SUSTech CS303 2023 Fall Project3 Report

Chunhui XU

I. Introduction

A. Problem Introduction

The problem studied in this project is the design of a Knowledge Graph(KG)-based Recommender System. This concept originates from the field of information filtering systems, which aim to predict and show user preferences for various items such as books, movies, or music.

A KG-based Recommender System can be characterized as a system that utilizes knowledge graphs as auxiliary information to enhance the accuracy and explainability of recommendations. Knowledge graphs are heterogeneous graphs that represent entities and relationships between them. By integrating user and item information, these systems can capture the relationships between users and items, as well as user preferences, more accurately.

B. Purpose

The purpose of this project is to design an algorithm that can construct a KG-based Recommender System. The algorithm should be able to predict whether a user would be interested in an item that they have not encountered before, based on their previous likes and dislikes, as well as the knowledge graph. The performance of the algorithm will be evaluated based on its accuracy in making predictions.

This project has real-world applications in various domains such as online streaming platforms, and content recommendation systems. By providing personalized recommendations, KG-based Recommender Systems can enhance user experience and engagement, leading to increased user satisfaction and potentially higher sales or user retention for businesses. Also, it allows us to have a practical understanding of knowledge graphs.

II. Preliminary

A. General Formulation (From Requirement Documentation)

Given an interaction record set Y_{train}, and a knowledge graph $\mathcal{G} = (V, E)$. For each interaction record $y_{uw} \in Y_{train}$, $u \in U$, $w \in W$, where U is the user set and W is the item set, it is a 0/1 value that represents whether the user is interested in the item or not, 1 means interested in and 0 means not. $\mathcal{G} = (V, E)$ is a knowledge graph about the items, V is the entity set and E is the relation set, which means that the entities in V are items and something that is related to the items, and the relations in E describe the relationship between the items or their attributes. Based on the given Y_{train} and $\mathcal{G}$, we are asked to design a recommender system with a score function $f(u, w)$, which is used to predict the interest level from user u to item w, the higher score means the higher interest level.

There are two tasks in this project we need to do:

1) Maximize the AUC metric of the score function $f(u, w)$ on a test dataset Y_{test}, i.e.,
$$\max_f \text{AUC}(f, Y_{test})$$
2) Maximize the nDCG@k metric of the score function $f(u, w)$ on a test dataset Y_{test}, while $k = 5$, i.e.,
$$\max_f \text{nDCG@5}(f, Y_{test})$$

B. IO Ananlyze

First, we need input:

- Interaction record set Y_{train}
- Knowledge graph $\mathcal{G} = (V, E)$

Then the requirement is to find a recommender system with a score function $f(u, w)$, where u is the user and w is the item.

Then we should:

In Click-Through-Rate (CTR) prediction task, we need to find the possibility of a user is interested in a item.

In the Top-k recommendation task, we need to find the Top k favorite item of the l users from U_{test}. We need to return a $l \times k$ matrix, indicate the top favorite result of each user.

III. Methodology

A. Workflow

In this problem, we have these steps to do:

1) Read the data and pre-process the data so that the relationship and data structure of knowledge graph can be established.
2) For each iteration, the training data is processed to obtain a batch of data, training is performed based on this batch of data, the loss is calculated, and the training loss is back-propagated. In this step, use Adam optimizer.
3) Repeat iterations for training.
4) For the trained model, each time data is input, give the corresponding score or top k ranking list.
5) Evaluate the model by AUG or nDG@k.

B. Algorithm

In this training process, we mainly use Adam Algorithm in `PyTorch`, as Algorithm 1.

C. Algorithm Analyze

To be honest, because the Adam algorithm is relatively complex, the mathematical analysis about it is retrieved from the Internet:

Assume that the time complexity of computing the gradient is O(g), where g is the number of parameters. In the Adam

Algorithm 1 Adam Optimizer

Require: γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective), λ (weight decay), $amsgrad$, $maximize$

$m_0 \leftarrow 0$ (first moment)
$v_0 \leftarrow 0$ (second moment)
$\widehat{v_0}^{max} \leftarrow 0$
for $t = 1$ to $\ldots$ **do**
  if $maximize$ **then**
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  end if
  if $\lambda \neq 0$ **then**
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  end if
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$
  $\widehat{v_t} \leftarrow v_t / (1-\beta_2^t)$
  if $amsgrad$ **then**
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / (\sqrt{\widehat{v_t}^{max}} + \epsilon)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon)$
  end if
end for
return θ_t

algorithm, batch gradient descent (Batch Gradient Descent) or mini-batch gradient descent (Mini-batch Gradient Descent) is usually used to calculate gradients. For batch gradient descent, each calculation of the gradient requires traversing the entire training set, so the time complexity is O(ng), where n is the size of the training set. To sum up, the time complexity of calculating the gradient can be O(ng).

IV. Experiments

A. Task 1

1) Evaluate Metrics: In AUC calculation, s. Suppose the samples of the test dataset T_{test} can be split to positive set S and negative set S' according to the ground truth, while all the samples in S are positive samples and all the samples in S' are negative samples. For a sample s, the score predicted will be marked as score(s), then we can define the calculation process of AUC is shown below:

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

where the function I is:

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

1) Your Experimental results:
2) Try to find through the experiments:
- the effect of different models (if any) or algorithms
- the effect of hyperparameters (if any).
- Analyze the effect of different algorithms/models and hyperparameters if you have corresponding experiments.

B. Task 2

1) Evaluate Metrics: Suppose for the i-th user in U_{test}, according to the ground truth, there is a positive item set S_i, which contains all the items that the user u is interested in.

Assume the return matrix is M, we can define the calculation process of nDCG@k as shown below and k is set to 5 in this project:

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

where $\text{iDCG}_i@k$ is the theoretical maximum value of $\text{DCG}_i@k$ when the ground truth for the i-th user is S_i. And $\text{DCG}_i@k$ is defined as

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

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

and $I(S_i, M_{ij})$ is:

$$I(S_i, M_{ij}) = \begin{cases} 1 & M_{ij} \text{ is in } S_i \\ 0 & M_{ij} \text{ is not in } S_i \end{cases}$$

C. Results

My final version until Dec. 26th has the evaluate value for demo:

- *batch_size*: 256
- *eval_batch_size*: 1024
- *neg_rate*: 1
- *emb_dim*: 128
- *l1*: True
- *margin*: 70
- *learning_rate*: 5e-3
- *weight_decay*: 1e-2
- *epoch_num*: 30

Result in demo: (0.81801, 0.090065)

- AUG:
-

D. hyperparameters

Origin (0.63591, 0.010230):

- *batch_size*: 256
- *eval_batch_size*: 1024
- *neg_rate*: 1
- *emb_dim*: 128
- *l1*: False

- *margin*: 15
- *learning_rate*: 1e-4
- *weight_decay*: 0
- *epoch_num*: 30

In this section, I change one hyperparameter once and give the result in (AUC, nDCG@k) from demo.

- *l1*: True (0.65162, 0.020763)
- *neg_rate*: 1 (0.66344, 0.019459)
- *leaning_rate*: 1e-3 (0.68068, 0.011561)
- *weight_decay*: 1e-3 (0.60451, 0.005703), 1e-5(0.63496. 0.007695)
- *margin* 50 (0.65147, 0.018283), 70(0.65933, 0.024075)

Use the absolute sum for score is better than square, same as the project 2 subtask 2, Euclidean distance may not be the best solution.

If I reduce the negative rate, the system will be better. It have more information about positive sample, it is suitable for a remommender system.

Weight decay will cause the model to decline. It may be a problem with the number of iterations, resulting in the model having no chance of over-fitting and more possibilities of under-fitting.

Higher margin have better accuracy in nDCG@k, because it appropriately increases the sensitivity of the data and makes the loss function more accurate.

But these are just discrete analyses. When the parameters are combined together, there will be different effects. For example, a higher number of iterations combined with weight deca'yay will make the results more accurate.

V. Conclusion

A. Model Comment

Overall, the KG-based RS has the significant advantages:

- **Ability to incorporate diverse data sources**: Knowledge graphs can integrate data from various sources. This allows for a more comprehensive understanding of users' preferences and item characteristics, leading to better recommendations.
- **Support for serendipitous recommendations**: Knowledge graphs can capture long-tail and niche information about items and users. This enables the discovery of unexpected and serendipitous recommendations that may not be apparent in traditional recommender systems.

B. Further thoughts

By studying the combination of knowledge graph and recommender system, I realized that knowledge graph can provide richer semantic information for the recommender system. By matching user and item attributes with entities and relationships in the knowledge graph, items that match the user's interests and needs can be more accurately recommended.

Maybe we can explore how to use technologies such as natural language processing and machine learning to automatically build and update knowledge graphs from large-scale text data to improve its coverage and practicality. For example, if a new relation appeared, how can we add the new head-relation-tail chain for the existed graph, that is, the recommender system can automatically extract deeper information from the knowledge graph.