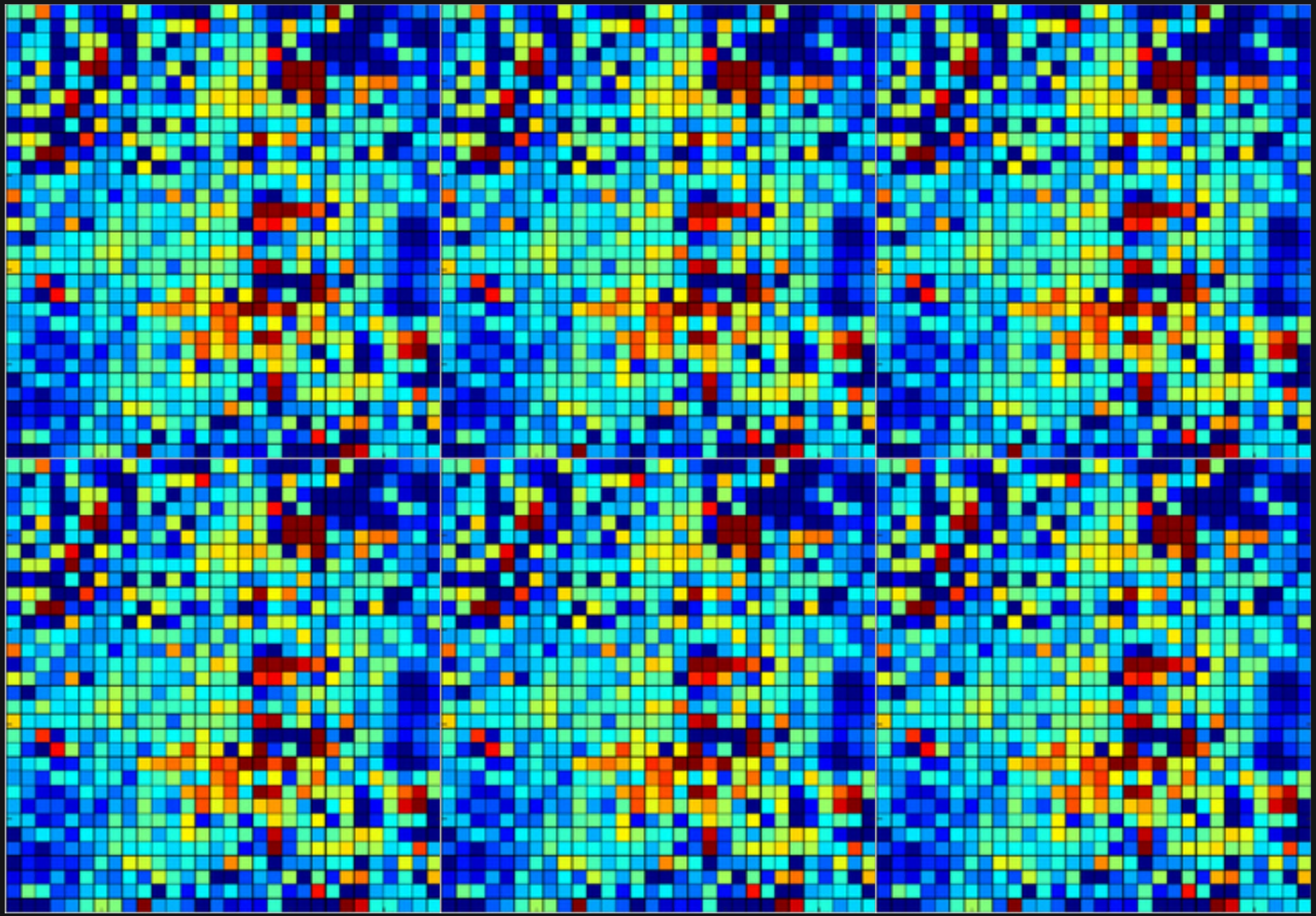
Geographical-based Friendship Network

Assignment 1d

Probability & Statistics SA 2023-2024

By Yuliya Ten



Inhabitant Map

1.Purely Random Map

1000 individuals within a unit square, [0, 1]^2

Each individual is represented by coordinates (x, y)

Inhabitant Map

2.Geographical based Random Map

Circular deserts

Elliptical mountain ranges

Each individual in this case is characterized by coordinates (x, y)

f (x, y) = geographical feature f characterizing (x, y)

Winding rivers

Rectangular cities

1.Friendship Network

a)nf (number of features = 8 (2 examples of each feature)

Feature 1 (Desert1): 9.2

Feature 2 (Desert2): 8.8

Feature 3 (Mountain Range1): 8.7

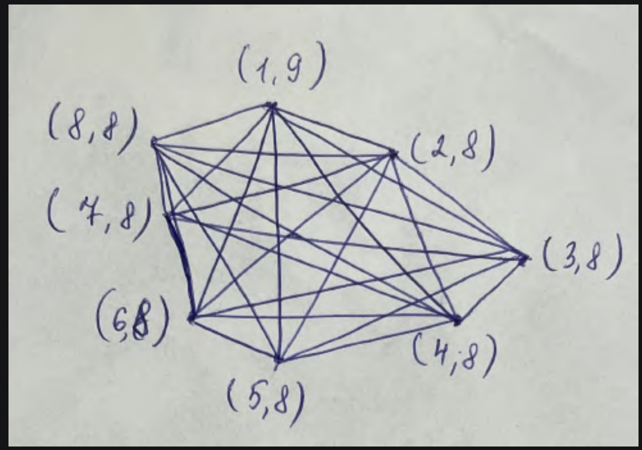
Feature 4 (Mountain Range2): 8.5

Feature 5 (Winding River1): 8.4

Feature 6 (Winding River2): 8.6

Feature 7 (City1): 8.3

Feature 8 (City2): 8.9



Erdos–Rényi model

$$P[\text{random person settles in } (x,y)] = \frac{s_{f(x,y)}}{\sum_{(x',y')\in \text{Map}} s_{f(x',y')}}$$ = 0.11

$$P(A - B) = \exp\{-\beta * \text{dist}[(x_A, y_A), (x_B, y_B)]\}$$ = 0.1

Try Pitch

Scenario 1

```
#Scenario 1
i)def SamplingIndividuals(n):
    points = []
    for i from 1 to n:
        x = random value between 0 and 1
        y = random value between 0 and 1
        add (x, y) to points
    return points

ii)def euclDistMatrix(individuals):
    n = number of individuals
    distance = n x n
    for i from 1 to n:
        for j from 1 to n:
            if i == j:
                distance[i][j] = 0
            else:
                distance[i][j] = euclidean_distance(individuals[i], individuals[j])
    return distance

 ------  def euclidean_distance(individuals[i], individuals[j]): ------
        dx = individuals[i]
        dy = individuals[j]
        distance = sqrt( (xi-xj)^2 + (yi-yj)^2 )
        return distance
 iii)def distMatrixFromN(n):
     for iteration from 1 to 100:
         individuals = SampleIndividuals(n)
         distance_matrix = euclDistMatrix(individuals)
         summary_info = compute_summary(distance_matrix)
         for measure in summary_measures:
             summary_measures[measure].append(summary_info[measure])
     return summary_measures

 ----- def compute_summary(distance):  -----
       min, max, average, total = 0
       for i from 1 to n:
         for j from i+1 to n:
           distance = distance[i][j]
           average = total / n
           if distance < min:
             min = distance
           if distance > max:
             max = distance


 iv)def erdosRenyiFromP(n, p):
     for i from 1 to n:
         for j from i+1 to n:
             if (random value between 0 and 1) < p:
                 add an edge between node i and node j in graph
     return graph
```

c) i) In my opinion, ErdosRenyiFromP model can be realistic for Friendship Network, because If look at this globally, people's communication in real life or online looks like a threads, which are connected and develop, and m*ake* a new nodes and edges from it.

c) ii)

Disadvantages:

- certain limit
- shortage of reality

Advantages:

- simple representation
- opportunity to count and analyse

Scenario2

```
#Scenario 2
i)def geographicalFeatureMap(n):
    map = nxn
    circularDesert(map)
    ellipsoidalMountain(map)
    sinusoidalRiver(map)
    rectangularCity(map, 3)
    return map

ii)def geographicalFeatureProbability(map):
    probability_value = nxn
    for each point in map:
        probability_value[point] = probability(point)
    return value

iii)def geographicalSample(map, n):
    individuals = n
    probability = geographicalFeatureProbability(map)
    for i from 1 to n:
        individual = mapIndividual(probability)
        individuals.append(individual)
    return individuals

iv)def euclDistMatrix(individuals):
    matrix_distance = nxn
    for i from 1 to n:
        for j from 1 to n:
            matrix_distance[i][j] = distance(individuals[i], individuals[j])
    return matrix_distance
```

```
v)def adjMatrixFromDist(matrix_distance, beta):
    adjacencyMatrix = nxn
    for A from 1 to n:
        for B from 1 to n:
            u = uniform[0, 1]
            probability = probability_A_B(matrix_distance[A][B], beta)
            if u < probability:
                adjacencyMatrix[A][B] = 1
            else:
                adjacencyMatrix[A][B] = 0
    return adjacencyMatrix

vi)def degreeFromAdjMatrix(adjacencyMatrix):
    total = 0
    for i from 1 to n:
        total = sum(adjacencyMatrix[i])
        totalDegree += degree
    meanDegree = total / n
    return meanDegree
```

c)Relationship between the geographical features and the groupings of friends in the network: probabilities values of individuals on the grid, which based on different factors, comparison.

It looks more realistic because based on individuals's values, limited applicability and show on the graphs all probabilities of forming friendship network.

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)