Music Artist Recommender

C608 – Recommender Systems

Group 8

TABLE OF CONTENTS

Data Sources

Grouplens – Last.fm 2k

- Artists metadata – name, URL (17,632 records)
- User – artist interactions (listen counts) (92,834 records)
- User – user pairs (social links) (25,434 records)
- Tags – tagID mapping to freetext tagValue (11,946 records)
- User – tagged – artists triples with dates (186,479 records)

Number of unique users = 1,892
Number of unique Artists = 17,632
Number of unique Tags = 11,946

Spotify – Scraped via API

- Artists metadata – Genre (10,275 records)

Motivation & Purpose

Implement Music Artist Recommender

- ➢ Increase user engagement by introducing the new artists which are relevant to users

Multi-relational Recommender System

- ➢ Integrates multiple types of data
  - User – item interactions (listen counts)
  - User generated content (tags)
  - Item metadata (genres)
  - Social relations

- ➢ Experiments conducted:
  - EASE (as benchmark)
  - RecWalk (multiple versions)

Data Preparation

Step 1: Build Raw Knowledge Graphs

- Build a list of triples (entity, relation, entity):
  - ‘userID’, ‘listened_to’, ‘artistID’
  - ‘artistID’, ‘has_tag’, ‘tagID’
  - ‘userID’, ‘tagged’, ‘artistID’
  - ‘userID’, ‘prefers_tag’, ‘tagID’
  - ‘user_ID’. ‘friends_with’, ‘userID’ – bidirectional
  - ‘artistID’, ‘has_genre’, ‘genre’

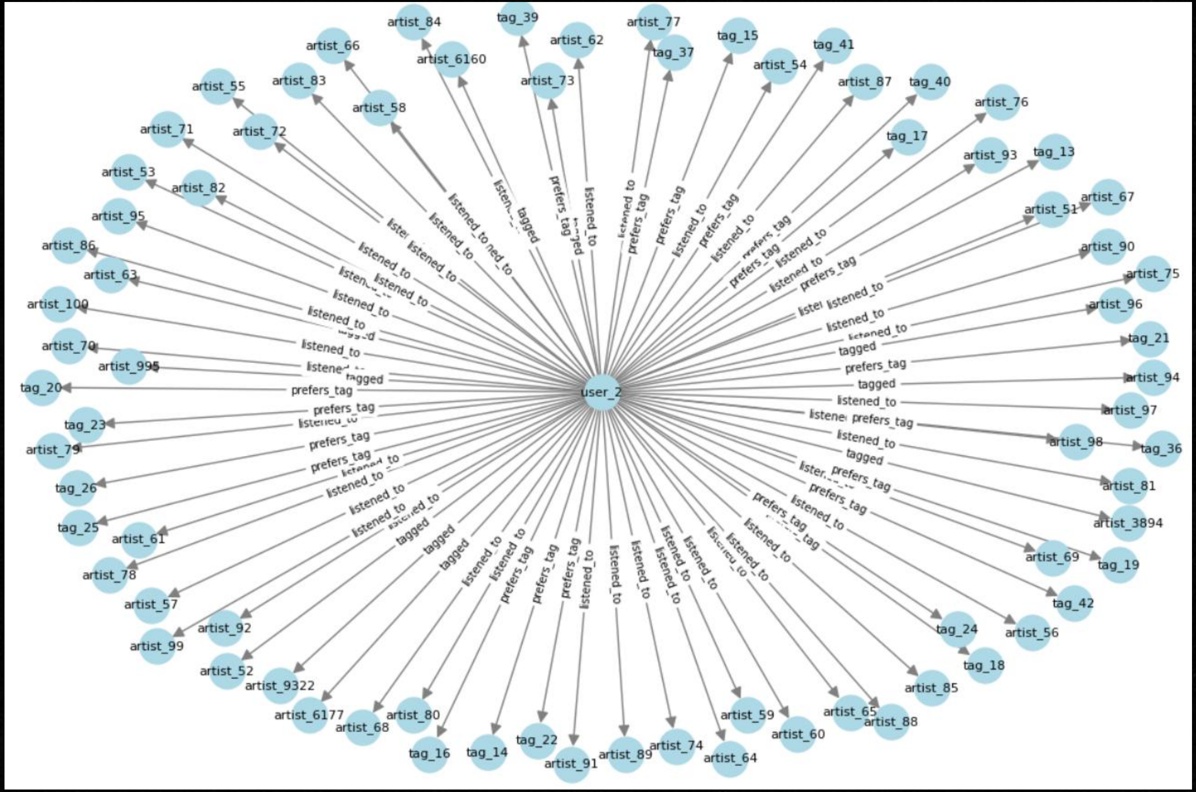
- Create entity ID mapping
  - Index all entities

Data Preparation

Knowledge Graph

```
📊 Knowledge Graph Statistics
Total Triples          : 345,173
Unique Entities        : 34,710
Unique Relations       : 6

Relation Type Distribution:
  listened_to        : 92,834
  has_tag            : 109,750
  tagged             : 71,064
  prefers_tag        : 35,816
  friends_with       : 25,434
  has_genre          : 10,275
```



Data Preparation

<u>KEY REASONS</u> for using Knowledge Graphs

- ➢ Enables multi-hop reasoning for better recommendations

User_A → friends_with → user_B → listened_to → artist_A → has_tag → tag_A ← has_tag ← artist_B

- ➢ Improved cold start handling

Newer or less-listened tracks can be recommended by reasoning over their relationships (genres, tags, etc.)

- ➢ Better interpretability / explainability

Recommendations explainable via paths in graph:

"You might like the song because you like songs by artists similar to X, who also belong to the same genre"

RecWalk – Genre Enhanced

Step 2: Build weighted adjacency matrix

Edge Type	Direction	Weights
User ↔ Artist	Bidirectional	Natural logarithm of listened counts
User ↔ Tag	Bidirectional	1.0
User ↔ Friend	Bidirectional	1.0
Artist ↔ Genre	Bidirectional	Artist → Genre : 1.0, Genre → Artist 0.5

- Bidirectional edges allow us to 'walk' between users / artists / content
    - Logarithmically smoothened listened counts used to prevent very popular artists from dominating
    - Asymetrical weights between Artist and Genre
        - Artists are more clearly described by their Genres – walking from Artist to Genre reflects this strong identity
        - Genres may describe many Artists – walking from Genre to Artist reflects a weaker connection

RecWalk – Genre Enhanced

Step 3: Convert to CSR Matrix

- Compressed Sparse Rows as a memory efficient storage of a sparse matrix
  - Make row-wise operations (random walks) fast and memory efficient
- Normalize weights
  - Transforms each row into transition probabilities between two nodes

Step 4: Perform Random Walk with Restarts and extract scores

```
for _ in range(30):
    x = (1 - self.alpha) * x + self.alpha * self.P.dot(x)
```

- Alpha is a tune-able hyperparameter governing restart probability
- Iteratively approximates the steady state distribution
- **Models likelihood of the model 'walking' from a particular user to a particular artist**
- Scores reflecting how connected artist is to the user is then extracted

RecWalk – Genre Enhanced

Step 5: Apply Genre-based Boosting

```
genre_score = self._genre_match_score(artist, user_genres)
artist_scores[artist] = base_score * (1 + self.genre_weight * genre_score)
```

- user_genres determined by which genres appear most frequently among artists user is connected to
- _genre_match_score computes similarity between artist's genres and user's top genres
- **Rationale** behind genre boosting
  - Tags can be noisy or sparse – user generated tags vary widely in quality and are non-standardized
  - Social links do not equate to shared taste and preferences – weak predictors
  - Genres provides semantic structure – generalize beyond known artists to stylistically aligned ones
  - Enhanced diversity and discovery – genre awareness reduces focus on popular (highly connected) artists

RecWalk – Similarities Enhanced

Layered enhancement over previous genre – based model

Step 6: Compute Artist Similarities

- Co-listening Similarity – cosine similarity computed to measure similarities in user (listener) base

Step 7: Addition of Artist – Artist Similarity Edges

- Bidirectional Artist – Artist edges based on similarity score and similarity weight hyperparameter
- Patches additional edges into existing graph if similarity score > defined threshold

RecWalk – Similarities Enhanced

Layered enhancement over previous genre – based model

Step 8: Apply similarity – based reranking

```
avg_sim = sim_score / len(user_artists) if user_artists else 0
# Boost original score by similarity (50% of similarity score)
scored_recs.append((artist, score * (1 + 0.5 * avg_sim)))
```

- Compute similarity score between current artist with artists user listened to
- Boosts final score (and ranking) based on average similarity

RecWalk - Model Explainability

```
recs = model.recommend(user_id=2, top_k=50)
print("Recommended artists:", [a for a, _ in recs])
```

```
Recommended artists: [83, 87, 79, 92, 74, 60, 78, 94, 62, 90, 73, 82, 91, 95,
13157, 13275, 77, 63, 13151, 84, 96, 99, 12760, 6781, 18226, 6734, 15771, 1317
```

```
recs_with_explanations = model.recommend(user_id=2, top_k=5, explain=True)
for rec in recs_with_explanations:
    print(f"Artist: {rec['artist']}, Score: {rec['score']:.4f}")
    for source in rec['sources']:
        print(f"  - {source['type']}: {source.get('artist', source.get('genre', ''))} (weight: {source['weight']:.4f})")
```

```
Artist: 83, Score: 0.1860
  - direct_listening:  (weight: 0.0173)
Artist: 87, Score: 0.1860
  - direct_listening:  (weight: 0.0170)
Artist: 79, Score: 0.1839
  - direct_listening:  (weight: 0.0175)
  - genre: genre_['new wave'] (weight: 0.5000)
Artist: 92, Score: 0.1830
  - direct_listening:  (weight: 0.0167)
  - genre: genre_['new wave'] (weight: 0.5000)
Artist: 74, Score: 0.1689
  - direct_listening:  (weight: 0.0181)
  - genre: genre_['smooth jazz'] (weight: 0.5000)
```

Evaluation Metrics

	EASE (Benchmark)	RecWalk (Genre)	RecWalk (Similarities)
Precision	0.0050	0.0029	0.0056
Recall@50	0.0293	0.0148	0.0284
NDCG@50	0.0588	0.0261	0.0627


DEMO

Questions?