KG-Based Recommender System

Hok Lay Heng

December 22, 2024

1 Introduction

A recommender system is a specialized tool designed to help users discover content that aligns with their personal tastes. It works by analyzing a user's past behavior, such as their browsing habits or selection history, and then predicting which items—like books, movies, or music—might appeal to them. The goal is to provide personalized suggestions that are relevant to the individual, enhancing their experience by introducing them to new content they are likely to enjoy. In the context of this project, the challenge is to develop an algorithm that leverages a Knowledge Graph (KG) to power a recommender system, offering a more informed approach to predicting a user's preferences.

In this project, the recommender system will be based on the principles of a Knowledge Graph (KG), which represents relationships and attributes of items in a structured way. The algorithm needs to predict whether a user would be interested in a new item they have not interacted with yet, using their historical preferences. This involves analyzing both the items they have previously liked or disliked and the connections between these items within the KG. By utilizing the KG's rich structure, the system can make more accurate recommendations, taking into account not just direct preferences but also the broader context of item relationships and user behavior.

2 Preliminary

Given a set of interaction records $\mathcal{Y}$ and a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the goal is to model user-item interactions. For each interaction record $\mathbf{y} \in \mathcal{Y}$, involving a user $u \in \mathcal{U}$ and an item $w \in \mathcal{W}$, where $\mathcal{U}$ represents the set of users and $\mathcal{W}$ denotes the set of items, the variable t_{train}^{uw} is a binary value (0/1). A value of 1 indicates that the user is interested in the item, while 0 signifies no interest. The knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ encapsulates information about items, where $\mathcal{V}$ is the set of entities (items and related attributes) and $\mathcal{E}$ represents relationships among them, describing connections between items or their features. Using the provided $\mathcal{Y}$ and $\mathcal{G}$, the objective is to design a recommender system with a scoring function $f(u, w)$, which predicts the degree of interest a user u has in an item w. Higher scores indicate stronger interest.

This project requires optimizing the performance of the scoring function $f(u, w)$ on two key evaluation metrics:

- Maximize the AUC (Area Under the Curve) metric of the score function $f(u, w)$ on a test dataset $\mathcal{Y}_{\text{test}}$, i.e.,

$$\max_f \text{AUC}(f, \mathcal{Y}_{\text{test}})$$

- Maximize the $nDCG@k$ (Normalized Discounted Cumulative Gain at rank k) metric of the score function $f(u, w)$ on a test dataset $\mathcal{Y}_{\text{test}}$, with $k = 5$, i.e.,

$$\max_f nDCG@5(f, \mathcal{Y}_{\text{test}})$$

3 Methodology

3.1 Workflow

The workflow for this methodology is structured into several key steps to ensure effective data processing, model training, and evaluation. Each step is described in detail below:

1. Dataset preparation: The data will be encoded into a format suitable for our implementation, specifically by constructing the knowledge graph.
2. Model training: The training process will involve optimizing the model using the training data and minimizing the loss function.
3. Model testing: Performance will be assessed on the test dataset using metrics such as AUC and nDCG@5.
4. Hyperparameter tuning: Parameters like the number of epochs, learning rate, and batch size will be fine-tuned to enhance model performance.

3.2 Algorithm and Model Design

3.2.1 Training Algorithm for TransE

The following algorithm describes the training procedure for the TransE model, a translation-based embedding method for knowledge graph representation. The model parameters are optimized using the Adam optimizer, and the process iteratively minimizes a loss function over multiple epochs. The training data is divided into batches, and the optimizer updates model parameters for each batch using backpropagation. Optional loss logging allows monitoring of the training progress.

```
Input: Number of epochs $N_{\text{epochs}}$, learning rate $\eta$, weight decay $\lambda$, training batches $\mathcal{B}$, optional log
    flag log_output
Output: Optimized TransE parameters
Initialize Adam optimizer with $\eta$ and $\lambda$;
for $t \leftarrow 1$ to $N_{epochs}$ do
    Retrieve training batches: $\mathcal{B} \leftarrow$ get_training_batch();
    Initialize empty list $\mathcal{L}$ for storing losses;
    for each batch $b \in \mathcal{B}$ do
        Compute loss $\ell \leftarrow$ optimize($b$);
        Zero gradients: optimizer.zero_grad();
        Backpropagate: loss.backward();
        Update parameters: optimizer.step();
        Append $\ell$ to $\mathcal{L}$;
    end
    if log_output is True then
        Compute mean loss: $\bar{\ell} \leftarrow$ mean($\mathcal{L}$);
        Log: print("Loss after epoch", $t$, "is", $\bar{\ell}$);
    end
end
```

Algorithm 1: Training Procedure for TransE

3.2.2 CTR Evaluation Algorithm

This algorithm evaluates the Click-Through Rate (CTR) by computing scores for batches of evaluation data. It leverages the 'forward' method of the model to compute prediction scores and processes data using the knowledge graph structure.

```
Input: Evaluation batches $\mathcal{B}_{\text{eval}}$
Output: Evaluation scores $\mathcal{S}$
Transpose each batch in $\mathcal{B}_{\text{eval}}$: $\mathcal{B}_{\text{eval}} \longleftarrow [b^T$ for each $b \in \mathcal{B}_{\text{eval}}]$;
Initialize empty list $\mathcal{S}$ for storing scores;
for each batch $b \in \mathcal{B}_{eval}$ do
    Retrieve relation $r \longleftarrow$ [rel_dict["feedback_recsys"] for each entity in batch];
    Compute adjusted user IDs $u \longleftarrow b[0]$ + n_entity;
    Compute scores: $s \longleftarrow$ torch.squeeze(forward($u, r, b[1]$), dim=-1);
    Append numpy version of $s$ to $\mathcal{S}$;
end
Concatenate scores: $\mathcal{S} \longleftarrow$ np.concatenate($\mathcal{S}$, axis=0);
return $\mathcal{S}$;
```

Algorithm 2: CTR Evaluation Algorithm

3.2.3 Top-K Evaluation Algorithm

The following algorithm describes the procedure for evaluating the top-K recommendations for a list of users. For each user, the algorithm identifies the top-K items that they are most likely to interact with, excluding items they have already interacted with in the training dataset. The implementation leverages the scoring function of the model and sorts the items based on their predicted scores.

```
Input: List of users users, number of top recommendations $k$ (default = 5)
Output: List of top-K recommended items for each user sorted_list
Retrieve item list and known positive items for each user:
  item_list, train_user_pos_item $\longleftarrow$ get_user_pos_item_list();
Initialize empty list sorted_list;
for each user user $\in$ users do
    Compute head entities: head $\longleftarrow$ [user + n_entity for _ in range(len(item_list))];
    Compute relations: rel $\longleftarrow$ [rel_dict["feedback_recsys"] for _ in range(len(item_list))];
    Set tail entities: tail $\longleftarrow$ item_list;
    Compute scores: scores $\longleftarrow$ torch.squeeze(forward(head, rel, tail), dim=-1);
    Sort scores in descending order:
      score_ast $\longleftarrow$ np.argsort(scores.cpu().detach().numpy(), axis=-1)[::-1];
    Initialize empty list sorted_items;
    for each index index $\in$ score_ast do
        if length of sorted_items $\geq k$ then
            break;
        end
        if user $\notin$ train_user_pos_item or item_list[index] $\notin$ train_user_pos_item[user] then
            Append item_list[index] to sorted_items;
        end
    end
    Append sorted_items to sorted_list;
end
Return sorted_list;
```

Algorithm 3: Top-K Evaluation Algorithm

4 Model Analysis

4.1 Optimality

The effectiveness of the Knowledge Graph-based recommender system relies heavily on the configuration of several hyperparameters, such as embedding dimensions, margin, learning rate, and weight decay:

- **Embedding Dimensions:** Larger embedding dimensions provide the capacity to encode more complex relationships between entities in the knowledge graph. However, overly large dimensions may lead to overfitting and increase computational costs.

- **Margin in TransE:** The margin parameter in the TransE loss function balances the separation between positive and negative samples. Fine-tuning this parameter is critical to avoid under- or over-penalizing incorrect predictions.
- **Learning Rate:** A well-chosen learning rate ensures steady convergence. Too high a rate risks overshooting the optimum, while too low a rate may cause slow convergence or getting stuck in suboptimal regions.
- **Weight Decay:** Regularization through weight decay prevents overfitting, ensuring generalization to unseen data. The decay value must balance model flexibility with robustness.
- **Batch Size:** Smaller batches provide more precise gradient updates but at the cost of higher computation per epoch. Larger batches offer computational efficiency but may smooth out gradients, hindering convergence.

4.2 Complexity

The computational complexity of the model is directly impacted by several factors:

- **Knowledge Graph Size:** The number of entities and relationships in the knowledge graph increases the number of computations required for embedding updates.
- **Embedding Dimensions:** Higher dimensions amplify the matrix operations' complexity, including dot products and backpropagation.
- **Training Algorithm:**
  - The TransE training algorithm requires computing losses for positive and negative samples, scaling linearly with the number of training samples per epoch.
  - Optimizations like batching help reduce memory footprint but may slightly increase per-iteration complexity due to batch operations.
- **Evaluation Metrics:** Calculating AUC and nDCG@5 involves ranking and sorting, adding $O(n \log n)$ complexity per evaluation batch.

4.3 Interpretability

A key advantage of the Knowledge Graph-based approach is its potential interpretability:

- **Relationship Insights:** The embeddings capture the semantics of relationships between entities, making it easier to explain why a certain item is recommended to a user.
- **Path Analysis:** By analyzing the paths between a user and an item in the graph, one can gain insights into the recommendation's reasoning.
- **Feature Contributions:** Attributes encoded in the knowledge graph provide a transparent basis for prediction, enabling explainability compared to black-box models like deep neural networks.

4.4 Limitations

Despite its strengths, the model has inherent limitations:

- **Limited Generalization:** The TransE approach may struggle with more complex graph structures (e.g., cyclic or hierarchical relationships) due to its linear distance-based embedding approach.
- **Bias in Historical Data:** Recommendations might reinforce existing biases in the dataset, as the model learns directly from historical user-item interactions.

- **Static Graph Assumption:** The model assumes a static knowledge graph, which might not reflect dynamic changes in user preferences or item availability.

By addressing these aspects through thoughtful design, tuning, and evaluation, the Knowledge Graph-based recommender system can achieve robust, scalable, and interpretable recommendations tailored to user preferences.

5 Experiments

5.1 Final Hyperparameters

The final hyperparameters used for the experiments are listed below:

Hyperparameter	**Value**
Batch Size	256
Evaluation Batch Size	1024
Negative Sampling Rate	1.5
Embedding Dimensions	16
L1 Regularization	True
Margin	30
Learning Rate	2e-3
Weight Decay	5e-4
Number of Epochs	35

Table 1: Final hyperparameter configuration.

5.2 Task 1: AUC Evaluation

5.2.1 Metric Description

The Area Under the Curve (AUC) metric evaluates the model's capability to distinguish between positive and negative samples. It reflects the ranking quality by determining the likelihood of assigning higher scores to positive examples compared to negative ones.

5.2.2 Experiment Results

The experiments utilized the TransE model to explore the impact of various hyperparameters on AUC. The model achieved an AUC of **0.7003**.

- **Embedding Dimension:** Increasing the embedding size tends to enhance AUC during local training but may not guarantee similar improvements on the test set.
- **Margin and Negative Sampling Rate:** Higher margin values and increased negative sampling rates are observed to boost AUC performance on the test dataset.
- **Weight Decay:** Setting the weight decay parameter above zero helps prevent overfitting, leading to more robust performance.

5.3 Task 2: nDCG@5 Evaluation

5.3.1 Metric Description

The Normalized Discounted Cumulative Gain at rank 5 (nDCG@5) assesses the ranking quality of the recommendation system. It evaluates how well the model prioritizes relevant items in the top five recommendations, considering their graded relevance.

5.3.2 Experiment Results

The TransE model achieved an nDCG@5 score of **0.1844**. Similar to the observations for AUC, the following trends were noted:

- **Embedding Dimension:** The effect on nDCG@5 follows the same pattern as AUC, where higher dimensions improve ranking during training but may not generalize as effectively.
- **Hyperparameter Interaction:** Parameters such as margin and negative sampling rate significantly influence ranking performance, as the score-based approach for both metrics is inherently linked.

6 Conclusion

The KG-based recommender system developed in this project demonstrated notable strengths in leveraging structured knowledge to enhance user-item predictions. One key advantage of this approach is its ability to incorporate rich contextual relationships between entities, which allows for more informed and precise recommendations. This contextual understanding surpasses traditional matrix factorization methods that rely solely on interaction data.

However, several challenges were observed. The computational complexity of the model, particularly during the training phase, posed scalability issues when applied to large-scale datasets. Additionally, the quality of the recommendations heavily depended on the completeness and accuracy of the knowledge graph, which might not always be achievable in practical applications.

Experimental results aligned with expectations, showcasing strong performance on metrics like AUC and $nDCG$@5. Nonetheless, there were cases where user preferences were underpredicted due to sparsity in the knowledge graph or interaction records.

Future improvements could focus on integrating graph neural networks (GNNs) to further exploit the structural properties of the knowledge graph. Additionally, fine-tuning hyperparameters, exploring alternative embeddings like TransH or RotatE, and incorporating auxiliary data such as user reviews or temporal information could potentially enhance the model's robustness and accuracy. Despite its limitations, the proposed system lays a solid foundation for knowledge graph-driven recommendation strategies.