PAFit: Nonparametric Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks

Package Vignette

Thong Pham, Paul Sheridan and Hidetoshi Shimodaira

July 9, 2016

1 Introduction

This tutorial demonstrates the use of the `R` package `PAFit` which implements the PAFit method in Refs. 1,2. This method estimates the preferential attachment (PA) function A_k in isolation [1], or estimate the PA function A_k and node fitness η_i jointly [2,3]. `PAFit` are written mainly in `C++` by using the package `Rcpp` [4,5]. It employs OpenMP for simple parallel processing. This makes the package applicable to large datasets. If you use this package in your projects, please run `citation("PAFit")` for citation information.

First we introduce the underlying generative network model in Section 2. In Section 3 we show how to estimate the PA function in isolation. The joint estimation of PA and node fitness is discussed in Section 4. Section 5 discusses some miscellaneous utilities in `PAFit`, including how to generate a wide range of temporal networks.

2 The General Temporal model

In PAFit, the growth process of the network is assumed to follow the General Temporal (GT) model, which is a generative network model for both undirected and directed networks. Here we only show the definition of the directed version. More details can be found in Ref. [3]. In the directed GT model, a node v_i with degree k and fitness η_i receives a new edge with probability proportional to the product of A_k and η_i:

$$P(\text{node } v_i \text{ receives a new edge}) \propto A_k \times \eta_i, \tag{1}$$

This model includes a wide range of existing generative network models as special cases (Tab. 1). Note that both PA function A_k and node fitness η_i are assumed to be time-invariant.

Generative Network Model	**PA Function**	**Fitness**	**Reference**
GT model	Free	Free	Pham *et al.* [1,2]
Callaway *et al.* (i.e. ER model with growth)	$A_k = 1$	$\eta_i = 1$	Callaway *et al.* [6]
BA model	$A_k = k$	$\eta_i = 1$	Barabási and Albert [7]
Extended BA model	$A_k = k^\alpha$	$\eta_i = 1$	Krapivsky *et al.* [8]
Krapivsky *et al.*	Free	$\eta_i = 1$	Krapivsky *et al.* [9]
Caldarelli model	$A_k = 1$	Free	Caldarelli *et al.* [10]
BB model	$A_k = k$	Free	Bianconi and Barabási [11]
Extended BB model	$A_k = k^\alpha$	Free	Not previously considered.

Table 1: **Some existing network models that are included as special cases of the GT model.**

3 Estimating the PA function in isolation

Estimating the PA function in isolation, i.e. assuming that $\eta_i = 1$ for all i, is an important problem in its own right [2]. Here we show how to use `PAFit` to estimate the PA function from a simulated network. The network is contained in the file "`data1.txt`" which can be found in the folder `inst\simdata` of the package.

First we read the file into the `R` environment:

```
library("PAFit")
data1 <- read.table(system.file("simdata",file = "data1.txt",package ="PAFit"))
```

The format of the data is a matrix where each row contains information of one edge in the form of `(from_node, to_node, time_stamp)`. `from_node` and `to_node` are the ids of the source node and destination node, respectively. `time_stamp` is the arrival time of the node. It is assumed that both ids are `integer` starting from 0. `time_stamp` can be either `numeric` or `string`. The only assumption is that a smaller `time_stamp` represents an earlier arrival time.

The simulated network has total number of nodes $N = 1000$, and the number of new edges at each time-step $m = 5$. The true PA function used is $A_k = k$.

We then use the function `GetStatistics` to get all summary statistics needed in estimation of A_k:

```
stats1  <- GetStatistics(data1, only_PA = TRUE)
```

One can explore `stats1` to view various summary statistics. Note that the option `only_PA = TRUE` will ignore statistics that not needed for the estimation of PA in isolation. This will help us save a lot of memory when the network is big.

Next we can estimate the PA function by:

```
result1 <- PAFit(stats1, only_PA = TRUE)
```

We can access the estimated attachment function $\hat{A}_k$ via `result$k` and `result$A`. Note that `PAFit` can also estimate the confidence intervals of $\hat{A}_k$. One can access the upper ends and the lower ends of these confidence intervals via `result$upper_A` and `result$lower_A`. These confidence intervals are calculated as two standard deviations from the estimated $\hat{A}_k$. The variances of $\hat{A}_k$ (square of standard deviations) are stored in `result$var_A`.

One can plot the estimated PA function together with the true function:

```
plot(result1,stats1)
alpha  <- 1
true_A <- pmax(result1$center_k^alpha,1)
lines(result1$center_k + 1,true_A, col = "red", lwd = 2)
```

Note that in the case of power-law PA function $A_k = k^\alpha$ like in this example, one can also access the estimated α via `result1$alpha`.

4 Joint estimation of the attachment function and node fitness

Here we show how to estimate the PA function and node fitness simultaneously from a simulated network. The network is stored in `data2.txt`, while the true node fitnesses are stored in `true-fitness.txt` in the folder `inst\simdata`. As described in Ref. 3, `PAFit` requires the setting of the regularization parameter r of A_k and the regularization parameter s of η_i.

4.1 When regularization parameters are known

Here we assume that we know the regularization parameters for `PAFit`. Since the simulated network is generated with a non-log-linear PA function $A_k = 3(\log \max(k, 1))^3 + 1$, we set a small value 0.1 for r. Node fitnesses are sampled from a gamma distribution with mean 1 and variance $1/s^* = 1/5$. Hence we set the regularization parameter $s = s^* = 5$. The following scripts estimate PA and node fitness simultaneously:

```
library("PAFit")
data2   <- read.table(system.file("simdata",file = "data2.txt",package ="PAFit"))
stats2  <- GetStatistics(data2, G = 50)
result2 <- PAFit(stats2, r = 0.1, s = 5)
```

We can plot the estimated attachment function and node fitnesses as follows.

```
plot(result2, stats2, plot = "A")
alpha <- 3
beta  <- 3
true_A <- sapply(result2$center_k,function(x) alpha*(log(max(x,1)))^beta + 1)
lines(result2$center_k + 1,true_A,lwd = 2, col = "red")
# User needs to open a new plotting device here
# loading the true fitnesses
true_fitness <- as.vector(as.matrix(read.table(system.file("simdata",file = "true-fitness.txt",
package ="PAFit"))))
plot(result2, stats2, true = true_fitness, plot = "true_f", high_deg = 5)
```

4.2 When regularization parameters are unknown

In real-world situations, we do not know the optimal pairs of r and s, so we have to perform a cross-validation(CV)-like approach to choose r and s [3].

First we create the data for CV:

```
library("PAFit")
data2   <- read.table(system.file("simdata",file = "data2.txt",package ="PAFit"))
CV_data <- CreateDataCV(data2)
```

Then we set the grid of (r, s) and perform CV:

```
r  <- c(0.01,0.05,0.1,0.2,0.5,1)
s  <- c(0.1,2,3,4,5,6,7,8)
#### This could take a while ########

CV_result <- performCV(CV_data, r = r, s = s, only_PAFit = TRUE)

#####################################
#opimal s
CV_result$s_optimal
#optimal_r
CV_result$r_optimal
```

Finally after finding the optimum pair of (r, s), we use this pair with the full data to obtain the final estimation of PA and fitness.

```
stats2  <- GetStatistics(data2, G = 50)
result3 <- PAFit(stats2,r = CV_result$r_optimal, s = CV_result$s_optimal)
plot(result3, stats2, plot = "A", high_deg = 1)
alpha <- 3
beta  <- 3
true_A <- sapply(result3$center_k[-1],function(x) alpha*(log(max(x,1)))^beta + 1)
lines(result3$center_k[-1] + 1,true_A,lwd = 2, col = "red")
# User needs to open a new plotting device here
# loading the true fitnesses
true_fitness <- as.vector(as.matrix(read.table(system.file("simdata",file = "true-fitness.txt",
package ="PAFit"))))
plot(result3, stats2, true = true_fitness, plot = "true_f", high_deg = 5)
```

5 Miscellaneous

5.1 Generating simulated networks

`PAFit` includes the function `GenerateNet` to generate networks from many important network models (Table 1). For example, the following script generates a network in which $A_k = k$, $\eta_i \sim Gamma(1,1)$, total number of nodes $N = 1000$, and number of new edges introduced at each time step is $m = 5$:

```
#mode = 1: A_k = k^alpha with alpha = 1 , eta_i from Gamma(1,1)
data1 <- GenerateNet(N = 1000, m = 5, alpha = 1, shape = 1, rate = 1,mode = 1)
```

The object `data1` is a list with components `data1$graph` and `data1$fitness`. `data1$graph` is a 3-column matrix where information about the edges is stored in each row. `data1$fitness` stores the true fitness value of each node. One then can use `data1$graph` as the input of `GetStatistics`.

If either `shape` or `rate` is 0, then node fitness is fixed at 1:

```
#mode = 1: A_k = k^alpha with alpha = 1 , eta_i = 1
data1 <- GenerateNet(N = 1000, m = 5, alpha = 1, shape = 0,mode = 1)
```

One can also generate networks from the attachment function $A_k = \min(k, \mathrm{sat\_at})^\alpha$ with $\alpha = 1$ and $\mathrm{sat\_at} = 100$ by specifying `mode = 2`.

```
#mode = 2: A_k = min(k,sat_at)^alpha with alpha = 1, sat_at = 100; eta_i from Gamma(1,1)
data2 <- GenerateNet(N = 1000, m = 5, alpha = 1, sat_at = 100, shape = 1, rate = 1, mode = 2)
```

Finally, the following script generates a network where the attachment function is $A_k = \alpha\log^\beta(k) + 1$ with $\alpha = 3$ and $\beta = 2$.

```
#mode = 3: A_k = A_k = alpha*log^beta(k) + 1 with alpha = 3, beta = 2 ; eta_i from Gamma(1,1)
data3 <- GenerateNet(N = 1000, m = 5, alpha = 3,beta = 2, shape = 1, rate = 1,, mode = 3)
```

Instead of fixing the number of new edges at each step m at `m = 5`, it might be more realistic to let m be a Poisson random variable, whose realized value varies at each time-step. This can be archived by specifying `prob_m = TRUE`. In this case, if the option `increase` is `FALSE` then the mean of this Poisson distribution is fixed at `m`, otherwise the mean itself will grow with the current size of the network. In the latter case, if `log = TRUE`, the mean will grow logarithmically with the current size, otherwise it will grow linearly.

5.2 Binning

Binning is an important pre-processing step of grouping together the statistics of k into bins. It is very useful in stabilizing the estimation of the PA A_k. `PAFit` employs logarithmic binning. Binning is performed when the statistics are summarized by the function `GetStatistics`. We specify `Binning = TRUE` and then specify the number of bins `G`.

```
data         <- GenerateNet(N = 1000, m = 5, alpha = 1, shape = 0, mode = 1)

#no binning
stats_nobin  <- GetStatistics(data$graph, Binning = FALSE)
result_nobin <- PAFit(stats_nobin,only_PA = TRUE)

#Number of bins is G = 50
stats_bin    <- GetStatistics(data$graph, Binning = TRUE, G = 50)
result_bin   <- PAFit(stats_bin,only_PA = TRUE)

plot(result_nobin, stats_nobin)
plot(result_bin,stats_bin)
```

References

[1] Thong Pham, Paul Sheridan, and Hidetoshi Shimodaira. *Nonparametric Estimation of the Preferential Attachment Function in Complex Networks: Evidence of Deviations from Log Linearity*, pages 141–153. Springer International Publishing, Cham, 2016.

[2] Thong Pham, Paul Sheridan, and Hidetoshi Shimodaira. PAFit: A statistical method for measuring preferential attachment in temporal complex networks. *PLOS ONE*, (9):e0137796, 9 2015.

[3] Thong Pham, Paul Sheridan, and Hidetoshi Shimodaira. Joint estimation of preferential attachment and node fitness in growing complex networks (submitted). 2016.

[4] Dirk Eddelbuettel and Romain Francois. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8), 2011.

[5] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp.* Springer, New York, NY, USA, 2013.

[6] Duncan S. Callaway, John E. Hopcroft, Jon M. Kleinberg, M. E. J. Newman, and Steven H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64:041902, Sep 2001.

[7] R Albert and AL Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[8] P. L. Krapivsky, S. Redner, and F. Leyvraz. Connectivity of growing random networks. *Phys. Rev. Lett.*, 85:4629–4632, Nov 2000.

[9] PL Krapivsky, GJ Rodgers, and S Redner. Organization of growing networks. *Physical Review E*, page 066123, 2001.

[10] G. Caldarelli, A. Capocci, P. De Los Rios, and M. A. Muñoz. Scale-free networks from varying vertex intrinsic fitness. *Phys. Rev. Lett.*, 89:258702, Dec 2002.

[11] G Bianconni and AL Barabási. Competition and multiscaling in evolving networks. *Europhys. Lett.*, 54:436, 2001.

[12] T Opsahl and P Panzarasa. Clustering in weighted networks. *Social Networks*, 31:155–163, May 2009.

[13] H Zhou, D Alexander, and K Lange. A quasi-Newton acceleration for high-dimensional optimization algorithms. *Statistics and Computing*, 21:261–273, 2011.