Project 3 Recommender System Report

Name: 钟志源
SID: 12110517

Introduction

A recommender system is an information filtering system that predicts and offers personalized suggestions for items such as books, movies, or music, relying on user preferences derived from browsing and selection histories.

KG-based recommender systems utilize knowledge graphs as supplementary data to enhance recommendation accuracy and explainability. Knowledge graphs, consisting of heterogeneous graphs with nodes symbolizing entities and edges denoting relationships, illustrate item relationships and attributes. They integrate user-side information, depicting precise user-item relationships and preferences.

In this project, we are required to design an algorithm for a Knowledge Graph (KG)-based Recommender System. It aims to predict a user's interest in previously unencountered items by leveraging the user's historical preferences toward items and incorporating information from a knowledge graph.

Preliminary

Problem Formulation

Given an interaction record set $\boldsymbol{Y}$, and a knowledge graph $\mathcal{G} = (V, E)$. For each interaction record $\boldsymbol{y} \in \boldsymbol{Y}$, $u \in \boldsymbol{U}$, $w \in \boldsymbol{W}$, where U is the user set and W is the item set, it uw $train$ 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 and the relationship between the items or their attributes. Based on the given $\boldsymbol{Y}$ and $\mathcal{G}$, we are asked to design a recommender system with $train$ 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:

- Maximize the AUC metric of your score function $f(u, w)$ on a test dataset Y_{test}, i.e.,
$$max_f AUC(f, Y_{test})$$
- Maximize the $nDCG@k$ metric of your score function $f(u, w)$ on a test dataset Y_{test}, i.e.
$$max_f nDCG@5(f, Y_{test})$$

Methodology

General Workflow

The proposed method is divided into *Data Preparation*, *Model training*, *Model evaluation*. *Data Preparation* involves random sampling, index shifting and negative data generation. *Model training* involves Embedding and forward/back propagation. *Model evaluation* involves AUC(ctr_eval) and nDCG@5.(topK_eval).

Algorithm Design

Class: Dataloader

Initialization

- Inputs: `train_pos`, `train_neg`, `kg_lines`, `train_batch_size`, `neg_rate`
- Initialize KG, entity dictionary, and entity count.
- Process training data, prepare entities, and load known interactions.
- Prepare negative examples and initialize variables.

Methods:

- `_add_recsys_to_kg()`
  - Adds interaction data as extra relations to the KG.
- `_load_ratings()`
  - Loads known interaction data, offsets user indices.
- `_convert_kg()`
  - Loads KG data, converts relation types to integers.
- `get_user_pos_item_list()`
  - Retrieves known positive items for each user, sample extra negative training samples.
- `get_training_batch()`
  - Prepares positive and negative training batches from the KG.
  - Generates negative samples based on tails and heads of existing facts.
  - Splits data into batches based on the training batch size.

Class: TransE

Initialization

- Inputs: `ent_num`, `rel_num`, `dataloader`, `dim`, `l1`, `margin`, `learning_rate`, `weight_decay`, `device_index`
- Initializes entity and relation embeddings using Embedding layers.

Methods:

- `forward(head, rel, tail) -> torch.Tensor`
    - Computes score based on entity embeddings and relation embeddings.
- `optimize(pos, neg) -> loss`
    - Calculates Margin Loss for positive and negative samples.
- `ctr_eval(eval_batches) -> scores`
    - Evaluates scores for given batches (feedback data).
- `top_k_eval(users, k) -> sorted_list`
    - Generates top-k recommendations for users based on their known positive items.
- `train_TransE(epoch_num, output_log)`
    - Trains the TransE model using Adam optimizer for a specified number of epochs.

Hyperparameters:

- `batch_size`: Batch size for training.
- `eval_batch_size`: Batch size for evaluation.
- `neg_rate`: Ratio of negative samples to positive samples.
- `emb_dim`: Embedding dimension.
- `l1`: Boolean indicating L1 or L2 norm usage.
- `margin`: Margin for the loss function.
- `learning_rate`: Learning rate for optimization.
- `weight_decay`: Weight decay for regularization.
- `epoch_num`: Number of epochs for training.

Class: KGRS

Methods:

- `__init__(train_pos, train_neg, kg_lines)`:
    - Initialize the KGRS algorithm with training data and knowledge graph lines.
- `training()`:
    - Train the recommendation system.
- `eval_ctr(test_data) -> np.array`:
    - Evaluate the Click-Through Rate (CTR) task based on test data, returning the predicted interest level.
- `eval_topk(users, k) -> List[List[int]]`:
    - Evaluate the Top-K recommendation task for specified users, returning recommended items sorted by user interest.

Analysis:

Optimality:

The model's performance heavily depends on hyperparameters like embedding dimensions, margin, learning rate, and weight decay. Tuning these parameters significantly impacts the model's performance.

Complexity:

Larger embedding dimensions may increase complexity but might capture finer relationships.

Performance Factor

- Batch Size: Influences the number of samples processed per iteration during training. Larger batch sizes can expedite training but might require more memory.
- Evaluation Batch Size: Similar to training batch size, but used for evaluation.
- Negative Sampling Rate (neg_rate): A higher rate increases the number of negative samples relative to positive ones. This parameter significantly impacts model learning by affecting the balance between positive and negative examples.
- Embedding Dimension (emb_dim): Represents the size of the latent space. Higher dimensions capture more intricate relationships but might increase computational complexity, and also overfitting.
- L1 Norm (l1): Determines whether the model uses L1 or L2 norm for distance calculation.
- Margin: Affects the margin used in the margin-based ranking loss. It controls the threshold where the model starts penalizing scores.
- Learning Rate: Governs the step size taken during optimization. Too high might lead to instability, while too low might slow convergence.
- Weight Decay: Controls overfitting by penalizing large weights in the model. An optimal value helps in generalization.
- Epoch Number: The number of times the algorithm goes through the entire dataset during training. It impacts how well the model fits the data and affects convergence.

Experiment

Task 1:

- AUC Metric:
  Measures the model's ability to rank positive examples higher than negative examples. Indicates the model's ability to discriminate between positive and negative samples.

Experiment Results:

- model: TransE
- Effect of Hyperparameters: Higher Embedding Dimension may increase AUC on local training, but may not be the case on the test set. Higher Margin and higher neg_rate helps the model to achieve higher AUC on the test set. Set decay_rate larger than 0 help the model to prevent overfitting.

Task 2:

- nDCG@5 Metric:
  Measures the ranking quality of a recommendation system. Evaluates how well the model ranks items by considering the graded relevance of the top 5 recommendations.

Experiment Results:

- model: TransE
- Effect of Hyperparameters: Similar to AUC, since the ranking is based on the score.

Final Hyperparameters

- `batch_size`: 256
- `eval_batch_size`: 1024
- `neg_rate`: 3.5
- `emb_dim`: 16
- `l1`: True
- `margin`: 35
- `learning_rate`: 3e-3
- `weight_decay`: 7e-4
- `epoch_num`: 36

Conclusion

Model Assessment:

- Advantages: TransE leveraged knowledge graph embeddings effectively, providing structured representation learning for recommendation systems.
- Disadvantages: The model performance is sensitive to hyperparameters and datasets. Certain models might struggle with scalability or handling diverse and sparse user-item interaction patterns.

Final Thoughts:

- `batch_size`: unchanged
- `eval_batch_size`: unchanged
- `neg_rate`: 2 -> 3.5, since more negative samples help the model to discriminate between positive and negative samples, because the dataset is sparse.
- `emb_dim`: 128 -> 16, a smaller embedding dimension is enough and helps to prevent overfitting.
- `l1`: False -> True
- `margin`: 15 -> 35, a higher margin helps the model to discriminate between positive and negative samples.
- `learning_rate`: 1e-4 -> 3e-3,a higher learning rate helps the model to converge faster and avoid local minimum.
- `weight_decay`: 0 -> 7e-4, a proper weight decay helps the model to prevent overfitting.
- `epoch_num`: 30 -> 36, a higher epoch_num helps the model to converge.