Bayesian_Reinforcement_Learning

Shoyeb_Khan

13/01/2017

```
library(ReinforcementLearning)

#ReinforcementLearning package performs model-free reinforcement learning in R.
#This implementation enables the learning of an optimal policy based on sample
#sequences consisting of states, actions and rewards. In addition, it supplies
#multiple predefined reinforcement learning algorithms, such as experience replay.

##################################Attempt01#########################****
##As of now the state data is randomly generated but when
##integrated with the "pymunk" it will contain the states and the
##rewards and actions of the angry bird.
state_tr_data <- sampleGridSequence(N=10000)

head(state_tr_data)
```

```
##   State Action Reward NextState
## 1    s2   down     -1        s2
## 2    s3   left     -1        s2
## 3    s3   down     -1        s3
## 4    s2   down     -1        s2
## 5    s1   left     -1        s1
## 6    s3     up     10        s4
```

```
             ###or###

# Define the state and the action sets
#states <- c("s1","s2","s3","s4")
#actions <- c("up","down","left","right")
#envr_n_bird <- gridworldEnvironment()


model <- ReinforcementLearning(data = state_tr_data,
                               s= "State",
                               a= "Action",
                               r = "Reward",
                               s_new = "NextState",iter = 10)

summary(model)
```

```
##
## Model details
```

```
## Learning rule:           experienceReplay
## Learning iterations:     10
## Number of states:        4
## Number of actions:       4
## Total Reward:            -2685
##
## Reward details (per iteration)
## Min:                     -2685
## Max:                     -2685
## Average:                 -2685
## Median:                  -2685
## Standard deviation:      0
```

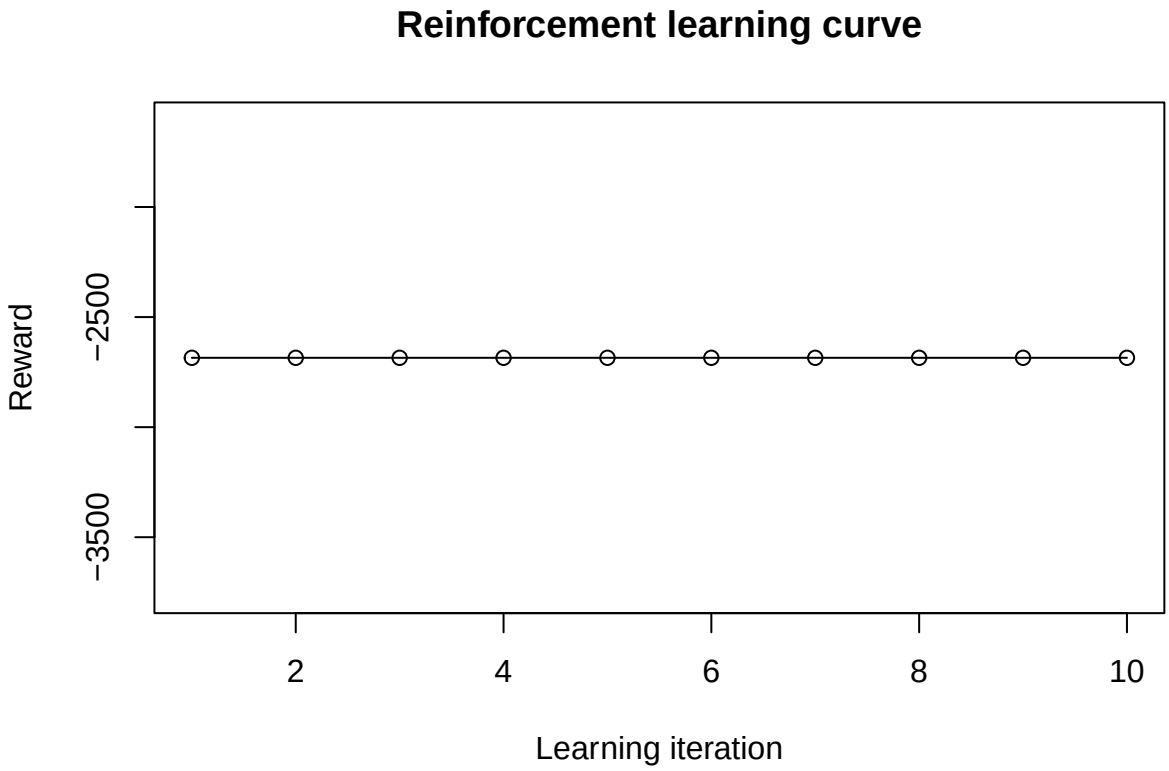
```
print(model)
```

```
## State-Action function Q
##          right        up        down      left
## s1 -1.10011111 -1.100111 -1.00111111 -1.100111
## s2 -0.01111111 -1.100111 -1.00111111 -1.001111
## s3 -0.01111111  9.888889 -0.01111111 -1.001111
## s4 -1.11111111 -1.111111 -1.11111111 -1.111111
##
## Policy
##      s1      s2      s3      s4
##  "down" "right"    "up" "right"
##
## Reward (last iteration)
## [1] -2685
```

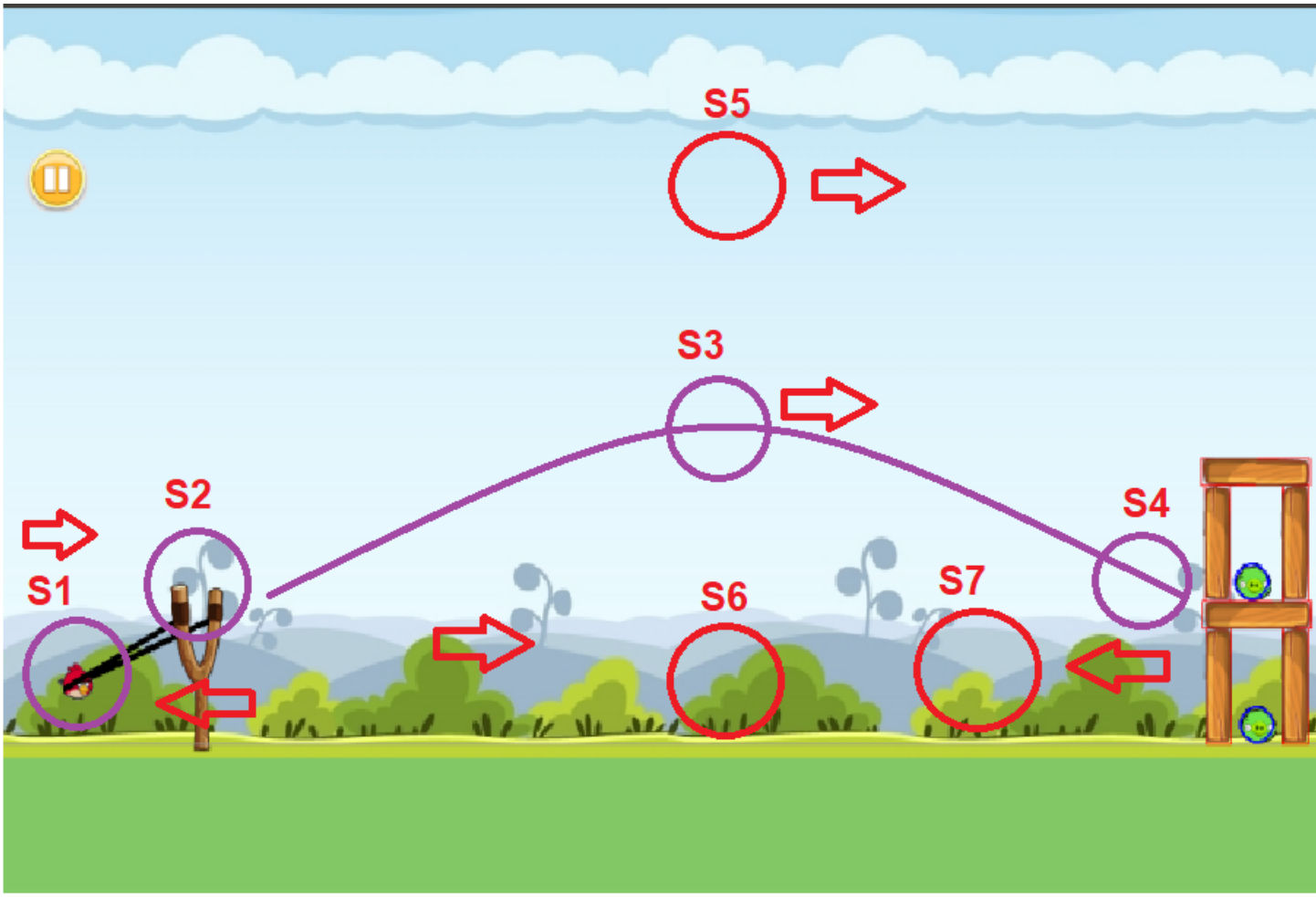
```
computePolicy(model)
```

```
##      s1      s2      s3      s4
##  "down" "right"    "up" "right"
```

```
plot(model)
```




```
##############################Attempt02#####################################

# The inputs that will be required are :
# 1. Definition of the environment
# 2. A list of states
# 3. A list of actions
# 4. Control parameters
# 5. epsilon : usually the value is 0.1(probability value)

# The output of the model is :
# A state action value matrix which indicates how good it is
# to be in particular state and take an action.

## Creating the environment from the snapshot of the angry bird simulation
## from the 'pymunk' API.

## For the definition of the states and the direction refer the png image
## attached in the GitHub repository.

knitr::include_graphics('C:/Users/khans/OneDrive/Documents/R_mentor/state_space_action.png')
```




```
y.env <- function(state,action) {

  next_state <- state

   if (state == state("s2") && action == "left")
    { next_state <- state("s1") }

   if (state == state("s1") && action == "right")
    { next_state <- state("s3") }

   if (state == state("s2") && action == "left")
    { next_state <- state("s1"); reward <-  5 }

   if (state == state("s2") && action == "right")
    { next_state <- state("s3") }

   if (state == state("s2") && action == "right")
    { next_state <- state("s3"); reward <-  10 }

   if (state == state("s3") && action == "left")
    { next_state <- state("s2") ; rewards <- -1 }
```

```
   if (state == state("s3") && action == "right")
    { next_state <- state("s4"); reward <- 20 }

   if (state == state("s4") && action == "left")
    { next_state <- state("s7") }

   if (state == state("s4") && action == "left")
    { next_state <- state("s7") ; rewards <- 10}

   if (state == state("s2") && action == "right")
    { next_state <- state("s6") ; rewards <- -2 }

   if (state == state("s2") && action == "right")
    { next_state <- state("s5") ; rewards <- -2 }


   if (state == state("s5") && action == "left")
    { next_state <- state("s2") ; reward <- -3 }

   if (state == state("s5") && action == "right")
    { next_state <- state("s2") ; reward <- -3 }

   if (state == state("s6") && action == "left")
    { next_state <- state("s2") ; reward <- -3 }

   if (state == state("s6") && action == "right")
    { next_state <- state("s2") ; reward <- -3 }

   if (next_state == state("s7") && state != state("s7"))
    { reward <- 10}

   else
    { reward <- -1}

   out <- list(NextState = next_state, Reward = reward)

   return(out)
}

states <- c("s1", "s2", "s3", "s4", "s5", "s6", "s7")

actions <- c("left","right")

data    <- sampleExperience(N=5000,env=y.env,states=states,actions=actions)
control <- list(alpha = 0.1, gamma = 0.8, epsilon = 0.1)
model   <- ReinforcementLearning(data, s = "State", a = "Action", r = "Reward",
                                 s_new = "NextState", control = control,iter = 10)

print(model)
```

```
## State-Action function Q
##       right    left
## s1  2.04000  0.6320
```

```
## s2 -1.39552  0.6320
## s3  3.80000 -0.4944
## s4  3.80000  6.0000
## s5 -0.49440 -0.4944
## s6 -0.49440 -0.4944
## s7 -5.00000 -5.0000
##
## Policy
##      s1      s2      s3      s4      s5      s6      s7
## "right"  "left" "right"  "left" "right" "right" "right"
##
## Reward (last iteration)
## [1] -985
```

```
computePolicy(model)
```

```
##      s1      s2      s3      s4      s5      s6      s7
## "right"  "left" "right"  "left" "right" "right" "right"
```

```
plot(model)
```