CS303 Project3 Report

Name: Tan Hao Yang

SID: 12212027

1. Introduction

Recommender systems are crucial for providing personalized recommendations across various industries, such as e-commerce and entertainment. Traditional systems often struggle to capture complex relationships between users and items. By integrating knowledge graphs, which encode entities and their relationships, recommender systems can achieve more accurate and context-aware recommendations.

This project focuses on developing a Knowledge Graph-based Recommender System (KGRS) using the TransE model. The goal is to predict user-item interactions and generate personalized recommendations by combining interaction data with a knowledge graph. The project aims to improve recommendation accuracy (measured by AUC and nDCG) while maintaining computational efficiency, making the system suitable for real-world applications like movie and product recommendations.

Through this report, we explore the application of knowledge graphs in recommender systems, evaluate the TransE model's performance, and discuss strategies for optimizing both accuracy and efficiency.

2. Preliminary

In this project, the problem of **Knowledge Graph-based Recommendation** can be formally formulated as a **link prediction** problem, where the goal is to predict missing links (user-item interactions) in a knowledge graph based on available interaction data and graph-based information.

The system aims to predict the likelihood that a user will interact with an item, given their past interactions and the underlying knowledge graph structure. The recommendation process can be framed as follows:

Problem Formulation:

Let $\mathcal{U}$ be the set of users, $\mathcal{I}$ be the set of items, and $\mathcal{R}$ be the set of relations in the knowledge graph. The knowledge graph $\mathcal{G}$ consists of triples of the form:

$$\mathcal{G} = \{(h, r, t) \mid h \in \mathcal{U} \cup \mathcal{I}, r \in \mathcal{R}, t \in \mathcal{U} \cup \mathcal{I}\}$$

where:

- h (head entity) is either a user or an item,
- r (relation) represents the type of relationship (e.g., "user likes item"),
- t (tail entity) is either a user or an item.

Given this, the recommendation task is to predict the probability $P(h \to t \mid r)$ for unobserved triples, i.e., the likelihood that a user h will interact with an item t under the relation r.

Terminology and Notation:

- **User** u: An individual who interacts with items in the system (e.g., a customer or viewer).
- **Item** i: The product, movie, or any other entity that a user can interact with.
- **Relation** r: A connection between entities in the knowledge graph (e.g., "user likes item" or "movie belongs to genre").
- **Embedding**: A low-dimensional vector representation of users, items, and relations in the knowledge graph that captures the semantic information encoded in the graph.
- **Embedding Dimension** d: The number of dimensions used to represent entities and relations in the embedding space.
- **Link Prediction**: The task of predicting whether an edge (interaction) exists between a user and an item in the knowledge graph.

Objective:

The objective of the recommender system is to learn embeddings for users, items, and relations such that the score for true triples is lower than that for false triples. This is achieved through a **margin-based loss function**, where the model minimizes the distance between embeddings of related entities and maximizes the distance between embeddings of unrelated entities.

Formally, the optimization goal is to minimize the following loss function:

$$L = \sum *(h, r, t) \in \mathcal{G} \sum *\text{negative sample } (h', r, t') \max(0, \gamma + f(h, r, t) - f(h', r, t'))$$

where:

- $f(h, r, t)$ is the scoring function (e.g., based on TransE),
- γ is the margin,
- Negative samples are generated by corrupting the triples (i.e., replacing h or t with a random entity).

The model learns to differentiate between observed and unobserved interactions, thereby generating recommendations for unseen user-item pairs.

3. Methodology

1. General Workflow

The proposed method for the **Knowledge Graph-based Recommender System (KGRS)** using the **TransE model** follows a structured workflow consisting of three primary steps: data preprocessing, model training, and evaluation.

1. **Data Preprocessing**:
   - The first step involves **loading and preprocessing** the data. This includes:
     - **Interaction Data**: User-item interactions (positive and negative samples) are loaded and split into training and test sets.
     - **Knowledge Graph**: The knowledge graph, which encodes relationships between entities (users, items, and relations), is also loaded and processed. The entities are mapped to unique IDs, and relations are encoded as integers.
     - **Negative Sampling**: Negative samples (non-interacted user-item pairs) are generated to balance the dataset.
2. **Model Training**:
   - The **TransE model** is used to learn the embeddings for users, items, and relations. The objective is to minimize the margin-based loss function, which encourages the model to distinguish between positive and negative samples. The training involves optimizing the embeddings using gradient descent and backpropagation.
3. **Evaluation**:
   - The trained model is evaluated using **AUC (Area Under Curve)** for the CTR task and **nDCG@5 (Normalized Discounted Cumulative Gain at rank 5)** for the recommendation task. The performance metrics are calculated by comparing the model's predictions with the true interactions in the test set.

2. Algorithm/Model Design

The **TransE model** is used for embedding entities (users and items) and relations in a low-dimensional space. The TransE model is based on the idea that the relationship between a head entity h, a relation r, and a tail entity t should satisfy:

$$\mathbf{h} + \mathbf{r} \approx \mathbf{t}$$

where $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ are the embeddings of the head entity, relation, and tail entity, respectively. The model learns these embeddings by minimizing the following **margin-based loss function**:

$$L = \sum *(h, r, t) \in \mathcal{G} \sum *\text{negative sample } (h', r, t') \max(0, \gamma + f(h, r, t) - f(h', r, t'))$$

where:

- $f(h, r, t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|\_p$ is the score function (typically L2 or L1 norm),
- γ is the margin,
- Negative samples (h', r, t') are generated by randomly replacing the head or tail entity.

The goal is to find the embeddings that minimize the loss, i.e., that make the score for true triples lower than the score for false triples.

Model Design:

1. **Input**:
    - Training data: Triples of the form (h, r, t) representing interactions and relationships.
    - Negative samples: Corrupted triples where either the head or the tail is replaced by a random entity.
2. **Training**:
    - The model optimizes the embeddings of the entities and relations by minimizing the margin-based loss.
3. **Output**:
    - Learned embeddings for users, items, and relations that can be used to predict future user-item interactions.

Pseudo-code for the TransE Training Algorithm:

```
# Pseudo-code for TransE Model Training
initialize entity_embeddings, relation_embeddings  # Initialize embeddings for entities and rel

for epoch in range(num_epochs):
    for batch in training_data:
        positive_samples = batch["positive_samples"]
        negative_samples = generate_negative_samples(batch["positive_samples"])

        # Compute scores for positive and negative samples
        pos_scores = compute_scores(positive_samples)
        neg_scores = compute_scores(negative_samples)

        # Compute loss using margin-based loss function
        loss = compute_loss(pos_scores, neg_scores, margin)

        # Perform backpropagation and optimize embeddings
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if epoch % log_interval == 0:
        print(f"Epoch {epoch}, Loss: {loss.item()}")
```

The **key steps** are:

1. **Compute scores**: The score for a sample (h, r, t) is computed as $||\mathbf{h} + \mathbf{r} - \mathbf{t}||\_p$.
2. **Compute loss**: The loss function penalizes the model when the positive sample score is lower than the negative sample score by more than the margin.
3. **Optimize embeddings**: The embeddings are updated by backpropagating the gradients through the model.

3. Analysis

1. **Optimality**:
    - The TransE model is relatively simple and efficient, as it uses vector addition to model relationships between entities. However, its simplicity can be a limitation, as it assumes the relation between entities is linear, which may not hold for all types of data.
    - The performance of TransE depends on the quality of the embeddings and how well the margin-based loss function can differentiate between true and false triples.
2. **Complexity**:

- **Time Complexity**: The time complexity of training the TransE model is $O(n \cdot d)$, where n is the number of entities (or triples) and d is the embedding dimension. For each epoch, the model processes all triples and updates embeddings using stochastic gradient descent (SGD).
- **Space Complexity**: The space complexity is $O(n \cdot d)$ due to the need to store embeddings for all entities and relations, where n is the number of entities and d is the embedding dimension.

3. **Deciding Factors for Performance**:
   - The **embedding dimension** d plays a crucial role in the model's ability to capture complex relationships between entities. Larger values of d allow for richer embeddings but require more computational resources.
   - The **margin** γ controls how strictly the model differentiates between positive and negative samples. A larger margin can help improve ranking quality but might slow down training.
   - The **learning rate** is another critical factor: too large, and the model may converge too quickly to suboptimal solutions; too small, and it might take too long to converge.

4. Experiments

4.1 Task 1: Maximizing AUC

1. Metrics

For **Task 1**, the goal is to **maximize AUC (Area Under the Curve)**. AUC measures how well the model distinguishes between positive and negative samples. AUC values range from `0.5` (random guesses) to `1.0` (perfect predictions).

Test Flow:

1. **Data Loading**: The test data, including both positive and negative samples, is loaded.
2. **Model Training**: The model is trained using both the **first set of hyperparameters (the orignally provided hyperparameters)** and the **second set of hyperparameters**.
3. **Evaluation**: After training, the model's performance is evaluated on the test set using **AUC** as the primary metric.

2. Experimental Results for Task 1

First Set of Hyperparameters:

- **AUC**: The model trained with the first set of hyperparameters achieved **AUC of ~0.58**. This indicates that the model could distinguish between positive and negative samples but with limited effectiveness.

Second Set of Hyperparameters:

- **AUC**: The model trained with the second set achieved **AUC of ~0.68**. This is a slight improvement over the first set, suggesting that increasing the **embedding dimension** and **margin** helped the model distinguish between positive and negative samples more effectively.

3. Analysis of Hyperparameters for AUC:

- **Effect of Embedding Dimension**: The **larger embedding dimension (196)** in the second set likely allowed the model to better capture the complexity of the relationships between users, items, and their interactions, leading to a slight improvement in **AUC**.
- **Effect of Margin**: The **larger margin (200)** in the second set might have helped the model create a more distinct boundary between positive and negative samples, which also likely contributed to the improvement in **AUC**.
- **Effect of Epochs**: Both sets used a similar number of epochs (`30` for the first set and `150` for the second set). The second set trained for longer, which likely helped the model improve, but the impact of other hyperparameters (embedding size and margin) were more influential.

4.2 Task 2: Maximizing nDCG

1. Metrics

For **Task 2**, the goal is to **maximize nDCG@5** (Normalized Discounted Cumulative Gain at rank 5), which evaluates the ranking quality of the top 5 recommendations. A higher nDCG value indicates that the top-ranked items are more relevant.

Test Flow:

1. **Data Loading**: The test data is prepared, including both positive and negative samples.
2. **Model Training**: The model is trained using both the **first set of hyperparameters** and the **second set of hyperparameters**.
3. **Evaluation**: After training, the model's performance is evaluated on the test set using **nDCG@5** as the primary metric.

2. Experimental Results for Task 2

First Set of Hyperparameters:

- **nDCG@5**: The model trained with the first set of hyperparameters achieved a **very low nDCG (~0.01)**, meaning that the top-ranked items were not highly relevant.

Second Set of Hyperparameters:

- **nDCG@5**: The model trained with the second set of hyperparameters achieved **nDCG of ~0.14**, indicating that the top 5 recommendations were much more relevant.

3. Analysis of Hyperparameters for nDCG:

- **Effect of Embedding Dimension**: The **larger embedding dimension (196)** in the second set enabled the model to better capture the nuances of the user-item interactions, leading to better ranking performance and an improvement in **nDCG**.
- **Effect of Margin**: The **larger margin (200)** helped to enforce a stricter separation between positive and negative samples, which likely contributed to better ranking (higher nDCG).
- **Effect of Epochs**: The **increased number of epochs (150)** in the second set provided more time for the model to refine its predictions and improve ranking, which directly improved **nDCG**.

5. Conclusion

In this project, we developed a **Knowledge Graph-based Recommender System (KGRS)** using the **TransE model**, which successfully incorporated both user-item interaction data and knowledge graph information to generate recommendations. The experimental results showed that **hyperparameter tuning** plays a crucial role in improving model performance, particularly for maximizing **nDCG** and **AUC**.

- The **TransE model** offers a simple yet effective approach to embedding knowledge graphs, with its main advantage being its ability to capture relationships between entities in a low-dimensional vector space. However, the model assumes linear relationships, which may limit its expressiveness for more complex datasets.
- The experimental results largely met our expectations: the **second set of hyperparameters** improved both **AUC** and **nDCG**, particularly for ranking accuracy (nDCG@5).
- Further improvements could involve exploring alternative models like **TransH** or **TransR** that capture more complex relationships between entities, as well as experimenting with **learning rate schedules** or more advanced regularization techniques.

In conclusion, while the model performed reasonably well, future work could focus on refining hyperparameters and exploring more advanced embedding techniques to further enhance recommendation quality and efficiency.