Reinforcement Learning

MDP (markov decision process)

Code available as a github gist: https://gist.github.com/smc77/8277155#file-gistfile1-txt from Shane Conway.

Reinforcement learning is a topic usually covered in AI classes because it is helpful for typical AI problems such as learning to play checkers, or teaching a robot to move through a room.

Reinforcement models typically have:

- E environment - the checker board, the room, etc.
- S state - the current state (such as location) of the agent
- A action - action available to the agent such as move north, stay put, etc.
- R reward - the reward given to the agent for the last action A

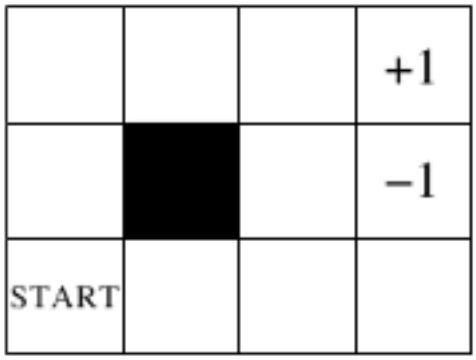

Figure 1: grid

```
actions <- c("N", "S", "E", "W")

x <- 1:4
y <- 1:3

rewards <- matrix(rep(0, 12), nrow=3)
rewards[2, 2] <- NA
rewards[1, 4] <- 1
rewards[2, 4] <- -1

values <- rewards # initial values

states <- expand.grid(x=x, y=y)

# Transition probability
transition <- list("N" = c("N" = 0.8, "S" = 0, "E" = 0.1, "W" = 0.1),
        "S"= c("S" = 0.8, "N" = 0, "E" = 0.1, "W" = 0.1),
        "E"= c("E" = 0.8, "W" = 0, "S" = 0.1, "N" = 0.1),
        "W"= c("W" = 0.8, "E" = 0, "S" = 0.1, "N" = 0.1))

# The value of an action (e.g. move north means y + 1)
action.values <- list("N" = c("x" = 0, "y" = 1),
        "S" = c("x" = 0, "y" = -1),
        "E" = c("x" = -1, "y" = 0),
        "W" = c("x" = 1, "y" = 0))
```

```r
# act() function serves to move the robot through states based on an action
act <- function(action, state) {
    action.value <- action.values[[action]]
    new.state <- state
    #
    if(state["x"] == 4 && state["y"] == 1 || (state["x"] == 4 && state["y"] == 2))
        return(state)
    #
    new.x = state["x"] + action.value["x"]
    new.y = state["y"] + action.value["y"]
    # Constrained by edge of grid
    new.state["x"] <- min(x[length(x)], max(x[1], new.x))
    new.state["y"] <- min(y[length(y)], max(y[1], new.y))
    #
    if(is.na(rewards[new.state["y"], new.state["x"]]))
        new.state <- state
    #
    return(new.state)
}


rewards
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    1
## [2,]    0   NA    0   -1
## [3,]    0    0    0    0
```

```r
bellman.update <- function(action, state, values, gamma=1) {
    state.transition.prob <- transition[[action]]
    q <- rep(0, length(state.transition.prob))
    for(i in 1:length(state.transition.prob)) {
        new.state <- act(names(state.transition.prob)[i], state)
        q[i] <- (state.transition.prob[i] * (rewards[state["y"], state["x"]] + (gamma * values[new.stat
    }
    sum(q)
}

value.iteration <- function(states, actions, rewards, values, gamma, niter) {
    for (j in 1:niter) {
        for (i in 1:nrow(states)) {
            state <- unlist(states[i,])
            if(i %in% c(4, 8)) next # terminal states
            q.values <- as.numeric(lapply(actions, bellman.update, state=state, values=values, gamma=ga
            values[state["y"], state["x"]] <- max(q.values)
        }
    }
    return(values)
}

final.values <- value.iteration(states=states, actions=actions, rewards=rewards, values=values, gamma=0

final.values
```

```
##           [,1]      [,2]      [,3]     [,4]
## [1,] 0.9516605 0.9651596 0.9773460  1.00000
## [2,] 0.9397944        NA 0.8948359 -1.00000
## [3,] 0.9266500 0.9150957 0.9027132  0.81989
```