Network and non-Network based Approach for Music Recommendation

Mengying Zhang
Yuying Li

SI608 Final Project

Research Question

- Whether network based model of non-network based model performs better in recommendation system
- Which network based model for recommendation system prevails
- What accuracy can a network-based model reach

Motivation

- Help explore music you love in a music platform
- Low frequency of existing recommendation
- Low accuracy of existing recommendation results
- Few recommendation systems based on network concept

Related Work

- Brandon Liu “Better than PageRank: Hitting Time as a Reputation Mechanism”, 2014
- Li Gao et all. “Recommendation with Multi-Source Heterogeneous Information”, 2018

DataSet

- Song Data: a set of 10000 song records with song metadata fields
  - Used metadata fields: duration, tempo, artist_hotness, key, mode, loudness
  - Dropped metadata fields: drop all remaining fields due to missing data

- User Taste Data: 48,373,586 records of 1,019,318 users' listening history
  - filter listening history records to only those with songs among the 10,000 song data
  - 418,252 users with 772,661 records remains, with listening count provided

User	Song	Listening count
abc	Sorry	1
cba	Timber	23

Sample data

Methods

- **Non-Network based Method**
    - Collaborative Filtering
- **Core: Network based Method**
    - BFS
    - Hitting Time

Method 1: Collaborative Filtering

- Intuition: How much do users similar to me like the song?
    - More listening counts, more likely to like the song!
- Challenges
    - Sparse → Not many people listened to the exact set of songs as you → correlation = 0.001?
    - Computation heavy if compare with everyone to find similar users or similar songs
- Solution
    - Dense Data Representation: Use k means clustering to cluster songs into 74 clusters. Represent a 124,257 x 74 matrix where each element is [user, within cluster listening count]
    - Computation speed up:
        - Use python sparse matrix that allows faster computation.
        - Apply early pruning to unpromising users that are not similar to you
        - Use clustering result to help narrow down search domain for similar songs

Method 1: Collaborative Filtering

- Dimension Reduction: K-means++ to cluster all songs to 74 clusters
    - Features: duration, tempo, artist_hotness, key, mode, loudness
    - Optimal # clusters: use np.sqrt(n_samples/2) and silhouette_score
- Algorithm: For each user in test data(user i, song s):
    - Step1: Find which cluster of music the person has ever listened to
    - Step2: Find all potentials who have also listened to the same clusters
    - Step3: Prune potentials who have listened to more than 2 clusters than the user
    - Step4:
        - Narrow down Very similar users: Among pruned potentials we find the cosine_similarity(potential, user) >= 0.6
        - Among very similar users:
            - Has Listened to the song s: look at his listening count
            - Not listened to the song s: find similar songs to song s using the clustering result and look for his listening count for similar songs

Method 1: Collaborative Filtering

- Prediction listening count for (**user i**, song s)

Sum over all very similar user j

$$\sum_{j} \frac{\text{cosine\_similarity}(\textbf{user i}, \text{user j}) * \text{cosine\_similarity}(\text{song s}, \text{song s'}) * \text{listening\_count}(\text{user j}, \text{song s'})}{\text{cosine\_similarity}(\textbf{user i}, \text{user j}) * \text{cosine\_similarity}(\text{song s}, \text{song s'})}$$

- If user j **has** listened to song s, **song s' = song s,** thus cosine_similarity(song s, song s') = 1
- If user j has **never** listened to song s, find similar song s' that is in the same cluster and find cosine_similarity(song s, song s')

Collaborative Filtering Result

Test data: 1485 records

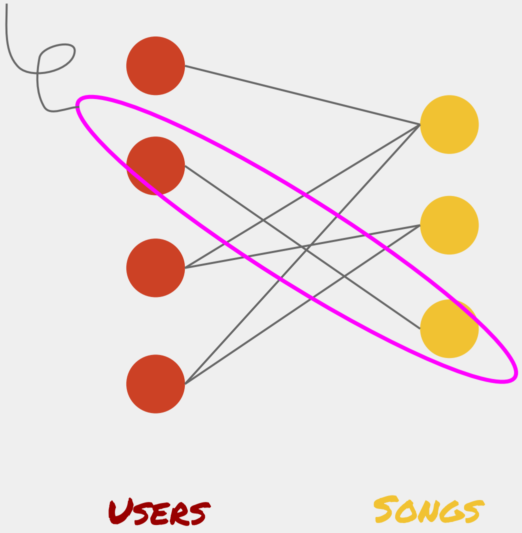
- Unpromising:
  - About half of the records, the system was unable to find similar songs in the same cluster that very similar user has listened to.
  - High Likeness Detection Accuracy: 13%
  - Overall Accuracy: 31%

Likeness	# prediction listening counts	# records
Low	<=2	396
Medium	<=4	366
High	>5	115
	NAN	608

Test mini: 877 records

Core: Network Representation

Bipartite Graph Representation:

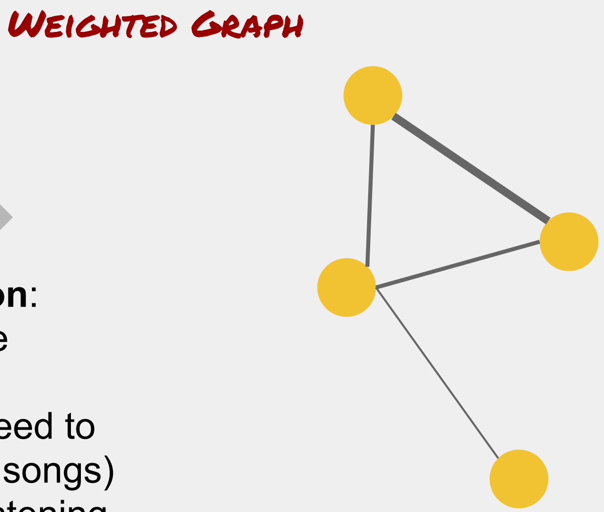


691 songs,
105,986 users,
286,480 records

Graph Creation Condition:

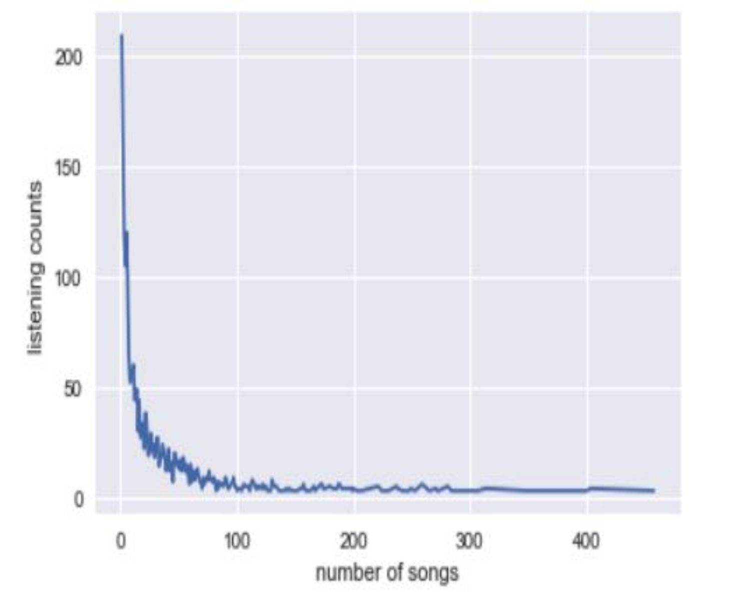
- ❖ All nodes need to be connected.
- ❖ Test data's songs need to be in the graph(344 songs)
- ❖ Test data's user's listening history need to be in the graph(564 songs)
- ❖ Networkx can handle the size

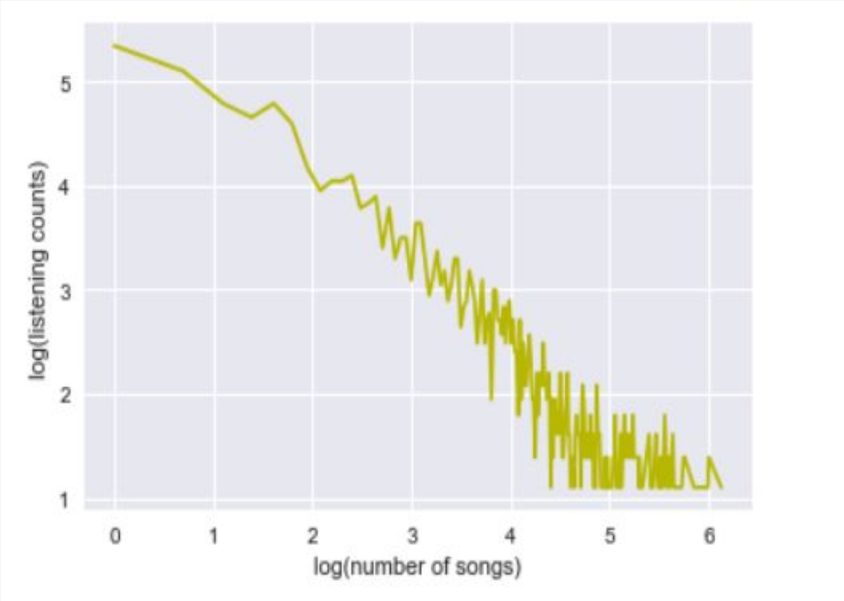
One Mode projection on Songs:



Network Statistics

- Degree distribution





Network Statistics

Number of nodes: 690

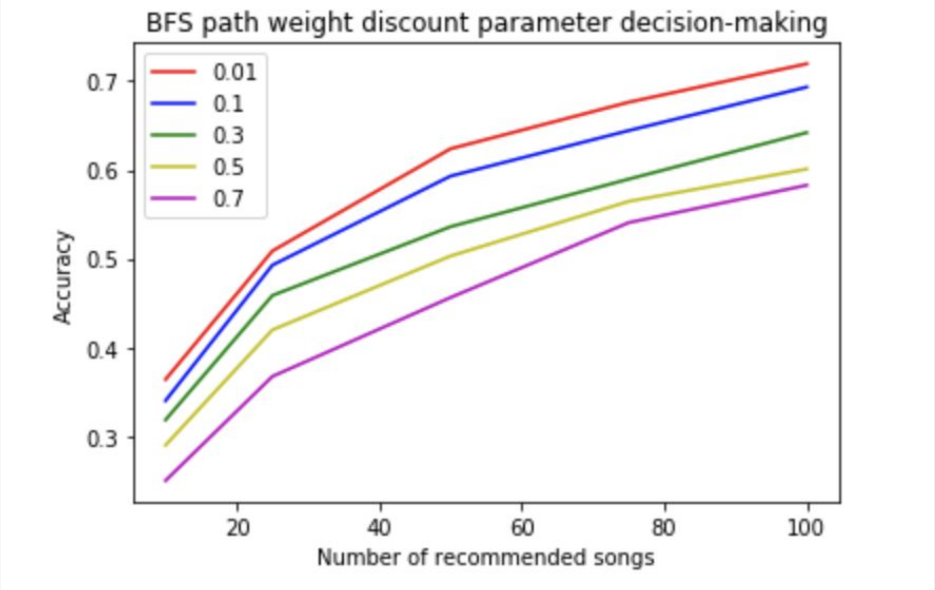
Number of edges: 69,912

Clustering coefficient: 0.703

Average shortest path length: 1.72

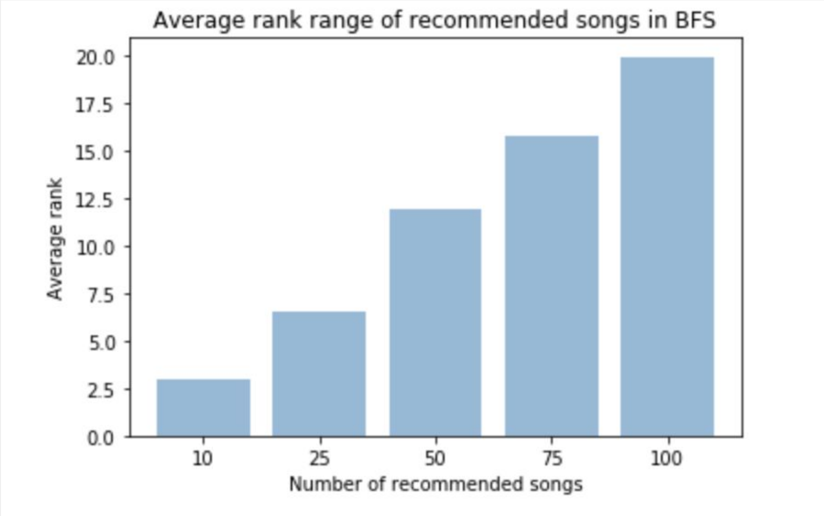
Method 2: BFS - Network based

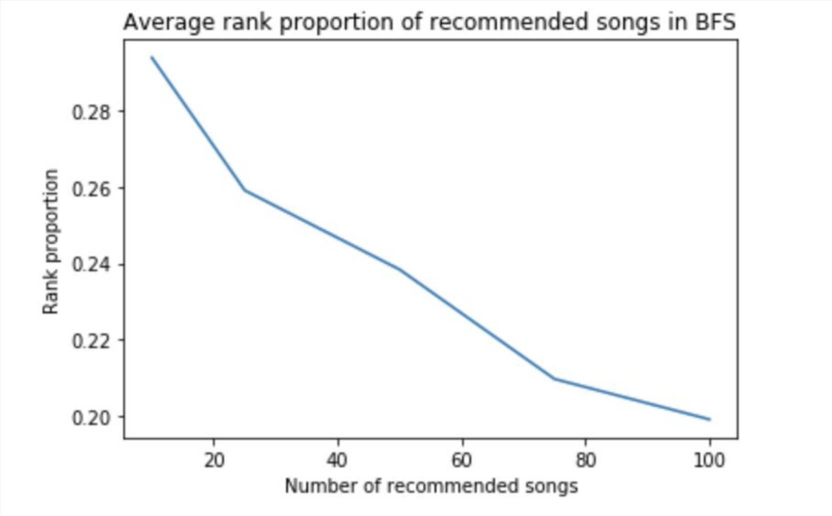
- Neighbors: Node with path length 1 & 2
- Path weight discount parameter: α
- Neighbor score
  - Node with path length 1: $\alpha * w1$
  - Node with path length 2: $\alpha * \alpha * (w1+w2)$
- Compromise for multiple occurences: *log(s1+s2)*
- Path weight discount parameter choice: α = 0.01



Method 2: BFS - Network based

- Challenge:
  - Choice making for path weight discount parameter
  - Recommendation score calculation
- Result with α = 0.01





Method 3: Hitting Time - Network based

Intuition: The lower the hitting time, the more likely it should be recommended to user.

- Hitting Time: Expected number of steps that a random walk i → j
- Can be Recursively defined. Complexity *O(|V|^3)*

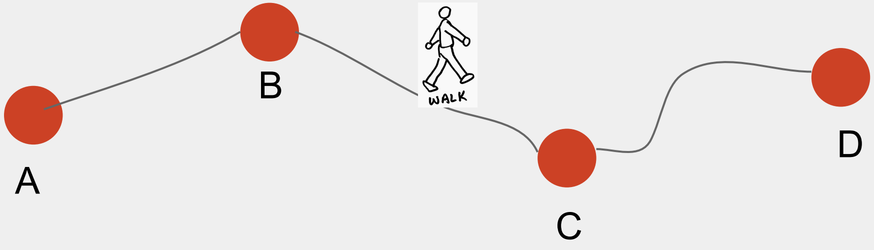
$$h_i^A = \begin{cases} \sum_{j \in V} \boxed{p(i \to j)}\boxed{h_j^A} + 1, & \text{for } i \notin A \\ 0, & \text{for } i \in A \end{cases}$$

Iterative Computation

Method 3: Hitting Time - Network based

Our Approach:

- Naive Monte Carlo Hitting Time
  - Simulate random walk from node i for k times
  - Estimate $h_{i,j}$ = (# random walks that reached j)/ k
- Multiwalk Monte Carlo Hitting Time
  - Observation: Each random walk can also be used to estimate sub-random walks! → **31874 walks!**



$h_{A,B} = 0.1$ ^ 1

$h_{A,C} = 0.1$ ^ 2

$h_{B,C} = 0.1$ ^ 1

Method 3: Hitting Time - Network based

Our Approach:

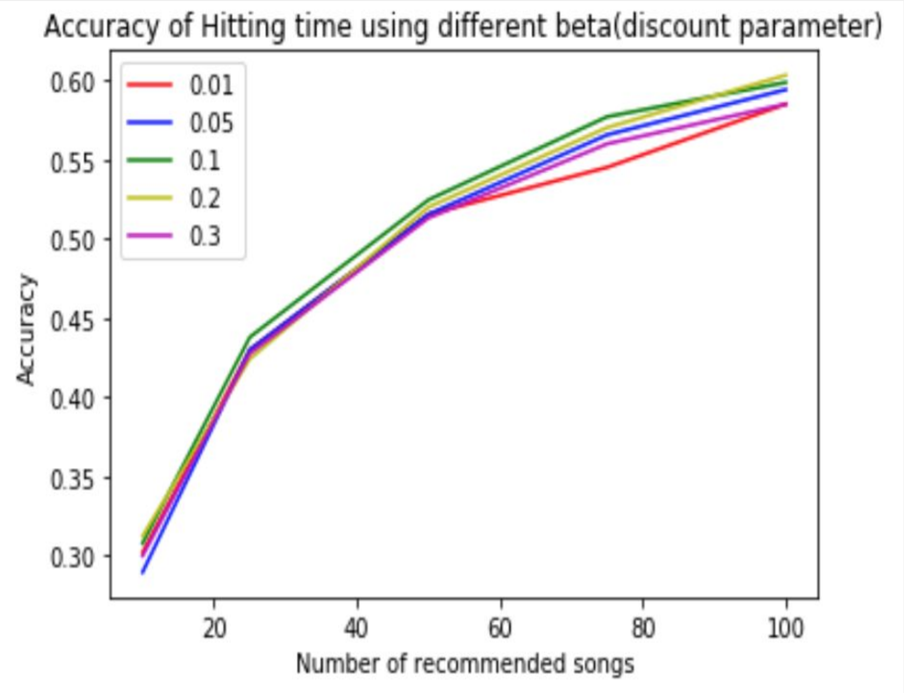
- Set-up:
  - For each user i in test data, we find all the songs she has listened to.
  - We set each of the listened song as an initial random walk start node.
  - Simulate α-terminating random walk k times. Edge weights is used to pick a path.
  - Record paths and the corresponding hitting time =β^step_size. → β = 0.1 Hitting time depends on both edge weights(implicitly) and step_size!
- α-Terminating Random Walk

$$\boxed{P(X_{t+1} = j | X_t = i) = \mathbb{1}[(1-\alpha)\frac{w_{i,j}}{\sum w_{i,j}}] \text{ if node } i, j \text{ is connected}}$$

- β-discounted length of Random Walk

Method 3: Hitting Time - Network based

Result:



- Evaluation:
- Best Parameter: $\beta = 0.1$
- Sharp increase in accuracy from top 10 to top 30.

1 Song 1

2 Song 2

3 Song 3

4 Song 4

Results

- Network based model performs better than non-network based ones in our recommendation mechanism.fif
    - Collaborative Filtering fails to find very similar users with similar songs.
- BFS performs better than Hitting Time among two network based models we study
- The highest accuracy in our study is reached by BFS with a value around 70% when p=0.01

Thank you!