

Part 1: Unveiling Patterns

Exploratory Data Analysis (EDA) is required for data understanding so that we can make modeling decisions. It also helps us in data preprocessing.

Visualizing the graph:



Finding the node degree and arranging them as a pandas data frame - by doing this we can see that there are certain nodes which have more importance as they have a high degree and then there are nodes with 1 or 0 degree, which are less important to our problem

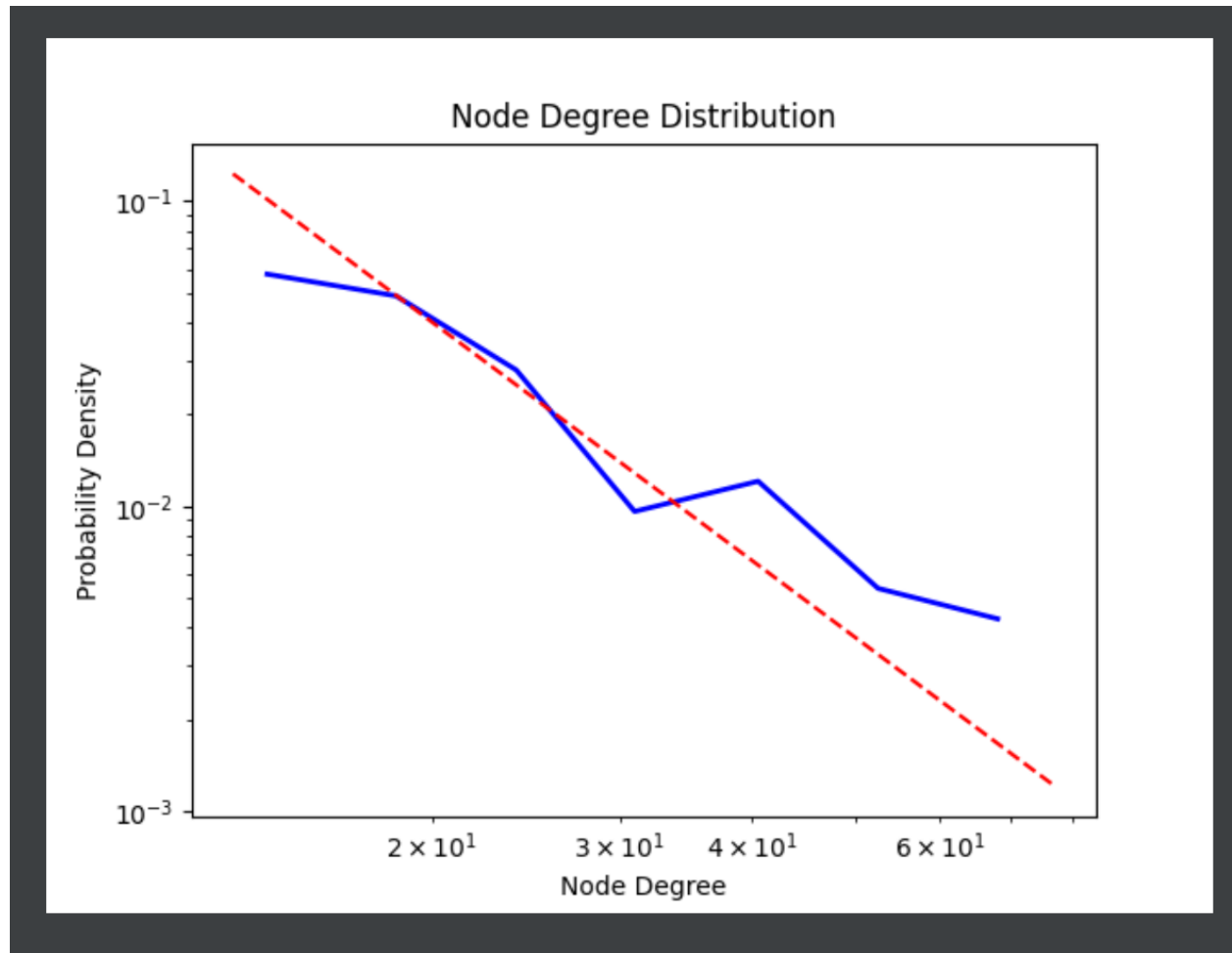
Checking if graph exhibits power law distribution -

Calculating best minimal value for power law fit

Power-law alpha: 2.5997905949181854

This is to check if the graph is possibly a subgraph of a larger graph. If it does not follow then there may be a potential sample selection bias.

But in our case since alpha is close to 2, we can say that we follow a power law distribution and hence we are fine



Finding the density and average degree of graph -

We can find it by dividing the no of edges by no of nodes. The higher the density, the richer the graph is which gives us more possibility in feature learning i.e. more options on hyper parameter tuning.

In our case we get the following result which is pretty good and since the average node degree is not that high it means feature learning is limited to the connections.

The density of the graph is: 7

The average degree of the graph is: 14.518731988472622

Checking if graph is empty, self loops and weighted -

We can see that the graph is not empty and the relationships are not weighted, and also no self loops therefore feature learning will not have a significant impact.

The self loops in the graph are: 0

Part 2: Crafting Tomorrow's Chapters

Here the problem statement is to provide fresh co-authoring opportunities to a given author who approaches us.

I have approached this problem with 2 solutions both of which are a recommendation system but one uses collaborative filtering and another one uses link prediction.

Link prediction directly uses node feature vectors but performs less better when compared to collaborative filtering approach in which we have to embed the feature vectors manually. In both the models, the performance can be improved further with hyperparameter tuning, using different optimizers or adding more layers to the GNN

Also as the problem statement is to give personalized co-author recommendations based on an author and his features, collaborative filtering excels in this area. But the other approach can also be used as well

For both the models, I will provide results for nodes with id 56, 100, 343. So that I can include as much variation as possible.

Results for model.py (Approach using Collaborative filtering) -

Top 5 recommendations for node 56: [112, 61, 127, 207, 3]

Likelihood scores: [tensor(0.1807), tensor(0.1032), tensor(0.0527), tensor(0.0491), tensor(0.0466)]

Top 5 recommendations for node 100: [148, 221, 42, 40, 44]

Likelihood scores: [tensor(0.2303), tensor(0.1122), tensor(0.0815), tensor(0.0597), tensor(0.0517)]

Top 5 recommendations for node 343: [61, 124, 180, 221, 52]

Likelihood scores: [tensor(0.2681), tensor(0.1506), tensor(0.0992), tensor(0.0518), tensor(0.0476)]

The screenshot shows an IDE with a file named `model.py` open. The code in the editor consists of 19 lines, each mapping a feature name to a numerical value, such as `"n.Feature31 as Feature31, n.Feature32 as Feature32, "`. Below the editor, the Run console displays the output of the program. It shows likelihood scores and top 5 recommendations for four different nodes (343, 344, 345, and 346). For each node, it lists five items with their likelihood scores. At the bottom of the console output, it states 'Hit Rate at 5: 0.0086' and 'Trained model, recommendations, and likelihood scores saved to 'model.h5''. The IDE interface includes a menu bar at the top, a sidebar on the left with 'Project' and 'Structure' views, and a status bar at the bottom showing system information like temperature (37°C) and date (02-09-2023).

```
88 "n.Feature31 as Feature31, n.Feature32 as Feature32, "  
89 "n.Feature33 as Feature33, n.Feature34 as Feature34, "  
90 "n.Feature35 as Feature35, n.Feature36 as Feature36, "  
91 "n.Feature37 as Feature37, n.Feature38 as Feature38, "  
92 "n.Feature39 as Feature39, n.Feature40 as Feature40, "  
93 "n.Feature41 as Feature41, n.Feature42 as Feature42, "  
94 "n.Feature43 as Feature43, n.Feature44 as Feature44, "  
95 "n.Feature45 as Feature45, n.Feature46 as Feature46, "  
96 "n.Feature47 as Feature47, n.Feature48 as Feature48, "  
97 "n.Feature49 as Feature49, n.Feature50 as Feature50, "  
98 "n.Feature51 as Feature51, n.Feature52 as Feature52, "  
99 "n.Feature53 as Feature53, n.Feature54 as Feature54, "  
100 "n.Feature55 as Feature55, n.Feature56 as Feature56, "  
101 "n.Feature57 as Feature57, n.Feature58 as Feature58, "  
102 "n.Feature59 as Feature59, n.Feature60 as Feature60, "  
103 "n.Feature61 as Feature61, n.Feature62 as Feature62, "  
104 "n.Feature63 as Feature63, n.Feature64 as Feature64, "  
105 "n.Feature65 as Feature65, n.Feature66 as Feature66, "  
106 "n.Feature67 as Feature67, n.Feature68 as Feature68, "  
107 "n.Feature69 as Feature69, n.Feature70 as Feature70, "
```

Run: model x

```
Likelihood scores: [tensor(0.0174), tensor(0.0127), tensor(0.0188), tensor(0.0180), tensor(0.0099)]  
Top 5 recommendations for node 343: [61, 124, 180, 221, 52]  
Likelihood scores: [tensor(0.2681), tensor(0.1506), tensor(0.0992), tensor(0.0518), tensor(0.0476)]  
Top 5 recommendations for node 344: [100, 61, 48, 207, 46]  
Likelihood scores: [tensor(0.0975), tensor(0.0841), tensor(0.0581), tensor(0.0575), tensor(0.0569)]  
Top 5 recommendations for node 345: [223, 167, 147, 19, 199]  
Likelihood scores: [tensor(0.2131), tensor(0.2019), tensor(0.1450), tensor(0.1018), tensor(0.0402)]  
Top 5 recommendations for node 346: [61, 112, 124, 48, 60]  
Likelihood scores: [tensor(0.5371), tensor(0.2198), tensor(0.1506), tensor(0.0118), tensor(0.0189)]  
Hit Rate at 5: 0.0086  
Trained model, recommendations, and likelihood scores saved to 'model.h5'
```

Results for model2.py (Approach using Link prediction) -

Top 5 recommendations for node 56: [20, 140, 180, 183, 167]

Likelihood scores: [0.010310431942343712, 0.007679259404540062, 0.007418027613312006, 0.007402912247925997, 0.007058937568217516]

Top 5 recommendations for node 100: [204, 60, 160, 208, 1]

Likelihood scores: [0.008443913422524929, 0.008035272359848022, 0.007700406480580568, 0.007544406224042177, 0.007389116566628218]

Top 5 recommendations for node 343: [15, 8, 43, 186, 7]

Likelihood scores: [0.006753073073923588, 0.006075149867683649, 0.00603825505822897, 0.005855973344296217, 0.00574180344119668]

```
43 # Compute a new edge feature named 'score' by a dot-product between the
44 # source node feature 'h' and destination node feature 'h'.
45 g.apply_edges(fn.u_dot_v('h', 'h', 'score'))
46 # u_dot_v returns a 1-element vector for each edge so you need to squeeze it.
47 return g.edata['score'][:, 0]
48
49
50 # setting up neo4j graph database connection
51 uri = "bolt://127.0.0.1:7687"
52 username = "neo4j"
53 password = "Sunny1758#"
54
55 driver = GraphDatabase.driver(uri, auth=(username, password), encrypted=False)
56
57 features_dict = {}
58 author_id_dict = {}
59
60 # this function allows us to load the nodes, edges, features of each node using cypher commands
61 def load_neo4j_data(driver):
62     with driver.session() as session:
```

Run: model2.py

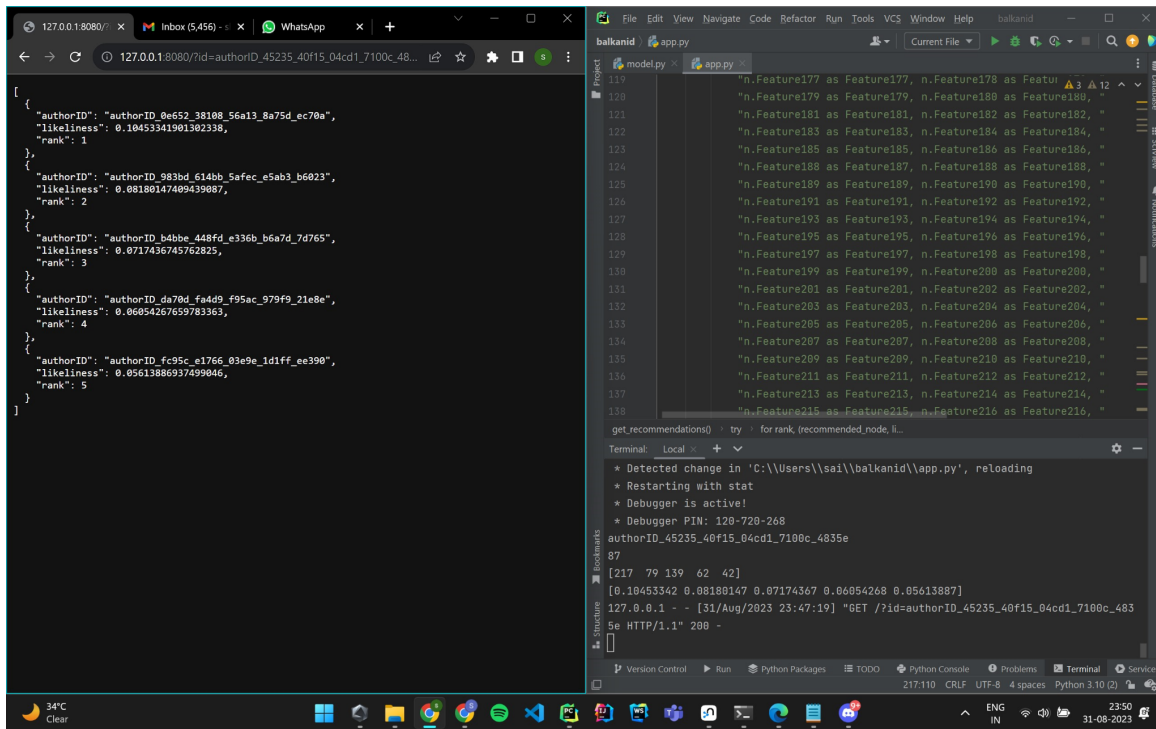
```
Top 5 recommendations for node 342: [43, 15, 113, 196, 7]
Likelihood scores: [0.007423189468681812, 0.006450525019317865, 0.006444701459258795, 0.006301746238023043, 0.006052941083908081]
Top 5 recommendations for node 343: [15, 8, 43, 186, 7]
Likelihood scores: [0.006753073073923588, 0.006075149867683649, 0.00603825505822897, 0.005855973344296217, 0.00574180344119668]
Top 5 recommendations for node 344: [103, 148, 15, 215, 71]
Likelihood scores: [0.005723483394831419, 0.005518114194273949, 0.0055097658187150955, 0.005500938277691603, 0.005453578196465969]
Top 5 recommendations for node 345: [103, 148, 15, 215, 71]
Likelihood scores: [0.005723483394831419, 0.005518114194273949, 0.0055097658187150955, 0.005500938277691603, 0.005453578196465969]
Top 5 recommendations for node 346: [210, 195, 2, 66, 184]
Likelihood scores: [0.007371255196630955, 0.007364379707723856, 0.006422464270144701, 0.006214916240423918, 0.006188481114804745]
Hit Rate at 5: 0.0086
```

Part 3: Cloud Chronicles

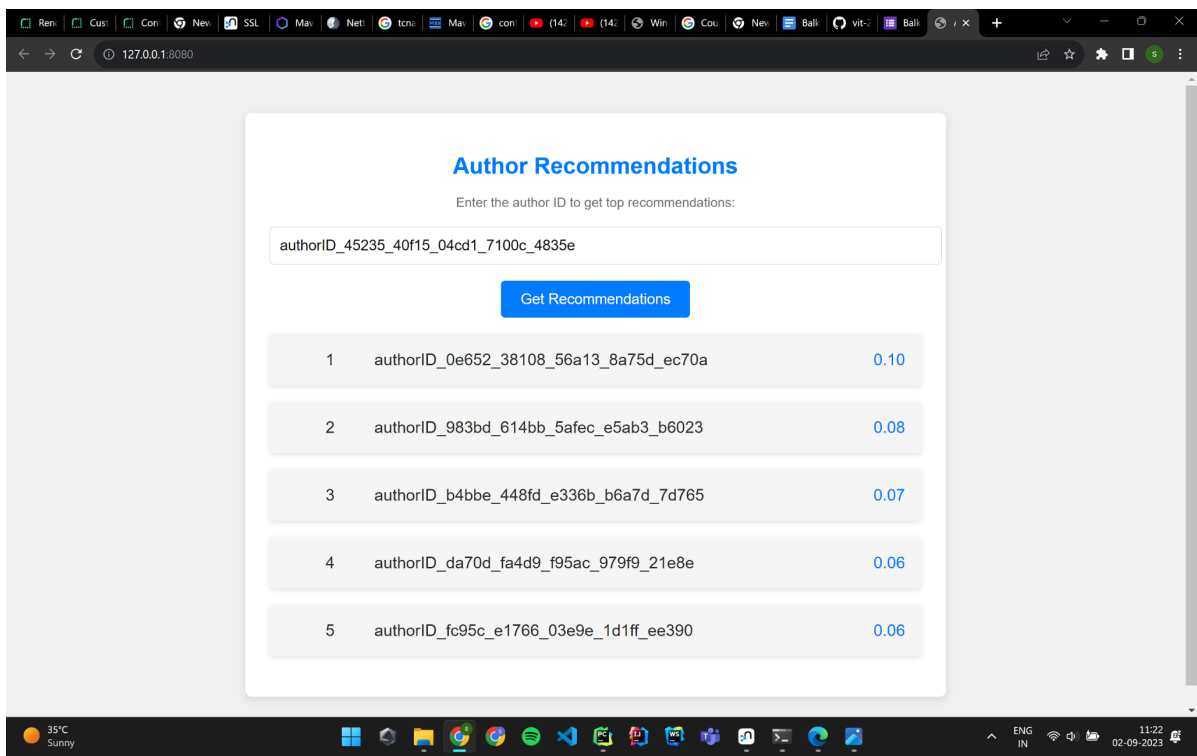
Created a flask application and used h5py module to store the model outputs (collaborative filtering as it performs better). The flask application takes the id parameter from GET Request, finds the nodeid for the given authorid and finds the recommended coauthor (nodes) and likelihood scores using the .h5 file.

All of the results mentioned are using model.py (collaborative filtering approach)

These results are stored in the JSON format response as mentioned in the file



After getting the response in the required format, then I included a index.html file as well style.css to improve the front end of the application.



```
C:\Users\sai>curl -X GET "http://127.0.0.1:8080/get_recommendations?id=authorID_45235_40f15_04cd1_7100c_4835e"
[
  {
    "authorID": "authorID_0e652_38188_56a13_8a75d_ec70a",
    "likeliness": 0.10453341901382338,
    "rank": 1
  },
  {
    "authorID": "authorID_983bd_614bb_5afec_e5ab3_b6023",
    "likeliness": 0.08180147409439087,
    "rank": 2
  },
  {
    "authorID": "authorID_b4bbe_448fd_e336b_b6a7d_7d765",
    "likeliness": 0.0717436745762825,
    "rank": 3
  },
  {
    "authorID": "authorID_da70d_fa4d9_f95ac_979f9_21e8e",
    "likeliness": 0.06054267659783363,
    "rank": 4
  },
  {
    "authorID": "authorID_fc95c_e1766_03e9e_1d1ff_ee390",
    "likeliness": 0.056138869377499046,
    "rank": 5
  }
]

C:\Users\sai>
```

```
181 for idx, node_id in enumerate(all_nodes):
182     recommended_nodes_dict[node_id] = recommended_nodes_data[idx]
183     likelihood_scores_dict[node_id] = likelihood_scores_data[idx]
184
185
186 p.route('/')
187 index():
188     return render_template('index.html')
189
190 # dd a route to serve static files (CSS, JavaScript, etc.)
191 p.route('/static/SPATH:path')
192 send_static(path):
193     return send_from_directory('static', path)
194
195 p.route('/get_recommendations', methods=['GET'])
196 get_recommendations():
197     try:
198         # Parse the author ID from the query parameter 'id'
199         author_id = request.args.get('id')
200         print(author_id)
201         # Find the corresponding node ID for the author ID
202         node_id = None
203         for id, a_id in author_id_dict.items():
204             if a_id == author_id:
```

```
Terminal: Local
authorID_45235_40f15_04cd1_7100c_4835e
87
[217 79 139 62 42]
[0.10453342 0.08180147 0.07174367 0.06054268 0.05613887]
127.0.0.1 - - [02/Sep/2023 11:27:46] "GET /get_recommendations?id=authorID_45235_40f15_04cd1_7100c_4835e HTTP/1.1" 200 -
```

After the successful deployment of the flask app on the local host, I tried the following cloud deployment websites to deploy my flask application in. They are pythonanywhere, render and google cloud.

But in all of the above mentioned deployment websites, I was getting the following error -

Sep 1 07:46:33 PM neo4j.exceptions.ServiceUnavailable: Couldn't connect to 127.0.0.1:7687 (resolved to ()): Sep 1 07:46:33 PM Failed to establish connection to ResolvedIPv4Address(('127.0.0.1', 7687)) (reason [Errno 111] Connection refused) - how to solve this error, that I am getting because neo4j database is not getting accessed.

I went through stack overflow, neo4j documentation, github and the only solution I was able to find was I needed to setup SSL certification and then use HTTPS instead of BOLT, as the deployment website was using HTTPS protocol.

<https://neo4j.com/docs/operations-manual/current/security/ssl-framework/#ssl-certificates>

I tried setting up the certificates and SSL over HTTPS by following the above documentation, But when I did follow all the steps mentioned above, I was getting the DBMS cannot be setup error in neo4j Desktop and could not find any solutions for it. Hence I was unable to setup the application of Cloud, in the given time frame. I had to use neo4j Desktop, instead of it's web version or sandbox, as only the desktop version had accepted the .dump file format which was provided to us in the dataset.

I would have resolved this issue, if I had more time, but I gave my best trying to debug it and solve it. The other solution, I think of as now, is to save the made dgl graph use it for getting nodes, edges and feature vectors, rather than importing neo4j and using `load_neo4j_data`