The diffusr tutorial

Simon Dirmeier

2016-11-27

Introduction

`diffusr` implements algorithms for network diffusion such as *Markov random walks with restarts* and *weighted neighbor classification*. Network diffusion has been studied extensively in bioinformatics, e.g. in the field of cancer gene prioritization. Network diffusion algorithms generally spread information in the form of node weights along the edges of a graph to other nodes. These weights can for example be interpreted as temperature, an initial amount of water, the activation of neurons in the brain, or the location of a random surfer in the internet. The information (node weights) is iteratively propagated to other nodes until a equilibrium state or stop criterion occurs.

Tutorial

First load the package:

```
library(diffusr)
```

Markov random walks

A *Markov random walk with restarts* calculates the stationary distribution:

$$\mathbf{p}_{t+1} = (1 - r)\mathbf{W}\mathbf{p}_t + r\mathbf{p}_0, \tag{1}$$

where $r \in (0, 1)$ is a *restart probability* of the Markov chain, $\mathbf{W}$ is a column-normalized stochastic matrix (we do the normalization for you) and $\mathbf{p}_0$ is the starting distribution of the Markov chain. We calculate the iterative updates, it is also possible to do the math using the nullspace of the matrix (comes later).

If you want to use Markov random walks just try something like this:

```
# count of nodes
n <- 5
# starting distribution (has to sum to one)
p0    <- as.vector(rmultinom(1, 1, prob=rep(.2, n)))
# adjacency matrix (either normalized or not)
graph <- matrix(abs(rnorm(n*n)), n, n)
# computation of stationary distribution
pt    <- random.walk(p0, graph)
```

The stationary distribution should have changed quite a bit from the starting distribution:

```
print(t(p0))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    1
```

```
print(t(pt))
```

```
##           [,1]       [,2]       [,3]      [,4]      [,5]
## [1,] 0.1421462 0.02882297 0.08152183 0.1744001 0.5731089
```

Nearest neighbors

Diffusion using *nearest neighbors* is done by traversing through a (weighted) graph and take all the neighbors of a node until a certain depths in the graph is reached. We find shortest paths using priority queues:

```
  # count of nodes
  n <- 10
  # indexes(integer) of nodes for which neighbors should be searched
  node.idxs <- c(1L, 5L)
  # the adjaceny matrix (does not need to be symmetric)
  graph <- rbind(cbind(0, diag(n-1)), 0)
  # compute the neighbors until depth 3
  neighs <- nearest.neighbors(node.idxs, graph, 3)
```

Let's see what which nodes we got:

```
  print(neighs)
```

```
## $`1`
## [1] 2 3 4
##
## $`5`
## [1] 6 7 8
```

Insulated heat diffusion

An *insulated heat diffusion process* calculates the stationary distribution:

$$\mathbf{h}_{t+1} = (r(\mathbf{I} - (1-r)\mathbf{W})^{-1})\mathbf{h}_t \tag{2}$$

where $r \in (0, 1)$ is a rate controlling the diffusion process, $\mathbf{W}$ is a column-normalized stochastic matrix (we do the normalization for you) and $\mathbf{h}_t$ is the current heat distribution.

If you want to use the heat diffusion use:

```
  # count of nodes
  n <- 5
  # starting distribution (has to sum to one)
  h0    <- as.vector(rmultinom(1, 1, prob=rep(.2, n)))
  # adjacency matrix (either normalized or not)
  graph <- matrix(abs(rnorm(n*n)), n, n)
  # computation of stationary distribution
  ht    <- insulated.heat.diffusion(h0, graph)
```

The result of the heat diffusion looks like this:

```
  print(t(h0))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
```

```
  print(t(ht))
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2429112 0.2063345 0.1860879 0.1160998 0.2485666
```

Heat diffusions and Markov random walks are related to each other and should yield similar results.

Laplacian heat diffusion

A *Laplacian heat diffusion process* calculates the heat distribution over a graph after at a specific time t:

$$h_i(t) = h_i(0) \exp(-\lambda_i t) \tag{3}$$

where $\mathbf{h}_0$ is the initial heat distribution, $\mathbf{h}_t$ is the heat distribution at time t and $\boldsymbol{\lambda}$ are the eigenvalues of the *Laplacian* of your graph.

You can use the *Laplacian heat diffusion process* like this:

```
  # count of nodes
  n <- 5
  # starting distribution (has to sum to one)
  h0    <- as.vector(rmultinom(1, 1, prob=rep(.2, n)))
  # adjacency matrix (either normalized or not)
  graph <- matrix(abs(rnorm(n*n)), n, n)
  # computation of stationary distribution
  ht    <- laplacian.heat.diffusion(h0, graph)
```

Here are the results:

```
  print(t(h0))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    1
```

```
  print(t(ht))
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.03774252 0.1640692 0.1243287 0.1756897 0.6586633
```