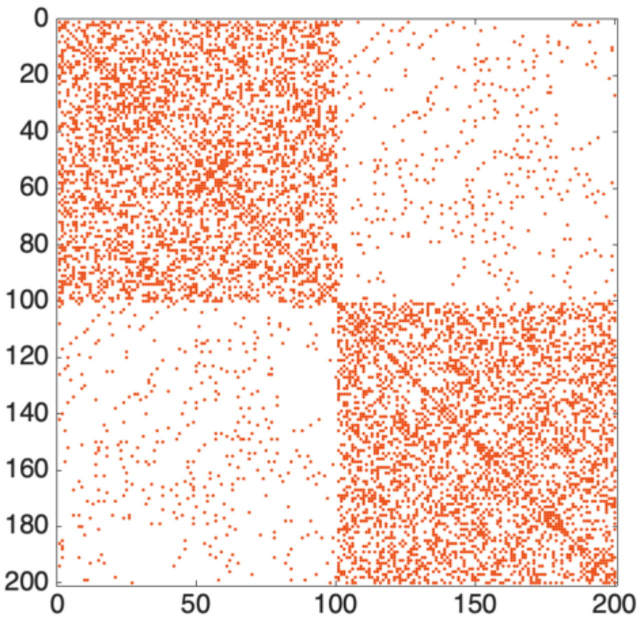
Selected Topics in Frontiers of Statistics

12111620 DingYixuan

Use Maximum Likelihood method to detect the community structure of the network:



from the network, we can observe that the network could be partitioned into two communities. From the requirement of the problems, the average degree inside community is 8, and the average degree between communities is 2.

Thus we can use networkx to construct graph:

First we can create each community:

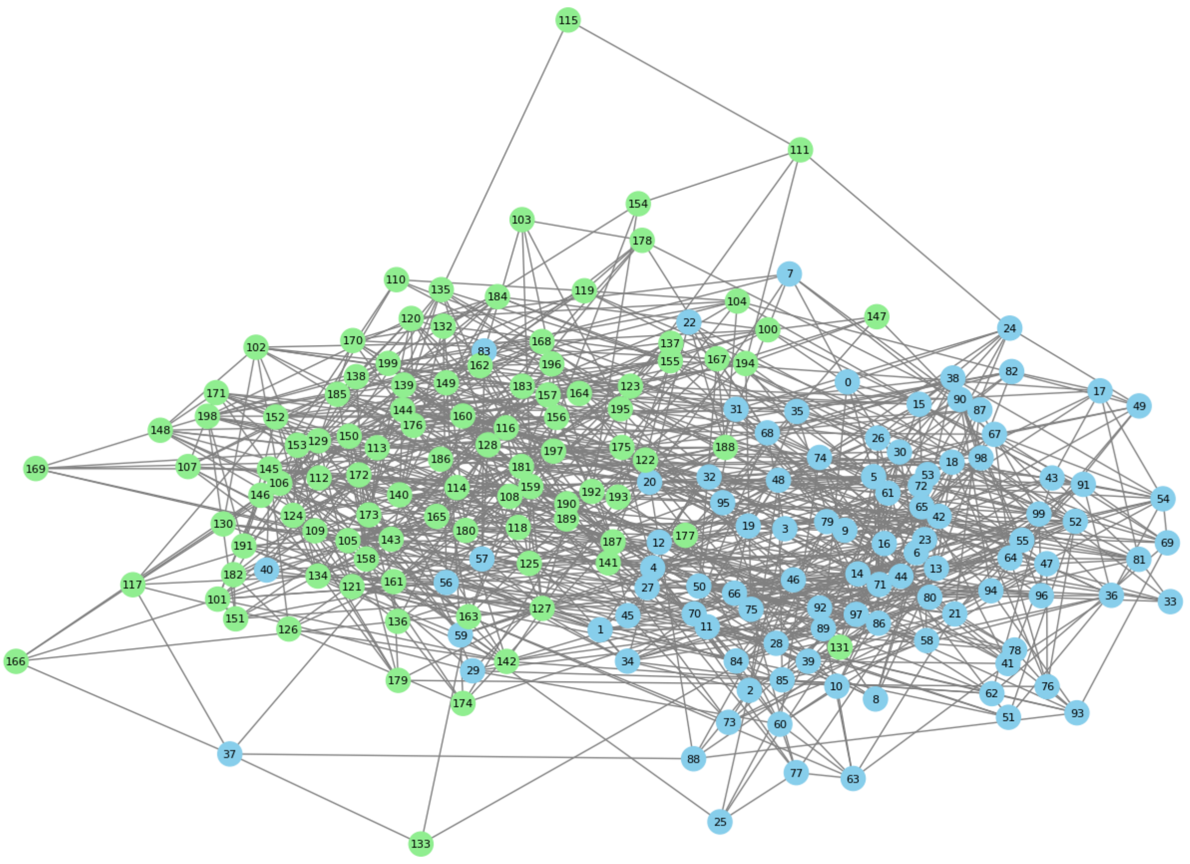
```
def create_community_graph(n_nodes, avg_degree):
    p = avg_degree / (n_nodes - 1)
    return nx.erdos_renyi_graph(n_nodes, p)
```

Then we can link the community:

```
def connect_communities(G1, G2, avg_inter_degree):
    n1 = len(G1)
    n2 = len(G2)
    n_inter_edges = int(avg_inter_degree * (n1 + n2) / 2)

    G = nx.disjoint_union(G1, G2)

    nodes_community1 = list(G1.nodes)
    nodes_community2 = list(G2.nodes)

    while n_inter_edges > 0:
        node1 = np.random.choice(nodes_community1)
        node2 = np.random.choice(nodes_community2) + n1
        if not G.has_edge(node1, node2):
            G.add_edge(node1, node2)
            n_inter_edges -= 1

    return G
```

Also, from the picture we can know that number of nodes in each community is 100.

Below is the generated network:



Before we actually implement coding, we should specify some statistics:

$$q_{ir} = \frac{\pi_r \prod_j \theta_{rj}^{A_{ij}}}{\sum_s \pi_s \prod_j \theta_{sj}^{A_{ij}}}$$

which denotes the probability that node i lies in class r.

$$\pi_r = \frac{1}{n} \sum_i q_{ir}$$

which denotes the probability that randomly selecting a notes lying in class r.

$$\theta_{rj} = \frac{\sum_i A_{ij} q_{ir}}{\sum_i k_i q_{ir}}$$

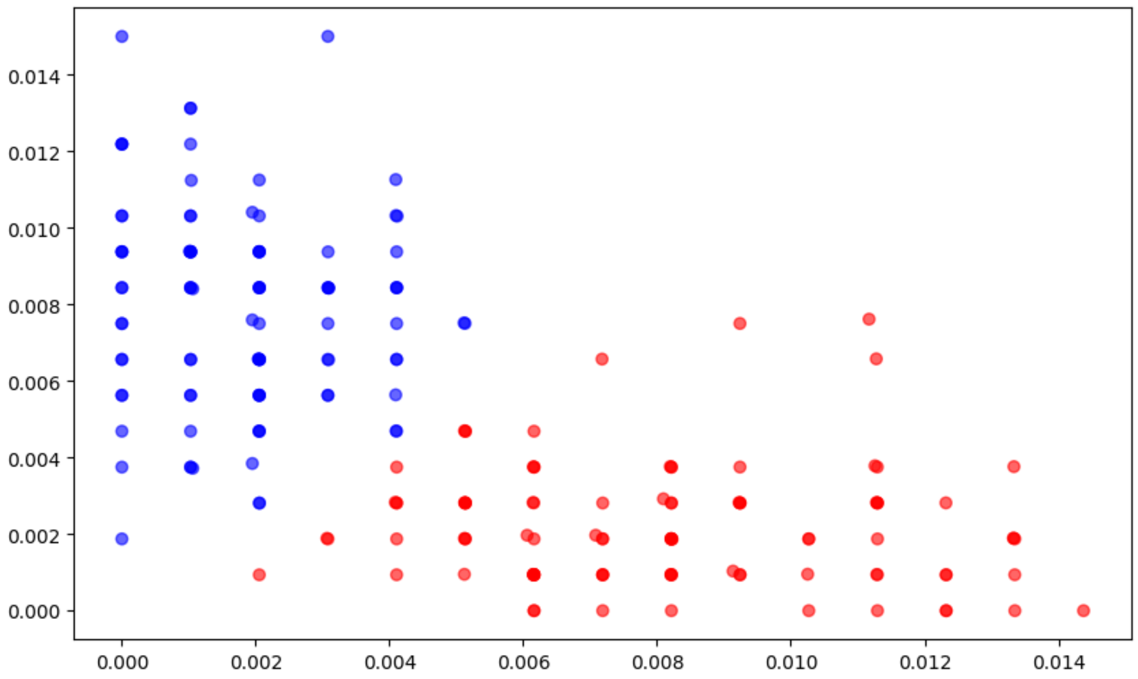
which denotes the probability that a link from a particular vertex in group r connects to vertex i.

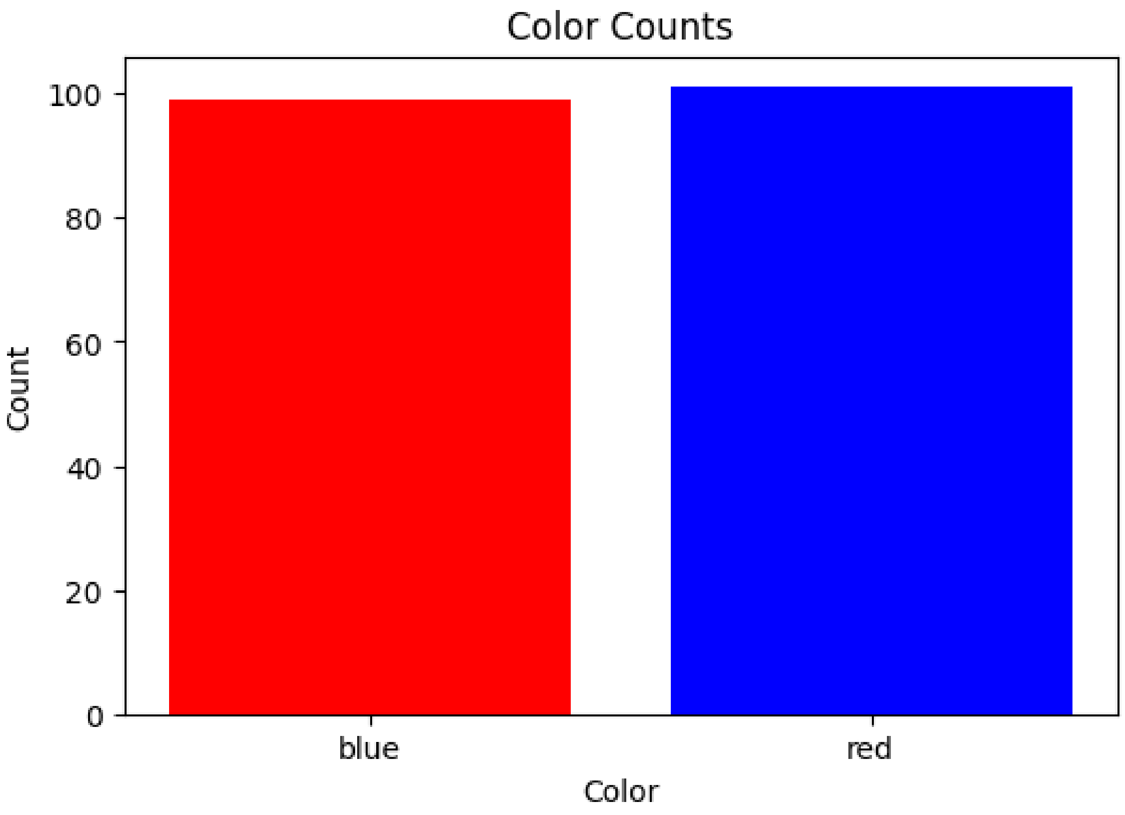
Note that the probability q_{ir} can be solved iteratively.

OK now, we can implement coding, specific annotation is included:

```
def iterate_likelihood(A, k, c, n, epsilon=1e-6, max_iter=1000):
    """
    Iterate the likelihood algorithm given the input matrices and parameters.

    Parameters:
        A (np.ndarray): The matrix A with shape (n, c).
        k (np.ndarray): The array k with length n.
        c (int): Number of columns in A (number of categories).
        n (int): Number of rows in A (number of samples).
        epsilon (float): Convergence threshold.
        max_iter (int): Maximum number of iterations.

    Returns:
        np.ndarray: The final values of π_r.
        np.ndarray: The final values of θ_rj.
    """
    # Initialize π and θ with small random perturbations around the symmetric choice
    np.random.seed(42)
    pi = np.random.rand(c)
    pi = pi / pi.sum()
    theta = np.random.rand(c, n)
    theta = theta / theta.sum(axis=1, keepdims=True)
    # print(pi)

    # Compute q_ir
    q = np.zeros((n, c))

    for iteration in range(max_iter):
        # print(iteration)
        # Save old values for convergence check
        pi_old = pi.copy()
        theta_old = theta.copy()

        for i in range(n):
            for r in range(c):
                numerator = pi[r] * np.prod(theta[r, :] ** A[i, :])
                denominator = sum(pi[s] * np.prod(theta[s, :] ** A[i, :]) for s in range
                q[i, r] = numerator / denominator

        # Update π_r and θ_rj
        pi = np.sum(q, axis=0) / n
        for r in range(c):
            for j in range(n):
                theta[r, j] = np.sum(A[:, j] * q[:, r]) / np.sum(k * q[:, r])

        # Check for convergence
        if np.allclose(pi, pi_old, atol=epsilon) and np.allclose(theta, theta_old, atol=
            break

    return pi, theta
```

It is worth noticing that the derived q_{ir} may not be the form of probability. Thus we should take softmax for two class, however, we can still observe the community detection performance by visualization below:





It shows that the community has been detected and two groups of nodes are separated.