test_LFR_partial-info

September 19, 2022

[ ]:
```python
# ============= SET-UP =====================
# --- Standard library ---
import sys  # System pathing
from copy import deepcopy

# --- Scientific ---
import numpy as np  # General computational tools
from sklearn import metrics  # Measuring classifier performance
from scipy.special import comb

# --- Network science ---
import networkx as nx  # General network tools
from node2vec import Node2Vec as N2V  # Embedding tools

# --- Project source code ---
sys.path.append("../src/")
from Utils import *  # Custom synthetic benchmarks
from glee import eigenmaps

# --- Data handling and visualization ---
import matplotlib.pyplot as plt

# --- Miscellaneous ---
accuracy = metrics.accuracy_score
auroc = metrics.roc_auc_score
```

[ ]:
```python
# Process parameters
N = 500
tau1 = 2.1
tau2 = 1.0
mu = 0.1
min_community = 1
average_degree = 5
max_degree = np.sqrt(N)
prob_relabel = 1.0

pfi = 0.5
```

```python
largest_component = True

dimensions = 2

# Form "raw" duplex
D, _sigma1, _sigma2, _mu_temp = lfr_multiplex(N, tau1, tau2, mu,␣
↪average_degree, max_degree, min_community, prob_relabel)

# Split into layers
G, H = duplex_network(D, 1, 2)

# Observe partial information
R_G, R_H, testset = partial_information(G, H, pfi)

# Restrict to largest connected component (if specified)
if largest_component:
    R_G_ = nx.Graph()
    R_H_ = nx.Graph()
    R_G_.add_nodes_from(R_G.nodes())
    R_H_.add_nodes_from(R_H.nodes())

    maxcc_R_G = max(nx.connected_components(R_G), key=len)
    maxcc_R_H = max(nx.connected_components(R_H), key=len)

    edges_R_G_ = set(R_G.subgraph(maxcc_R_G).edges())
    edges_R_H_ = set(R_H.subgraph(maxcc_R_H).edges())
    R_G_.add_edges_from(edges_R_G_)
    R_H_.add_edges_from(edges_R_H_)

    testset = {
        edge: gt_
        for edge, gt_ in testset.items()
        if edge in edges_R_G_ & edges_R_H_
    }
else:
    R_G_ = R_G
    R_H_ = R_H
```

```
setting… -N 500
setting… -mu 0.1
setting… -maxk 22.360679774997898
setting… -k 5
setting… -t1 2.1
setting… -t2 1.0

**************************************************************
```

```
number of nodes:        500
average degree: 5
maximum degree: 22
exponent for the degree distribution:   2.1
exponent for the community size distribution:   1
mixing parameter:       0.1
number of overlapping nodes:    0
number of memberships of the overlapping nodes: 0
**************************************************************

-----------------------------------------------------------
community size range automatically set equal to [3 , 22]
building communities…
connecting communities…
recording network…


---------------------------------------------------------------------------
network of 500 vertices and 1092 edges;  average degree = 4.368

average mixing parameter: 0.0973671 +/- 0.166681
p_in: 0.506756  p_out: 0.000737044

Segmentation fault (core dumped)
```

[ ]:
```python
c = set(R_G_.edges()) & set(R_H_.edges())
R_G_cut_c = set(R_G_.edges()) - c
R_H_cut_c = set(R_H_.edges()) - c

print(
    len(c & set(testset.keys())) / len(set(testset.keys())),
    len(R_G_cut_c)/len(set(R_G_.edges())),
    len(R_H_cut_c)/len(set(R_H_.edges()))
)
```

```
1.0 0.3484472049689441 0.34231974921630093
```

[ ]:
```python
embedding = "n2v"

if embedding == "glee":
    E_alpha = eigenmaps(R_G_, dimensions, method='glee', return_vals=False)
    E_beta = eigenmaps(R_H_, dimensions, method='glee', return_vals=False)
elif embedding == "n2v":
    E_alpha = N2V(R_G_, dimensions=dimensions, walk_length=30, num_walks=200,␣
 ↪workers=4, quiet=True).fit(window=10, min_count=1, batch_words=4).wv.vectors
    E_beta = N2V(R_H_, dimensions=dimensions, walk_length=30, num_walks=200,␣
 ↪workers=4, quiet=True).fit(window=10, min_count=1, batch_words=4).wv.vectors
```

[ ]:
```python
def reconstruct_system(testset, G, H, G_, H_, metric="negexp"):
    cls = []
    scores = []
    gt = []

    for edge, gt_ in testset.items():
        i, j = edge
        gt.append(gt_)

        v_G_i = G_[i, :]
        v_G_j = G_[j, :]
        v_H_i = H_[i, :]
        v_H_j = H_[j, :]

        d_G = np.linalg.norm(v_G_i - v_G_j)
        d_H = np.linalg.norm(v_H_i - v_H_j)

        # * Thresholding to avoid inactive nodes, embedded at origin, having␣
 ↪incorrect placements
        if d_G <= 1e-1:
            d_G = sys.maxsize
        if d_H <= 1e-1:
            d_H = sys.maxsize

        if metric == "inverse":
            d_G = 1 / d_G
            d_H = 1 / d_H
        elif metric == "negexp":
            d_G = np.exp(-d_G)
            d_H = np.exp(-d_H)

        t_G = d_G / (d_G + d_H)
        t_H = 1 - t_G

        scores.append(t_G)

        cls_ = np.random.randint(2)
        if t_G != t_H:
            if np.random.rand() <= t_G:
                cls_ = 1
            else:
                cls_ = 0
        cls.append(cls_)

    return cls, scores, gt

def measure_performance(cls, scores, gt):
```

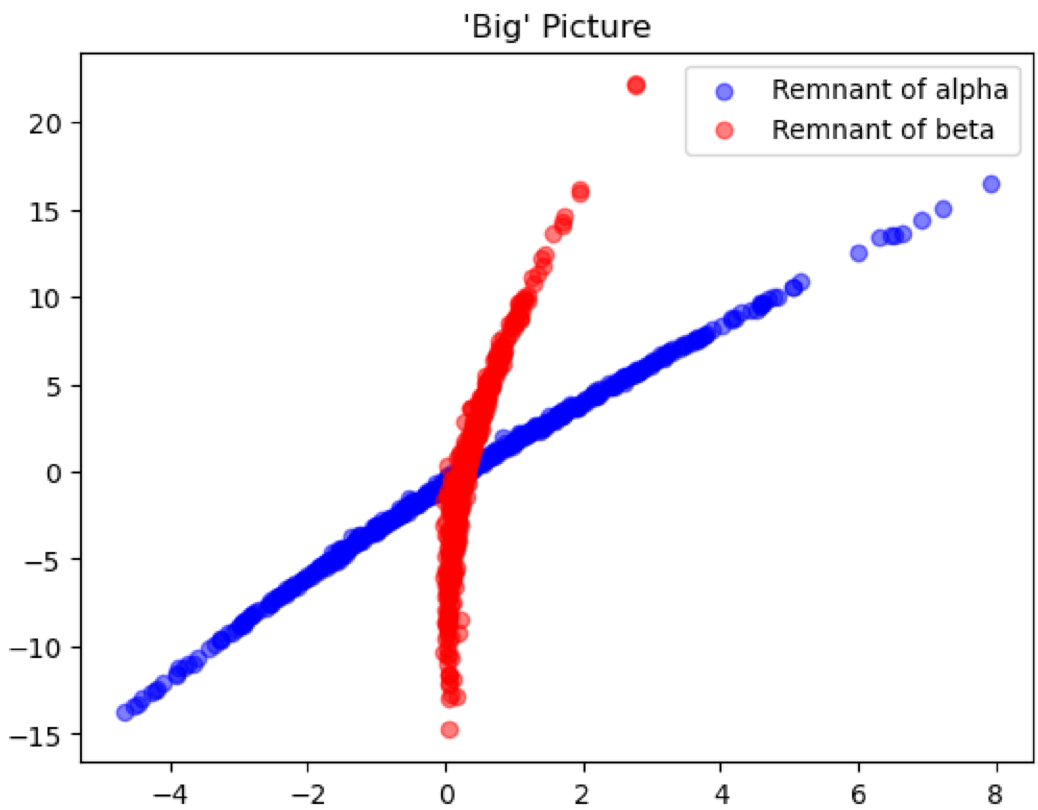
```
    acc = accuracy(gt, cls)
    auc = auroc(gt, scores)

    return acc, auc
```

[ ]:
```
cls, scores, gt = reconstruct_system(testset, G, H, E_alpha, E_beta, "inverse")
acc, auc = measure_performance(cls, scores, gt)
print(acc, auc)
```

```
0.5487179487179488 0.5183458237418274
```

[ ]:
```
plt.figure()
plt.scatter(
    x=[v[0] for v in E_alpha],
    y=[v[1] for v in E_alpha],
    color="blue",
    label="Remnant of alpha",
    alpha=0.5
)
plt.scatter(
    x=[v[0] for v in E_beta],
    y=[v[1] for v in E_beta],
    color="red",
    label="Remnant of beta",
    alpha=0.5
)
plt.legend()
plt.title("'Big' Picture")
```

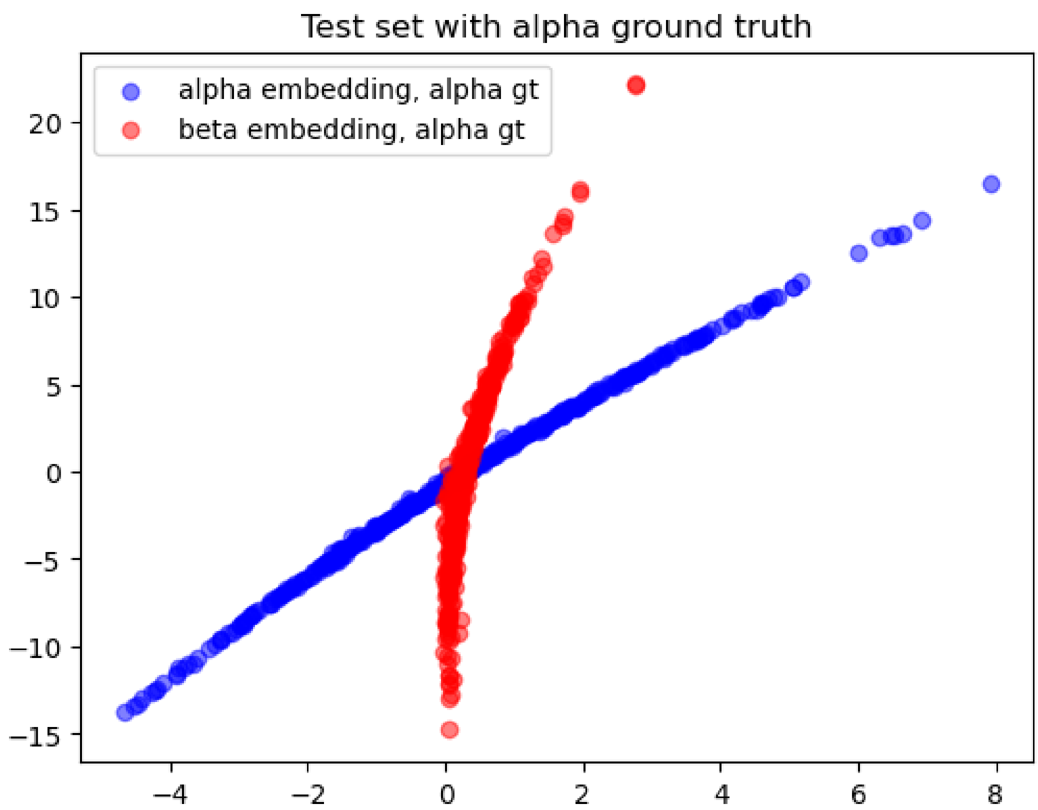
[ ]:
```
Text(0.5, 1.0, "'Big' Picture")
```

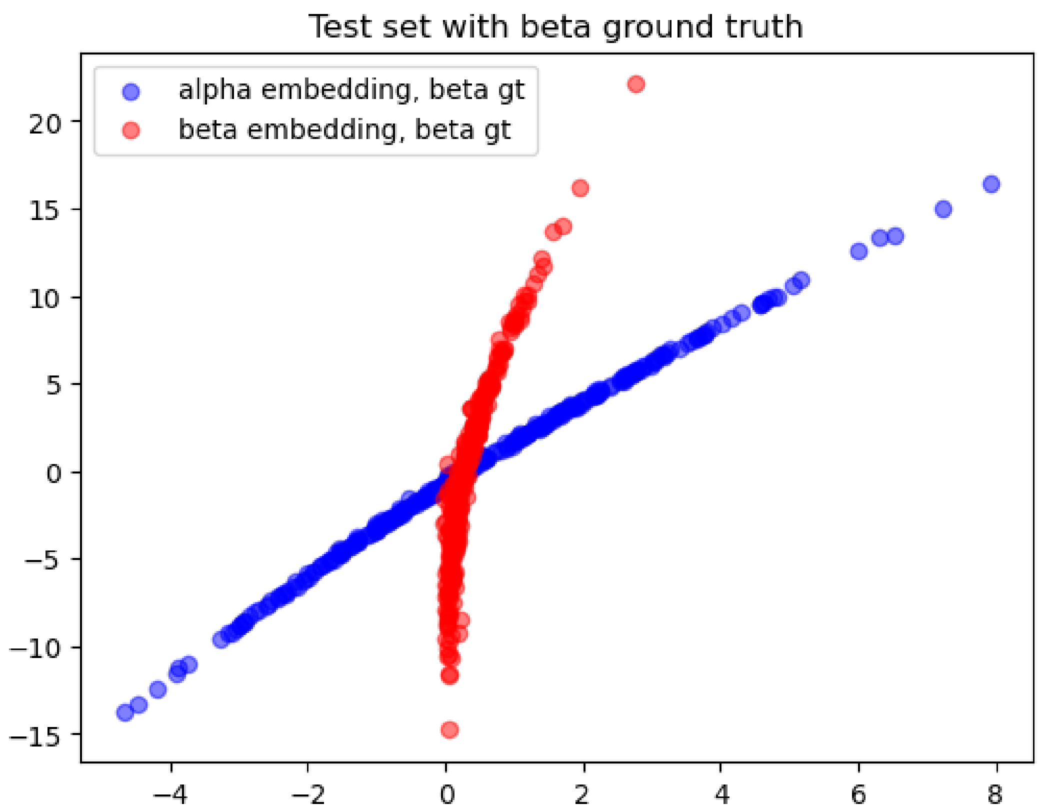


[ ]:
```
edges_gt_alpha = [
    edge
    for edge, gt in testset.items() if gt == 1
]
edges_gt_beta = [
    edge
    for edge, gt in testset.items() if gt == 0
]

nodes_gt_alpha = set([x[0] for x in edges_gt_alpha]) | set([x[1] for x in␣
 ↪edges_gt_alpha])
nodes_gt_beta = set([x[0] for x in edges_gt_beta]) | set([x[1] for x in␣
 ↪edges_gt_beta])


plt.figure()
plt.scatter(
    x=[E_alpha[v][0] for v in nodes_gt_alpha],
    y=[E_alpha[v][1] for v in nodes_gt_alpha],
    color="blue",
    label="alpha embedding, alpha gt",
```

```
    alpha=0.5
)
plt.scatter(
    x=[E_beta[v][0] for v in nodes_gt_alpha],
    y=[E_beta[v][1] for v in nodes_gt_alpha],
    color="red",
    label="beta embedding, alpha gt",
    alpha=0.5
)
plt.legend()

plt.title("Test set with alpha ground truth")

# ----------------


plt.figure()
plt.scatter(
    x=[E_alpha[v][0] for v in nodes_gt_beta],
    y=[E_alpha[v][1] for v in nodes_gt_beta],
    color="blue",
    label="alpha embedding, beta gt",
    alpha=0.5
)
plt.scatter(
    x=[E_beta[v][0] for v in nodes_gt_beta],
    y=[E_beta[v][1] for v in nodes_gt_beta],
    color="red",
    label="beta embedding, beta gt",
    alpha=0.5
)
plt.legend()

plt.title("Test set with beta ground truth")
```

[ ]: Text(0.5, 1.0, 'Test set with beta ground truth')

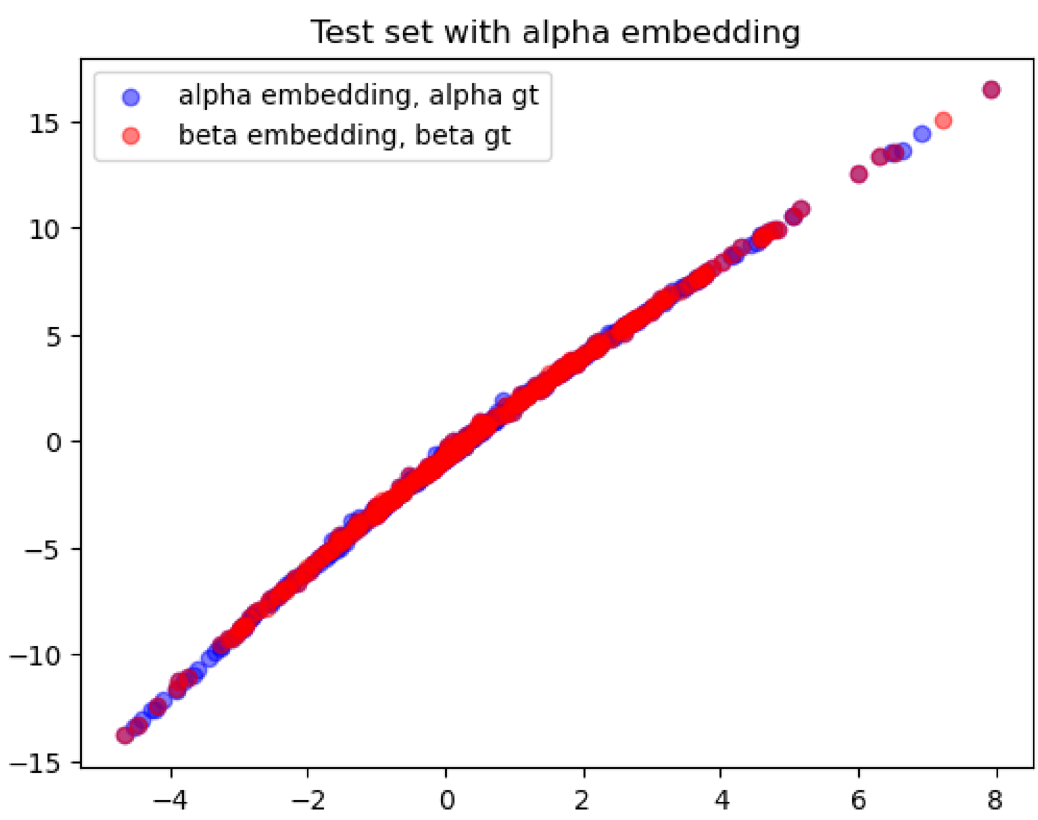

Test set with alpha ground truth
alpha embedding, alpha gt
beta embedding, alpha gt
20
15
10
5
0
−5
−10
−15
−4
−2
0
2
4
6
8

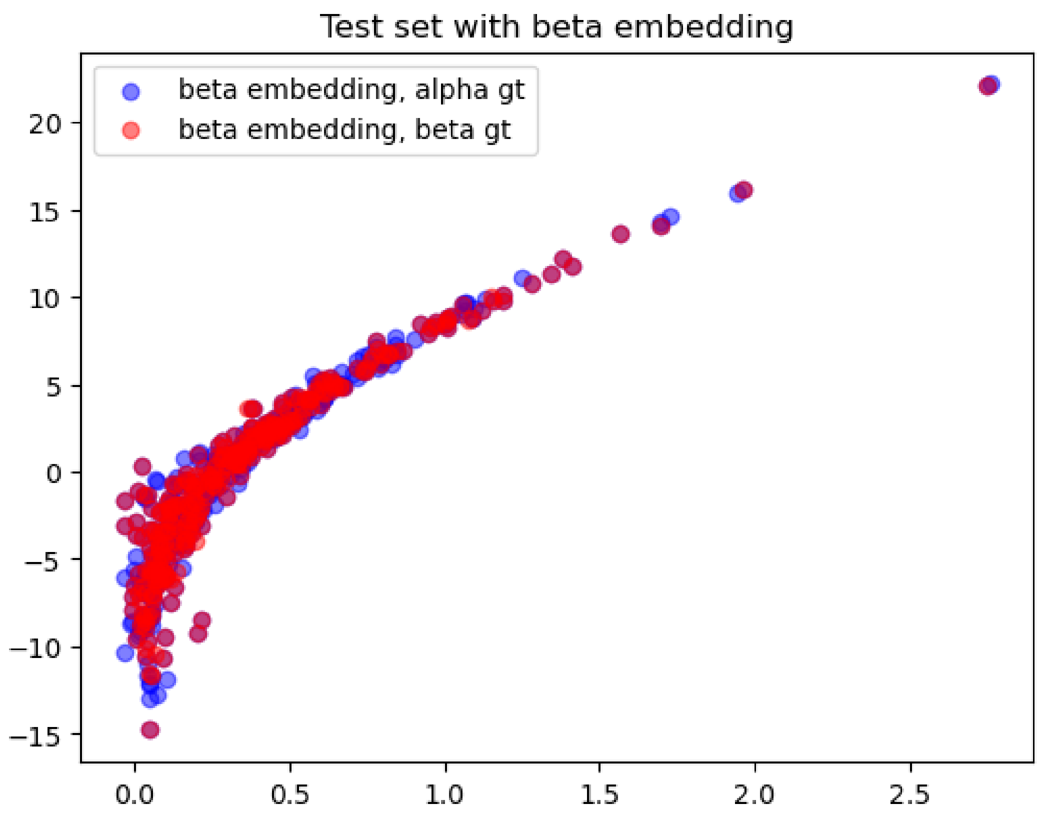



```
[ ]: plt.figure()
plt.scatter(
    x=[E_alpha[v][0] for v in nodes_gt_alpha],
    y=[E_alpha[v][1] for v in nodes_gt_alpha],
    color="blue",
    label="alpha embedding, alpha gt",
    alpha=0.5
)
plt.scatter(
    x=[E_alpha[v][0] for v in nodes_gt_beta],
    y=[E_alpha[v][1] for v in nodes_gt_beta],
    color="red",
    label="beta embedding, beta gt",
    alpha=0.5
)
plt.legend()

plt.title("Test set with alpha embedding")

# ----------------
```

```
plt.figure()
plt.scatter(
    x=[E_beta[v][0] for v in nodes_gt_alpha],
    y=[E_beta[v][1] for v in nodes_gt_alpha],
    color="blue",
    label="beta embedding, alpha gt",
    alpha=0.5
)
plt.scatter(
    x=[E_beta[v][0] for v in nodes_gt_beta],
    y=[E_beta[v][1] for v in nodes_gt_beta],
    color="red",
    label="beta embedding, beta gt",
    alpha=0.5
)
plt.legend()

plt.title("Test set with beta embedding")
```

```
[ ]: Text(0.5, 1.0, 'Test set with beta embedding')
```



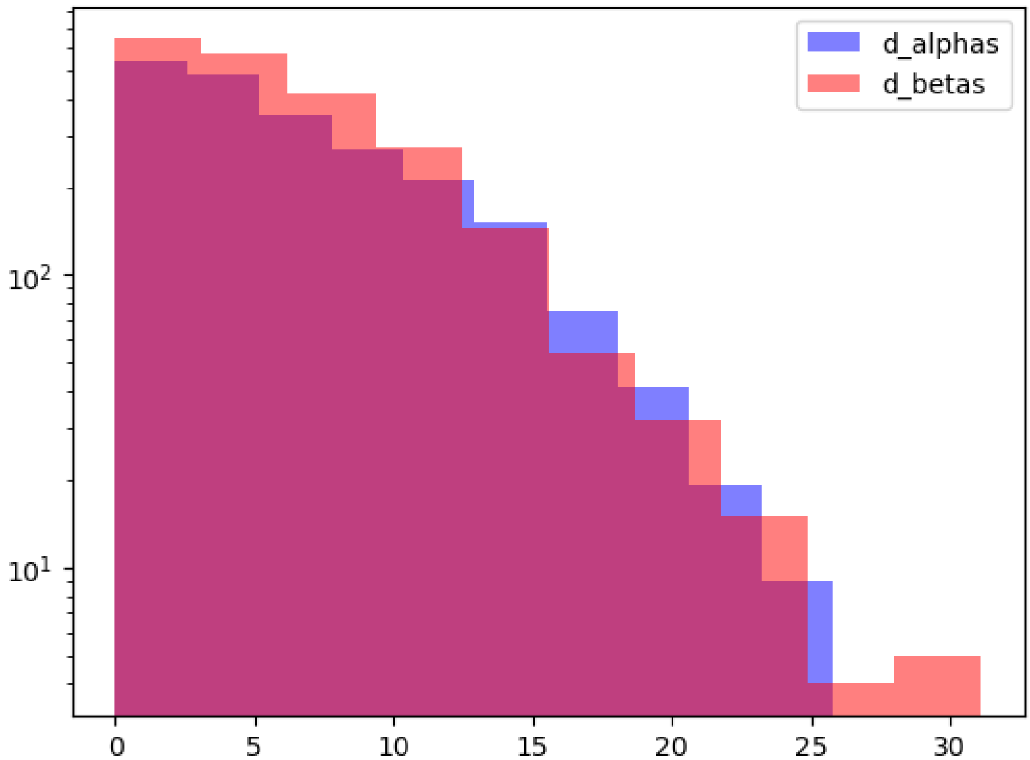


[ ]:
```
deltas_alpha = {edge: None for edge in set(R_G_.edges())}
deltas_beta = {edge: None for edge in set(R_H_.edges())}

for edge in deltas_alpha.keys():
    i, j = edge
    d_alpha = np.linalg.norm(E_alpha[i] - E_alpha[j])
    d_beta = np.linalg.norm(E_beta[i] - E_beta[j])
    deltas_alpha[edge] = (d_alpha, d_beta)

for edge in deltas_beta.keys():
    i, j = edge
    d_alpha = np.linalg.norm(E_alpha[i] - E_alpha[j])
    d_beta = np.linalg.norm(E_beta[i] - E_beta[j])
    deltas_beta[edge] = (d_alpha, d_beta)

deltas = deepcopy(deltas_alpha)
deltas.update(deltas_beta)
```

[ ]:
```
plt.figure()
plt.hist([x[0] for x in deltas.values()], color="blue", label="d_alphas",␣
 ↪alpha=0.5)
```

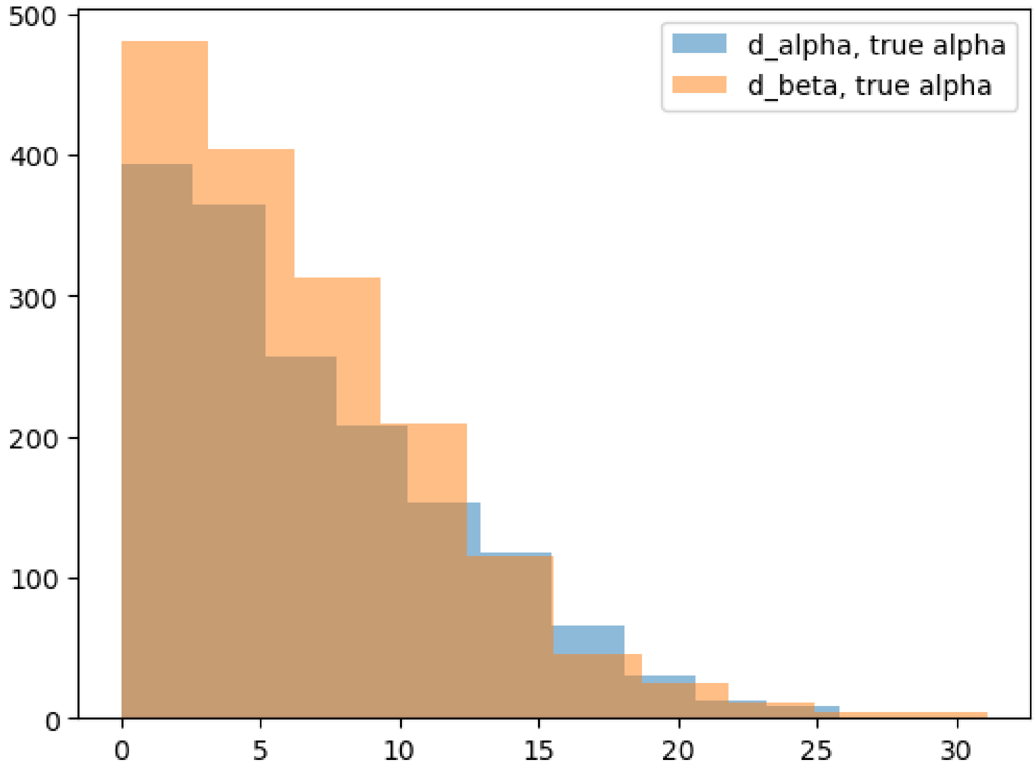
```
plt.hist([x[1] for x in deltas.values()], color="red", label="d_betas", alpha=0.
  ↪5)
plt.yscale("log")
plt.legend()
```

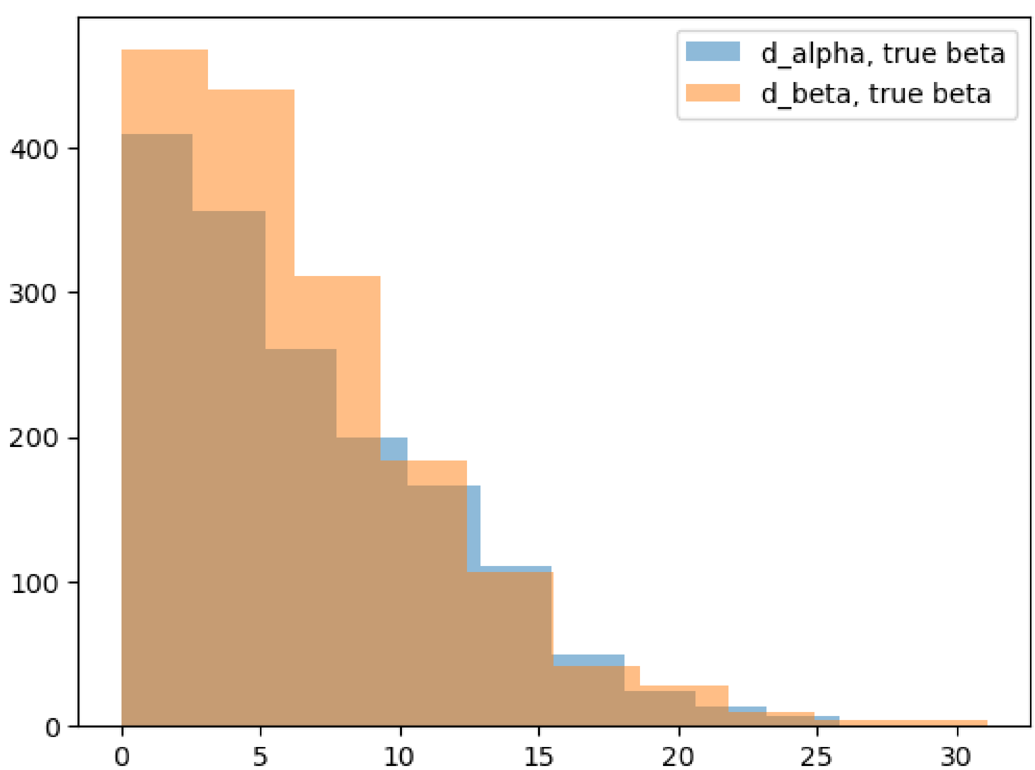
[ ]: <matplotlib.legend.Legend at 0x7f8bfd22fb80>



[ ]:
```
plt.figure()
plt.hist([x[0] for x in deltas_alpha.values()], label="d_alpha, true alpha",␣
  ↪alpha=0.5)
plt.hist([x[1] for x in deltas_alpha.values()], label="d_beta, true alpha",␣
  ↪alpha=0.5)
plt.legend()

plt.figure()
plt.hist([x[0] for x in deltas_beta.values()], label="d_alpha, true beta",␣
  ↪alpha=0.5)
plt.hist([x[1] for x in deltas_beta.values()], label="d_beta, true beta",␣
  ↪alpha=0.5)
plt.legend()
```

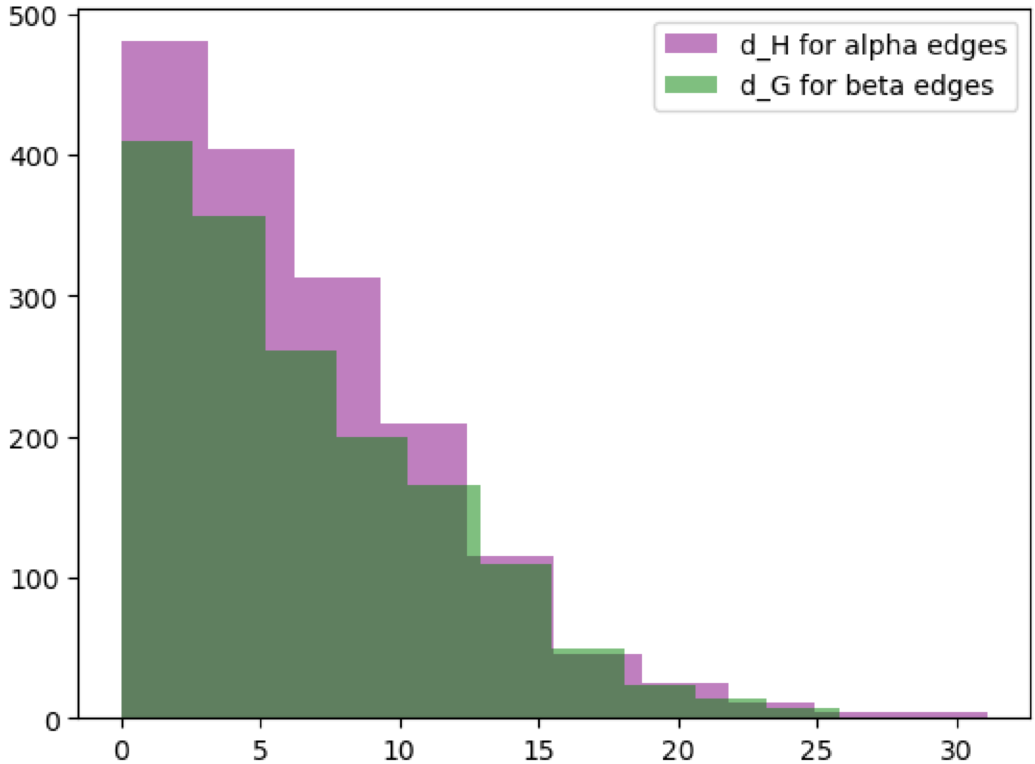
[ ]: <matplotlib.legend.Legend at 0x7f8bfd0073a0>


d_alpha, true alpha
d_beta, true alpha
500
400
300
200
100
0
0
5
10
15
20
25
30


d_alpha, true beta
d_beta, true beta
400
300
200
100
0
0
5
10
15
20
25
30

[ ]:
```python
d_a_bad = []
d_b_bad = []


# Within 0 to 99, alpha stays same, beta goes to 100 + i
for edge in R_G_.edges():
    i, j = edge
    _h = np.linalg.norm(E_beta[i] - E_beta[j])
    d_a_bad.append(_h)


# Within 100 to 199, alpha stays same, beta goes to 100 - i
for edge in R_H_.edges():
    i, j = edge
    _g = np.linalg.norm(E_alpha[i] - E_alpha[j])
    d_b_bad.append(_g)

plt.figure()
plt.hist(d_a_bad, color="purple", label="d_H for alpha edges", alpha=0.5)
plt.hist(d_b_bad, color="green", label="d_G for beta edges", alpha=0.5)
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f8ba7626dd0>
```


d_H for alpha edges
d_G for beta edges
500
400
300
200
100
0
0
5
10
15
20
25
30

[ ]: