1. 用有向图展示用户之间的关联关系

为了直观的展现用户之间的关联, 我们用有向图来描述这种关系.

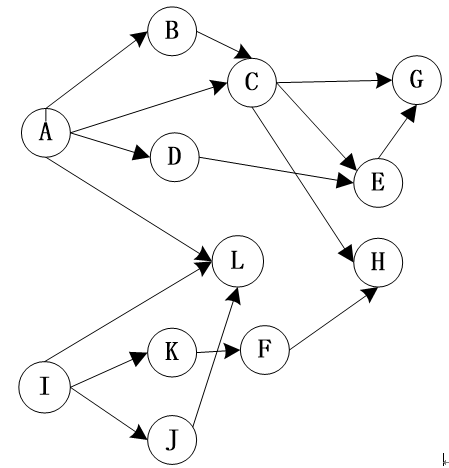
我们用有向图中的节点来代表用户实体, 用有向图中的边来代表用户之间的相互关注, 这样, 用户之间的关联关系就表示成了一个巨大的有向图(见图4.4.1.1), 我们下面的分析也是基于这个有向图来做的.

图4.4.1.1

在这里, 用户之间的距离是指在由用户节点和关注边所构成的有向图中, 某个用户节点与另外一个用户节点之间的最短距离.

1. 计算用户的关联性

在代表用户关联的有向图中, 我们通过floyd算法来计算任意两点之间的最短距离

1. 矩阵的存储

由于矩阵比较大, 存储在数据库中不方便进行运算, 运算时临时生成又比较耗时, 因此我们把矩阵序列化之后以JSON的形式存储在文件中, 运算的时候从文件中读入并且进行反序列化以及初始化.

1. 矩阵的运算

从数据库中根据用户之间相互关注的信息, 生成表示有向图的矩阵, 下面所进行的运算都是基于该矩阵来进行的.

首先生成有向图的节点, 该过程中对有向图的边进行遍历是要去除重复的节点.

const mongodb = require('../mongodb')

const fs = require('fs')

const path = require('path')

const $conn = mongodb.$conn

;(async function()

{

const $db = await $conn

const $follow\_sex = $db.collection('follow\_sex')

const idSet = new Set()

const cursor = $follow\_sex.find().limit(1 \* 10000)

cursor.batchSize(1000)

let n = 0

while(await cursor.hasNext())

{

const e = await cursor.next()

idSet.add(e.from.userId)

idSet.add(e.to.userId)

n ++

console.log(n)

}

console.log(idSet.size)

const idList = []

let index = 0

idSet.forEach(async function(e)

{

const item =

{

index: index,

userId: e

}

idList.push(item)

index ++

console.log(index)

})

// await $vertex.insert(idList)

fs.writeFileSync(path.resolve(\_\_dirname, 'vertex.json'), JSON.stringify(idList))

})()

然后对节点进行处理, 便于编号和索引, 方便下面所要进行的运算

const fs = require('fs')

const path = require('path')

const vertex = JSON.parse(fs.readFileSync('vertex.json').toString())

const list = []

vertex.forEach(function(e)

{

console.log(e.index)

list.push(e.userId)

})

console.log(list)

fs.writeFileSync('vertex.list.json', JSON.stringify(list))

然后遍历有向图的边来生成有向图的矩阵, 代码如下:

const mongodb = require('../mongodb')

const fs = require('fs')

const $conn = mongodb.$conn

const vertex = JSON.parse(fs.readFileSync('vertex.list.json').toString())

const Inf = Infinity

const findIndex = function(userId)

{

return vertex.indexOf(userId)

}

const createMatrix = function(n, init = Inf)

{

const result = []

for(let i = 0; i < n; i ++)

{

const row = []

for(let i = 0; i < n; i ++)

{

row.push(init)

}

result.push(row)

}

return result

}

;(async function()

{

const $db = await $conn

const $follow\_sex = $db.collection('follow\_sex')

const cursor = $follow\_sex.find().limit(1 \* 10000)

cursor.batchSize(1000)

const matrix = createMatrix(vertex.length)

while(await cursor.hasNext())

{

const e = await cursor.next()

const row = findIndex(e.from.userId)

const col = findIndex(e.to.userId)

console.log(`[${row}][${col}] 1`)

try

{

matrix[row][col] = 1

}

catch(e)

{

}

}

console.log(matrix)

fs.writeFileSync('matrix.json', JSON.stringify(matrix))

})()

最后根据生面生成的有向图来运用floyd算法计算出距离矩阵和路径矩阵, 代码如下:

const Inf = Infinity

const fs = require('fs')

const matrix = JSON.parse(fs.readFileSync('matrix.json').toString())

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] === null)

{

matrix[row][col] = Inf

}

}

}

const numberOfNotInf = function(matrix)

{

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] !== Inf)

{

n ++

}

}

}

return n

}

console.log(numberOfNotInf(matrix))

// console.log(matrix)

const createSameSize = function(matrix, init = 0)

{

const result = []

for(let i = 0; i < matrix.length; i ++)

{

const row = []

for(let i = 0; i < matrix.length; i ++)

{

row.push(init)

}

result.push(row)

}

return result

}

// console.log(createSameSize(matrix))

const floyd = function(matrix)

{

const dist = createSameSize(matrix, Inf)

const path = createSameSize(matrix, Inf)

for (let i = 0; i < matrix.length; i ++)

{

for (let j = 0; j < matrix.length; j ++)

{

dist[i][j] = matrix[i][j]

path[i][j] = j // 顶点i"到"顶点j"的最短路径是经过顶点j, path[i][j] = i 也是一样的

}

}

for (let i = 0; i < matrix.length; i ++) // 循环遍历每个节点

{

console.log(`i: ${i}`)

for (let j = 0; j < matrix.length; j ++) // 每一行

{

for (let k = 0; k < matrix.length; k ++) // 每一列

{

const temp = (dist[i][k] === Inf || dist[k][j] === Inf) ? Inf : dist[i][k] + dist[k][j]

if(dist[i][j] > temp)

{

dist[i][j] = temp

path[i][j] = k

}

}

}

}

return [dist, path]

}

const result = floyd(matrix)

fs.writeFileSync('result.json', JSON.stringify(result))

console.log(numberOfNotInf(result[0]))

console.log(numberOfNotInf(result[1]))

至此, 有向图的运算已经完成, 我们下面的分析都是基于该运算结果来进行的.

1. 对矩阵的运算结果进行分析

在前面矩阵的运算结果的基础上, 对矩阵进行分析.

代码如下:

const fs = require('fs')

const result = JSON.parse(fs.readFileSync('result.json').toString())

const dist = result[0]

const Inf = Infinity

for(let row = 0; row < dist.length; row ++)

{

for(let col = 0; col < dist.length; col ++)

{

if(dist[row][col] === null)

{

dist[row][col] = Inf

}

}

}

const numberOfNotInf = function(matrix)

{

let n = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] !== Inf)

{

n ++

}

}

}

return n

}

const maxValueOfMatrix = function(matrix)

{

let max = 0

for(let row = 0; row < matrix.length; row ++)

{

for(let col = 0; col < matrix.length; col ++)

{

if(matrix[row][col] > max && matrix[row][col] !== Inf)

{

max = matrix[row][col]

}

}

}

return max

}

console.log(dist.length)

const nxn = dist.length \* dist.length

const noni = numberOfNotInf(dist)

const max = maxValueOfMatrix(dist)

console.log(nxn, noni, max)

有向图共有7586个节点, 10000条有向边

有7586 \* 7586 = 57547396对节点

可以联通的节点有2457173对

因此任意一个节点到另外一个节点有路径的概率为2457173 / 57547396 = 4.3%

最长的路径长度为7

由此可以看出网易云音乐用户之间的关联度极高.