Graph Neural Network-Based Recommender System for Social Media Content

Authored by : Abhay Kashyap

***Abstract*—This report presents a Graph Neural Network (GNN)-based recommender system implemented in PyTorch Geometric for social media content recommendation. The architecture integrates heterogeneous graph data representing user-item interactions with multi-modal node and edge features. We comprehensively detail the data acquisition and cleaning pipeline, feature normalization techniques, model architecture with specialized embeddings, message passing mechanisms, and training methodology. The proposed system employs time-aware edge weights, contextual feature embedding, and multiple specialized GNN layers to generate personalized and cold-start recommendations. Extensive evaluation demonstrates the system's effectiveness in capturing complex interaction patterns and providing relevant content suggestions across diverse user segments.**

I. Introduction

Recommender systems are essential in filtering and suggesting relevant content to users in social media platforms. Traditional collaborative filtering and content-based models often fail to capture complex user-item interactions and temporal dynamics. Graph Neural Networks (GNNs) offer a powerful paradigm for modeling such relationships using graph-structured data, which naturally represents the interactions between users and content.

This work implements a heterogeneous GNN-based recommender system using the PyTorch Geometric library. The system models user-content interactions as a heterogeneous graph with multiple interaction types (viewing, liking, inspiring, and rating) and leverages various graph convolution operations to learn rich feature representations. Our implementation specifically addresses several critical challenges:

- Temporal dynamics of user interactions through time-aware edge weighting
- Cold-start problem through specialized mood-based embeddings
- Multi-modal feature integration through specialized encoding architectures
- Scalability concerns through efficient batch processing and caching

II. Data Acquisition and Preprocessing

A. Data Sources and API Integration

The system collects data from a social media platform API with multiple endpoints for different interaction types:

- User profile information with engagement metrics
- Content metadata with categorization and performance metrics
- User-content interactions: views, likes, inspirations, and ratings

Data is fetched using pagination to handle large datasets, with authentication via token-based access. Each endpoint returns structured JSON data that is normalized into appropriate dataframes.

B. Data Cleaning and Transformation

The preprocessing pipeline includes multiple stages:

- **Timestamp Standardization:** Interaction timestamps are normalized to a consistent format and column name.
- **Deduplication:** User-content interaction pairs are deduplicated to prevent redundant edges.
- **Categorical Feature Extraction:** Content categories are extracted from nested JSON structures.
- **Interaction Weighting:** Different interaction types are assigned appropriate weights based on engagement significance (viewing=1, liking=3, inspiring=4, rating=normalized value).
- **Temporal Normalization:** Interaction timestamps are converted to recency scores in the $[0, 1]$ range using min-max normalization.
- **Edge Strength Calculation:** Final edge weights combine interaction weight and recency using the formula: $\text{strength} = \text{weight} \times (1 + \text{recency})$

C. Feature Engineering

For both users and content items, the system applies specialized feature engineering:

1) User Features: User nodes incorporate activity and social metrics:

- Content creation count (post_count)
- Social graph metrics (following_count, follower_count)

2) Content Features: Content nodes integrate categorical and engagement metrics:

- Content category ID (extracted from nested structure)
- Engagement metrics (view_count, upvote_count)
- Quality signals (average_rating)

3) Feature Normalization: The system applies distinct normalization techniques to different feature types:

- User features: Z-score normalization for numerical features
- Content engagement metrics: Log transformation for count-based metrics (view_count, upvote_count)
- Rating features: Min-max scaling for the $[0, 1]$ range
- Categorical features: Preserved as integer indices for embedding lookup

III. Graph Construction

A. Node and Edge Encoding

The system constructs a heterogeneous graph with the following components:

- **User Nodes**: Indexed by unique user IDs with feature tensor $X_u \in \mathbb{R}^{|\mathcal{U}| \times d_u}$
- **Content Nodes**: Indexed by unique content IDs with feature tensor $X_i \in \mathbb{R}^{|\mathcal{I}| \times d_i}$
- **Interaction Edges**: Multiple edge types representing different interactions (viewing, liking, inspiring, rating)
- **Edge Attributes**: Edge weight tensor based on interaction strength and recency

B. Similarity Edges

Beyond direct user-content interactions, the system enriches the graph with:

- **User-User Similarity**: Edges connecting users with similar activity patterns, weighted by cosine similarity of feature vectors
- **Content-Content Similarity**: Edges connecting content items with similar attributes, weighted by cosine similarity of feature vectors

Similarity edges are constructed using thresholded cosine similarity with a minimum value of 0.5 to avoid excessive edge density.

C. Time-Aware Train-Validation-Test Split

Unlike random splitting, the system implements a temporal split:

- Interactions are sorted chronologically
- Train set: First 70% of interactions (oldest)
- Validation set: Next 20% of interactions
- Test set: Last 10% of interactions (most recent)

This approach ensures the model is evaluated on its ability to predict future interactions, mimicking the real-world recommendation scenario.

IV. Mathematical Foundations

Let $\mathcal{G} = (\mathcal{U}, \mathcal{I}, \mathcal{E})$ be a heterogeneous graph where:

- $\mathcal{U}$: Set of user nodes
- $\mathcal{I}$: Set of content nodes
- $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{I} \times \mathcal{T}$: Set of edges representing interactions, where $\mathcal{T}$ is the set of interaction types

Let $X_u \in \mathbb{R}^{|\mathcal{U}| \times d_u}$ and $X_i \in \mathbb{R}^{|\mathcal{I}| \times d_i}$ be initial feature matrices, where d_u and d_i are the user and item feature dimensions respectively.

A. Feature Transformation

The initial feature matrices are transformed using specialized embedding layers:

$$\hat{X}_u = \text{MLP}_u(X_u) \tag{1}$$

$$\hat{X}_i = \text{ContentEmbedding}(X_i) \tag{2}$$

where ContentEmbedding includes category embedding and feature normalization:

$$\text{ContentEmbedding}(X_i) = \text{Concat}(\text{CategoryEmbed}(X_i^{\text{cat}}), \text{BatchNorm}(X \tag{3}$$

B. Message Passing Mechanism

The multi-layer heterogeneous message passing mechanism is defined as:

For each layer ℓ, the node features are updated according to:

$$h_v^{(\ell)} = \sigma\left(\sum_{t \in \mathcal{T}} \sum_{u \in \mathcal{N}^t(v)} \alpha_{uv}^t W_t^{(\ell)} h_u^{(\ell-1)}\right) \tag{4}$$

where:

- $h_v^{(\ell)}$ is the representation of node v at layer ℓ
- $\mathcal{N}^t(v)$ is the neighborhood of node v under edge type t
- $W_t^{(\ell)}$ is the learnable weight matrix for edge type t at layer ℓ
- α_{uv}^t is the attention coefficient for edge (u, v) of type t
- σ is a non-linear activation function (ReLU)

The attention mechanism in Graph Attention Network (GAT) layers calculates attention coefficients:

$$\alpha_{uv}^t = \frac{\exp(\text{LeakyReLU}(a^T[W_t h_u \| W_t h_v]))}{\sum_{w \in \mathcal{N}^t(v)} \exp(\text{LeakyReLU}(a^T[W_t h_w \| W_t h_v]))} \tag{5}$$

where a is a learnable attention vector and $\|$ represents concatenation.

V. Model Architecture

A. Feature Embedding Layers

The model implements specialized embedding components:

1) User Embedding:

- Linear projection of user features to hidden dimension
- Batch normalization for training stability
- ReLU activation and dropout for regularization

2) Content Embedding:

- Category embedding for categorical features
- Separate processing for numerical features with batch normalization
- Concatenation of categorical and numerical embeddings
- Projection to unified hidden dimension

3) Mood Embedding: For cold-start recommendations:

- Embedding lookup table for mood IDs
- Xavier initialization for better convergence
- Integration with GNN for mood-based content recommendation

B. Graph Convolution Layers

The model employs heterogeneous graph convolution with multiple edge types:

- User $\rightarrow$ Content: GAT convolution with 2 attention heads
- Content $\rightarrow$ User: GAT convolution with 2 attention heads
- User $\rightarrow$ User: GraphSAGE convolution for user similarity

- Content → Content: GraphSAGE convolution for content similarity

Each convolution aggregates messages from the respective edge type, with attention mechanisms focusing on relevant connections. Multiple layers enable higher-order interactions.

C. Prediction Layers

Two specialized prediction components:

1) Regular Prediction: For users with interaction history:

- Separate projection layers for user and content embeddings
- Element-wise multiplication (Hadamard product) of projected embeddings
- Multi-layer prediction network with batch normalization
- Final sigmoid activation for interaction probability

2) Cold-Start Prediction: For new users without interaction history:

- Mood embedding as a proxy for user preferences
- Vector normalization of mood and content embeddings
- Concatenation of normalized vectors
- Specialized prediction network to map mood-content pairs to interaction probability

VI. Training Methodology

A. Data Loaders

The system implements specialized data loaders for GNN training:

- Edge-based sampling with source (user) and destination (content) indices
- Normalization of edge attributes to $[0, 1]$ range
- Batch processing with configurable batch size
- Separate loaders for train, validation, and test sets

B. Loss Function

Binary Cross-Entropy with Logits Loss:

$$\mathcal{L} = -\frac{1}{|\mathcal{E}_{\text{train}}|} \sum_{(u,i)\in\mathcal{E}_{\text{train}}} [y_{ui} \log(\sigma(\hat{y}_{ui})) + (1 - y_{ui}) \log(1 - \sigma(\hat{y}_{ui}))] \tag{6}$$

where:

- y_{ui} is the ground truth interaction strength
- $\hat{y}_{ui}$ is the predicted logit
- σ is the sigmoid function

This loss function provides better numerical stability than MSE for binary prediction tasks.

C. Optimization

The training process employs:

- Adam optimizer with weight decay (L2 regularization)
- Learning rate scheduling with ReduceLROnPlateau
- Gradient clipping to prevent exploding gradients
- Early stopping based on validation loss
- Model checkpointing to save the best-performing model

D. Training Loop

The training loop performs the following for each epoch:

- Forward pass through GNN to compute node embeddings
- Extraction of batch-specific user and content embeddings
- Prediction of interaction probabilities
- Loss calculation and backpropagation
- Parameter updates with gradient clipping
- Validation on held-out edges
- Learning rate adjustment based on validation performance

VII. Recommendation System Implementation

A. Recommendation Generation

The system implements two recommendation approaches:

1) Personalized Recommendations: For existing users:

- Forward pass through GNN to compute user and content embeddings
- Extraction of target user's embedding
- Scoring all available content items through the prediction layer
- Filtering previously interacted content (optional)
- Ranking content by prediction score
- Pagination for efficient delivery

2) Cold-Start Recommendations: For new users:

- Mood-based embedding as user preference proxy
- Vector normalization of mood and content embeddings
- Scoring content through cold-start predictor
- Ranking and pagination of results

B. Caching Mechanism

To optimize performance and reduce computational load:

- Timestamp-based caching of recommendations
- Configurable cache time-to-live (TTL)
- Automatic invalidation on new user interactions
- Separate caches for regular and mood-based recommendations

C. Evaluation Metrics

The system evaluation employs multiple metrics:

- **Mean Absolute Error (MAE):** Average absolute deviation of predictions from ground truth
- **Root Mean Squared Error (RMSE):** Square root of the average squared deviation
- **Area Under the ROC Curve (AUC):** Measure of model's ability to distinguish between positive and negative interactions
- **F1 Score:** Harmonic mean of precision and recall at optimal threshold

The system also implements an optimal threshold selection based on F1 score maximization on the validation set.

VIII. Deployment Considerations

A. Model Export

For deployment, the model weights can be exported to a lightweight format:

- Conversion of model parameters to efficient C arrays
- Storage in header (.h) files for embedded platforms
- Flattened weight representation for optimized memory access

B. Real-time Updates

The system supports dynamic updates:

- Incremental graph updates with new user interactions
- Automatic cache invalidation for affected user-content pairs
- Timely recommendation refreshing based on interaction patterns

IX. Conclusion and Future Work

We developed a comprehensive GNN-based recommender system for social media content using PyTorch Geometric. The system effectively models heterogeneous user-content interactions with specialized graph convolution operations and attention mechanisms. Time-aware edge weights and cold-start handling through mood embeddings address key recommendation challenges.

Future work may include:

- Incorporating explicit temporal modeling through Temporal Graph Networks
- Implementing multi-modal content representation for image and video features
- Exploration of self-supervised learning techniques for improved cold-start performance
- Large-scale deployment optimization through graph sampling and distributed computing
- Integration of explainability mechanisms to provide transparent recommendation rationales

References