Recommendation Algorithm: Deep Walk-Based Movie Recommendation System

The movie recommendation system was developed based on a **Deep Walk-style Graph Neural Network (GNN)** approach. The system was designed to recommend movies to users by leveraging their historical preferences and the intricate relationships between users and movies, uncovering nuanced patterns in taste.

Algorithm Overview

At its core, the system's operation began with the construction of a **bipartite graph**. In this graph, both users and movies were represented as individual nodes. When a user interacted with a movie (for instance, by rating it), an edge formed between that user's node and the corresponding movie's node. This approach allowed for a clear and direct representation of user-movie interactions.

Once the graph was established, I proceeded to generate many random walks across it. These walks can be conceptualized as paths taken through the network, moving sequentially from a user to a movie they interacted with, then to another user who also interacted with that movie, and so on. These random walks were crucial as they effectively captured the local neighborhood structure and the complex, often indirect, relationships between users and movies. Specifically, I configured the system to generate 10 walks per node, each with a length of 80 steps. For the Movie Lens 100k dataset, this resulted in approximately 26,250 total walks, providing a comprehensive exploration of the graph's structure.

These sequences of nodes, derived from the random walks, were subsequently fed into a **Word2Vec model**. I treated each walk as a "sentence" and each user or movie node as a "word." This process enabled the model to learn a numerical representation, or "embedding," for every user and movie. A key outcome of this step was that users and movies with similar characteristics or preferences would possess embeddings closely positioned in this learned vector space. The embeddings were 128-dimensional vectors, and a window size of 5 was used, considering five neighboring nodes in each direction during the embedded learning process.

Finally, to generate recommendations, I calculated the **cosine similarity** between a user's embedding and the embeddings of all movies they had not yet rated. Movies with higher similarity scores were considered more relevant and were then ranked to provide the top recommendations to the user.

Key Assumptions

My approach was founded on several fundamental assumptions:

1. I assumed that users who enjoyed the same movies were likely to share similar preferences for other movies. This principle formed the basis for connecting users and movies within the constructed graph, allowing for the inference of shared tastes.
2. If User A liked Movie X, and User B also liked Movie X, I inferred that User A and User B might possess similar tastes beyond just Movie X. The random walks were instrumental in capturing these transitive relationships through multi-hop paths within the graph.
3. I believed that the immediate connections and neighborhood around a user or movie node in the graph contained the most relevant information for generating recommendations. The design of the random walks, with their relatively short length (80 steps), specifically focused on exploring these local structures effectively rather than attempting to capture the entire global graph structure.
4. **Embedding Space Semantics**: I assumed that the learned embeddings accurately reflected the underlying preferences. This meant that movies and users with similar tastes would indeed be positioned closely in the multidimensional embedding space, and that core similarity in this space would correlate directly with preference similarity. This assumption was critical for the effectiveness of the recommendation generation phase.

Performance

Through rigorous evaluation, the model achieved a F1-Score of 83.72%, demonstrating a strong balance between recommendation precision and recall. The key metrics were:

- **Accuracy**: 83.24%
- **Precision**: 81.75%
- **Recall**: 85.77%

These results confirm the model's effectiveness in identifying relevant movies while maintaining a favorable tradeoff between suggesting truly relevant items (precision) and capturing the full range of relevant items (recall).

The evaluation strategy used a binary classification task:

- **Positive samples**: User-movie pairs with ratings ≥ 3.5
- **Negative samples**: Randomly sampled non-interactions
- **Features**: Combined user+movie embeddings (256-dimensional vectors)
- **Classifier**: Logistic Regression with 3-fold cross-validation

Hyperparameters for the Deep Walk model (embedding size, walk length, and window size) were systematically tuned to optimize performance. This rigorous validation approach ensures the reported metrics reliably reflect the model's recommendation capabilities.

Technical Design Decisions

During the development of this recommendation system, I made several key technical design decisions that influenced its architecture and performance:

1. I chose to represent the user-movie interactions as an **unweighted bipartite graph**. This decision was made to simplify the random walk generation process while still preserving the essential user-movie relationships. While weighted edges based on rating values could have provided more granular information, I opted for an unweighted approach to maintain simplicity and computational efficiency in this initial implementation. This choice allowed for a clear and direct mapping of user-movie interactions without adding the complexity of the rating scale.

2. I implemented a strategy of **uniform random walks**. This approach ensured an unbiased exploration of the graph structure, allowing the model to discover patterns without imposing prior assumptions about the importance of certain nodes or edges. While alternative strategies like biased walks (e.g., Node2Vec) could have been explored for different exploration behaviors, uniform random walks provided a solid baseline for capturing the inherent structure of the user-movie interaction graph.

3. For learning the node embeddings, I selected the **Word2Vec Skip-gram model**. This choice was based on its proven effectiveness in learning meaningful

embeddings from sequential data, which aligned perfectly with treating the random walks as "sentences" and the nodes as "words." The Skip-gram architecture, particularly, was chosen for its ability to predict context nodes given a target node, which is highly suitable for capturing the relationships within the graph. While other graph-specific embedding methods (such as GraphS AGE or Graph Convolutional Networks) exist, Word2Vec provided a robust and well-understood solution for this application.

4. **Recommendation Scoring** used **cosine similarity** between user and movie embeddings for recommendation scoring. This method was chosen because it effectively captures the semantic similarity in the learned embedding space. Movies with embeddings closer to a user's embedded in this multidimensional space were considered more relevant. **Confidence Metric (`prob`)**: The `prob` field in recommendations represents prediction confidence (0–100%) derived from the logistic regression classifier during evaluation. It estimates the likelihood that a user would rate the movie ≥3.5, based on the combined user-movie embedding. This provides an interpretable confidence measure alongside the cosine similarity score.
5. **Scalability Considerations**: The system was designed with scalability principles in mind, though optimized for the MovieLens 100k dataset:
    - **Graph construction** uses efficient sparse matrix storage (O(|E|) memory)
    - **Random walks** use batch processing for memory efficiency
    - **Embeddings** leverage Gensim's optimized Word2Vec for larger corpora
    - **Recommendations** use vectorized cosine similarity (O(n) per user)

Note: Production deployment would require distributed computing for walk generation and approximate nearest-neighbor search for huge item catalogs.

Justification for GENSIM

The choice of `gensim` for the Word2Vec (Skip-gram) implementation was justified by its robust, efficient, and scalable capabilities, which I found particularly well-suited for the Movie Lens 100k dataset and similar recommendation tasks.

1. **Optimized for Large Datasets**: gensim's Word2Vec implementation is highly optimized for processing large corpora. Though MovieLens 100k is a relatively small dataset in terms of total interactions (100,000), the underlying graph can still be significant. `gensim` efficiently handled the random walks generated from the graph (which I treated as "sentences"), allowing for fast and memory-efficient training of embeddings.
2. `gensim's` Word2Vec is a mature and widely-used library, thoroughly tested and optimized for speed and accuracy in learning word (or, in this case, node) embeddings. This stability provided confidence in the quality of the learned Deep Walk embeddings.
3. `gensim` offered a straightforward API for training Word2Vec models, making it easy to integrate into the existing Python-based project. This allowed me to focus on the recommendation logic rather than low-level implementation details of the embedding algorithm.
4. **Flexibility**: `gensim` provided various parameters for Word2Vec training (e.g., `vector_size` , window, min_count, `window` , `min_count` , window, min_count, sg for `skip-gram`). This flexibility allowed for fine-tuning the embedding process to achieve optimal performance for the specific characteristics of the movie interaction graph.
5. As an open-source library with an active community, `gensim` benefited from continuous improvements and readily available support, which I found valuable for maintaining and extending the project.

In summary, `gensim` provided a reliable, performant, and user-friendly solution for learning Deep Walk embeddings, making it an excellent choice for this movie recommendation engine, especially considering its ability to scale to larger datasets in future iterations.
