Lesson #3: Markov Models: The basic idea, representations, examples, exercises

What's It All About?

This lesson provides your next level introduction to Markov processes. As terminology, representations, and algorithms are presented, TS in ABC Land will be used as a running example.

Markov Process / Markov Model

Loosely speaking, a **Markov process**, or **Markov model** is a stochastic process (which means it consists of a sequence of probabilistically determined events) which is characterized as being "memoryless".

Somewhat more formally, A **Markov process**, or **Markov model**, is a stochastic process that possesses the "Markov property". A process satisfies the **Markov property** if one can make predictions for the future of the process based solely on its present state just as well as one could knowing the process's full history.

Examples

We conceived to TS's behavior in ABC land as a Markov process, since predicting where TS will be tomorrow depends only on where TS is today, not on where TS was yesterday, or on any previous days.

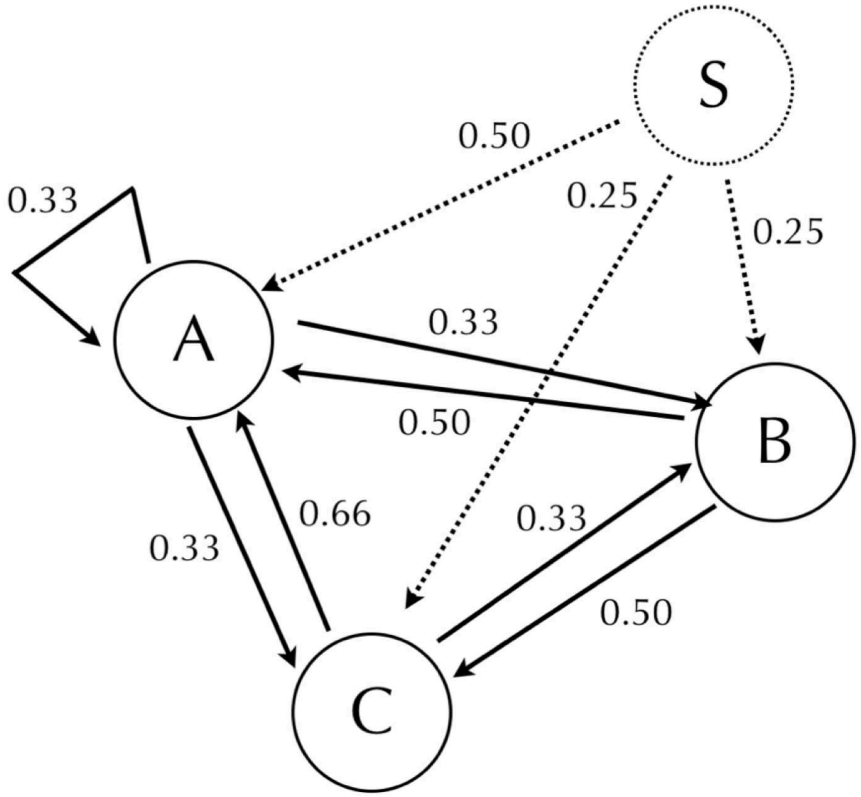
Consider the card game called "Call the Color". This is a solitaire game that is played with a full deck of cards. To start the game, a card is dealt. There after, 51 times, the player declares whether or not the next card will be black or red. If the player guesses correctly, they get a dollar. If they guess incorrectly, they loose a dollar. I claim that this game is **not** a Markov process. Why? **Please, in a sentence or two, do your best to defend my claim.**

Consider a tonal melody, like a classical piece, or a musical theater piece, or an Argentine tango, or a piece of rock and roll music. Do you believe that tonal melodies are Markov processes? That is, do they possess the Markov property? Please say something in defense of your answer.

State transition graph for a Markov model

One popular knowledge representation for the *declarative* bits of a Markov process is the transition graph. Such a graph indicates the probability of transitioning from one state to another state. A "start node" is, if appropriate, added to inform the processing agent how to begin the process.

Here is an example transition graph for TS in ABC land:



State transition probability matrix for a Markov model

Another popular knowledge representation for the *declarative* bits of a Markov process is the state transition matrix. Such a matrix indicates the probability of transitioning from one state to another state. A "special row" is, if appropriate, added to the matrix to inform the processing agent how to begin the process.

Here is an example transition graph for TS in ABC land:

		TO		
		A	B	C
FROM	A	0.33	0.33	0.33
	B	0.50	0.00	0.50
	C	0.67	0.33	0.00
	S	0.50	0.25	0.25

Review question

What is a knowledge representation?

State transition distribution matrix for a Markov model

Yet another popular knowledge representation for the *declarative* bits of a Markov process, **one that is particularly suited to algorithmic processing**, is with the state transition distribution matrix. This matrix may be obtained from the state transition probability matrix by cumulatively adding values in the rows from left to right.

Here is an example transition graph for TS in ABC land:

		TO A	TO B	TO C
FROM	A	0.33	0.67	1.00
	B	0.50	0.50	1.00
	C	0.67	1.00	1.00
	S	0.50	0.75	1.00

Review question

Given a state transition **probability** matrix, how to you construct the functionally equivalent state transition **distribution** matrix?

Pseudocode for simulating a sequence of state transitions based on the state transition distribution matrix

The following pseudocode method can be used to simulate a sequence of transitions with a state transition distribution matrix.

```
set S to the start state
while ( not finished ) do
   set R to a random real in (0,1)
   set S to the first column for which R is less than the cell value in row S of the STDM
   record S
end of while
```

Practice thing #1: Pseudocode execution

Suppose that you are in state `S` and that you generate 0.8121 as a random number. Thus: S → Ⓢ and R → 0.8121. What will the variable `S` be bound to after execution of the fourth statement of the code?

```
set S to the first column for which R is less than the cell value in row S of the STDM
```

FROM \ TO	A	B	C
A	0.33	0.67	1.00
B	0.50	0.50	1.00
C	0.67	1.00	1.00
S	0.50	0.75	1.00

```
set S to the start state
while ( not finished ) do
   set R to a random real in (0,1)
   set S to the first column for which R is less than the cell value in row S of the STDM
   record S
end of while
```

Practice thing #2: Pseudocode execution

Suppose that you are in state `A` and that you generate 0.5845 as a random number. Thus: S $\rightarrow$ Ⓐ and R $\rightarrow$ 0.5845
What will the variable `S` be bound to after execution of the fourth statement of the code?

```
set S to the first column for which R is less than the cell value in row S of the STDM
```

FROM \ TO	A	B	C
A	0.33	0.67	1.00
B	0.50	0.50	1.00
C	0.67	1.00	1.00
S	0.50	0.75	1.00

```
set S to the start state
while ( not finished ) do
   set R to a random real in (0,1)
   set S to the first column for which R is less than the cell value in row S of the STDM
   record S
end of while
```

Yet another simulation exercise for TS in ABC Land

Simulate **two weeks** of traveling for TS in ABC land, based on the following sequence of random numbers, by making good use of the given pseudocode for performing simulations based on Markov models which references the state transition distribution matrix for TS in ABC land.

```
0.819 0.400 0.648 0.952 0.264
0.083 0.742 0.224 0.274 0.837
```

Week 1 simulation solution - with trace

```
set S to the start state
while ( not finished ) do
   set R to a random real in (0,1)
   set S to the first column for which R is less than the cell value in row S of the STDM
   record S
end of while
```

```
0.819 0.400 0.648 0.952 0.264
```

FROM \ TO	A	B	C
A	0.33	0.67	1.00
B	0.50	0.50	1.00
C	0.67	1.00	1.00
S	0.50	0.75	1.00

1. S → Ⓢ
2. R → 0.819
3. S → Ⓒ
4. Ⓒ
5. R → 0.400
6. S → Ⓐ
7. Ⓐ
8. R → 0.648
9. S → Ⓑ
10. Ⓑ
11. R → 0.952
12. S → Ⓒ
13. Ⓒ
14. R → 0.264
15. S → Ⓐ
16. Ⓐ

Week 1 result: <**C A B C A**>

Week 2 simulation solution - with trace

```
set S to the start state
while ( not finished ) do
   set R to a random real in (0,1)
   set S to the first column for which R is less than the cell value in row S of the STDM
   record S
end of while
```

```
0.083 0.742 0.224 0.274 0.837
```

FROM \ TO	A	B	C
A	0.33	0.67	1.00
B	0.50	0.50	1.00
C	0.67	1.00	1.00
S	0.50	0.75	1.00

1. S → Ⓢ
2. R → 0.083
3. S → Ⓐ
4. Ⓐ
5. R → 0.742
6. S → Ⓒ
7. Ⓒ
8. R → 0.224
9. S → Ⓐ
10. Ⓐ
11. R → 0.274
12. S → Ⓐ
13. Ⓐ
14. R → 0.837
15. S → Ⓒ
16. Ⓒ

Week 2 result: <**A C A A C**>

Lisp program to perform simulation of TS in ABC land

```
; ------------------------------------------------------------------------------
; File: ts_abc_simulation.l
; ------------------------------------------------------------------------------
; Simulate 1 week of traveling behavior for TS in ABC land based on the state
; transition distribution matrix.

; ------------------------------------------------------------------------------
; library imports

( load "../libraries/lp.l" )

; ------------------------------------------------------------------------------
; Function to establish the state transition distribution matrix (STDM)

( defun establish-stdm ()
  ( setf ( symbol-plist 'a ) ( list 'a 0.3333 'b 0.6667 'c 1.0000 ) )
  ( setf ( symbol-plist 'b ) ( list 'a 0.5000 'b 0.5000 'c 1.0000 ) )
  ( setf ( symbol-plist 'c ) ( list 'a 0.6667 'b 1.0000 'c 1.0000 ) )
  ( setf ( symbol-plist 's ) ( list 'a 0.5000 'b 0.7500 'c 1.0000 ) )
  nil
)

( establish-stdm )

; ------------------------------------------------------------------------------
; Function to simulate one week's traveling for TS in ABC land. The result will be a
; list of five cities drawn from {A, B, C}.

( defun one-week (&aux s r cities)
  ( setf cities () )
  ( setf s 's )
  ( dotimes ( i 5 )
    ( setf r ( random 1.0 ) )
    ( setf s ( find-column r s ) )
    ( setf cities ( snoc s cities ) )
  )
  cities
)

( defun find-column ( n row )
  ( cond
    ( ( < n ( get row 'a ) ) 'a )
    ( ( < n ( get row 'b ) ) 'b )
    ( ( < n ( get row 'c ) ) 'c )
  )
)

; ------------------------------------------------------------------------------
; Function to compute some probabilities

( defun probabilities ( n &aux data monday tuesday wednesday thursday friday )
```

```
  ( setf data ( mapcar #'eval ( make-list n :initial-element '( one-week ) ) ) )
  ( setf monday ( the-probabilities data 0 ) )
  ( setf tuesday ( the-probabilities data 1 ) )
  ( setf wednesday ( the-probabilities data 2 ) )
  ( setf thursday ( the-probabilities data 3 ) )
  ( setf friday ( the-probabilities data 4 ) )
  ( list monday tuesday wednesday thursday friday )
)

( defun the-probabilities ( data location &aux n a-count b-count c-count )
  ( setf n ( length data ) )
  ( setf a-count ( count 'a ( mapcar ( selector location ) data ) ) )
  ( setf b-count ( count 'b ( mapcar ( selector location )  data ) ) )
  ( setf c-count ( count 'c ( mapcar ( selector location )  data ) ) )
  ( list ( / a-count ( float n ) ) ( / b-count ( float n ) ) ( / c-count ( float n ) )  )
)

( defun selector ( location )
  ( cond
    ( ( = location 0 ) #'first )
    ( ( = location 1 ) #'second )
    ( ( = location 2 ) #'third )
    ( ( = location 3 ) #'fourth )
    ( ( = location 4 ) #'fifth )
  )
)


; ------------------------------------------------------------------------------
; Demo 1

; bash-3.2$ clisp
;
; [1]> ( load "ts_abc_simulation.l" )
; ;; Loading file ts_abc_simulation.l ...
; ;;  Loading file ../libraries/lp.l ...
; ;;  Loaded file ../libraries/lp.l
; ;; Loaded file ts_abc_simulation.l
; T
; [2]> ( trace find-column )
; ;; Tracing function FIND-COLUMN.
; (FIND-COLUMN)
; [3]> ( one-week )
; 1. Trace: (FIND-COLUMN '0.49432975 'S)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.09167117 'A)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.69649065 'A)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.45998025 'C)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.44040638 'A)
; 1. Trace: FIND-COLUMN ==> B
; (A A C A B)
```

```
; [4]> ( one-week )
; 1. Trace: (FIND-COLUMN '0.46515113 'S)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.13419455 'A)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.6360787 'A)
; 1. Trace: FIND-COLUMN ==> B
; 1. Trace: (FIND-COLUMN '0.65475416 'B)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.36542284 'C)
; 1. Trace: FIND-COLUMN ==> A
; (A A B C A)
; [5]> ( one-week )
; 1. Trace: (FIND-COLUMN '0.75314605 'S)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.46932304 'C)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.9159984 'A)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.6214771 'C)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.5426824 'A)
; 1. Trace: FIND-COLUMN ==> B
; (C A C A B)
; [6]> ( one-week )
; 1. Trace: (FIND-COLUMN '0.9586098 'S)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.40656453 'C)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.37267494 'A)
; 1. Trace: FIND-COLUMN ==> B
; 1. Trace: (FIND-COLUMN '0.92834675 'B)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.032627344 'C)
; 1. Trace: FIND-COLUMN ==> A
; (C A B C A)
; [7]> ( one-week )
; 1. Trace: (FIND-COLUMN '0.6135827 'S)
; 1. Trace: FIND-COLUMN ==> B
; 1. Trace: (FIND-COLUMN '0.923228 'B)
; 1. Trace: FIND-COLUMN ==> C
; 1. Trace: (FIND-COLUMN '0.015564144 'C)
; 1. Trace: FIND-COLUMN ==> A
; 1. Trace: (FIND-COLUMN '0.39204293 'A)
; 1. Trace: FIND-COLUMN ==> B
; 1. Trace: (FIND-COLUMN '0.22885722 'B)
; 1. Trace: FIND-COLUMN ==> A
; (B C A B A)
; [8]> (bye)
; Bye.
; bash-3.2$

; ------------------------------------------------------------------------------
; Demo 2
```

```
; bash-3.2$ clisp
; ...
; [1]> ( load "ts_abc_simulation.l" )
; ;; Loading file ts_abc_simulation.l ...
; ;;  Loading file ../libraries/lp.l ...
; ;;  Loaded file ../libraries/lp.l
; ;; Loaded file ts_abc_simulation.l
; T
; [2]> ( one-week )
; (A C B A A)
; [3]> ( one-week )
; (C A C B C)
; [4]> ( one-week )
; (A A C A B)
; [5]> ( one-week )
; (A A B C A)
; [6]> ( one-week )
; (C A C A B)
; [7]> ( one-week )
; (C A B C A)
; [8]> ( one-week )
; (B C A B A)
; [9]> ( one-week )
; (C A A A B)
; [10]> ( one-week )
; (A B A B A)
; [11]> ( one-week )
; (A B A C A)
; [12]> (bye)
; Bye.
; bash-3.2$

; ------------------------------------------------------------------------------
; Demo 3

; bash-3.2$ clisp
; ...
;
; [1]> ( load "ts_abc_simulation.l" )
; ;; Loading file ts_abc_simulation.l ...
; ;;  Loading file ../libraries/lp.l ...
; ;;  Loaded file ../libraries/lp.l
; ;; Loaded file ts_abc_simulation.l
; T
; [2]> ( trace the-probabilities )
; ;; Tracing function THE-PROBABILITIES.
; (THE-PROBABILITIES)
; [3]> ( probabilities 5 )
; 1. Trace: (THE-PROBABILITIES '((C B C A A) (B A A A B) (A B C A B) (C B A B A) (A A A A B)) '0)
; 1. Trace: THE-PROBABILITIES ==> (0.4 0.2 0.4)
; 1. Trace: (THE-PROBABILITIES '((C B C A A) (B A A A B) (A B C A B) (C B A B A) (A A A A B)) '1)
; 1. Trace: THE-PROBABILITIES ==> (0.4 0.6 0)
; 1. Trace: (THE-PROBABILITIES '((C B C A A) (B A A A B) (A B C A B) (C B A B A) (A A A A B)) '2)
```

```
; 1. Trace: THE-PROBABILITIES ==> (0.6 0 0.4)
; 1. Trace: (THE-PROBABILITIES '((C B C A A) (B A A A B) (A B C A B) (C B A B A) (A A A A B)) '3)
; 1. Trace: THE-PROBABILITIES ==> (0.8 0.2 0)
; 1. Trace: (THE-PROBABILITIES '((C B C A A) (B A A A B) (A B C A B) (C B A B A) (A A A A B)) '4)
; 1. Trace: THE-PROBABILITIES ==> (0.4 0.6 0)
; ((0.4 0.2 0.4) (0.4 0.6 0) (0.6 0 0.4) (0.8 0.2 0) (0.4 0.6 0))
; [4]> ( untrace )
; (THE-PROBABILITIES)
; [5]> ( probabilities 10000 )
; ((0.4999 0.2479 0.2522) (0.4583 0.2504 0.2913) (0.4713 0.248 0.2807) (0.4663 0.2516 0.2821)
;  (0.4695 0.2546 0.2759))
; [6]> (bye)
; Bye.
; bash-3.2$
```