LASTFM USER ANALYSIS

Overview

Whether a user of LastFM would follow another user

It can be used as a recommendation engine to improve user engagement and satisfaction

What is in the data

- **Nodes** are **LastFM users** from Asian countries.
- **Edges** are **mutual follower relationships** between them. This is *undirected* graph.
- There also exist **node features**, based on the **artists liked** by the users.
- And the **country label** to which a user belongs.
- Data is available for **7.6K users** and **28K mutual follower relationships** exist among them.

Data source: SNAP

Predicting User Connections and Communities

Link prediction based on existing links to identify if two users would **follow each-other**.

Apply **user community detection** technique to identify major influencers or hubs within the network.

Apply multinomial **node classification** to predict the **users' country**.

Note

- We will consider this data as a **static graph**, assuming it is not changing over time, to make the task simpler
- **Agnostic of retraining graph embedding** (which we plan to implement), as no new node will enter the graph. Otherwise, we would have to take a different approach of graph embedding (using graph neural network?)

Approach

Explore graph attributes

- Check topological attributes of the graph.
  - Degree distribution, average degree, average clustering coefficient, closeness centrality, etc.
- Explore open-source software Gephi to visualize this network besides using `spring_layout` of NetworkX.

Problem formulation – *link prediction*

- Remove some edges randomly from the graph and attempt to predict them.

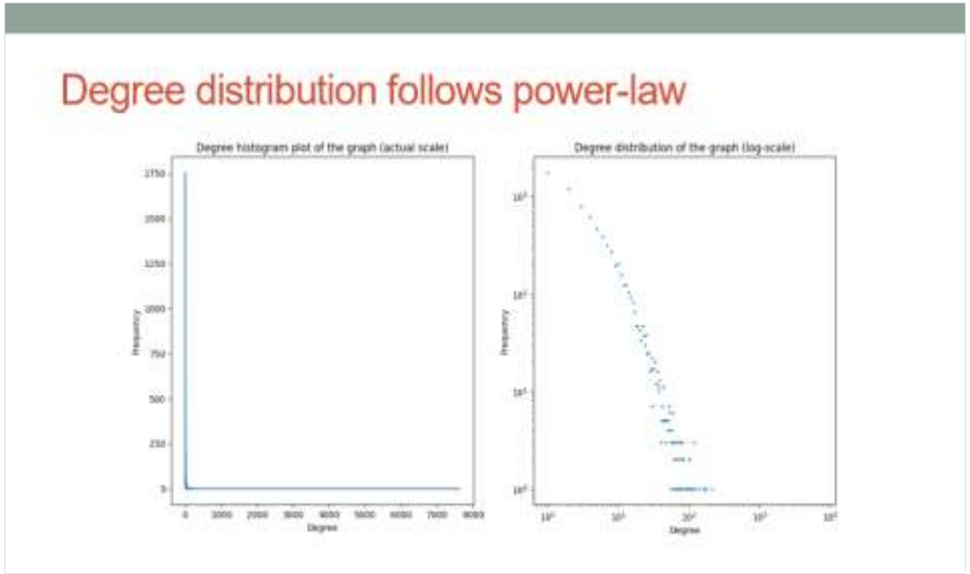
Possible approaches – *link prediction*

- *Adamic-Adar index*: A descriptive based metric that assigns large weights to common neighbors of *u* and *v* which themselves have few neighbors (weights rare features heavily).
- *Graph Embedding*: Train an embedding for each node and use the embedding vector to find the closest node on the embedding space.

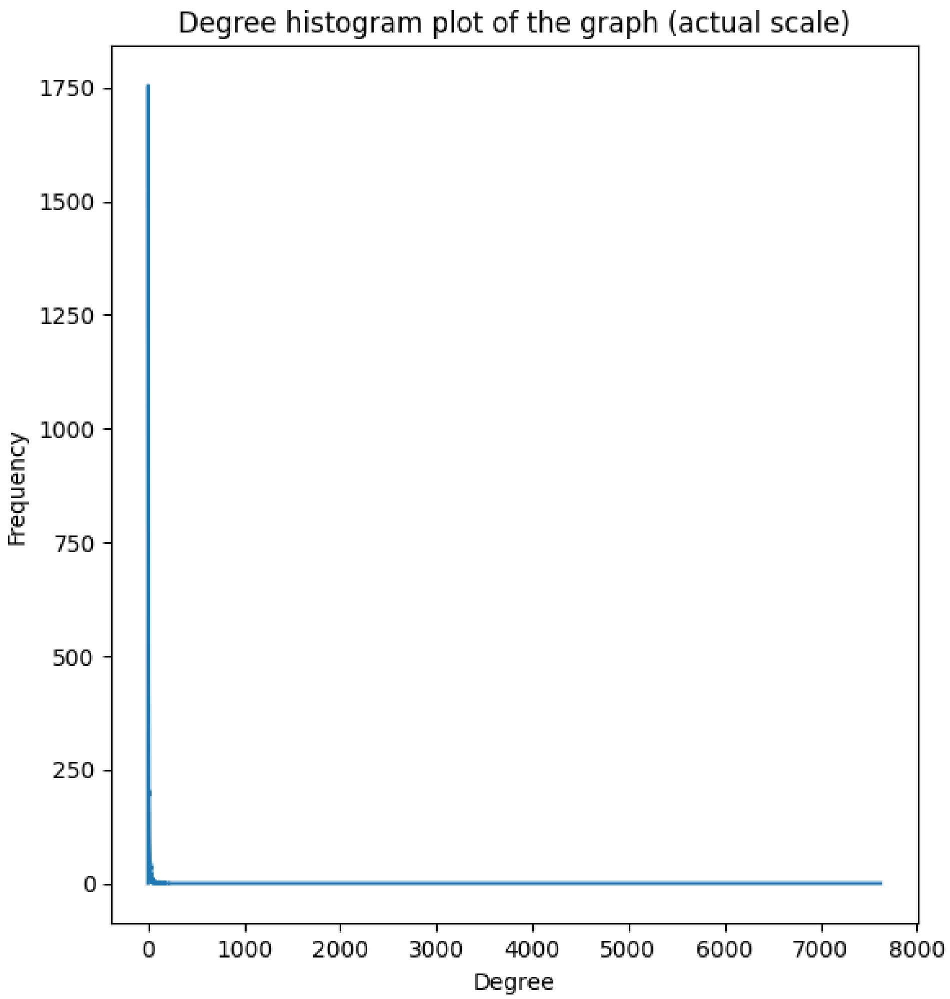
Note
Referred to the Stanford CS224 course page and their homework Colab1 to train an embedding on an undirected graph

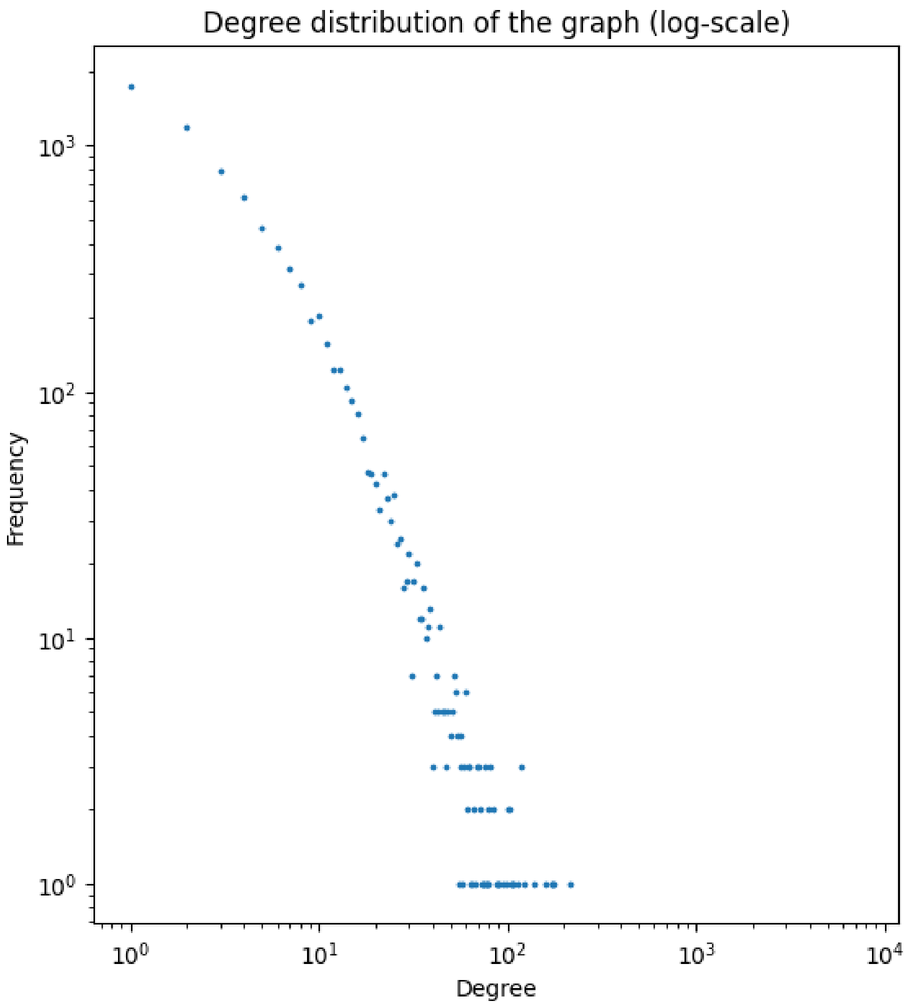
Network Characteristics

- **Number of nodes**: 7,624; **Edges**: 27,806
- **Average degree** (2*E/N): 7.29
- **Degree assortativity coefficient**: 0.02
  - **Cannot correlate** linked nodes based on their degrees.
- **Average clustering coefficient**: 22%
  - **Probability** that two randomly selected followers of a user are followers of each other is **not too small**

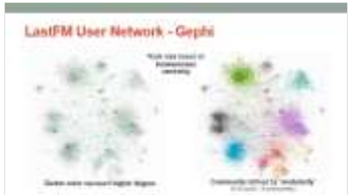


Degree distribution follows power-law





User 7199 connects multiple communities

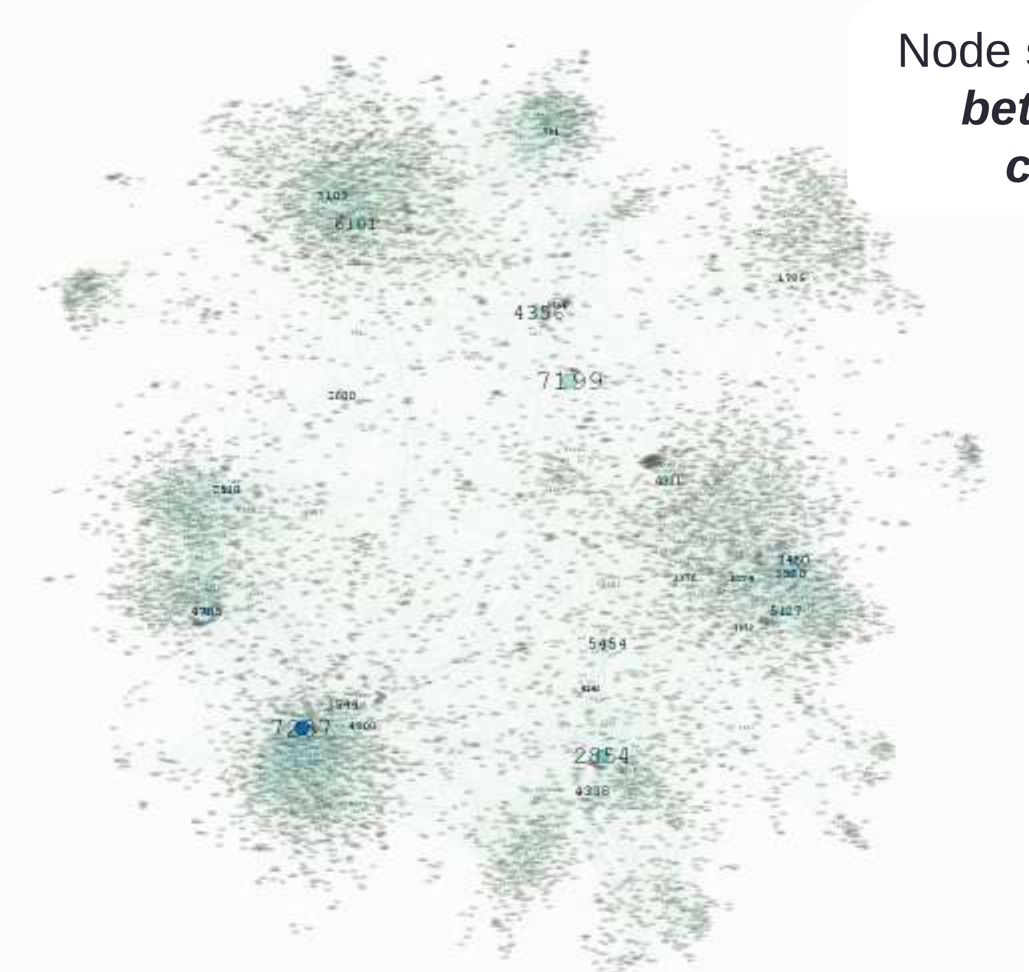


- *Degree centrality*:
  - **7237: 2.8%**, 3530: 2.3%, 4785: 2.3%, 524: 2.3%, 3450: 2.1%
- *Eigenvector centrality*:
  - **7237: 0.26**, 3240: 0.20, 3597: 0.19, 763: 0.18, 378: 0.16
- *Betweenness centrality*:
  - **7199: 0.09**, 7237: 0.09, 2854: 0.08, 4356: 0.07, 6101: 0.05

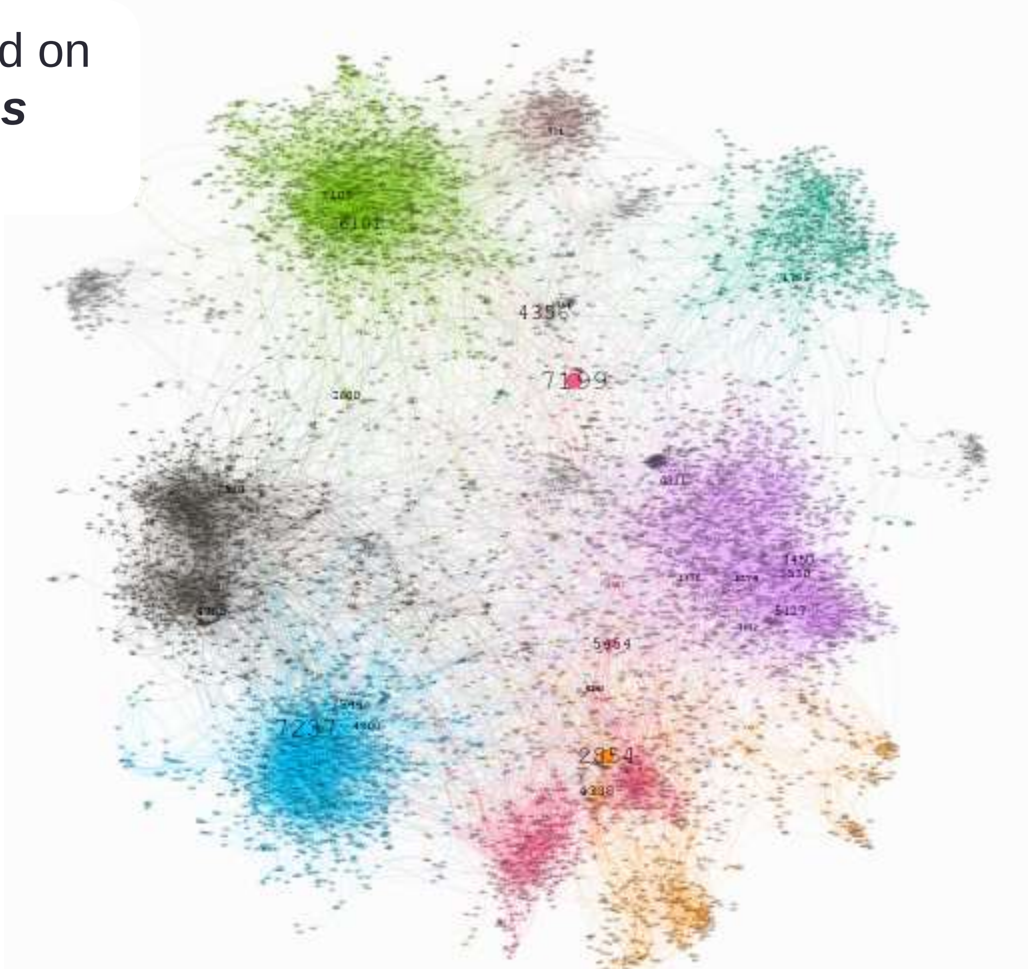
- **User \`7237\`** has the **highest degree of 216** and is mutual follower with 2.8% of all the users. The hubs in the graph are not as huge.
- **User \`7237\`** has the highest degree as well as **highest eigenvector centrality** (measuring **how connected a node is to other important nodes** in the network). Also, has one of the highest betweenness centralities, making the user very popular and influential to other nodes.
- User \`3240\`, \`763\` must have some very popular followers as they themselves do not have that many direct followers, but they have high eigenvector centrality.
- **User \`7199\`** has the **highest betweenness centrality** (percentage of all the **shortest paths of any two nodes** which **pass through given node**), but it is **not** in the **top-10** list of **degree centrality** neither in eigenvector centrality. This user is efficiently **connecting separate communities** in the user base.

LastFM User Network - Gephi

Node **size** based on ***betweenness centrality***



Darker color represent **higher degree**



Community defined by **`modularity`**
(81% score, 13 communities)

Link Prediction Overview

Prepare Training and Test Data

- Create negative samples randomly, up to the size of positive samples from original graph – the edges which do not exist are negative edges
- Remove some edges (10%) randomly, to be able to evaluate performance. These are positive edges for test set
- Include some edges (10%) randomly from the negative sample as well for test set

Node Embedding

- Each node gets mapped to d-dimensional vector space. The mapping will ensure "similar" nodes are placed "nearby“
- Methods to be explored:
  - Graph Factorization, DeepWalk, Node2Vec. Random walk based strategies work on undirected graph, not directed!

Descriptive Metrics

- Multiple metrics exist: Jaccards's coefficient, Adamic-Adar index, rooted PageRank, Katz centrality
- We will focus on Adamic-Adar index

Train node embedding – Graph Factorization*

“similarity” based on adjacency matrix

- Initialize an embedding in some d-dimension (64) space using random sample from U(0,1)
- Define target label as 1 for positive edges and 0 for negative edges
- Apply sigmoid on dot-product of the embedding weights representing an edge in training data
- Cross-entropy loss for binary classification
- Gradient descent (SGD, Adam, etc.) to update embedding vector minimizing loss
- Measure accuracy on the node pairs whose edges were removed earlier (test data). Also check performance on training data

Graph Factorization *as proposed by Ahmed et al., 2013*

Node Embedding Results – Graph Factorization

“similarity” based on adjacency matrix

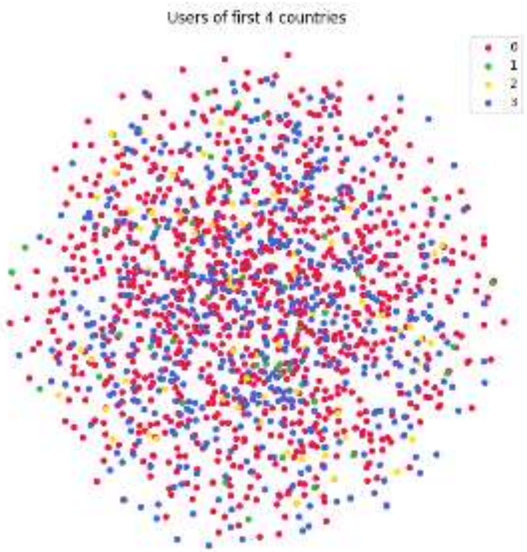
Hyperparameters

- d: 64, epochs: 150, optimizer: Adam, learning rate: 0.005

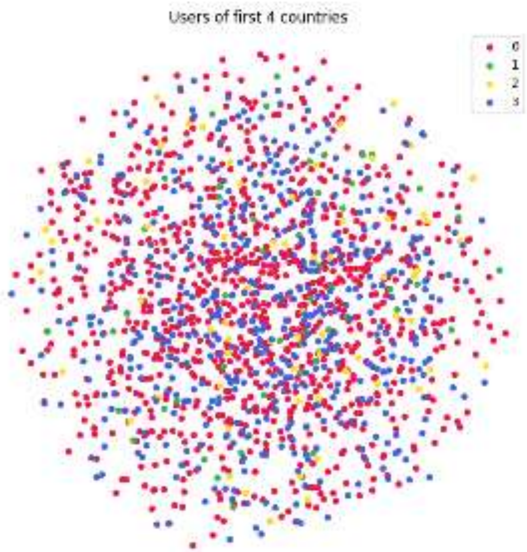
Performance

- *Training data*: F1-score 96%, Accuracy 96%
- *Test data*: F1-score 67%, Accuracy 55%

Not able to distinguish users (nodes) by their country (node label)



User embeddings **before training**

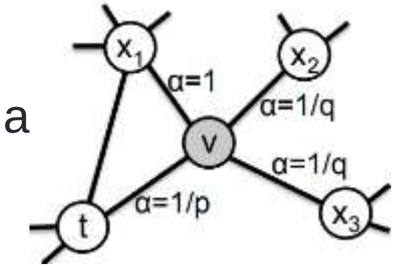


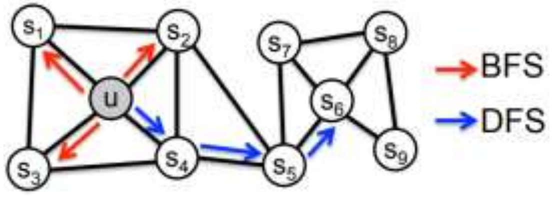
User embeddings **after training**

Train Node Embedding – DeepWalk*, Node2Vec*

"similarity" based on neighborhood

- Generate random walks (*biased in case of node2vec*) from each node of pruned graph for given length
- Initialize an embedding in some d-dimension space using random sample
- Update weights so that the probability of visiting neighborhood(u) is maximized
- Extract the node embeddings for each node participating in an edge - positive and nega
- Already have the true label (positive edge: 1, negative edge: 0)
- Multiple options to define the respective feature of the edge between nodes u and v
- Build logistic regression model to get the prediction probability of a link
- Measure accuracy on the node pairs whose edges were removed earlier (test data). Also check performance on training data





*__DeepWalk__ as proposed by [Perozzi et al., 2014](), *__Node2Vec__ as proposed by [Grover-Leskovec, 2016]()

Node Embedding Results – DeepWalk, Node2Vec

“similarity” based on neighborhood

Hyperparameters

- d: 64, epochs: 150, optimizer: SparseAdam, learning rate: 0.01
- batch size: 108, walk length: 6, context size: 4, walks per node: 20
- p: 1, q: 1 (*DeepWalk*); p: 4, q: 0.5 (*Node2Vec*)

Performance

DeepWalk

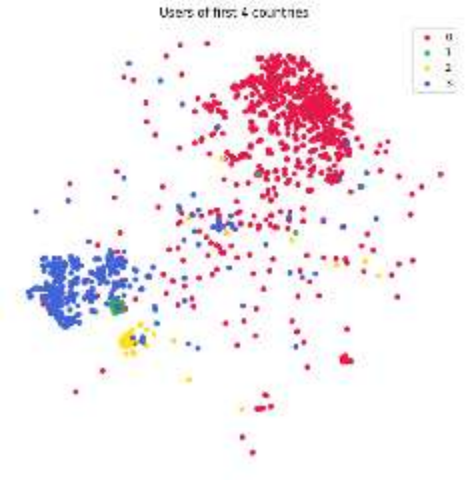
- *Training data*: F1-score 96%, Accuracy 96.3%
- *Test data*: F1-score 84%, Accuracy 86%

Node2Vec

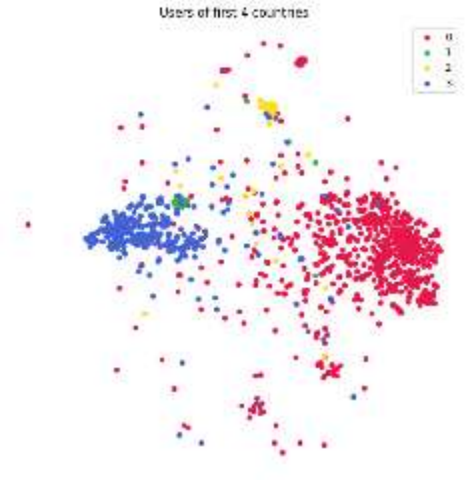
- *Training data*: F1-score of 96%, Accuracy of 96.4%
- *Test data*: F1-score 84%, Accuracy 86%

Referred to [Pytorch Geometric example on Node2Vec] for executing

✓ Users from same country tend to follow each other
✓ Able to distinguish users by their country very clearly



User embeddings **DeepWalk**



User embeddings **node2vec**

Link prediction using descriptive metrics

According to the paper The Link Prediction Problem for Social Networks (2004), the Adamic-Adar index had perfomed well on average across multiple social network based datasets they explored.

Approach:

- Figure out the list of ***nodes in the test set of edges*** which were removed and the test set of edges which did not exist in original graph
- ***Compute*** the Adamic-Adar index **for each** of those nodes **with all the nodes present in the graph**
- **Sort** the score in descending order, the **top scored is the predicted link for a given source** node
- Check **how many edges are predicted correctly** (positive test edge in predicted set, or negative test edge not in predicted set) out of all the edges in test set, that is the accuracy

Evaluation

- This approach predicted **overall 55.7% correctly** (**only 11.5% correct for positive edges** in test)
- It performed a lot worse than the node embedding based approaches, this one is biased towards negative edges

Next steps

- We assumed the potential new edge candidate is known, and checked whether that would happen or not. It is different from predicting new edges in future time with respect to the overall graph.
- Only the egde information has been used to predict a link, but node features also exists (artists liked by the user). Graph Neural Network can be utilized to use that information.
- Explored approaches would not work if a new node comes into the network, as that would require retraining from scratch. Again, Graph Neural Network may overcome this problem as long as the feature dimension does not change.

APPENDIX

Model iterations, NetworkX visuals

Hyperparameter experiments and epochs

Graph Factorization

		Training		Test	
Optimizer	Learning rate	F1-score	Accuracy	F1-score	Accuracy
Adam	0.005	96.2%	96.0%	67.1%	54.5%
SGD	2.5	66.8%	50.4%	66.6%	50.0%
SGD	5	76.0%	69.2%	66.1%	50.8%

```
epoch 010: loss 7.965736 accuracy 0.500000
epoch 020: loss 5.285167 accuracy 0.500000
epoch 030: loss 4.043911 accuracy 0.500000
epoch 040: loss 3.050901 accuracy 0.500000
epoch 050: loss 2.257896 accuracy 0.500000
epoch 060: loss 1.652953 accuracy 0.500040
epoch 070: loss 1.218219 accuracy 0.501359
epoch 080: loss 0.921730 accuracy 0.511468
epoch 090: loss 0.722815 accuracy 0.548749
epoch 100: loss 0.585928 accuracy 0.624710
epoch 110: loss 0.487021 accuracy 0.717354
epoch 120: loss 0.411868 accuracy 0.807900
epoch 130: loss 0.352404 accuracy 0.878327
epoch 140: loss 0.303969 accuracy 0.926037
epoch 150: loss 0.263737 accuracy 0.957904
```

*Training accuracy

DeepWalk

		Training		Test	
batch size	Operator	F1-score	Accuracy	F1-score	Accuracy
64	Concat	64.3%	64.6%	64.7%	64.5%
64	Hadamard	96.1%	96.2%	83.6%	85.4%
64	Sum	63.9%	64.3%	64.5%	64.6%
64	L2 norm	91.0%	91.0%	76.4%	79.3%
108	Hadamard	96.3%	96.3%	84.3%	86.0%

```
epoch 010: loss 1.011605 train acc 0.7659 test acc 0.5910
epoch 020: loss 0.795622 train acc 0.9089 test acc 0.7579
epoch 030: loss 0.764745 train acc 0.9503 test acc 0.8241
epoch 040: loss 0.756547 train acc 0.9608 test acc 0.8455
epoch 050: loss 0.754188 train acc 0.9635 test acc 0.8498
epoch 060: loss 0.753185 train acc 0.9641 test acc 0.8527
epoch 070: loss 0.754238 train acc 0.9646 test acc 0.8576
epoch 080: loss 0.754624 train acc 0.9640 test acc 0.8565
epoch 090: loss 0.754224 train acc 0.9641 test acc 0.8513
epoch 100: loss 0.755112 train acc 0.9638 test acc 0.8552
epoch 110: loss 0.754570 train acc 0.9636 test acc 0.8588
epoch 120: loss 0.754982 train acc 0.9636 test acc 0.8586
epoch 130: loss 0.754742 train acc 0.9638 test acc 0.8588
epoch 140: loss 0.754880 train acc 0.9634 test acc 0.8574
epoch 150: loss 0.754068 train acc 0.9634 test acc 0.8597
```

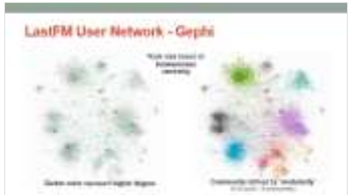
Loss curve hit a plateau?

Node2Vec

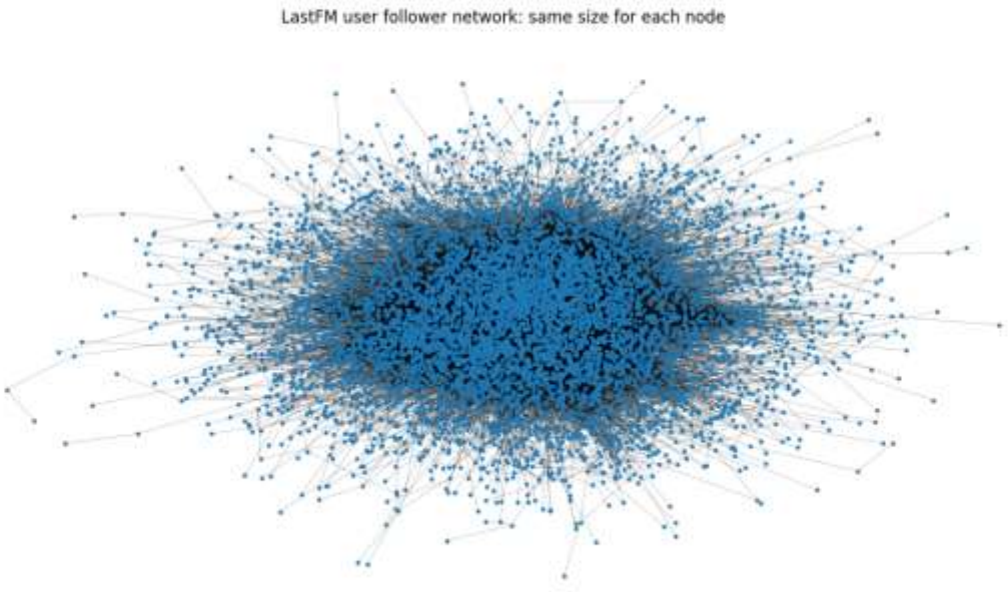
			Training		Test	
batch size	p	q	F1-score	Accuracy	F1-score	Accuracy
64	2	0.5	96.2%	96.2%	84.3%	86.0%
64	0.5	2	96.1%	96.1%	84.3%	85.9%
64	0.5	4	96.1%	96.1%	84.6%	86.2%
108	0.5	2	96.3%	96.4%	84.2%	85.9%
108	4	0.5	96.3%	96.4%	84.3%	86.0%
108	0.5	4	96.3%	96.3%	84.2%	85.9%

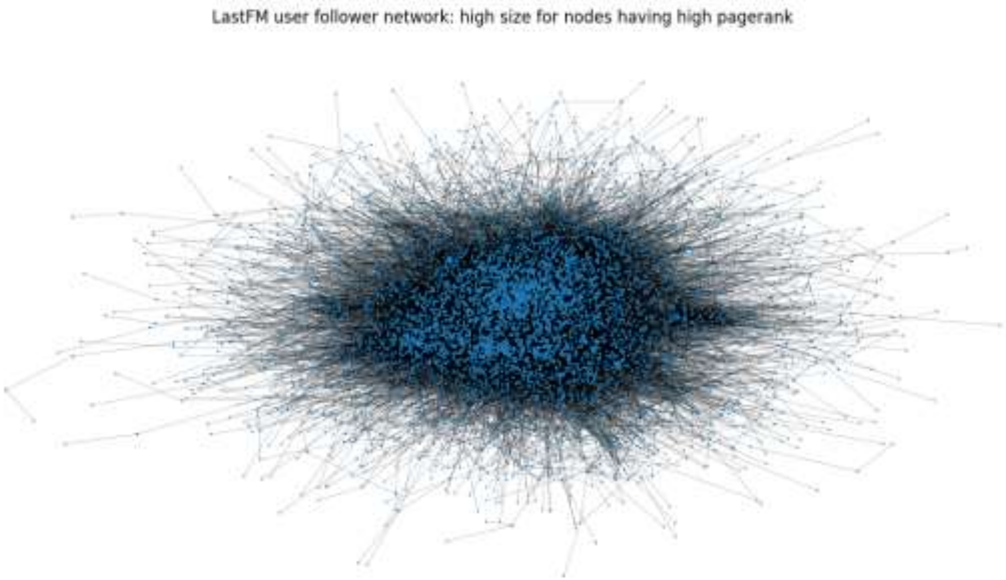
```
epoch 010: loss 1.012191 train acc 0.7647 test acc 0.5872
epoch 020: loss 0.795741 train acc 0.9106 test acc 0.7523
epoch 030: loss 0.764458 train acc 0.9515 test acc 0.8169
epoch 040: loss 0.756190 train acc 0.9610 test acc 0.8455
epoch 050: loss 0.754223 train acc 0.9632 test acc 0.8541
epoch 060: loss 0.752643 train acc 0.9627 test acc 0.8567
epoch 070: loss 0.754106 train acc 0.9643 test acc 0.8588
epoch 080: loss 0.753853 train acc 0.9640 test acc 0.8561
epoch 090: loss 0.754736 train acc 0.9635 test acc 0.8556
epoch 100: loss 0.754661 train acc 0.9631 test acc 0.8585
epoch 110: loss 0.755257 train acc 0.9627 test acc 0.8561
epoch 120: loss 0.754234 train acc 0.9626 test acc 0.8603
epoch 130: loss 0.754834 train acc 0.9639 test acc 0.8610
epoch 140: loss 0.754172 train acc 0.9640 test acc 0.8610
epoch 150: loss 0.754665 train acc 0.9636 test acc 0.8595
```

NetworkX visuals - 1



Used `spring_layout` to visualize the network

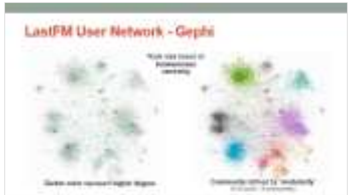




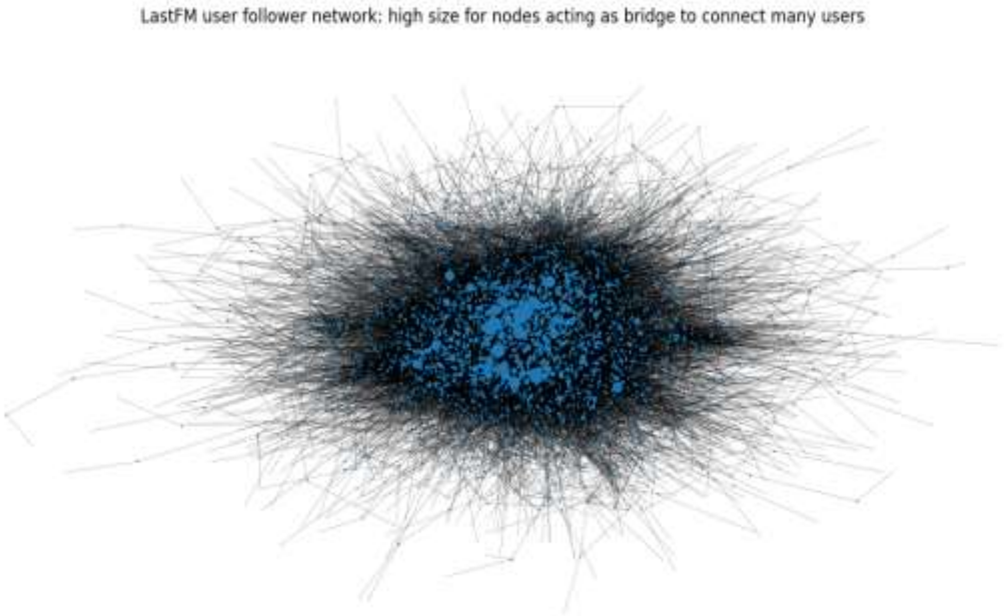
No apparent clusters visible in this layout

There are few nodes having high pagerank (which resembles the degree of that node in this case)

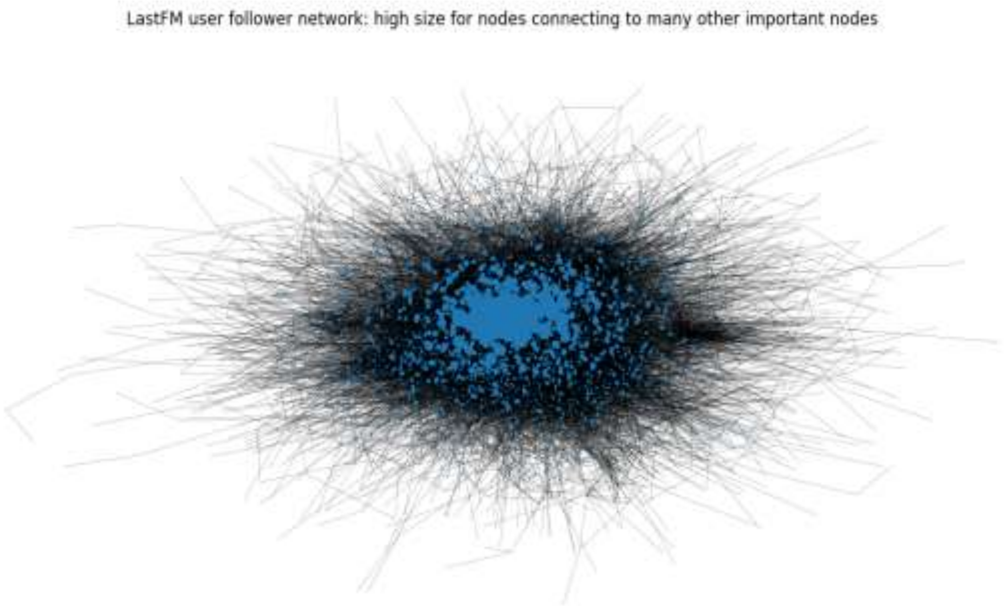
NetworkX visuals - 2



Used `spring_layout` to visualize the network



A handful of nodes have comparatively high betweenness centrality - they are forming bridge to connect different communities. This chart may help in understanding some cluster of users.



There are decent number of users who follow other popular users; they themselves do not necessarily have high number of mutual followers.