CS303 Artifitial Intelligence 2024F Project3 Report

Ben Chen
chenb2022@mail.sustech.edu.cn

December 6, 2024

1 Introduction

1.1 Problem Description

A recommendation system infers the preference of a user based on the historical data and relevant information of the user. More specifically, recommendation systems are score functions that calculate the probability of a user liking an item, according to the user's historical interaction data and the item's information. In this project, we will put more attention on the knowledge graph, i.e., the hybrid of user-item interaction history and item features, to achieve better recommendation performance.

1.2 Purpose

The purpose of this project is to implement a recommendation system based on the knowledge graph. We will use the user-item interaction history and item features to predict the probability of a user liking an item. We will use the interaction record to model the user-item interaction history, along with additional knowledge graph and train a neural network to learn the representation of the user and item. We will use the learned representation to predict the probability of a user being interested in some item and the top-k items that the user may like.

2 Preliminary

Given a interaction record Y_{train} with postive and negative samples, and a knowledge graph $G = (V, E)$, we need to train on interaction record and optimize the accuracy with additional information provided by the knowledge graph. For each record in Y_{train}, we have

$$y_{uw} \in \{0, 1\}, \quad u \in U, w \in W$$

where U is the set of users and W is the set of items, to indicates whether user u likes item w.

The knowledge graph G contains the information of the items, which can be represented as a set of triples (u, r, w), where u is the user, r is the relation, and w is the item. The relation r can be encoded as a weight value to indicate the importance of the relation.

In the first situation, we need to predict the click-through rate, which ideally the interest of the user to the item. It will be evaluates using the AUC metric. The AUC, Area Under Curve metric is defined as follows. Given a test sets with postive sample S and negative sample S',

$$\text{AUC} = \frac{\sum_{s \in S, s' \in S'} I(s, s')}{|S| \times |S'|}$$

where function I is defined as

$$I(s, s') = \begin{cases} 0, & f(s) < f(s') \\ 0.5, & f(s) = f(s') \\ 1, & f(s) > f(s') \end{cases}$$

And in the second situation, we need to predict the top-k items that the user may like, which is instead of given a scalar between 0 and 1, to emit a vector to indicate the top k items the user might be interested in, which will be evaluated using the nDCG@k metric. The nDCG@k, normalized Discounted Cumulative Gain metric is defined as follows. Given a test sets with a list of users with $|U| = l$ and a postive item set S, and the score function returns $M = \mathbb{R}^{l \times k}$

$$\text{nDCG@k} = \frac{1}{l} \sum_{i=1}^{l} \frac{\text{DCG}_i\text{@k}(S_i, M_i)}{\text{iDCG}_i\text{@k}(S_i)}$$

where $\text{iDCG}_i\text{@k}(S_i)$ is the ideal DCG@k of the i-th user, and $\text{DCG}_i\text{@k}(M_i, S_i)$ is the DCG@k of the i-th user. The DCG@k is defined as

$$\text{DCG}_i\text{@k}(S_i, M_i) = \sum_{j=1}^{k} \frac{I(S_i, M_{ij})}{\log_2(j+1)}$$

$$\text{iDCG}_i\text{@k}(S_i) = \sum_{j=1}^{\min(k, |S_i|)} \frac{1}{\log_2(j+1)}$$

and the function I is defined as

$$I(S_i, M_{ij}) = \begin{cases} 1, & M_{ij} \in S_i \\ 0, & \text{otherwise} \end{cases}$$

Our job, generally speaking, is to design a score function $f(u, w)$ on the input u as user and w as item, to predict the interest level $y_{\text{test}} \in [0, 1]$ to maximize two metrics in two real-world situation.

1. Maximize the AUC metric on score function f

$$\max_f \text{AUC}(f, Y_{\text{test}})$$

2. Maximize the nDCG@k metric on score function f with $k = 5$

$$\max_f \text{nDCG@5}(f, Y_{\text{test}})$$

3 Methodology

To achieve the goal, we generally will follow the following steps

Algorithm 1: Training

```
Input: epoch_num: Number of epochs, output_log: Boolean for logging progress (default False)
Output: Model
Function Train(epoch_num, output_log):
    Initialize Adam optimizer with learning rate learning_rate and weight decay weight_decay;
    for epoch = 1 to epoch_num do
        train_batches ← Get training batches from dataloader;
        Initialize an empty list losses;
        foreach batch in train_batches do
            loss ← Optimize with current batch;
            Update model parameters by performing backpropagation and optimizer step;
            Append loss to losses;
        end
        if output_log then
            Print average loss for this epoch;
        end
    end
```

1. Preprocess the data. We will encode the data to our representation in code, i.e., construct the knowledge graph.
2. Train the model. We will train the model on the training data, and calculate the loss function to optimize the model.
3. Evaluate the model. We will evaluate the model on the test data, and calculate the AUC and nDCG@5 metric.
4. Tune the hyperparameters. We will tune the hyperparameters to achieve better performance, i.e., epoch, learning rate, batch size, etc.
5. Submit the result. We will submit the result to the OJ server to evaluate the performance.

Firstly, encoding the knowledge graph is a trivial process as it can easily be done by (1) mapping the relation to weights and (2) constructing the graph. The next step is to randomly split the datasets into batches and train the model. The training algorithm is shown in the above algorithms. Here, we adapt a nerual network with forward pass and Adam optimizer to optimize the model.

Algorithm 2: Optimization

```
Input: pos: Positive samples, neg: Negative samples
Output: loss: Calculated loss
1 Function Optimize(pos, neg):
2     pos_score ← Forward pass for pos;
3     neg_score ← Forward pass for neg;
4     pos_matrix ← Multiply pos_score by transpose of ones matrix with same size as neg_score;
5     neg_matrix ← Multiply neg_score by transpose of ones matrix with same size as pos_score;
6     loss ← Mean of $(\max(\text{neg\_matrix} - \text{pos\_matrix} + \text{margin}, 0))$;
7     return loss
```

The details of Adam optimizer is ignored here since it's not much relevant to the project. The subprocesses are shown below.

Algorithm 3: Forward Pass

```
Input: head: List of head entities, rel: List of relations, tail: List of tail entities
Output: score: Similarity score of the input triple
1 Function ForwardPass(head, rel, tail):
2     head_emb ← Embedding of head;
3     tail_emb ← Embedding of tail;
4     rel_emb ← Embedding of rel;
5     if l1 then
6         score ← $\sum |(\text{head\_emb} + \text{rel\_emb}) - \text{tail\_emb}|$;
7     end
8     else
9         score ← $\sum ((\text{head\_emb} + \text{rel\_emb}) - \text{tail\_emb})^2$;
10    end
11    return −score
```

4 Experiments

In experiments, we adapt a compound testing methods to evaluate the effectiveness of the algorithms, which is to train the model locally and test remotely. For local testing, we use the following environment:

And for remote testing, we use the following environment:

Model	MacBook Air M3 13'
CPU	Apple M3 4E + 4P 2.4GHz-3.7GHz
Memory	LPDDR5-6400 8GB Unified Memory

Model	Online Judge
CPU	Intel Xeon E5-2680 2.2GHz * 2
Memory	8GB DDR4 2666MHz

Because the test data is currently unavailable, we will be using the parts of training data as test data to evaluate the performance of the model. The hyperparameters will be tuned locally and the result will be submitted to the OJ server to evaluate the performance. In the training dataset, there are 26,638 positive samples and 24,037 negative samples, and 6729 nodes and 20195 relations in the knowledge graph.

Algorithm 4: Predict CTR

```
Input: eval_batches: List of evaluation batches
Output: scores: Array of evaluation scores
Function CTR_Eval(eval_batches):
    foreach batch in eval_batches do
        batch ← transpose of batch;
    end
    Initialize empty list scores;
    foreach batch in eval_batches do
        Initialize rel as list of 'feedback_recsys' relations for each element in the batch;
        score ← Forward pass using batch[0] + number of entities, rel, and batch[1];
        Append score to scores;
    end
    scores ← Concatenate all scores in scores along axis 0;
    return scores
```

4.1 Task 1

The evaluation metric of AUC is described previously. The l1 norm is enabled to prevent overfitting. The batch size and other three hyperparameters remains the same is it doesn't show much difference in accuracy locally.

Table 1: Identical hyperparameters in model

Batch Size	Evaluation Batch Size	Negative Rate	Dimension of Embedding
256	1024	1	128

The results shown below are the result of the model with various hyperparameters.

Table 2: Result with hyperparameters in AUC

Learning Rate	Weight Decay	Epoch Number	Margin	Time Cost	AUC Score
2e-3	1e-4	25	70	80.68s	0.688
2e-3	1e-4	30	70	86.70s	0.686
1e-3	5e-3	30	70	88.23s	0.704

It clear that number of epoch does not significantly affects the performance of the model and over-trained model may even lead to worse accuracy. Based on our observation of the training data, our intuitive gives that the model is likely to have been over-trained. So to boost accuracy, the model together with less learning rate and more weight decay will probably lead to better performance, which is proven in the result.

4.2 Task 2

The metric is also defined in the previous section and the hyperparameters are the same as in Task 1. The The result is shown below, from which we can see that similar to the result in Task 1, the model with less learning rate and more weight decay will lead to better performance.

Table 3: Result with hyperparameters in nDCG@5

Learning Rate	Weight Decay	Epoch Number	Margin	Cost	nDCG@5 Score
2e-3	1e-4	25	70	80.68s	0.181
2e-3	1e-4	30	70	86.70s	0.178
1e-3	5e-3	30	70	88.23s	0.182

5 Conclusion

In overall sense, the knowledge graph-based recommendation system has the significant advantages:

Rich information source The knowledge graph provides a comprehensive perspective of the user-item interaction history and item features, which is better and highly explainable.

Algorithm 5: Predict Top-k

```
Input: users: List of users, k: Number of top items to return (default 5)
Output: sorted_list: List of top-k items for each user
1  Function TopK_Eval(users, k):
2      item_list, train_user_pos_item ← Get positive item list from dataloader;
3      Initialize empty list sorted_list;
4      foreach user in users do
5          Initialize head as the user ID + number of entities for each item in the item list;
6          Initialize rel as list of 'feedback_recsys' relations for each item in the item list;
7          tail ← item_list;
8          scores ← Forward pass using head, rel, and tail;
9          score_ast ← Sort scores in descending order;
10         Initialize empty list sorted_items;
11         foreach index in score_ast do
12             if length of sorted_items ≥ k then
13                 break;
14             end
15             if user is not in train_user_pos_item or item at item_list[index] is not in
                 train_user_pos_item[user] then
16                 Append item_list[index] to sorted_items;
17             end
18         end
19         Append sorted_items to sorted_list;
20     end
21     return sorted_list
```

Serendipitous recommendation The KG-based RS possesses the relation graph with long-tail information, which enables the discovery of new item for users, while the plain RS may only recommend popular items.

Through this project, I acquired the knowledge of recommendation system and the benefits of knowledge graph in recommendation system and ways to embed the optimization with it. By mapping the user-item relation, we may achieve a better recommendation in small-scale dataset. In the future, we may explore the real-world knowledge graph-based recommendation system and its application in large-scale dataset.