by the model remains unclear. The model may be learning complex features of path characteristics or it may be just be learning the number of paths. The complexity of the neural network architecture makes this type of question hard to answer.

In addition to this, we're uncertain how useful our current model would be for actual recommendations. After our experiments we tested our model by choosing a user in our dataset and replacing their interactions with a different set of interactions. We observed that the model tends to recommend the same popular songs despite changes in the input interactions. With input interactions that are less common, the recommended songs become more personalized, but the very popular songs still ranks among the top of the recommended list. An important detail to note here is that in the input data we trained the model with, user interaction is listening history not user preference. Listening to a song doesn't necessarily indicates liking the song. Thus, if the model is to be put in real life, it is important to train the model with user preference data. Since the current model is performing quite well in predicting the songs users interacted with, there is the potential for it to perform well in predicting songs that users would really like if it is trained on this preference data.

To further explore the KPRN model in the future, we would like to test the model out on a wider variety of dataset types. These datasets should have a larger number of relation and entity types, since ours consisted solely of users, songs, and artists. This will allow us to validate the actual advantage of making use of the relations between the different entities. As mentioned above, it would also be helpful to include preference data instead of just interaction data. Then, it could be interesting to research and test out other methods of evaluation. Given preference data, these could encompass both the ranking of a liked item, and the ranking of items a user especially dislikes. Performing these types of studies would continue to help us understand how well models such as the KPRN can generalize, and hopefully lead to more replicable work in the future.

References

[1] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5329–5336, 2019.

[2] Word embeddings: Encoding lexical semantics. `https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`.

[3] Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

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

[5] KKBOX. Wsdm - kkbox's music recommendation challenge, 2018. `https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data`.

[6] Ranking metrics. `http://ethen8181.github.io/machine-learning/recsys/2_implicit.html`.