Continuous Time Markov Chains

Sai Bhargav Yalamanchi, Giorgio Alfredo Spedicato

2015-08-24

The *markovchain* package provides functionality for continuous time Markov chains (CTMCs). This vignette aims to provide a brief mathematical introduction to the same as well as how to use the package functionality.

Mathematical introduction

CTMCs are a generalisation of discrete time Markov chains (DTMCs) in that we allow time to be continuous. We assume a finite state space S (for an infinite state space wouldn't fit in memory). We can think of CTMCs as Markov chains in which state transitions can happen at any time.

More formally, we would like our CTMCs to satisfy the following two properties

- The Markov property - let $F_{X(s)}$ denote the information about X upto time s. Let $j \in S$ and $s \leq t$. Then, $P(X(t) = j|F_{X(s)}) = P(X(t) = j|X(s))$
- Time homogenity - $P(X(t) = j|X(s) = k) = P(X(t - s) = j|X(0) = k)$

If both the above properties are satisfied, it is referred to as a time-homogeneous CTMC. If a transition occurs at time t, then $X(t)$ denotes the new state and $X(t) \neq X(t-)$.

Now, let $X(0) = x$ and let T_x be the time a transition occurs from this state. We are interested in the distribution of T_x. For $s, t \geq 0$, it can be shown that

$$P(T_x > s + t|T_x > s) = P(T_x > t)$$

This is the memory less property that only the exponential random variable exhibits. Therefore, this is the sought distribution, and each state $s \in S$ has an exponential holding parameter $\lambda(s)$. Since $\mathrm{E}T_x = \frac{1}{\lambda(x)}$, higher the rate $\lambda(x)$, smaller the expected time of transitioning out of the state x.

However, specifying this parameter alone for each state would only paint an incomplete picture of our CTMC. To see why, consider a state x that may transition to either state y or z. The holding parameter enables us to predict when a transition may occur if we start off in state x, but tells us nothing about which state will be next.

To this end, we also need transition probabilities associated with the process, defined as follows (for $y \neq x$) -

$$p_{xy} = P(X(T_s) = y|X(0) = x)$$

Note that $\sum_{y \neq x} p_{xy} = 1$. Let Q denote this transition matrix ($Q_{ij} = p_{ij}$). What is key here is that T_x and the state y are independent random variables. Let's define

$$\lambda(x, y) = \lambda(x)p_{xy}$$

We now look at Kolmogorov's backward equation. Let's define

$$P_{ij}(t) = P(X(t) = j|X(0) = i)$$

for $i, j \in S$. The backward equation is given by (it can be proved)

$$P_{ij}(t) = \delta_{ij}e^{-\lambda(i)t} + \int_0^t \lambda(i)e^{-\lambda(i)t} \sum_{k \neq i} Q_{ik}P_{kj}(t - s)ds$$

Basically, the first term is non-zero if and only if $i = j$ and represents the probability that the first transition from state i occurs after time t. This would mean that at t, the state is still i. The second term accounts for any transitions that may occur before time t and denotes the probability that at time t, when the smoke clears, we are in state j.

This equation can be represented compactly as follows

$$P'(t) = AP(t)$$

where A is the **generator** matrix.

$$A(i,j) = \begin{cases} \lambda(i,j) & \text{if } i \neq j \\ -\lambda(i) & \text{else.} \end{cases}$$

Observe that the sum of each row is 0. A CTMC can be completely specified by the generator matrix.

Stationary Distributions

The following theorem guarantees the existence of a unique stationary distribution for CTMCs. Note that $X(t)$ being irreducible and recurrent is the same as $X_n(t)$ being irreducible and recurrent.

Suppose that $X(t)$ is irreducible and recurrent. Then $X(t)$ has an invariant measure η, which is unique up to multiplicative factors. Moreover, for each $k \in S$, we have

$$\eta_k = \frac{\pi_k}{\lambda(k)}$$

where π is the unique invariant measure of the embedded discrete time Markov chain Xn. Finally, η satisfies

$$0 < \eta_j < \infty, \forall j \in S$$

and if $\sum_i \eta_i < \infty$ then η can be normalised to get a stationary distribution.

Fitting

Let the data set be

$$D = \{(s_0, t_0), (s_1, t_1), ..., (s_{N-1}, t_{N-1})\}$$

where $N = |D|$. Each s_i is a state from the state space S and during the time $[t_i, t_{i+1}]$ the chain is in state s_i. Let the parameters be represented by

$$\theta = \{\lambda, P\}$$

where λ is the vector of holding parameters for each state and P the transition matrix of the embedded discrete time Markov chain.

Then the probability is given by

$$Pr(D|\theta) \propto \lambda(s_0)e^{-\lambda(s_0)(t_1-t_0)}Pr(s_1|s_0) \,.\, \lambda(s_1)e^{-\lambda(s_1)(t_2-t_1)}Pr(s_2|s_1) \,...\, \lambda(s_{N-2})e^{-\lambda(s_{N-2})(t_{N-1}-t_{N-2})}Pr(s_{N-1}|s_{N-2})$$

Let $n(j|i)$ denote the number of i->j transitions in D, and $n(i)$ the number of times s_i occurs in D. Let $t(s_i)$ denote the total time the chain spends in state s_i.

Then the MLEs are given by

$$\hat{\lambda(s)} = \frac{n(s)}{t(s)}, \hat{Pr(j|i)} = \frac{n(j|i)}{n(i)}$$

Usage

To create a CTMC object, you need to provide a valid generator matrix. The CTMC object has the following slots - states, generator, byrow, name (look at the documentation object for further details). Consider the following example in which we aim to model the transition of a molecule from the σ state to the σ^* state. When in the former state, if it absorbs sufficient energy, it can make the jump to the latter state and remains there for some time before transitioning back to the original state. Let us model this by a CTMC -

```
library(markovchain)
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3,
                       1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
molecularCTMC <- new("ctmc", states = energyStates,
                  byrow = byRow, generator = gen,
                  name = "Molecular Transition Model")
```

To generate random CTMC transitions, we provide an initial distribution of the states. This must be in the same order as the dimnames of the generator. The output can be returned either as a list or a data frame.

```
statesDist <- c(0.8, 0.2)
rctmc(n = 3, ctmc = molecularCTMC, initDist = statesDist, out.type = "df", include.T0 = FALSE)
```

```
##       states              time
## 1 sigma_star 0.832612183523886
## 2      sigma  1.89396224393581
## 3 sigma_star  2.07761474036683
```

n represents the number of samples to generate. There is an optional argument T for *rctmc*. It represents the time of termination of the simulation. To use this feature, set n to a very high value, say *Inf* (since we do not know the number of transitions before hand) and set T accordingly.

```
statesDist <- c(0.8, 0.2)
rctmc(n = Inf, ctmc = molecularCTMC, initDist = statesDist, T = 2)
```

```
## [[1]]
## [1] "sigma"      "sigma_star"
##
## [[2]]
## [1] 0.0000000 0.4905904
```

To obtain the stationary distribution simply invoke the *steadyStates* function

```
steadyStates(molecularCTMC)
```

```
##      sigma sigma_star
## [1,]  0.25       0.75
```

For fitting, use the *ctmcFit* function. It returns the MLE values for the parameters along with the confidence intervals.

```r
data <- list(c("a", "b", "c", "a", "b", "a", "c", "b", "c"), c(0, 0.8, 2.1, 2.4, 4, 5, 5.9, 8.2, 9))
ctmcFit(data)
```

```
## $estimate
## An object of class "ctmc"
## Slot "states":
## [1] "a" "b" "c"
##
## Slot "byrow":
## [1] TRUE
##
## Slot "generator":
##            a          b          c
## a -0.9090909  0.6060606  0.3030303
## b  0.3225806 -0.9677419  0.6451613
## c  0.3846154  0.3846154 -0.7692308
##
## Slot "name":
## [1] ""
##
##
## $errors
## $errors$dtmcConfidenceInterval
## $errors$dtmcConfidenceInterval$confidenceLevel
## [1] 0.95
##
## $errors$dtmcConfidenceInterval$lowerEndpointMatrix
##   a b c
## a 0 0 0
## b 0 0 0
## c 0 0 0
##
## $errors$dtmcConfidenceInterval$upperEndpointMatrix
##           a b         c
## a 0.0000000 1 0.8816179
## b 0.8816179 0 1.0000000
## c 1.0000000 1 0.0000000
##
##
## $errors$lambdaConfidenceInterval
## $errors$lambdaConfidenceInterval$lowerEndpointVector
## [1] 0.04576665 0.04871934 0.00000000
##
## $errors$lambdaConfidenceInterval$upperEndpointVector
## [1] 1 1 1
```