real_multiplex_analysis

April 13, 2022

1 Prereqs and config

[ ]:
```
# Standard library

# Scientific computing
import numpy as np
import powerlaw
from sklearn.metrics import normalized_mutual_info_score as nmiscore

# Network science
import networkx as nx
import community as louvain

# Data management
import pandas as pd
from tabulate import tabulate

# Data viz
from IPython.display import Image
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

2 Choosing and loading multiplex

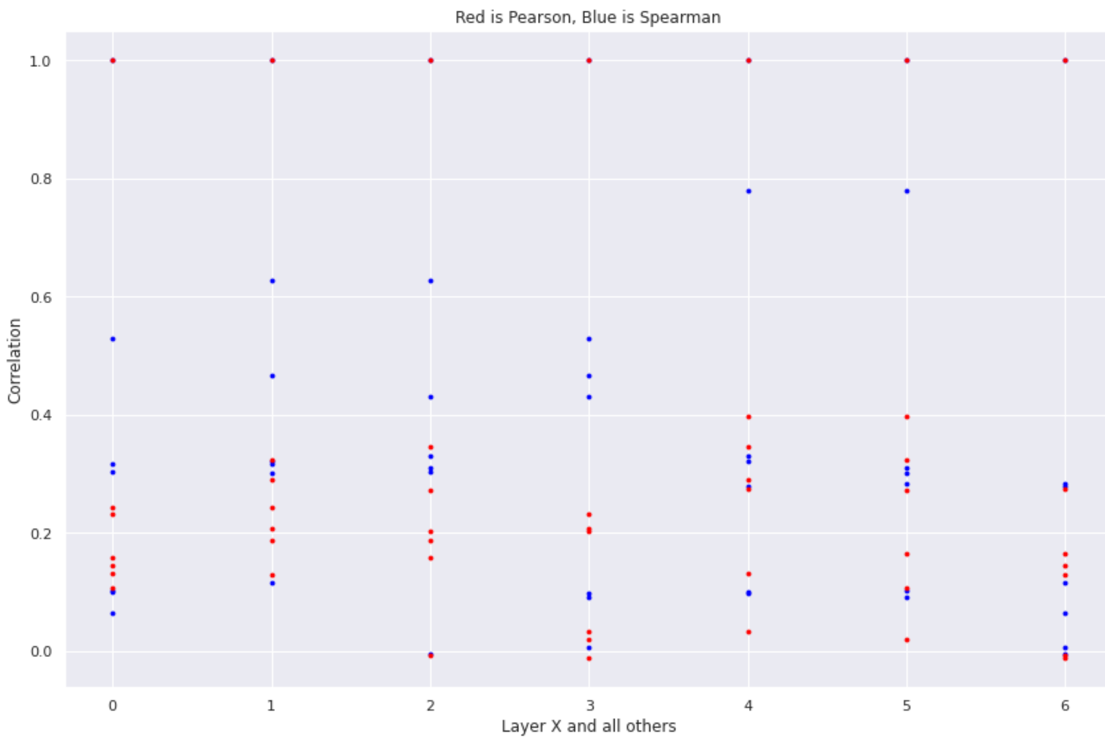
[ ]:
```
# Network config
networks = ["arxiv", "celegans", "drosophila", "london"]  # Common names for␣
↪multiplexes (alphabetized)
NUM_LAYERS_ = {
    "arxiv": 13,
    "celegans": 3,
    "drosophila": 7,
    "london": 3,
}

# * Choosing a network! Remmebr Python is 0 indexed.
```

```
IDX = 2
NUM_LAYERS = NUM_LAYERS_[networks[IDX]]

# Loading network
## Reading raw text edgelist
with open("data/multiplexes/multiplex_network-{}.edgelist".
↪format(networks[IDX])) as _f:
    lines = _f.readlines()

## Formatting as dictionary {layer: edgelist}
lines= [line.split(" ") for line in lines]
multiplex = {int(line[0]): [] for line in lines}
for line in lines:
    edge = (int(line[1]), int(line[2]))
    multiplex[int(line[0])].append(edge)


# Creating array of nx.Graph objects for each layer
S_nx = [
    nx.Graph(edgelist)
    for edgelist in multiplex.values()
]
```

2.1 Don't look at this, I am ashamed

[ ]:
```
from scipy.stats import spearmanr as sp
from scipy.stats import pearsonr as pr
plt.figure(figsize=(12,8))
for c in range(NUM_LAYERS):
    rhos = []
    rhos_p = []
    for d in range(NUM_LAYERS):
        deg1 = dict(S_nx[c].degree())
        deg2 = dict(S_nx[d].degree())
        for node in set(deg1.keys()) | set(deg2.keys()):
            if node not in deg1:
                deg1[node] = 0
            if node not in deg2:
                deg2[node] = 0

        a = list(deg1.values())
        b = list(deg2.values())
        if len(b) < len(a):
            b = b + [0]*(len(a)-len(b))
        else:
            a = a + [0]*(len(b)-len(a))
```

```
        rho, _ = sp(a, b)
        rho_p, _ = pr(a, b)
        rhos.append(rho)
        rhos_p.append(rho_p)
    plt.scatter([c]*len(rhos), rhos, label="{}".format(c), s=8, color="blue")
    plt.scatter([c]*len(rhos), rhos_p, label="{}".format(c), s=8, color="red")
#plt.legend()
plt.xlabel("Layer X and all others")
plt.ylabel("Correlation")
plt.title("Red is Pearson, Blue is Spearman")
plt.tight_layout()
```



2.2 Layerwise summary analysis

2.2.1 Basic stats

[ ]:
```
names=[str(x) for x in range(1, NUM_LAYERS+1)]
df_data=[]

for no,g in enumerate(S_nx):
    n,e=g.number_of_nodes(),g.number_of_edges()
    comps=len([len(c) for c in sorted(nx.connected_components(g), key=len,␣
 ↪reverse=True)])
```

```
    GCC=[len(c) for c in sorted(nx.connected_components(g), key=len,␣
↪reverse=True)]
    gcc=GCC[0]/n
    comm_het = powerlaw.Fit(GCC, verbose=False)
    comm_het = comm_het.power_law.alpha
    d=[j for _,j in g.degree()]
    d2=[j**2 for j in d]
    het=np.mean(d2)/(np.mean(d))**2
    df_data.append((names[no],n,e,comps,gcc,het,comm_het))

df=pd.DataFrame(df_data,columns=['Layer␣
↪Id','Nodes','Edges','Components','GCC','Degree Heterogeneity','Disconnected␣
↪Components power-law fit'])
print(tabulate(df, headers='keys', tablefmt='psql', showindex=False))
```

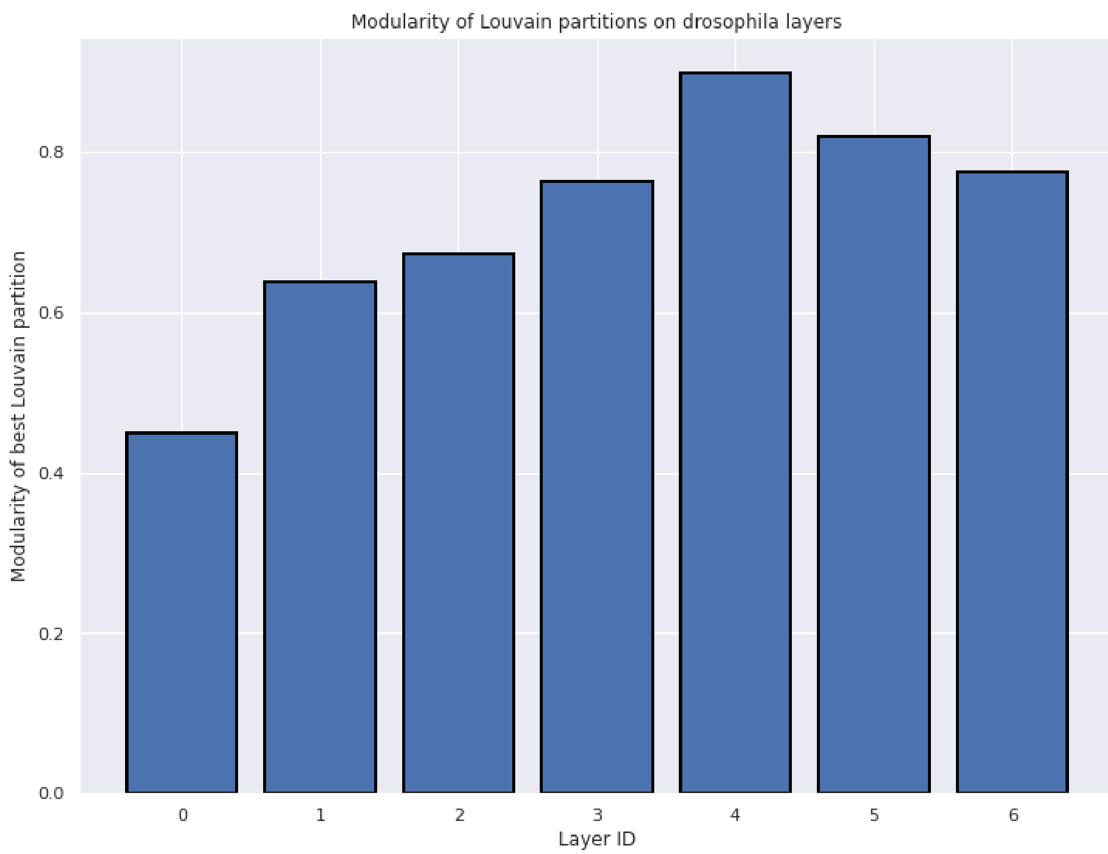
Layer Id	Nodes	Edges	Components	GCC	Degree Heterogeneity	Disconnected Components power-law fit
1	7356	23977	57	0.98491	3.55865	5.23996
2	839	1864	37	0.896305	3.71568	4.16608
3	755	1425	47	0.838411	3.44758	4.06924
4	2851	12818	157	0.857594	3.97	4.0385
5	85	71	29	0.164706	1.50069	4.85995
6	72	66	14	0.236111	2.26446	2.62527
7	12	7	5	0.25	1.10204	7.16576

2.2.2 Community structure

[ ]:
```
# Get partitions
partitions = [louvain.best_partition(layer) for layer in S_nx]

# Get modularity for each partition
graph_partition_pairs = zip(S_nx, partitions)
modularities = [
    louvain.modularity(partition, graph)
```

```python
    for graph, partition in graph_partition_pairs
]

# Quick distribution of modularity in each layer
## Plotting data
plt.figure(figsize=(12,9))
plt.bar(
    x=range(len(modularities)), height=modularities,
    edgecolor="black", linewidth=2
)

## Adjusting ticks and labels
plt.xticks(range(len(modularities)))
plt.xlabel("Layer ID")
plt.ylabel("Modularity of best Louvain partition")
plt.title("Modularity of Louvain partitions on {} layers".format(networks[IDX]))
```

[ ]: Text(0.5, 1.0, 'Modularity of Louvain partitions on drosophila layers')



3 Inter-layer analysis

3.1 Computations

[ ]:

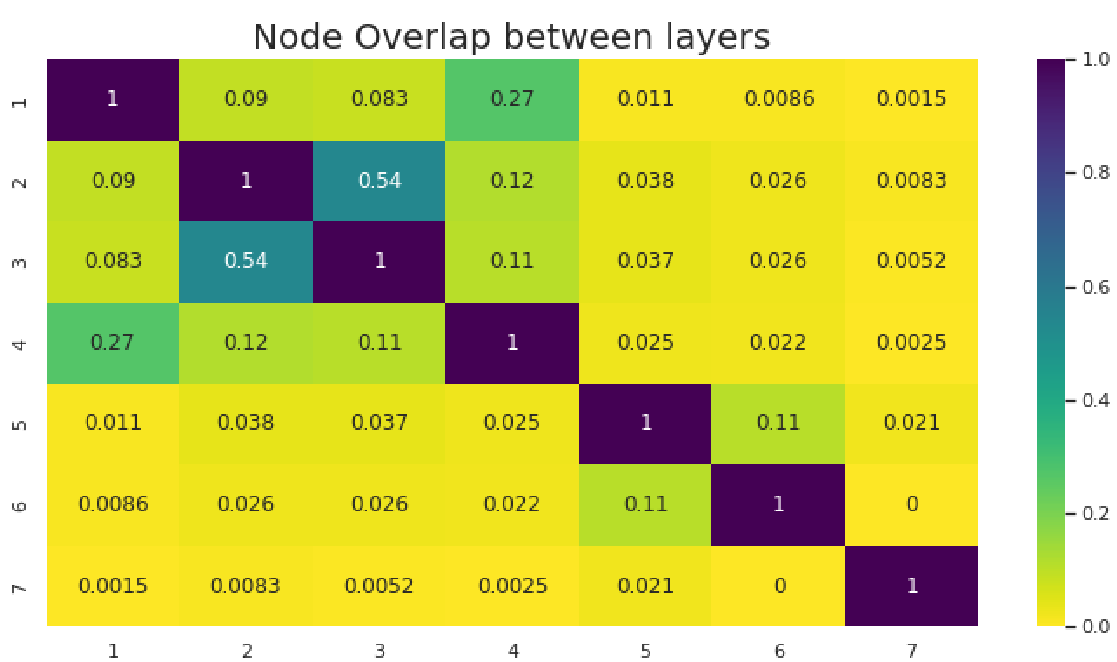
```
# Book-keeping
nodeoverlap_df_ = []  # Array tracking node overlap
edgeoverlap_df_ = []  # Array tracking edge overlap
ratios_df_ = []  # Array tracking ratio of average degrees between two layers
NMIs_df_ = []  # Array tracking NMI of partitions

# Looping over layer comparisons (left layer)
for no1,g1 in enumerate(S_nx):
    # Set up some intermediary data struct before dataframe
    temp_nodes = [0]*NUM_LAYERS  # One by one layer pairing node overlap
    temp_edges = [0]*NUM_LAYERS  # Same for edges
    temp_ratios = [0]*NUM_LAYERS  # Same for ratio of average degrees
    temp_nmis = [0]*NUM_LAYERS  # Same for partition NMIs

    # Looping over layer comparisons (right layer)
    for no2,g2 in enumerate(S_nx):
        # Get node overlap
        S1,S2=set(g1.nodes()),set(g2.nodes())
        J=len(S1.intersection(S2))/len(S1.union(S2))
        temp_nodes[no2]=J

        # Get edge overlap
        S1,S2=set(g1.edges()),set(g2.edges())
        J=len(S1.intersection(S2))/len(S1.union(S2))
        temp_edges[no2]=J

        # Get ratios of average degrees
        av1 = np.mean([x[1] for x in g1.degree()])
        av2 = np.mean([x[1] for x in g2.degree()])
        if av1 > av2:
            temp_ratios[no2] = av2/av1
        else:
            temp_ratios[no2] = av1/av2

        # Get partition NMIs
        ## Get relevant partitions
        partitions_ = (partitions[no1], partitions[no2])

        ## Restrict to common nodes between these partitions
        partition_maps_ = [
            [
                map_[common_node]
                for common_node in set(g1.nodes()) & set(g2.nodes())
            ]
```

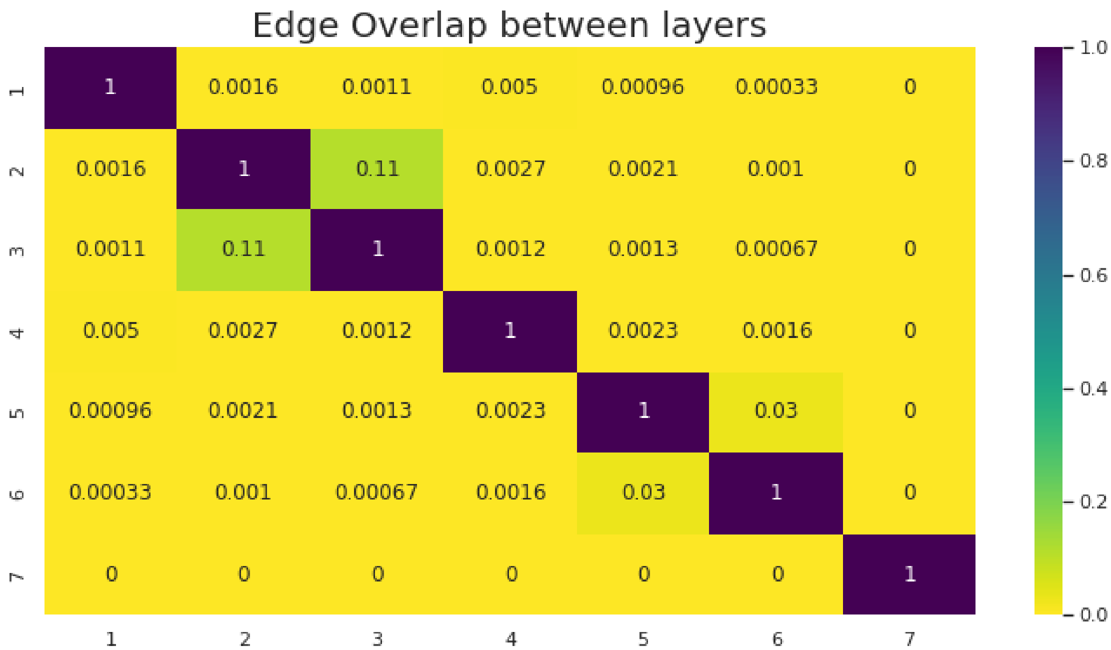
```
            for map_ in partitions_
        ]

        ## Plug into NMI calculation
        temp_nmis[no2] = nmiscore(*partition_maps_)


    # Cumulatively add to dataframe
    nodeoverlap_df_.append(temp_nodes)
    edgeoverlap_df_.append(temp_edges)
    ratios_df_.append(temp_ratios)
    NMIs_df_.append(temp_nmis)

nodeoverlap_df = pd.DataFrame(nodeoverlap_df_, columns=names,index=names)
edgeoverlap_df = pd.DataFrame(edgeoverlap_df_, columns=names,index=names)
ratios_df = pd.DataFrame(ratios_df_, columns=names,index=names)
NMIs_df = pd.DataFrame(NMIs_df_, columns=names,index=names)
```

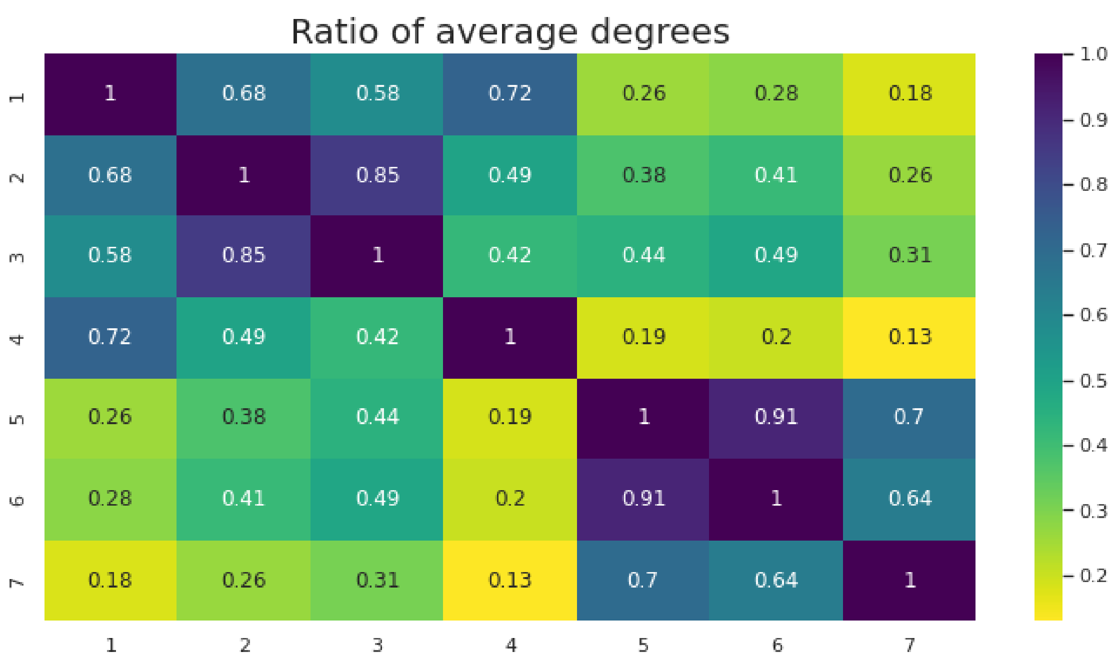
3.2 Plotting

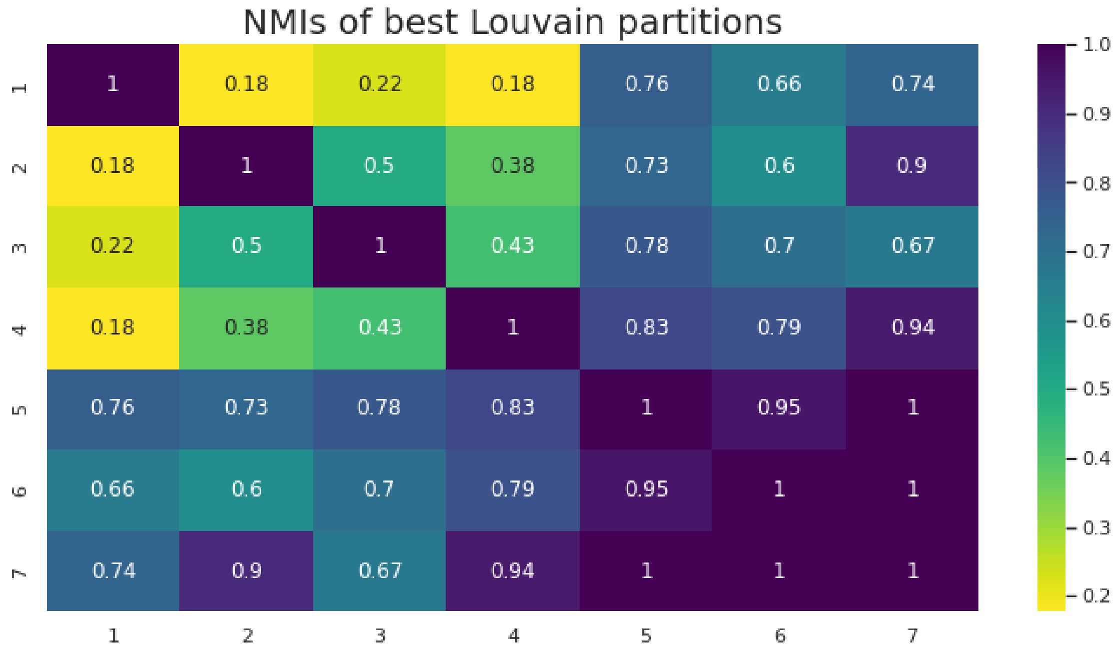
[ ]:

```python
# Node overlap
if 1:  # * If statement purely to toggle plot visibility
    plt.figure()
    sns.set(rc = {'figure.figsize':(12,6)})
    sns.heatmap(nodeoverlap_df, annot=True, cmap='viridis_r')
    plt.title('Node Overlap between layers',fontsize=20)

# Edge overlap
if 1:
    plt.figure()
    sns.set(rc = {'figure.figsize':(12,6)})
    sns.heatmap(edgeoverlap_df, annot=True, cmap='viridis_r')
    plt.title('Edge Overlap between layers',fontsize=20)

# Ratios of average degrees
if 1:
    plt.figure()
    sns.set(rc = {'figure.figsize':(12,6)})
    sns.heatmap(ratios_df, annot=True, cmap='viridis_r')
    plt.title('Ratio of average degrees',fontsize=20)

# NMIs of partitions
if 1:
    plt.figure()
    sns.set(rc = {'figure.figsize':(12,6)})
    sns.heatmap(NMIs_df, annot=True, cmap='viridis_r')
    plt.title('NMIs of best Louvain partitions',fontsize=20)
```



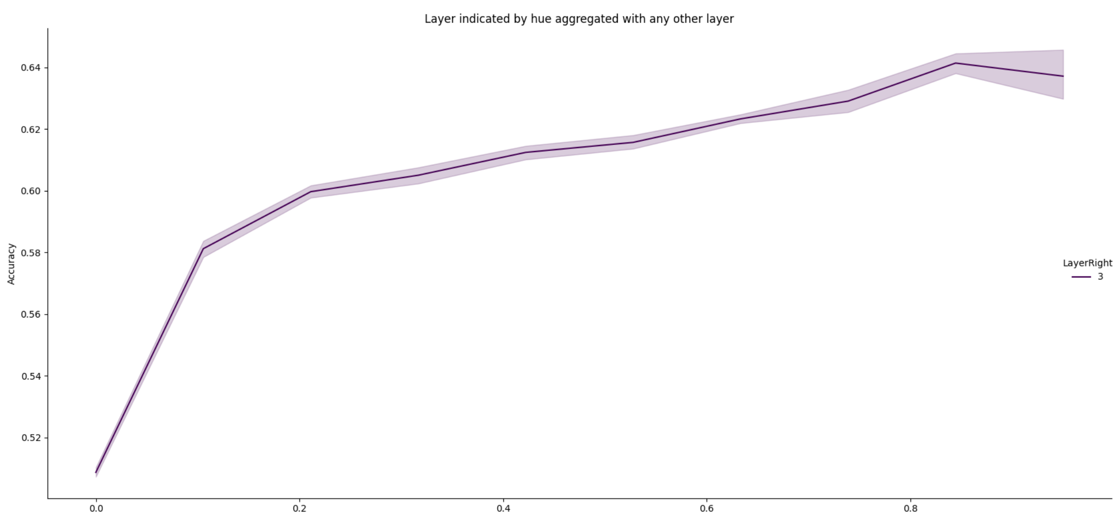

Edge Overlap between layers
1 0.0016 0.0011 0.005 0.00096 0.00033 0
0.0016 1 0.11 0.0027 0.0021 0.001 0
0.0011 0.11 1 0.0012 0.0013 0.00067 0
0.005 0.0027 0.0012 1 0.0023 0.0016 0
0.00096 0.0021 0.0013 0.0023 1 0.03 0
0.00033 0.001 0.00067 0.0016 0.03 1 0
0 0 0 0 0 0 1
1 2 3 4 5 6 7
1.0 0.8 0.6 0.4 0.2 0.0


Ratio of average degrees
1 0.68 0.58 0.72 0.26 0.28 0.18
0.68 1 0.85 0.49 0.38 0.41 0.26
0.58 0.85 1 0.42 0.44 0.49 0.31
0.72 0.49 0.42 1 0.19 0.2 0.13
0.26 0.38 0.44 0.19 1 0.91 0.7
0.28 0.41 0.49 0.2 0.91 1 0.64
0.18 0.26 0.31 0.13 0.7 0.64 1
1 2 3 4 5 6 7
1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2



[ ]:
```
# ! FOR SPECIFICALLY DROSOPHILA LAYERS 2 AND 3
Image(filename='results/drosophila_left-2_right-3.png')
```

[ ]:



3.3 Merging dataframes

[ ]:
```
# * Combining melted dataframes of all these measure
## Setting up convenient book-keeping for labels
vals = {
    0: "Node Overlap",
    1: "Edge Overlap",
    2: "Ratio",
    3: "NMI",
}

## Melting each dataframe into "long form"
melted_dfs = []
### ! ORDER IS IMPORTANT, MATCHES ABOVE LABELS DICT
for c, df_ in enumerate([nodeoverlap_df, edgeoverlap_df, ratios_df, NMIs_df]):
    ### Initial melt
    df_melted = df_.melt()

    ### Rename default column names (I can't find docs on header kwarg for␣
↪melt())
    df_melted = df_melted.rename(columns={"variable": "LayerRight", "value":␣
↪vals[c]})

    ### Grab index, which truly maps to one of the layer ids, and make it a␣
↪column for convenience
    df_melted["LayerLeft"] = df_melted.index

    ### Add to arracy of melted dfs
    melted_dfs.append(df_melted)

## Merge melted (long form) dataframes
tmp_ = melted_dfs[0]
for tmp in melted_dfs[0:]:
    tmp_ = pd.merge(tmp_, tmp, how="inner", on=["LayerLeft", "LayerRight"])

## Tidy up merged columns
### Remove extraneous column
### * As far as I can tell, this is a remnant of the first step of merging? Not␣
↪sure why it is here...
tmp_ = tmp_.drop(columns={"Node Overlap_y"})

### Rename the not-extraeous column to remove specifying identifier
df_melted_merged = tmp_.rename(columns={"Node Overlap_x": "Node Overlap"})

### Adjust LayerLeft from copied index values (initialized uniquely) to layer␣
↪ids
```

```
### * The "+ 1" in the lambda function application is because most datasets␣
↪from De Demonico's repository are indexed from 1
df_melted_merged["LayerLeft"] = df_melted_merged["LayerLeft"].apply(lambda x:␣
↪(x % NUM_LAYERS)+1)

### Quick rearrangement of columns for tabulation print to look nicer
### * Not necessary for calculations, just for aesthetics!
df_melted_merged = df_melted_merged[["LayerLeft", "LayerRight", "Node Overlap",␣
↪"Edge Overlap", "Ratio", "NMI"]]

## Pretty-print to admire my hardwork
print(tabulate(df_melted_merged, headers='keys', tablefmt='psql',␣
↪showindex=False))
```

```
+-------------+--------------+----------------+----------------+----------+-----
-----+
|   LayerLeft |   LayerRight |   Node Overlap |   Edge Overlap |    Ratio |
NMI |
|-------------+--------------+----------------+----------------+----------+-----
-----|
|           1 |            1 |     1          |    1           | 1        | 1
|
|           2 |            1 |     0.0899056  |    0.00155033  | 0.681602 |
0.177504 |
|           3 |            1 |     0.0834892  |    0.00114295  | 0.579048 |
0.223366 |
|           4 |            1 |     0.266377   |    0.00497091  | 0.724987 |
0.17797  |
|           5 |            1 |     0.0107308  |    0.000957336 | 0.256263 |
0.756788 |
|           6 |            1 |     0.00855397 |    0.000332848 | 0.281228 |
0.658087 |
|           7 |            1 |     0.00149517 |    0           | 0.178963 |
0.735069 |
|           1 |            2 |     0.0899056  |    0.00155033  | 0.681602 |
0.177504 |
|           2 |            2 |     1          |    1           | 1        | 1
|
|           3 |            2 |     0.537126   |    0.110024    | 0.84954  |
0.499657 |
|           4 |            2 |     0.116152   |    0.00266339  | 0.494152 |
0.381041 |
|           5 |            2 |     0.0382022  |    0.00207147  | 0.375972 |
0.728074 |
|           6 |            2 |     0.0259009  |    0.00103734  | 0.412598 |
0.596884 |
|           7 |            2 |     0.00829384 |    0           | 0.262563 |
```

					0.9035
1	3	0.0834892	0.00114295	0.579048	0.223366
2	3	0.537126	0.110024	0.84954	0.499657
3	3	1	1	1	1
4	3	0.108515	0.001195	0.419802	0.426504
5	3	0.037037	0.00133869	0.442559	0.77731
6	3	0.0260546	0.000671141	0.485673	0.703929
7	3	0.00524246	0	0.309064	0.666667
1	4	0.266377	0.00497091	0.724987	0.17797
2	4	0.116152	0.00266339	0.494152	0.381041
3	4	0.108515	0.001195	0.419802	0.426504
4	4	1	1	1	1
5	4	0.0251397	0.00225505	0.185787	0.825545
6	4	0.022028	0.00163259	0.203886	0.791478
7	4	0.00245098	0	0.129746	0.939945
1	5	0.0107308	0.000957336	0.256263	0.756788
2	5	0.0382022	0.00207147	0.375972	0.728074
3	5	0.037037	0.00133869	0.442559	0.77731
4	5	0.0251397	0.00225505	0.185787	0.825545
5	5	1	1	1	1
6	5	0.105634	0.0300752	0.91123	0.952559
7	5	0.0210526	0	0.698357	1
1	6	0.00855397	0.000332848	0.281228	0.658087
2	6	0.0259009	0.00103734	0.412598	0.596884
3	6	0.0260546	0.000671141	0.485673	

					0.703929
4	6	0.022028	0.00163259	0.203886	0.791478
5	6	0.105634	0.0300752	0.91123	0.952559
6	6	1	1	1	1
7	6	0	0	0.636364	1
1	7	0.00149517	0	0.178963	0.735069
2	7	0.00829384	0	0.262563	0.9035
3	7	0.00524246	0	0.309064	0.666667
4	7	0.00245098	0	0.129746	0.939945
5	7	0.0210526	0	0.698357	1
6	7	0	0	0.636364	1
7	7	1	1	1	1