Final Project Presentation

Inferring Social Ties in Large Social Networks

Moitrayee Chatterjee

Supra Jyotsna Jampa

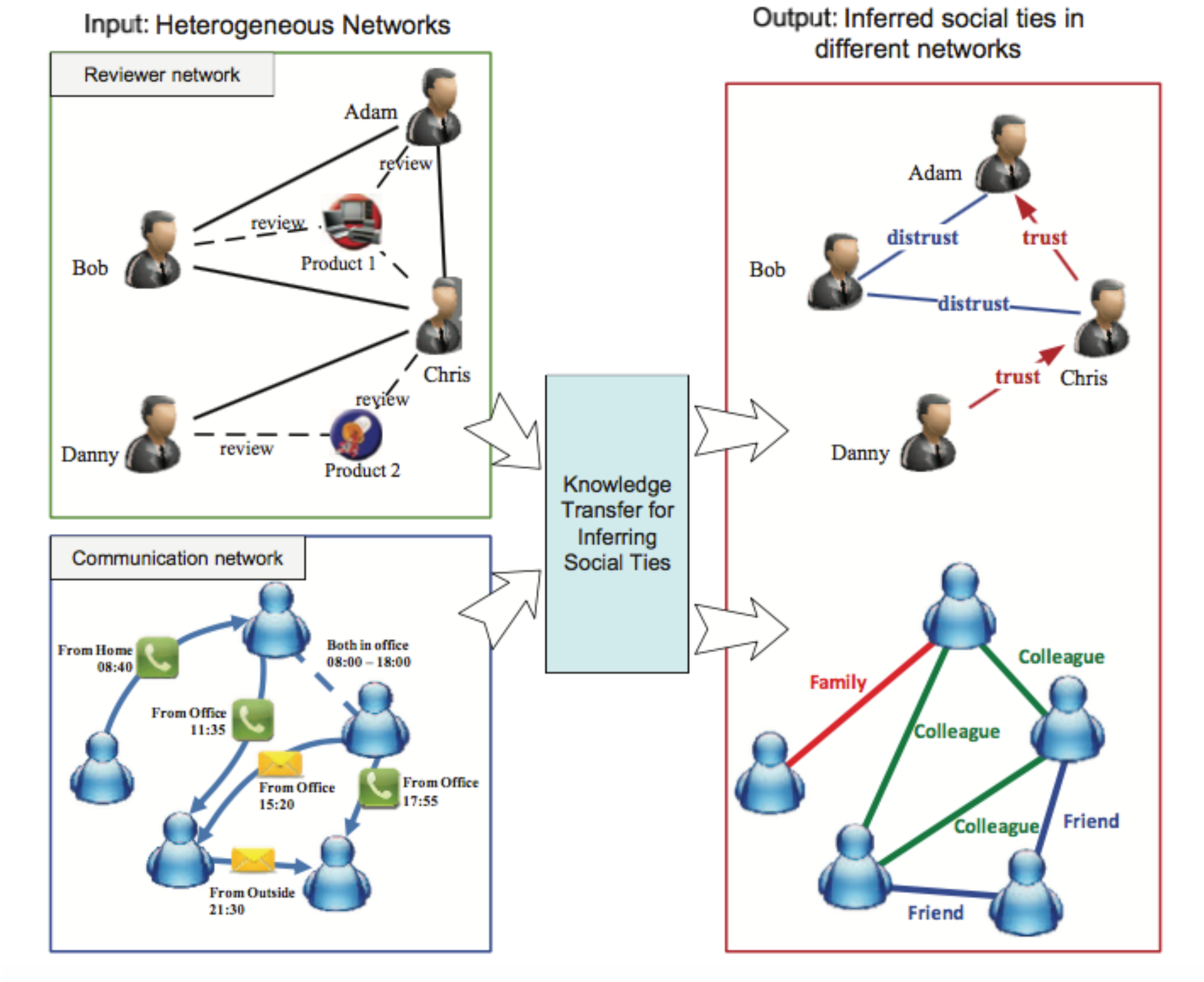
Introduction:

- People are connected with different types of social ties in different networks.
- Examples : Facebook, Twitter, LinkedIn, LiveJournal, etc.
- Existence of multiple networks result in overlapping networks.

Problem Statement

- Lack of labelled Social Networking nodes.
- Different types of social ties have essentially different influence between people.
- Awareness of these different types of social relationships can benefit many applications.
- The source network to help infer social ties in the target network.

Example:



Input: Heterogeneous Networks
Reviewer network
Adam
review
Bob
review
Product 1
Chris
review
Danny
review
Product 2
Communication network
From Home 08:40
Both in office 08:00 – 18:00
From Office 11:35
From Office 15:20
From Office 17:55
From Outside 21:30
Knowledge Transfer for Inferring Social Ties
Output: Inferred social ties in different networks
Adam
distrust
trust
Bob
distrust
trust
Chris
Danny
Family
Colleague
Colleague
Colleague
Friend
Friend

Literature Review

- Use of psychological theories to analyze the network based correlation.
- Statistics for calculating the network based correlations
  - Social Balance

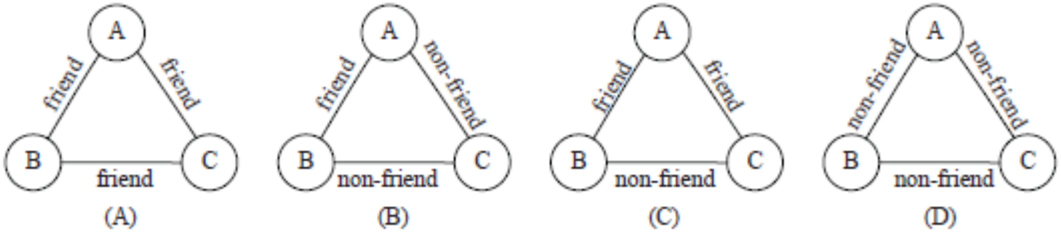


Figure 2: Illustration of structural balance theory. (A) and (B) are balanced, while (C) and (D) are not balanced.

- Structural Hole
- Social Status

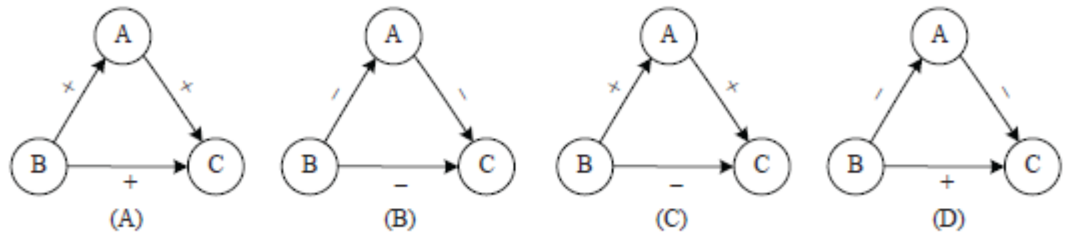


Figure 5: Illustration of status theory. (A) and (B) satisfy the status theory, while (C) and (D) do not satisfy the status theory. Here positive "+" denotes the target node has a higher status than the source node;

- Two step flow

Motivation:

- Using social theories to infer social ties in the target network.
- Previous works focus on mining particular types of relationships in specific domains.
- Predict associations based on existing social ties.
- Use of Matrix Factorization to learn the nature of relationship.

Matrix Factorization:

- Used to discover latent features underlying the interactions between two (or more) kinds of entities.
- R is a matrix of size |U|*|D
- K is the number of features to be discovered
- Find two matrices P(|U|*K) and Q(|D| *K)

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

Example of Matrix Factorization:

- Main application : Collaborative Filtering

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	1	4

	D1	D2	D3	D4
U1	4.97	2.98	2.18	0.98
U2	3.97	2.40	1.97	0.99
U3	1.02	0.93	5.32	4.93
U4	1.00	0.85	4.59	3.93
U5	1.36	1.07	4.89	4.12

DATASET: Slashdot social network, February 2009

- https://snap.stanford.edu/data/soc-Slashdot0902.html

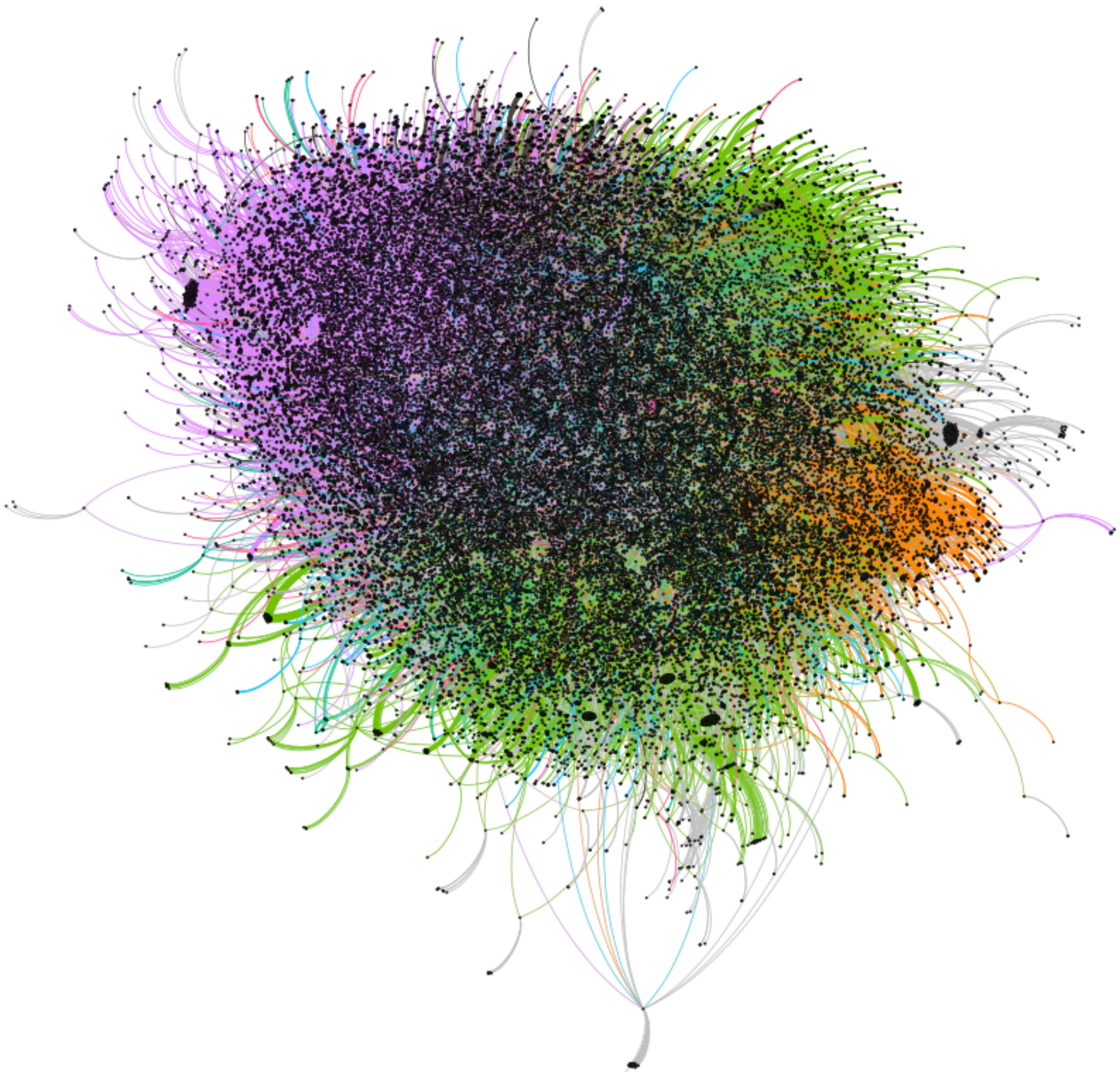
Name	Type	Nodes	Edges	Description
soc-Slashdot0922	Directed	82,168	948,464	Slashdot social network from February 2009

Dataset statistics	
Nodes	82168
Edges	948464
Nodes in largest WCC	82168 (1.000)
Edges in largest WCC	948464 (1.000)
Nodes in largest SCC	71307 (0.868)
Edges in largest SCC	912381 (0.962)
Average clustering coefficient	0.0603
Number of triangles	602592
Fraction of closed triangles	0.008168
Diameter (longest shortest path)	11
90-percentile effective diameter	4.7

Slashdot0902.txt

```
# Directed graph (each unordered pair of
nodes is saved once): Slashdot0902.txt
# Slashdot Zoo social network from
February 0 2009
# Nodes: 82168 Edges: 948464
# FromNodeId    ToNodeId
0       0
0       1
0       2
0       3
0       4
0       5
0       6
0       7
0       8
0       9
0       10
0       11
0       12
0       13
0       14
0       15
0       16
0       17
0       18
0       19
0       20
0       21
0       22
```

VISUALIZATION OF THE DATASET



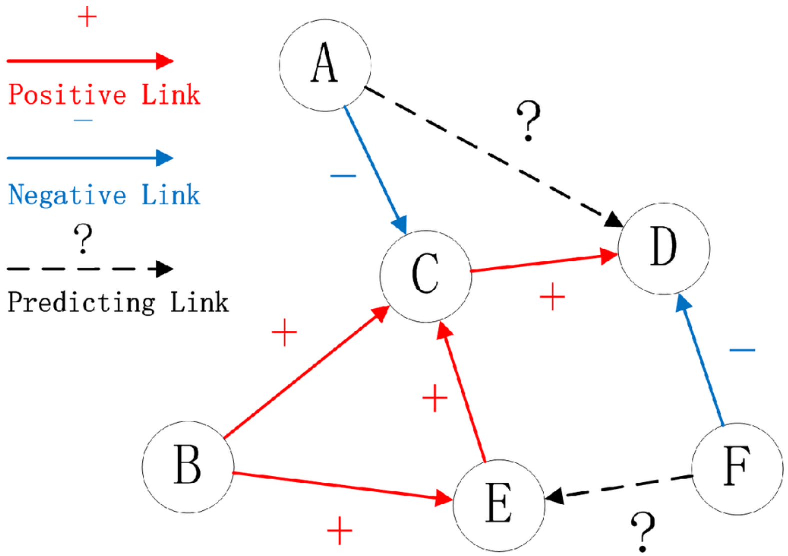
IMPLEMNTATION: Pre-processing

- **Done using R.**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
17	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
19	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
20	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
21	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
22	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
23	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
25	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

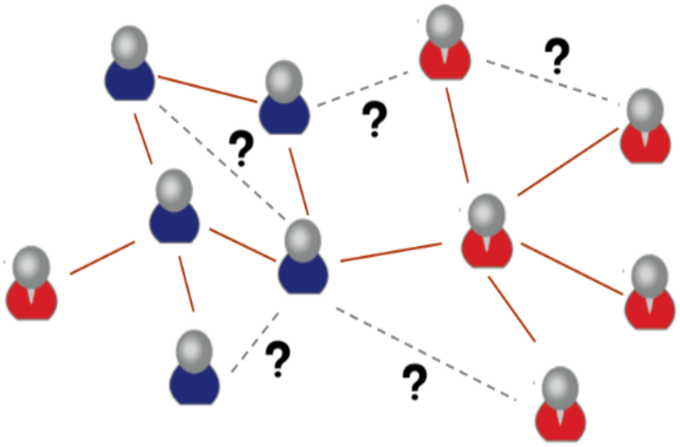
IMPLEMNTATION: Learning

- Implemented using Python.
  - *NumPy*: The fundamental package for scientific computing with Python.
  - *xrange(start, stop[, step]):* This is an opaque sequence type which yields the values, without actually storing them all simultaneously.



KEY LEARNING PARAMETERS

- Update each element P and Q.
- Use learning rate $\alpha := 0.002$.
- Steps := 400
- Regularization parameter $\beta := 0.02$



ALGORITHM

```
→ Initialize P and Q with random small numbers
→ for step until max_steps:
    for row, col in R:
      if R[row][col] > 0:
        compute error of element
        compute gradient from error
        update P and Q with new entry

      compute total error
      if error < some threshold:
        break
  return P, Q.T
```

CODE SNAPSHOT

```
def matrix_factorization(R, P, Q, K, steps=400, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta *
P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta *
Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in xrange(K):
                        e = e + (beta/2) * ( pow(P[i][k],2) + pow(Q[k][j],2) )
        if e < 0.001:
            break
    return P, Q.T
```

INPUTS

```
>>> R = [
         [1,0,0,1],
         [0,0,0,1],
         [1,1,0,0],
         [1,0,0,1],
         [0,1,1,1],
        ]

    R = numpy.array(R)

    N = len(R)
    M = len(R[0])
    K = 2

    P = numpy.random.rand(N,K)
    Q = numpy.random.rand(M,K)

    nP, nQ = matrix_factorization(R, P, Q, K)
```

OUTPUT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	1.189	1.091	0.934	0.852	0.939	1	1.227	0.887	1.041	1.117	1.017	0.803	1.041	1.086	1.061	1.196	0.846	1.081	1.025	1.114	1.032	0.915	0.979	0.877	0.642	0.869
1	1.097	1.076	0.831	0.811	0.878	0.864	1.135	0.841	0.997	0.974	1.027	0.682	0.941	0.923	0.95	1.092	0.761	1.026	0.918	0.997	0.989	0.875	0.874	0.808	0.646	0.756
2	1	0.643	0.907	0.621	0.744	1.075	1.019	0.659	0.731	1.162	0.508	0.913	0.955	1.225	1.005	1.053	0.786	0.795	0.973	1.058	0.72	0.647	0.941	0.745	0.33	0.914
3	0.962	1.011	0.7	0.735	0.782	0.701	0.999	0.759	0.91	0.8	0.986	0.54	0.806	0.734	0.807	0.946	0.65	0.928	0.779	0.845	0.904	0.797	0.738	0.707	0.618	0.619
4	1.157	1.047	0.916	0.824	0.912	0.987	1.194	0.859	1.005	1.1	0.971	0.795	1.018	1.075	1.039	1.167	0.827	1.046	1.004	1.091	0.996	0.883	0.96	0.854	0.613	0.856
5	0.922	0.618	0.825	0.582	0.69	0.97	0.941	0.615	0.687	1.051	0.5	0.82	0.873	1.101	0.916	0.967	0.718	0.743	0.887	0.964	0.677	0.608	0.857	0.687	0.324	0.826
6	0.945	0.865	0.744	0.677	0.746	0.798	0.975	0.704	0.826	0.89	0.805	0.641	0.829	0.867	0.845	0.951	0.673	0.858	0.816	0.887	0.819	0.726	0.78	0.698	0.508	0.693
7	0.974	1.025	0.708	0.744	0.792	0.709	1.012	0.769	0.922	0.81	0.999	0.546	0.816	0.742	0.816	0.958	0.658	0.94	0.788	0.856	0.916	0.808	0.747	0.716	0.626	0.625
8	1.044	0.967	0.817	0.751	0.826	0.872	1.078	0.782	0.918	0.974	0.904	0.699	0.912	0.945	0.928	1.049	0.741	0.953	0.897	0.975	0.911	0.807	0.857	0.77	0.57	0.758
9	1.051	0.855	0.874	0.715	0.812	0.977	1.079	0.749	0.862	1.076	0.76	0.805	0.952	1.084	0.982	1.076	0.777	0.91	0.95	1.032	0.853	0.76	0.912	0.778	0.484	0.841
10	0.844	0.76	0.67	0.6	0.664	0.723	0.87	0.625	0.731	0.805	0.703	0.584	0.744	0.788	0.759	0.852	0.605	0.761	0.734	0.797	0.724	0.643	0.702	0.623	0.444	0.627
11	0.795	0.61	0.677	0.528	0.608	0.77	0.815	0.555	0.633	0.843	0.529	0.64	0.73	0.861	0.758	0.82	0.598	0.673	0.733	0.797	0.625	0.558	0.706	0.59	0.338	0.66
12	0.951	0.756	0.799	0.641	0.732	0.9	0.976	0.672	0.771	0.988	0.666	0.744	0.866	1.002	0.896	0.977	0.708	0.816	0.867	0.942	0.762	0.68	0.833	0.705	0.424	0.773
13	1.073	1.057	0.812	0.795	0.86	0.842	1.111	0.824	0.977	0.95	1.009	0.664	0.92	0.899	0.929	1.068	0.744	1.006	0.897	0.974	0.97	0.858	0.854	0.79	0.635	0.737
14	0.999	0.795	0.838	0.674	0.769	0.944	1.025	0.707	0.81	1.037	0.701	0.78	0.91	1.05	0.941	1.026	0.744	0.857	0.91	0.989	0.801	0.714	0.875	0.74	0.447	0.811
15	0.97	0.853	0.779	0.683	0.76	0.848	0.999	0.712	0.83	0.942	0.783	0.688	0.86	0.929	0.881	0.982	0.7	0.867	0.852	0.925	0.822	0.73	0.815	0.717	0.496	0.734
16	1.12	1.146	0.828	0.844	0.904	0.841	1.161	0.873	1.042	0.956	1.108	0.655	0.947	0.889	0.951	1.106	0.764	1.067	0.918	0.997	1.035	0.914	0.872	0.823	0.695	0.74
17	1.083	0.813	0.931	0.713	0.825	1.066	1.109	0.751	0.853	1.164	0.698	0.888	1	1.195	1.04	1.121	0.82	0.909	1.007	1.094	0.842	0.753	0.97	0.804	0.447	0.912
18	1.021	0.936	0.803	0.732	0.807	0.861	1.054	0.762	0.893	0.961	0.872	0.692	0.895	0.935	0.912	1.028	0.727	0.928	0.881	0.958	0.885	0.785	0.842	0.754	0.551	0.748
19	0.919	0.598	0.83	0.573	0.685	0.982	0.936	0.607	0.675	1.062	0.475	0.833	0.875	1.118	0.92	0.966	0.72	0.733	0.891	0.969	0.665	0.597	0.862	0.684	0.308	0.835
20	0.958	0.741	0.814	0.638	0.734	0.924	0.982	0.671	0.766	1.012	0.644	0.767	0.879	1.032	0.911	0.987	0.719	0.813	0.882	0.958	0.757	0.675	0.848	0.711	0.412	0.793
21	1.138	0.988	0.92	0.796	0.89	1.006	1.172	0.831	0.967	1.115	0.902	0.818	1.013	1.104	1.039	1.155	0.825	1.011	1.004	1.091	0.957	0.85	0.962	0.842	0.572	0.87
22	1.048	1.108	0.759	0.802	0.852	0.757	1.088	0.829	0.994	0.866	1.082	0.582	0.876	0.792	0.875	1.029	0.706	1.014	0.845	0.918	0.988	0.871	0.801	0.77	0.678	0.669
23	0.935	0.79	0.764	0.647	0.727	0.844	0.961	0.676	0.783	0.933	0.714	0.69	0.838	0.93	0.861	0.952	0.683	0.822	0.833	0.905	0.775	0.689	0.799	0.691	0.453	0.728
24	0.891	0.791	0.712	0.629	0.699	0.772	0.918	0.656	0.766	0.858	0.729	0.625	0.788	0.844	0.806	0.901	0.641	0.799	0.779	0.846	0.759	0.673	0.745	0.658	0.461	0.669
25	1.084	1.033	0.835	0.79	0.863	0.88	1.12	0.821	0.969	0.988	0.976	0.7	0.939	0.947	0.952	1.084	0.761	1.001	0.92	0.999	0.961	0.851	0.877	0.799	0.615	0.767

CROSS VALIDATION OF OUTPUT

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
3	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1
4	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
6	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
8	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0
10	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
13	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0

$\approx$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1.19	1.09	0.93	0.85	0.94	1	1.23	0.89	1.04	1.12	1.02	0.8	1.04	1.09	1.06	1.2
1	1.1	1.08	0.83	0.81	0.88	0.86	1.14	0.84	1	0.97	1.03	0.68	0.94	0.92	0.95	1.09
2	1	0.64	0.91	0.62	0.74	1.08	1.02	0.66	0.73	1.16	0.51	0.91	0.95	1.23	1	1.05
3	0.96	1.01	0.7	0.73	0.78	0.7	1	0.76	0.91	0.8	0.99	0.54	0.81	0.73	0.81	0.95
4	1.16	1.05	0.92	0.82	0.91	0.99	1.19	0.86	1	1.1	0.97	0.8	1.02	1.07	1.04	1.17
5	0.92	0.62	0.83	0.58	0.69	0.97	0.94	0.62	0.69	1.05	0.5	0.82	0.87	1.1	0.92	0.97
6	0.95	0.86	0.74	0.68	0.75	0.8	0.98	0.7	0.83	0.89	0.81	0.64	0.83	0.87	0.84	0.95
7	0.97	1.03	0.71	0.74	0.79	0.71	1.01	0.77	0.92	0.81	1	0.55	0.82	0.74	0.82	0.96
8	1.04	0.97	0.82	0.75	0.83	0.87	1.08	0.78	0.92	0.97	0.9	0.7	0.91	0.94	0.93	1.05
9	1.05	0.85	0.87	0.72	0.81	0.98	1.08	0.75	0.86	1.08	0.76	0.8	0.95	1.08	0.98	1.08
10	0.84	0.76	0.67	0.6	0.66	0.72	0.87	0.63	0.73	0.81	0.7	0.58	0.74	0.79	0.76	0.85
11	0.79	0.61	0.68	0.53	0.61	0.77	0.81	0.56	0.63	0.84	0.53	0.64	0.73	0.86	0.76	0.82
12	0.95	0.76	0.8	0.64	0.73	0.9	0.98	0.67	0.77	0.99	0.67	0.74	0.87	1	0.9	0.98
13	1.07	1.06	0.81	0.79	0.86	0.84	1.11	0.82	0.98	0.95	1.01	0.66	0.92	0.9	0.93	1.07
14	1	0.8	0.84	0.67	0.77	0.94	1.02	0.71	0.81	1.04	0.7	0.78	0.91	1.05	0.94	1.03
15	0.97	0.85	0.78	0.68	0.76	0.85	1	0.71	0.83	0.94	0.78	0.69	0.86	0.93	0.88	0.98

ANALYSIS OF RESULTS

- Positive un-labelled learning.
- $O(n^2)$ hidden variable to estimate.
- Mean Square vs Precision-recall.

	Data Volume	Time Taken
Training and test	500 * 500	61.11
	5k * 5k	~ 52 minutes

FUTURE WORK and CONCLUSION

- Visualize the new network using Gephi
- RoC plotting.
- Compare with other prediction methods.
- Extensible to any recommendation system.
- Use labelled dataset.
- Transfer knowledge to a target network.
- Parallelize to work with massive dataset.
- GitHub:

https://github.com/moicha/Inferring-Ties-Between-Disconnected-Nodes

TEXAS TECH UNIVERSITY™

THANK YOU!