report_four-real-multiplexes

April 18, 2022

0.1 # Preface

0.2 Document purpose

This notebook is designed to be a central location for generation of figures utilized in the current working manuscript. Data is presumed to be current and no version control matching checks are made in this document.

0.3 Document Metadata

- Created: 2022-04-18 11:04:00 Eastern
- Last modified: 2022-04-18 11:10:00 Eastern
- Known issues:
  - My life is a walking issue

0.4 Setup

This notebook will be located in an isolated subdirectory of a relevant project directory. Make sure any dataframes are accessible in that directory *and your Google Drive is mounted when working on this notebook!*

0.5 Formatting content

Please stick to the predescribed section content to keep the document organized (I will shame you mercilessly if you do not.) Roughly, the format is as follows:

- Figure identifier ("# Figure [NUMBER]" in a markdown/text notebook cell.)
  - Description of figure ("## Figure description" in a markdown/text notebook cell.)
    - Markdown/text cell describing the figure at a high-level intended for authors, not for publication. Separate cell description allows for reorganizing figure in manuscript and being able to easily change the corresponding labels here.
  - Figure data handling ("## Data handling" in a markdown/text notebook cell.)
    - Code cells to load, format, and preprocess data relevant to *that* particular figure. Use naming conventions for your dataframes that signify what the data is and disallow accidental overlap (e.g. `df = pd.read_csv("data_1.csv")` will be shamed and changed.)
  - Figure plotting ("## Plotting" in a markdown/text notebook cell.)
    - Code cells to take processed data and create manuscript-quality figures. Include high-level journal-specific configurations in a separate configuration cell, e.g. font size or image dimensions (this allows "fine-tuning" and switching to/from double/single column format, for example, to be altering a minimal amount of cells).

- Figure caption ("## Caption" in a markdown/text notebook cell.)
  - Markdown/text cells to prototype a figure's caption. This is less of a necessity, as version control on large blocks of text in Overleaf is simpler than here.

There are section headings here to make navigation in notebook tools easier - Jupyter lab, VSCode, Google Collab, etc. all allow you to compress markdown headings and jump between them very quickly with a sidebar. ***Do not neglect them, even if they are tedious!***

On this note, do **not** use any of the Google Collab section organization tools - they are external to embedded notebook structure and may not be present when reading thi notebook in other tools such as VSCode or Jupyter Lab!

0.6 Imports and Universal Properties

[ ]:
```
# Scientific calculation
import numpy as np

# Dataframes
import pandas as pd

# Visualization
## Core packages
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

## MPL extras for fine-tuning
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
from cycler import cycler
```

[ ]:
```
# Plot configs
## Pyplot RC Params
rc_dict = {
  "savefig.dpi": 600,  # Saved figure dots-per-inch. 600 is "HD"
  "savefig.facecolor": "white",  # This, combined with transparent setting,␣
 ↪keeps saved figs from looking like trash on dark backgrounds
  "savefig.transparent": False,
  "figure.figsize": (10, 8),  # Default (width, height) of figure
  "axes.titlesize" : 28,  # Title size for each (sub)plot
  "axes.labelsize" : 25,  # X and Y axis label sizes (for the label, not the␣
 ↪ticks!)
  "lines.linewidth" : 1,  # Plotted line widths (1 is default)
  "lines.markersize" : 7,  # Size of markers on plotted data
  "xtick.labelsize" : 18,  # Size of annotations of X ticks
  "ytick.labelsize" : 18,  # Size of annotations of Y ticks
```

```
  "font.size": 18,  # Size of text elements
  "legend.labelspacing": 0.1,  # Vertical spacing of legend entries
  "legend.fontsize": 14,  # Text size of legend entries (smaller so legend␣
 ↪boxes don't overlap plot content)
}
plt.rcParams.update(rc_dict)

## Colormaps and markers
markers = ['*','o','s','x','v'] # No linestyle included
viridis = cm.get_cmap('viridis', 5)
colors = viridis(np.linspace(0, 1, 5))

# Network list
networks = ["arxiv", "celegans", "drosophila", "london"]
```

0.7 # arXiv

0.8 Data handling

[ ]:
```
df_accs = pd.read_csv("data/accuracy_networks_concat.csv")

# Select data by parameter value
network = "arxiv"
left = [8,9]  # None if want to consider all lefts
right = [8,9,10,11,12,13]  # None if want to consider all rights
# Query based on input values
## Restrict to multiplex
df_perf = df_accs.query("Multiplex == @network")
## If left specified, restrict
if type(left) is list:
    df_perf = df_perf.query("LayerLeft.isin(@left)")
elif type(left) is int:
    df_perf =df_perf.query("LayerLeft == @left")

## If right specified, restrict
if type(right) is list:
    df_perf = df_perf.query("LayerRight.isin(@right)")
elif type(right) is int:
    df_perf =df_perf.query("LayerRight == @right")

# ---
df_basicstats = pd.read_csv("results/dataframes/basic_stats.csv")
# Select data by parameter value
layers = [8,9,10,11,12,13]  # None if want to consider all layers
# Query based on input values
## Restrict to multiplex
df_bs = df_basicstats.query("Network == @network")
```

```
## If layers specified, restrict
if type(layers) is list:
    df_bs = df_bs.query("Layer.isin(@layers)")
elif type(layers) is int:
    df_bs = df_bs.query("Layer == @layers")


# ---
df_structuralcomparison = pd.read_csv("results/dataframes/
 ↪layer_structure_comparison.csv")
# Query based on input values
## Restrict to multiplex
df_sc = df_structuralcomparison.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_sc = df_sc.query("LayerLeft.isin(@layers) & LayerRight.isin(@layers)")
elif type(layers) is int:
    df_sc = df_sc.query("LayerLeft == @layers & LayerRight == @layers")
```

0.9 Plotting

[ ]:
```
# Summary stats for each layer
df_basicstats.query("Network == @network")
```

[ ]:

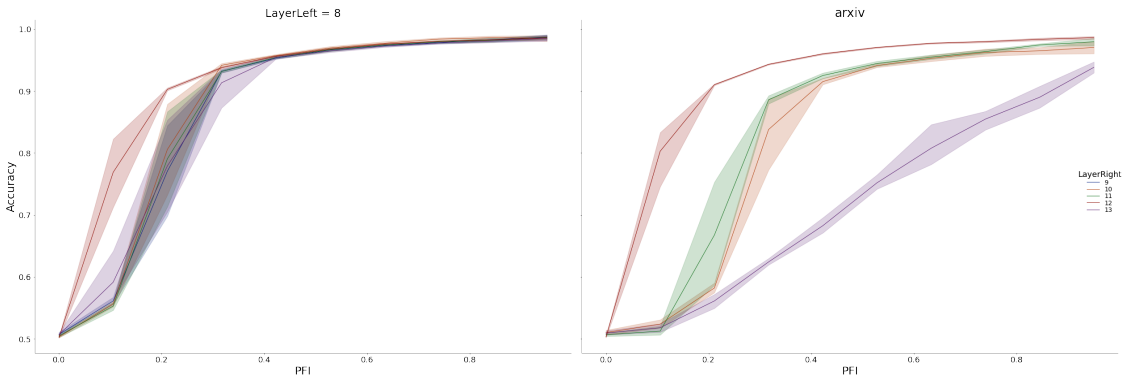
	Layer	Nodes	Edges	Components	GCC	Degree Heterogeneity	\
0	1	1594	3019	310	0.080928	1.518385	
1	2	5465	14485	508	0.671363	2.505357	
2	3	2956	6097	348	0.519959	1.919156	
3	4	361	592	83	0.036011	1.232985	
4	5	1605	4427	218	0.378816	3.306746	
5	6	3506	7341	394	0.604963	2.128181	
6	7	1451	2582	259	0.231564	1.822912	
7	8	1905	4423	264	0.340157	1.825839	
8	9	660	868	177	0.030303	1.428319	
9	10	700	1145	179	0.101429	1.550543	
10	11	1270	1953	232	0.311811	1.557613	
11	12	4946	11600	588	0.562475	2.092328	
12	13	377	494	100	0.053050	1.556440	

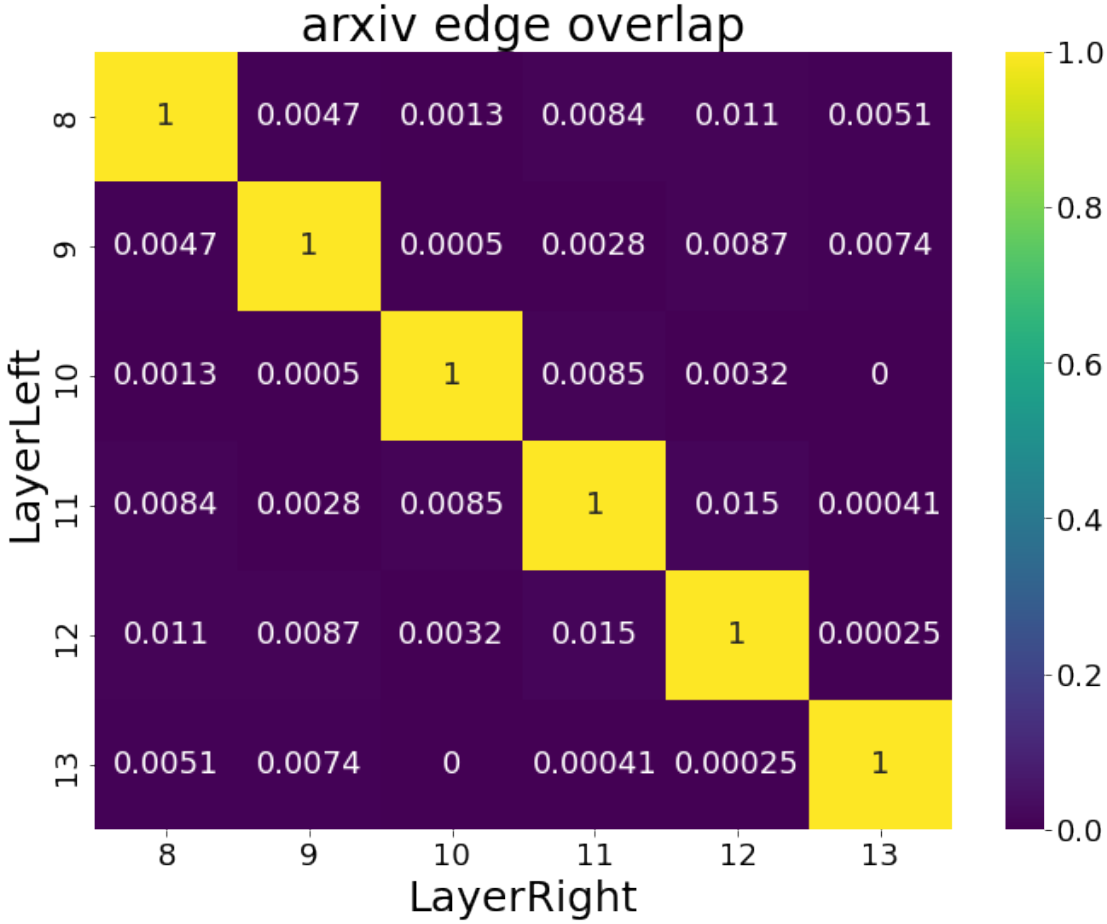
	Disconnected Components	power-law fit	Network	Modularities
0		3.187989	arxiv	0.982390
1		3.964531	arxiv	0.861008
2		3.983451	arxiv	0.926490
3		10.122177	arxiv	0.979233
4		3.195512	arxiv	0.894384
5		3.651826	arxiv	0.910554

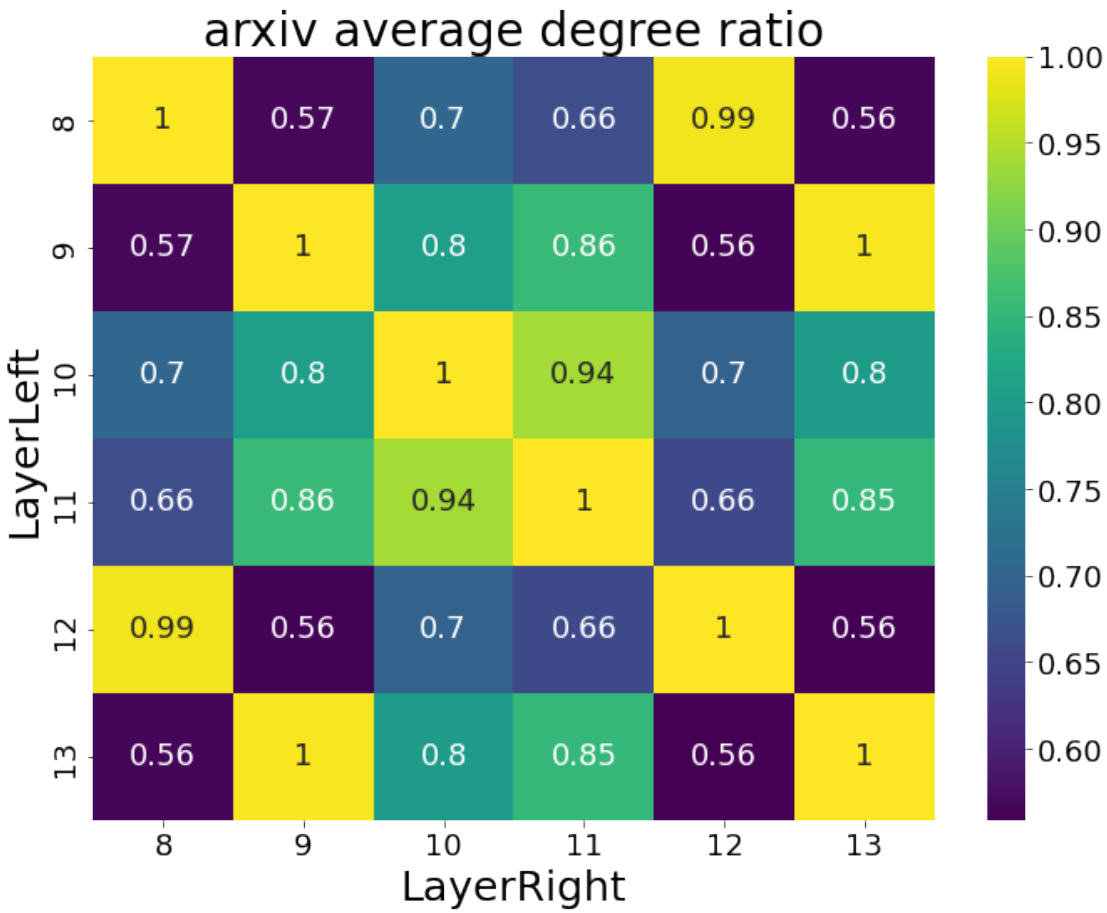
```
6                                   3.088868    arxiv       0.960407
7                                   3.158711    arxiv       0.968238
8                                   4.115962    arxiv       0.983688
9                                   5.261081    arxiv       0.968557
10                                  3.462025    arxiv       0.958312
11                                  3.418866    arxiv       0.908892
12                                  4.822711    arxiv       0.965407
```

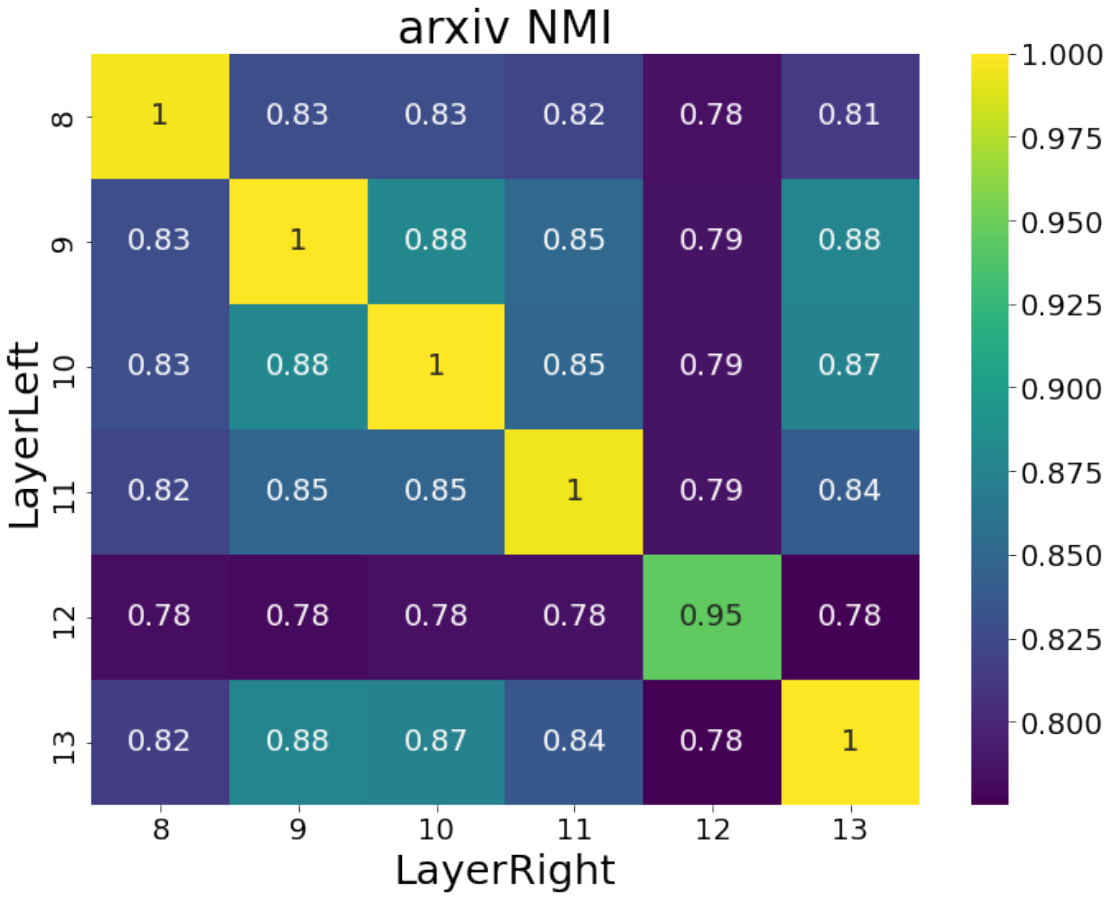
[ ]:
```
plt.figure()
sns.relplot(data=df_perf, kind="line",
            x="PFI", y="Accuracy",
            col="LayerLeft",
            hue="LayerRight",
            palette="dark",
            height=12, aspect=1.4
    )
plt.title(network)
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Edge␣
 ↪Overlap")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} edge overlap")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Ratio")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} average degree ratio")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="NMI")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} NMI")
plt.tight_layout()
```

```
<Figure size 720x576 with 0 Axes>
```


LayerLeft = 8
arxiv
Accuracy
PFI
PFI
LayerRight


arxiv edge overlap
LayerLeft
LayerRight
8
9
10
11
12
13
1
0.0047
0.0013
0.0084
0.011
0.0051
0.0047
1
0.0005
0.0028
0.0087
0.0074
0.0013
0.0005
1
0.0085
0.0032
0
0.0084
0.0028
0.0085
1
0.015
0.00041
0.011
0.0087
0.0032
0.015
1
0.00025
0.0051
0.0074
0
0.00041
0.00025
1
1.0
0.8
0.6
0.4
0.2
0.0


arxiv average degree ratio
LayerLeft
LayerRight
8
9
10
11
12
13
1
0.57
0.7
0.66
0.99
0.56
0.57
1
0.8
0.86
0.56
1
0.7
0.8
1
0.94
0.7
0.8
0.66
0.86
0.94
1
0.66
0.85
0.99
0.56
0.7
0.66
1
0.56
0.56
1
0.8
0.85
0.56
1
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60



0.10 Observations

Layers 9 and 12 are easily reconstructed and have (relative to layer 9 with 10-13) low NMI and also low ratio of average degrees. Layers 9 and 13 are difficult to reconstruct and have (relatrively) high NMI and also ratio of degrees = 1. What about other dataframe things? Looking at the dataframe basic stats suggests that layer 12 is quite qualitatively different.

0.11 # Drosophila

0.12 Data handling

[ ]:
```
df_accs = pd.read_csv("data/accuracy_networks_concat.csv")

# Select data by parameter value
network = "drosophila"
left = [1,2]  # None if want to consider all lefts
right = [2,3,4]  # None if want to consider all rights
# Query based on input values
## Restrict to multiplex
```

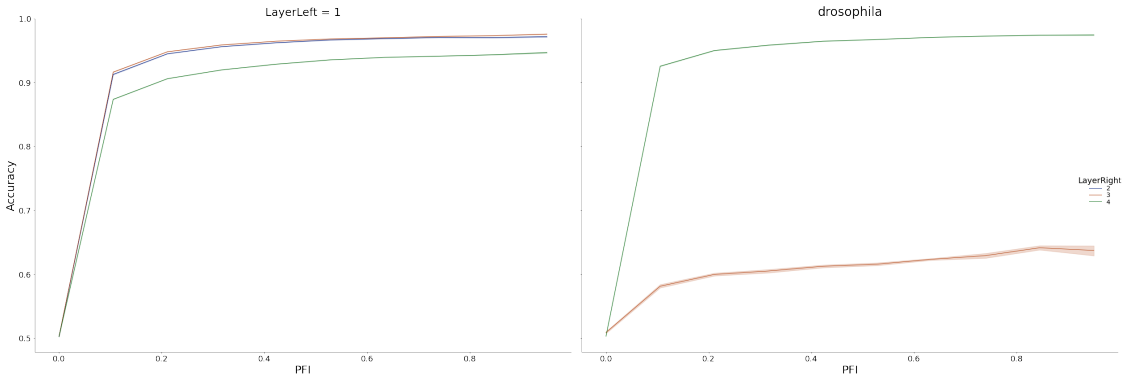
```
df_perf = df_accs.query("Multiplex == @network")
## If left specified, restrict
if type(left) is list:
    df_perf = df_perf.query("LayerLeft.isin(@left)")
elif type(left) is int:
    df_perf =df_perf.query("LayerLeft == @left")

## If right specified, restrict
if type(right) is list:
    df_perf = df_perf.query("LayerRight.isin(@right)")
elif type(right) is int:
    df_perf =df_perf.query("LayerRight == @right")

# ---
df_basicstats = pd.read_csv("results/dataframes/basic_stats.csv")
# Select data by parameter value
layers = [1,2,3,4]  # None if want to consider all layers
# Query based on input values
## Restrict to multiplex
df_bs = df_basicstats.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_bs = df_bs.query("Layer.isin(@layers)")
elif type(layers) is int:
    df_bs = df_bs.query("Layer == @layers")


# ---
df_structuralcomparison = pd.read_csv("results/dataframes/
 ↪layer_structure_comparison.csv")
# Query based on input values
## Restrict to multiplex
df_sc = df_structuralcomparison.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_sc = df_sc.query("LayerLeft.isin(@layers) & LayerRight.isin(@layers)")
elif type(layers) is int:
    df_sc = df_sc.query("LayerLeft == @layers & LayerRight == @layers")
```

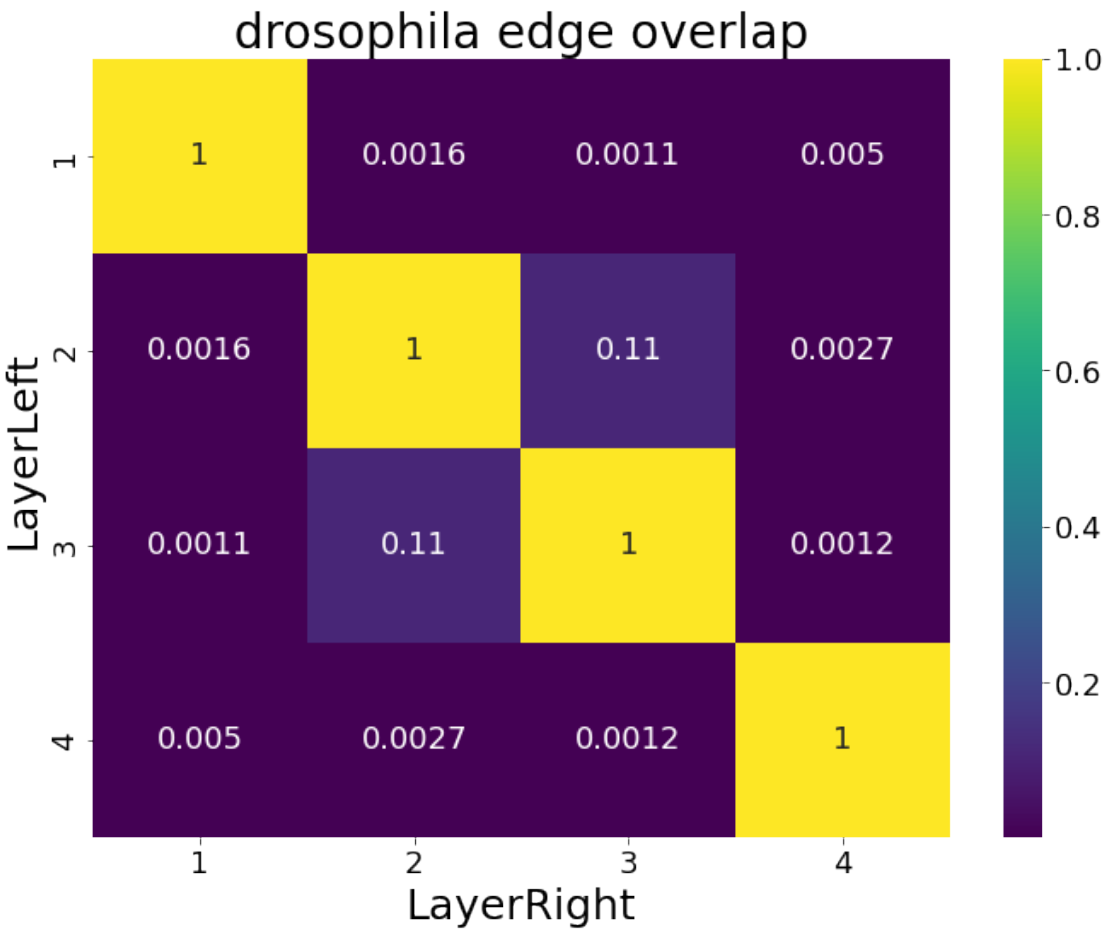
0.13 Plotting

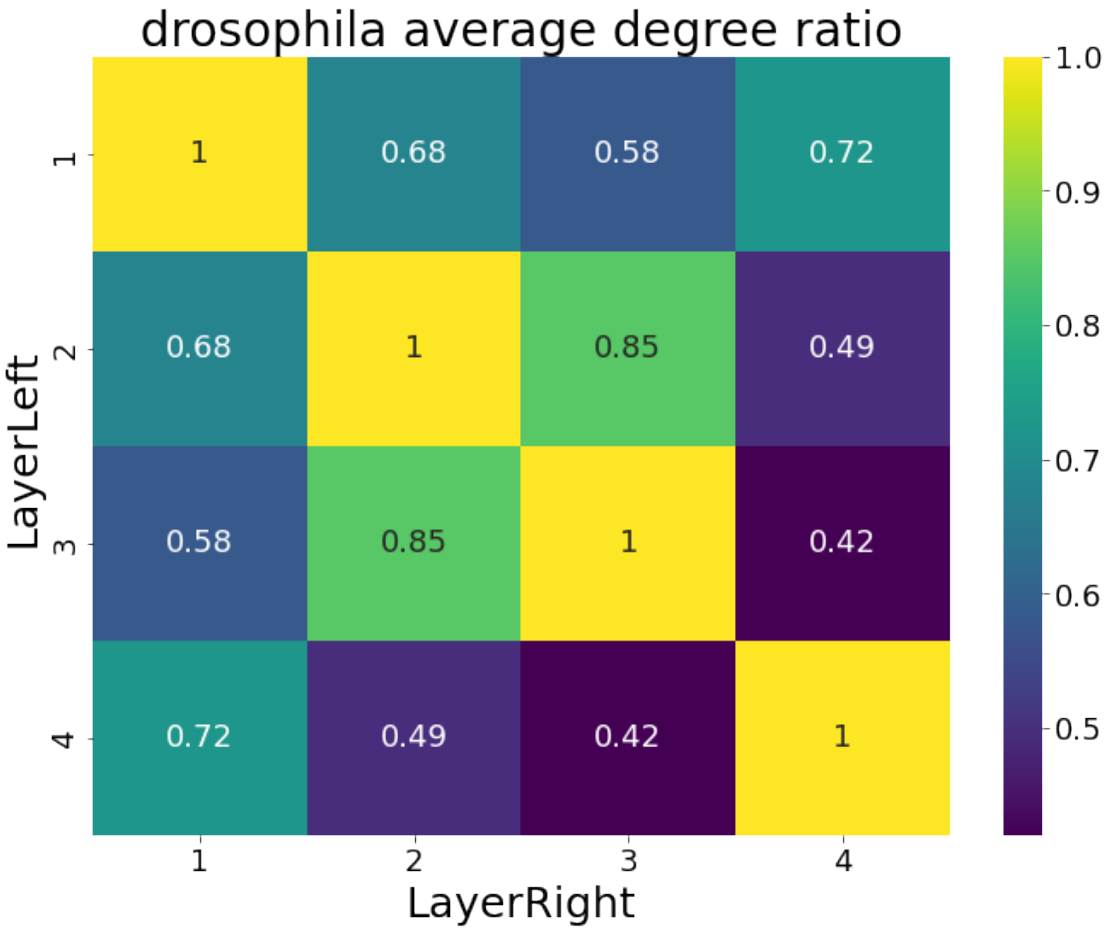
[ ]:
```
plt.figure()
sns.relplot(data=df_perf, kind="line",
            x="PFI", y="Accuracy",
            col="LayerLeft",
            hue="LayerRight",
            palette="dark",
```

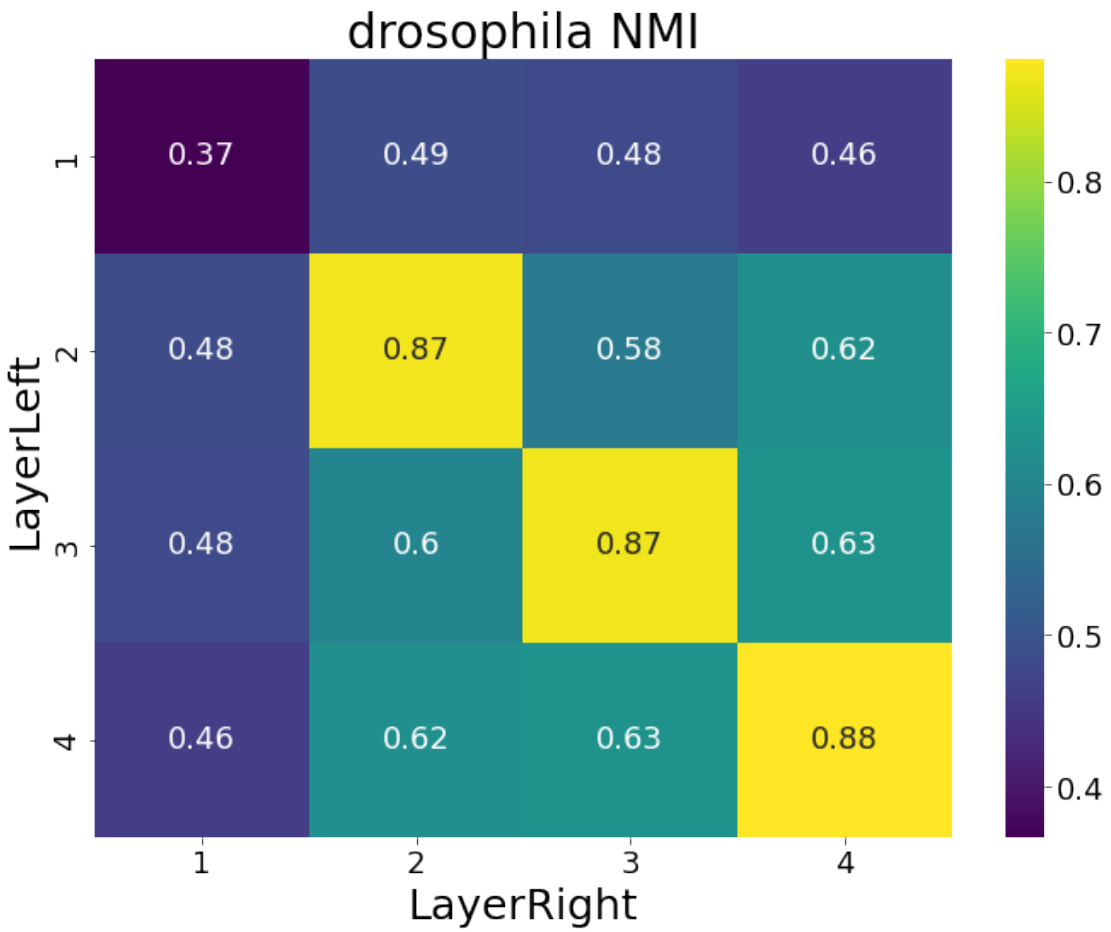
```
            height=12, aspect=1.4
    )
plt.title(network)
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Edge␣
↪Overlap")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} edge overlap")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Ratio")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} average degree ratio")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="NMI")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} NMI")
plt.tight_layout()
```

```
<Figure size 720x576 with 0 Axes>
```




drosophila edge overlap
1
0.0016
0.0011
0.005
0.0016
1
0.11
0.0027
0.0011
0.11
1
0.0012
0.005
0.0027
0.0012
1
LayerLeft
LayerRight
1
2
3
4
1.0
0.8
0.6
0.4
0.2


drosophila average degree ratio
1
0.68
0.58
0.72
0.68
1
0.85
0.49
0.58
0.85
1
0.42
0.72
0.49
0.42
1
LayerLeft
LayerRight
1
2
3
4
1.0
0.9
0.8
0.7
0.6
0.5



0.14 Observation

Layers 1 and 4 of 'drosophila' appear to have low NMI (low community correlation), a large number of overlapping nodes, and reconstruct well (quickly and high accuracy.)

Layers 2 and 3 have high NMI (0.97!) and low performance, a good signal we expect, however, layers 2 and 4 also have high NMI and good performance. This is likely because there is significantly lower node overlap with layers 2 and 4 than with 2 and 3 (0.12 versus 0.54) and correspondingly low edge overlap as well (0.0027 versus 0.11)

0.15 # Celegans

0.16 Data handling

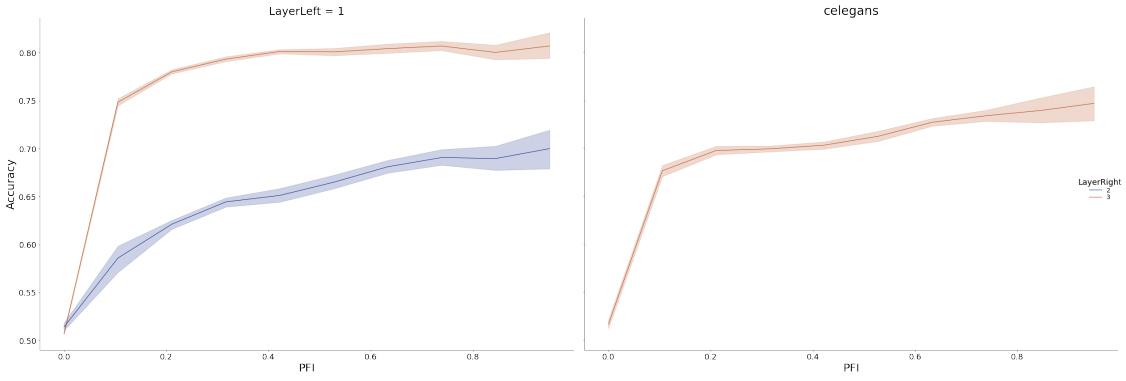
[ ]:
```
df_accs = pd.read_csv("data/accuracy_networks_concat.csv")

# Select data by parameter value
network = "celegans"
left = [1,2]  # None if want to consider all lefts
```

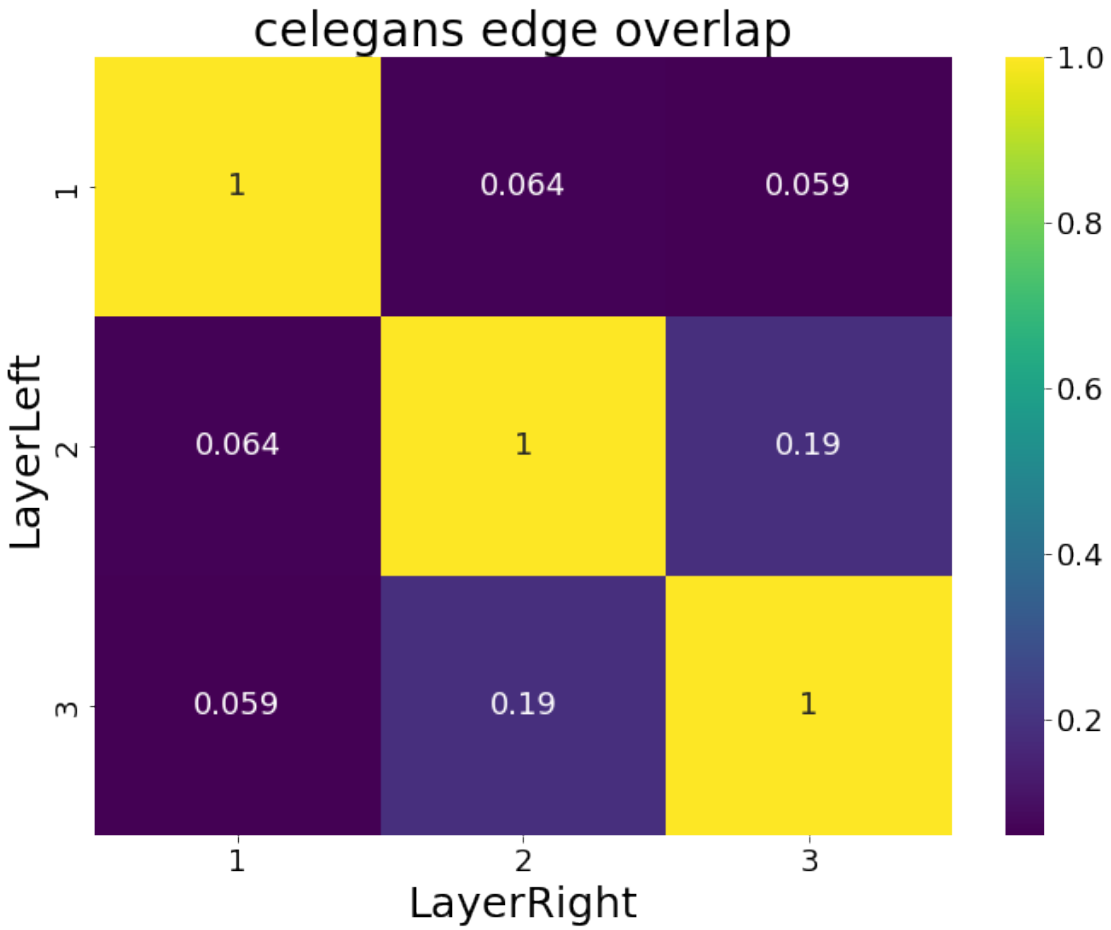
```
right = [2,3]  # None if want to consider all rights
# Query based on input values
## Restrict to multiplex
df_perf = df_accs.query("Multiplex == @network")
## If left specified, restrict
if type(left) is list:
    df_perf = df_perf.query("LayerLeft.isin(@left)")
elif type(left) is int:
    df_perf =df_perf.query("LayerLeft == @left")

## If right specified, restrict
if type(right) is list:
    df_perf = df_perf.query("LayerRight.isin(@right)")
elif type(right) is int:
    df_perf =df_perf.query("LayerRight == @right")

# ---
df_basicstats = pd.read_csv("results/dataframes/basic_stats.csv")
# Select data by parameter value
layers = [1,2,3]  # None if want to consider all layers
# Query based on input values
## Restrict to multiplex
df_bs = df_basicstats.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_bs = df_bs.query("Layer.isin(@layers)")
elif type(layers) is int:
    df_bs = df_bs.query("Layer == @layers")


# ---
df_structuralcomparison = pd.read_csv("results/dataframes/
 ↪layer_structure_comparison.csv")
# Query based on input values
## Restrict to multiplex
df_sc = df_structuralcomparison.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_sc = df_sc.query("LayerLeft.isin(@layers) & LayerRight.isin(@layers)")
elif type(layers) is int:
    df_sc = df_sc.query("LayerLeft == @layers & LayerRight == @layers")
```

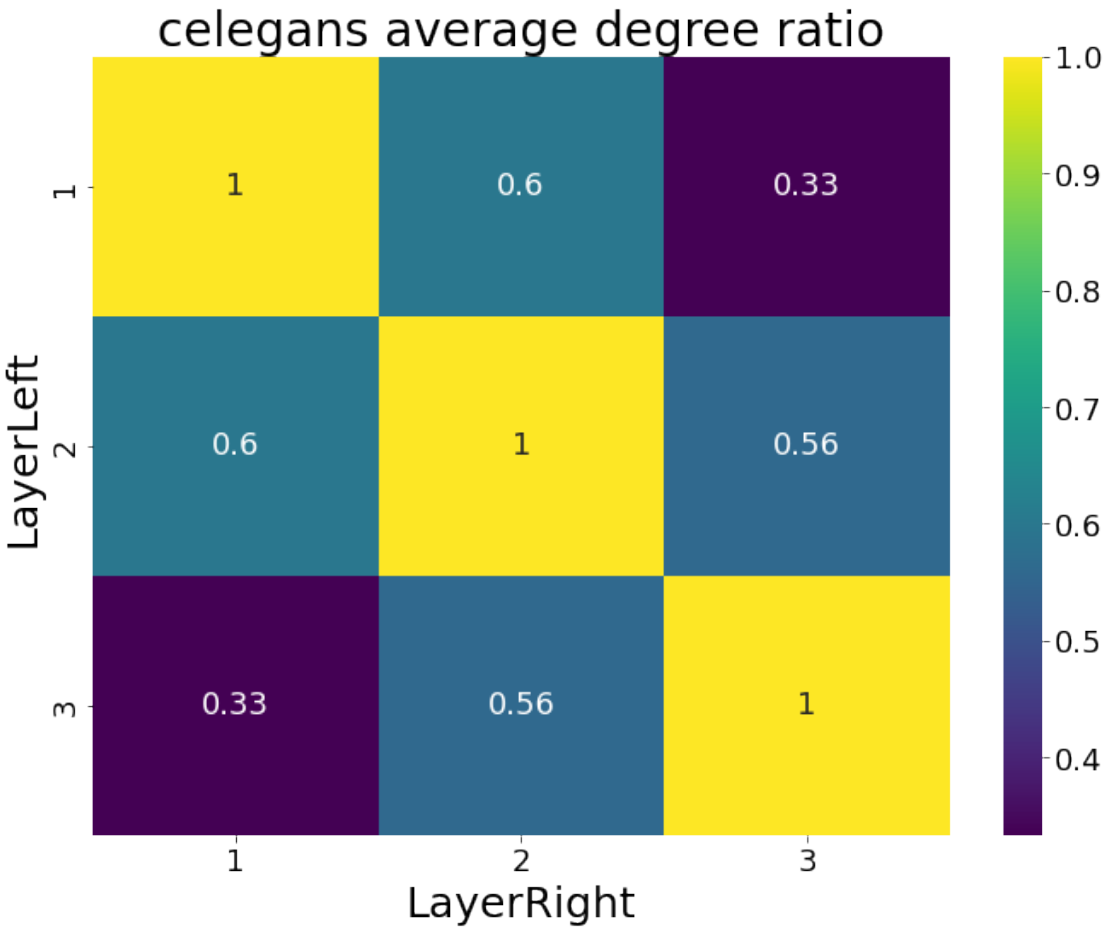
0.17 Plotting

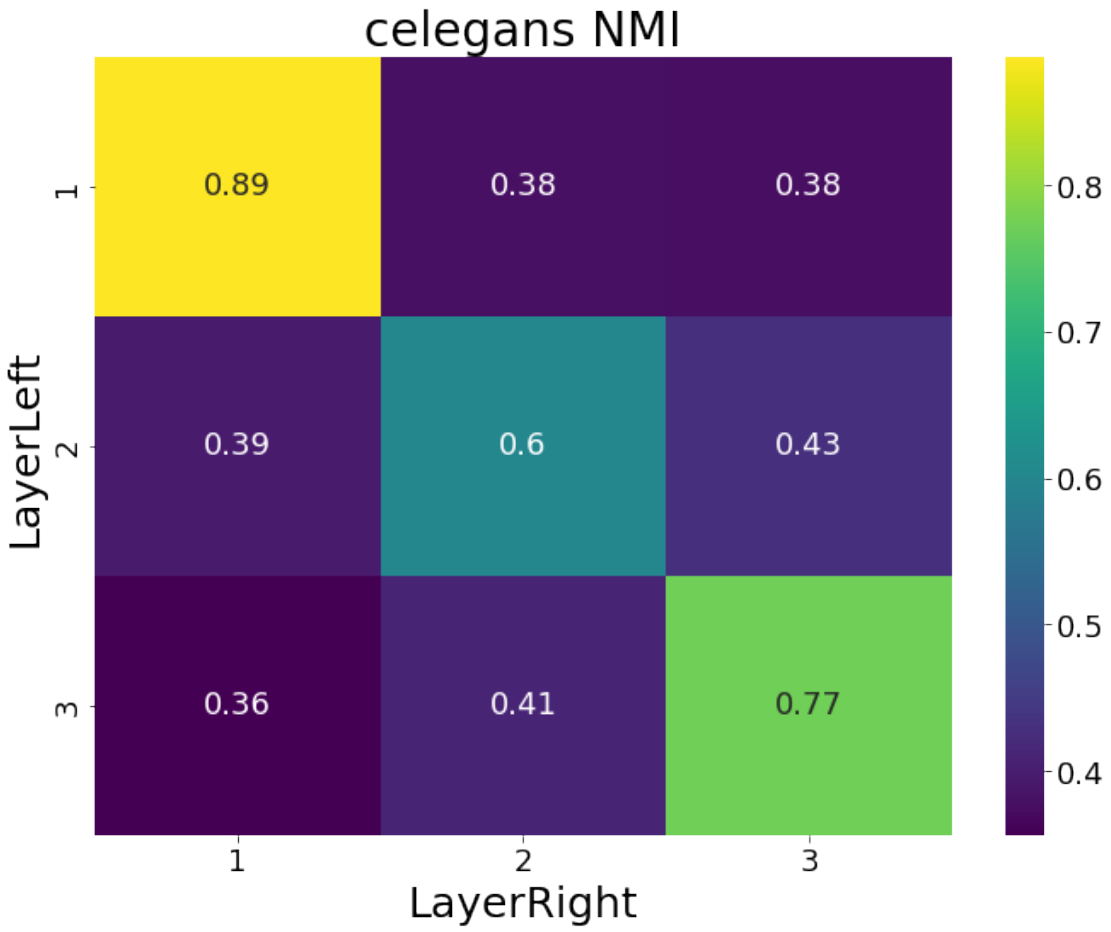
[ ]:
```
plt.figure()
sns.relplot(data=df_perf, kind="line",
            x="PFI", y="Accuracy",
            col="LayerLeft",
            hue="LayerRight",
            palette="dark",
            height=12, aspect=1.4
    )
plt.title(network)
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Edge␣
↪Overlap")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} edge overlap")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Ratio")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} average degree ratio")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="NMI")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} NMI")
plt.tight_layout()
```

```
<Figure size 720x576 with 0 Axes>
```




celegans edge overlap
1
0.064
0.059
0.064
1
0.19
0.059
0.19
1
LayerLeft
LayerRight
1
2
3
1.0
0.8
0.6
0.4
0.2


celegans average degree ratio
LayerLeft
LayerRight
1
2
3
1
0.6
0.33
0.6
1
0.56
0.33
0.56
1
1.0
0.9
0.8
0.7
0.6
0.5
0.4



0.18 Observation

This appears to break the desired pattern indicated by community correlations. It has been intimiately explored in Wu et al., however, and they find pretty decent performance. There may be (probably are?) signals Wu et al. can detect that we cannot. I guess its multiplex entropy - we don't get as sophisticated a picture, though the core of ratios and overlap are there?

0.19 # London

0.20 Data handling

[ ]:
```
df_accs = pd.read_csv("data/accuracy_networks_concat.csv")

# Select data by parameter value
network = "london"
left = [1,2]  # None if want to consider all lefts
right = [2,3,4]  # None if want to consider all rights
# Query based on input values
```

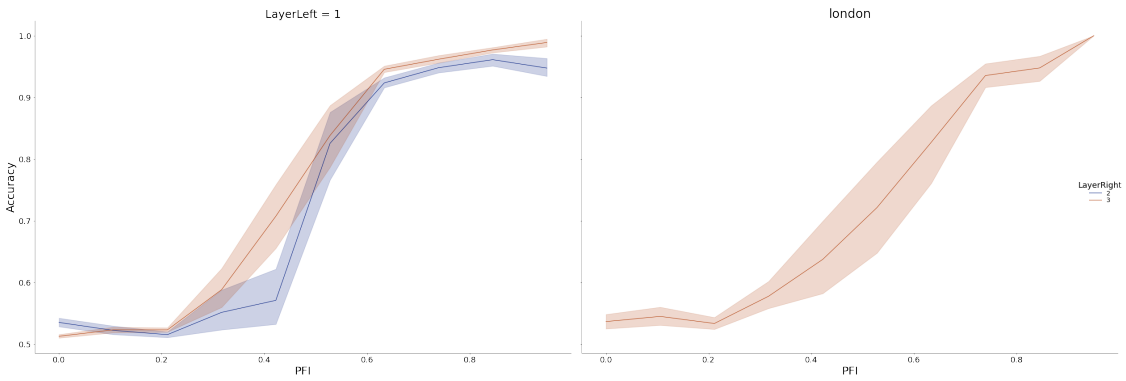
```
## Restrict to multiplex
df_perf = df_accs.query("Multiplex == @network")
## If left specified, restrict
if type(left) is list:
    df_perf = df_perf.query("LayerLeft.isin(@left)")
elif type(left) is int:
    df_perf =df_perf.query("LayerLeft == @left")

## If right specified, restrict
if type(right) is list:
    df_perf = df_perf.query("LayerRight.isin(@right)")
elif type(right) is int:
    df_perf =df_perf.query("LayerRight == @right")

# ---
df_basicstats = pd.read_csv("results/dataframes/basic_stats.csv")
# Select data by parameter value
layers = [1,2,3,4]  # None if want to consider all layers
# Query based on input values
## Restrict to multiplex
df_bs = df_basicstats.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_bs = df_bs.query("Layer.isin(@layers)")
elif type(layers) is int:
    df_bs = df_bs.query("Layer == @layers")


# ---
df_structuralcomparison = pd.read_csv("results/dataframes/
 ↪layer_structure_comparison.csv")
# Query based on input values
## Restrict to multiplex
df_sc = df_structuralcomparison.query("Network == @network")
## If layers specified, restrict
if type(layers) is list:
    df_sc = df_sc.query("LayerLeft.isin(@layers) & LayerRight.isin(@layers)")
elif type(layers) is int:
    df_sc = df_sc.query("LayerLeft == @layers & LayerRight == @layers")
```

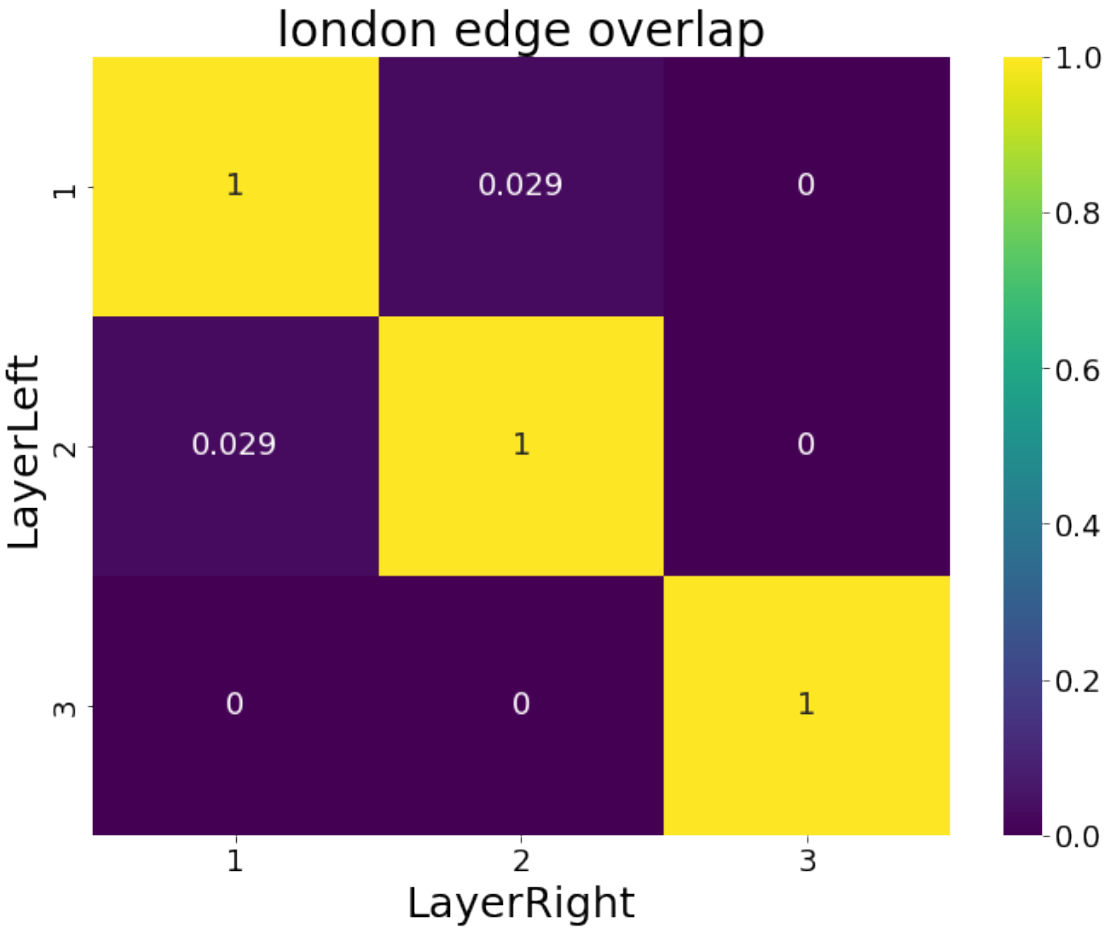
0.21 Plotting

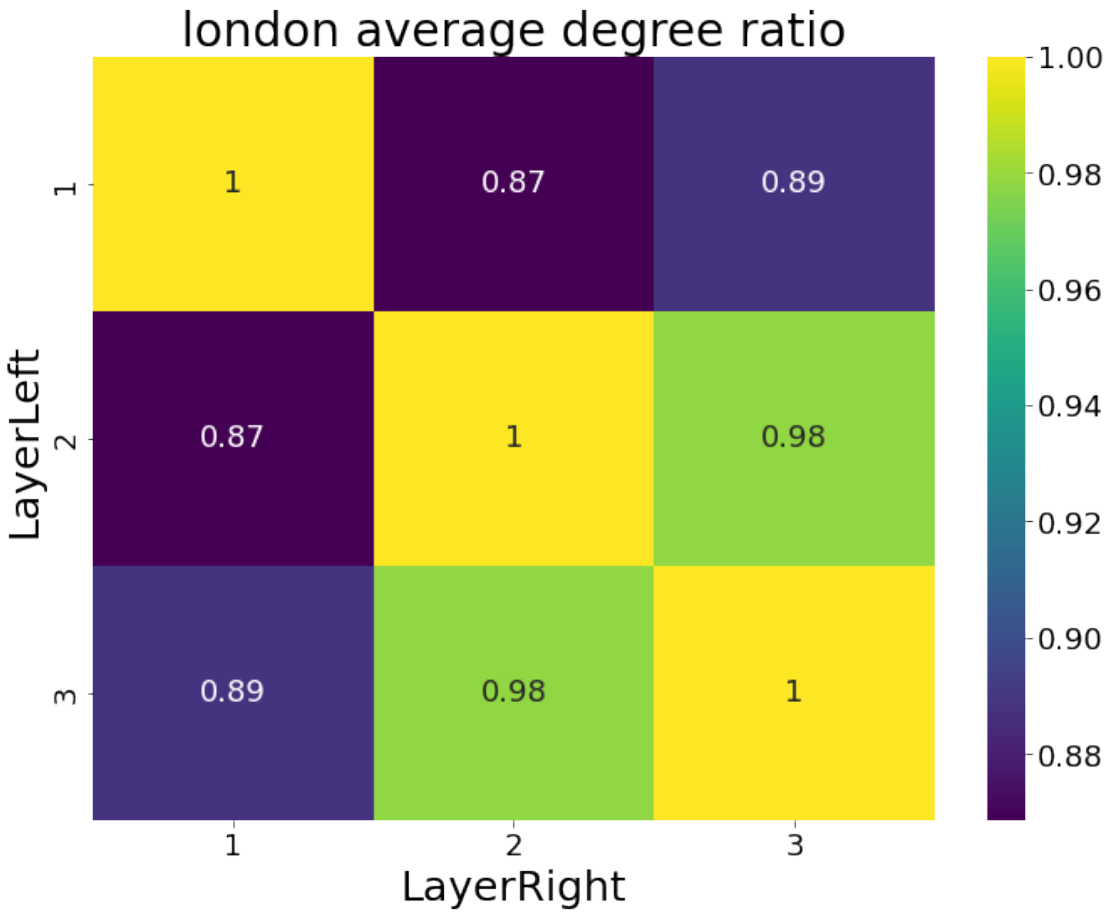
[ ]:
```
plt.figure()
sns.relplot(data=df_perf, kind="line",
            x="PFI", y="Accuracy",
            col="LayerLeft",
            hue="LayerRight",
```

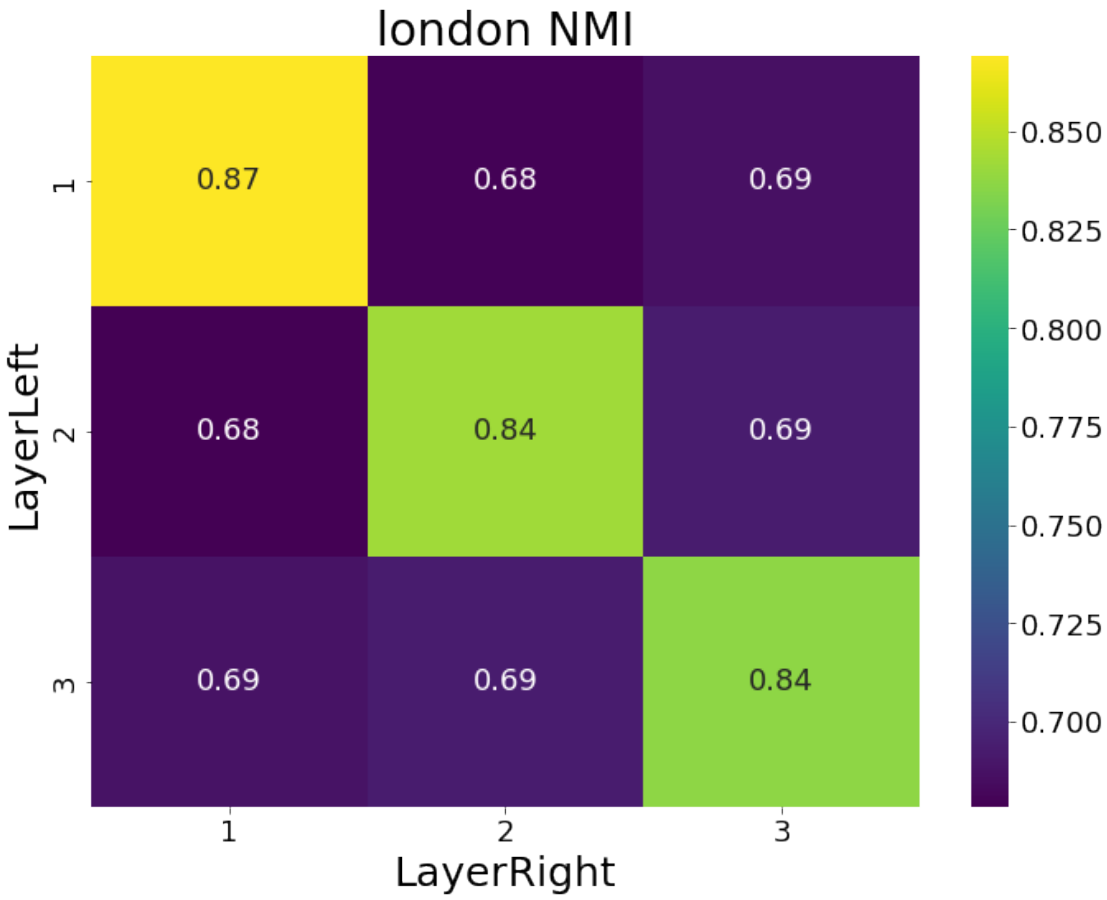
```
            palette="dark",
            height=12, aspect=1.4
    )
plt.title(network)
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Edge␣
 ↪Overlap")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} edge overlap")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="Ratio")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} average degree ratio")
plt.tight_layout()

heatmap = df_sc.pivot(index="LayerLeft", columns="LayerRight", values="NMI")
plt.figure()
sns.heatmap(heatmap, cmap="viridis", annot=True)
plt.title(f"{network} NMI")
plt.tight_layout()
```

```
<Figure size 720x576 with 0 Axes>
```




london edge overlap
1
0.029
0
0.029
1
0
0
0
1
LayerLeft
1
2
3
LayerRight
1
2
3
1.0
0.8
0.6
0.4
0.2
0.0


london average degree ratio
1
0.87
0.89
0.87
1
0.98
0.89
0.98
1
LayerLeft
LayerRight
1
2
3
1.00
0.98
0.96
0.94
0.92
0.90
0.88



0.22 Observation

Very atypical network structure. Told to ignore for now.