Model Explainability

Model inference (Explaining the model output for a single input i.e. one node)

- Model inference answers the question:- Which features of the input are the most influential in governing its output
- In this use case it will answer:- which nodes affect a node's output the most?
- It also analyses the importance of each feature from the feature_vector of the node. However this is not as meaningful because node2vec embeddings used as features don't hold semantic information.

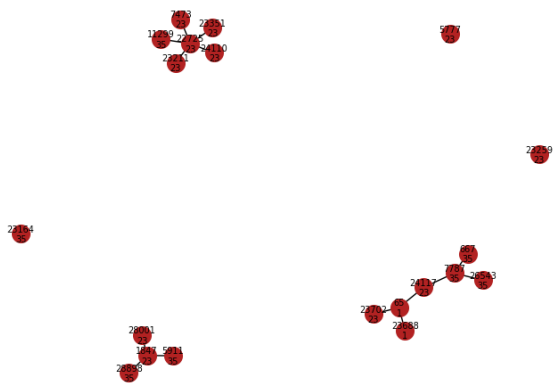

Fig 1. Graph representing top 20 most "influential" nodes for node 65

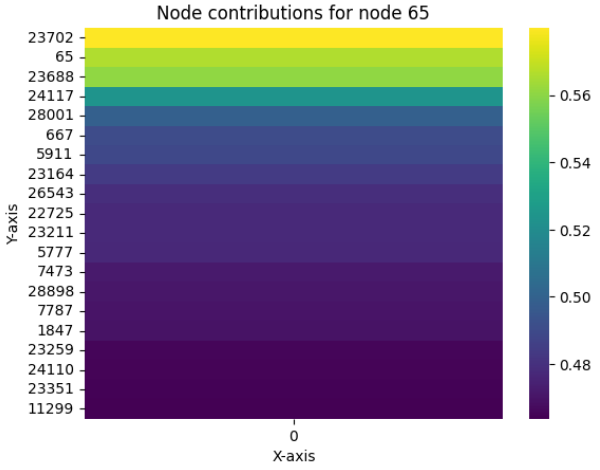

Fig 2. HeatMap representing the top 20 most "influential" nodes for node 65

Observations

- As expected, one hop neighbours are the most influential for node 65.
- Not only neighbourhood nodes, but nodes further away also influence a nodes output.
- Patterned structures are often influential even if they are further away. For example, a five and three pointed star formations influence the output of node 65.

pytorch_geometric's GNNExplainer is used to compute the explanation for a node.
This explanation is in the form of a weighted mask which has a value proportional to its influence of the node (0 in the case of no influence and positive indicating how much influence)

Grouped inference (Visualising how the model distinguishes different communities)

The following is the procedure to perform grouped inference:

1. Compute model inferences (masks) for nodes across communities
2. Embed these masks into a lower dimension (2 dimensions for a scatter plot). umap module was used for dimensionality reduction
3. Perform clustering and computation of silhouette score to gauge how well the model can differentiate between the different classes

The following are the uses:

- To visualise the ability of the model to differentiate between the classes
- To gauge where the model is getting "confused" (i.e. which classes are getting mixed/similar). This can help us decide what to do to make the model understand clearly. For example we can collect more data for specific classes that the model is getting confused with.
- To understand the way the model "sees" or has modelled the input data.

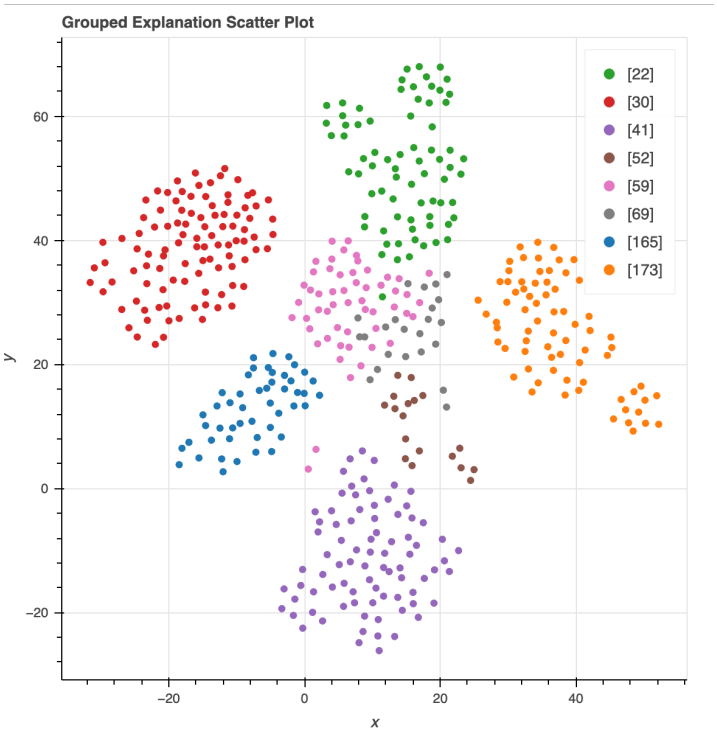

Fig 3. Grouped Explanation scatter for 8 communities

Observations
- The model can easily distinguish between the following communities: 30, 41, 165, 173
- The model cannot easily distinguish between the following communities(having overlaps): 22, 52 ,59, 69

Clustering results
- K-means
  - Silhouette score: 0.473

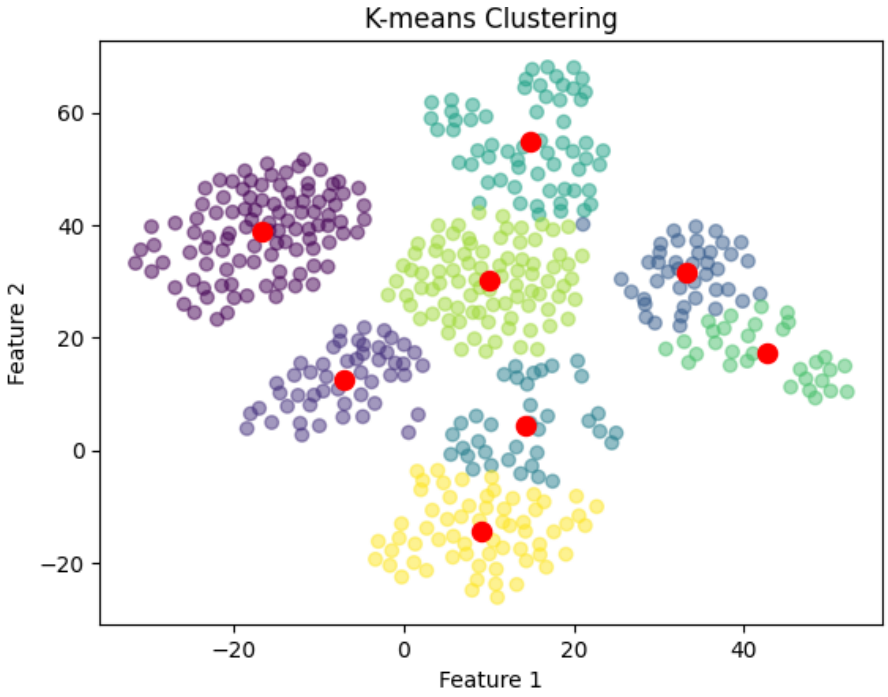

Fig 4. K-means clustering result

- DBSCAN clustering
  - Silhouette score: 0.517

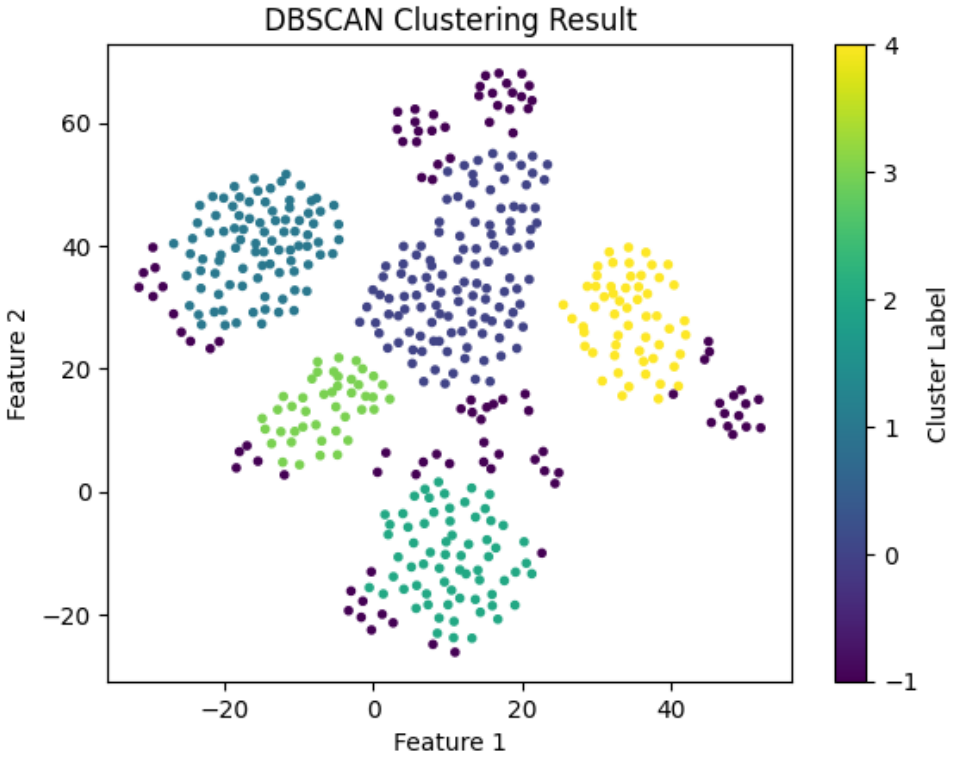

Fig 5. DBScan clustering result

- Agglomerative Hierarchical Clustering
  - Silhouette Score: 0.517

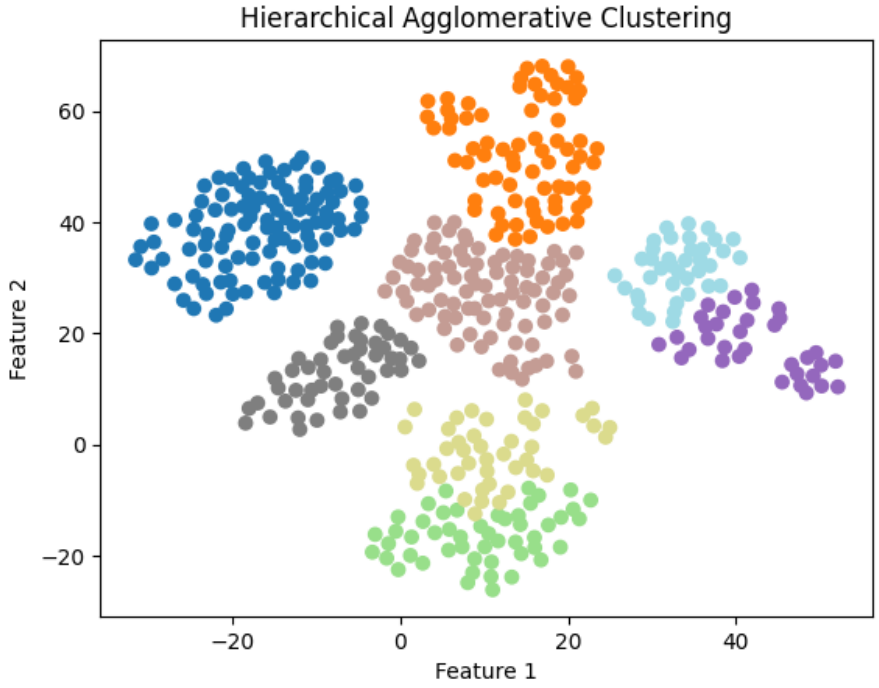

Fig 6. Agglomerative clustering result

Observation from clustering results:

- Low value of silhouette scores
  - Our use case is multi-label classification (one node can be a member of multiple communities) so by the nature of the problem communities can be overlapping. The nature of the problem causes clear cut groups to not exist.
  - The use case for the model is recommender system which is anticipatory in nature. Hence, the overlaps have a positive effect on the quality of recommendations.
  - Increasing the dimension of the umap embedding (2 in the above figures) does not significantly increase the clustering performance further indicating the natural overlap of communities.

APIs

Secure APIs are created for safe access to the model. Currently, 3 APIs have been exposed:

1. /query_node
   - To query an existing node
   - Returns a list of suggested/recommended communities (apart from the ones it is already a part of)

2. /query_community
   - To query a community
   - Returns a list of suggested/recommended nodes (apart from the ones already a part of the community)

3. /query_new_node
   - To query a new node (which is not already in the database)
     - A new node is defined by its neighbours. So this API endpoint only takes in a list of nodes the new node is connected to.
     - The new nodes feature vector is computed as the average of feature vectors of all its neighbours and new edges are added to the edge_list.
     - The forward pass is computed on this appended dataset and results are obtained.
   - Returns a list of suggested/recommended communities likely for the new node to join