~~~
---
title: "R Notebook"
output: html_notebook
---

```{r}
#' Computes the reinforcement learning policy
#'
#' Computes reinforcement learning policy from a given state-action table Q.
#' The policy is the decision-making function of the agent and defines the learning
#' agent's behavior at a given time.
#'
#' @param x Variable which encodes the behavior of the agent. This can be
#' either a \code{matrix}, \code{data.frame} or an \code{\link{rl}} object.
#' @seealso \code{\link{ReinforcementLearning}}
#' @return Returns the learned policy.
#' @examples
#' # Create exemplary state-action table (Q) with 2 actions and 3 states
#' Q <- data.frame("up" = c(-1, 0, 1), "down" = c(-1, 1, 0))
#'
#' # Show best possible action in each state
#' computePolicy(Q)
#'
#' @rdname computePolicy
#' @export
computePolicy <- function(x) {
  UseMethod("computePolicy", x)
}

#' @export
computePolicy.matrix <- function(x) {
  policy <- colnames(x)[apply(x, 1, which.max)]
  names(policy) <- rownames(x)
  return(policy)
}

#' @export
computePolicy.data.frame <- function(x) {
  return(computePolicy(as.matrix(x)))
}

#' @export
computePolicy.rl <- function(x) {
  return(computePolicy(x$Q))
}

#' @export
computePolicy.default <- function(x) {
  stop("Argument invalid.")
}

#' Computes the reinforcement learning policy
~~~

~~~
#' @export
policy <- function(x) {
  .Deprecated("computePolicy")
  computePolicy(x)
}

```
```{r}
# This function contains all of the parameters in one location so that it is easy to
# update the model as needed
LoadParameters<-function()
{
Parameters = data.frame(matrix(vector(), 1, 16, dimnames=list(c(), c("seed",
"cols","rows","percpop","biasYN","bias","percgroup","vulnamount",
"daylength","traveltime","detaintime","alpha","gamma","epsilon","pop","popdensity"))),string
Parameters$seed=7013         # Seed
Parameters$biasYN=0           # Social Bias: {0=No,1=Yes}
Parameters$bias=.1          # Amount Bias
Parameters$perctype=.3     # Percentage of Biased Group
Parameters$perctarget=.3     # Percentage of Biased Group
Parameters$indirect1=.25
Parameters$indirect2=.75
#Parameters$vulnamount=.05    # Vulnerability
#Parameters$perccrim=.1      # Possibly Increase in Criminal due to Vulnerability
#Parameters$percsusp=.25    # Possibly Increase in Suspicious due to Vulnerability
Parameters$daylength=10        # Length of Day
Parameters$traveltime=1      # Travel Time
Parameters$detaintime=3       # Detain Time
Parameters$alpha = .6        # Learning Rate [0,1]
Parameters$gamma = .8        # Thoughtfulness Factor [0,1]
Parameters$epsilon = .2      # Exploration Parameter [0,1]
Parameters$N=1000            # Number of Iterations of Sample States
Parameters$MoveReward=0

#Parameters$pop=Parameters$rows*Parameters$cols     # Total Population: pop
Parameters$pop=999
Parameters$popdensity=Parameters$percpop*Parameters$cols*Parameters$rows  #Density of popula

return(Parameters)
}
```
# This file creates the population data file for the Aletheia2019 project.

```{r}
#setwd("/Users/gatliffe/Documents/Research/Aletheia2019")
library(binaryLogic)
library(stats)
library(dplyr)
library(hash)
library(ggplot2)
library(ReinforcementLearning)
library(testthat)
~~~

~~~
# Load parameters.

```{r}
Parameters<-LoadParameters()
```

# This is the population seeding routine.

```{r}

set.seed=(Parameters$seed)

Population = data.frame(matrix(vector(), 0, 11, dimnames=list(c(), c("ID", "Type","Target",
"Indirect","ClueNoInfo", "ClueType", "ClueIndirect", "TimesOnMap","IgnoreHistory", "DetainHi
))),stringsAsFactors=F)
Person = data.frame(matrix(vector(), 1, 11, dimnames=list(c(), c("ID", "Type","Target", "Ran
"Indirect","ClueNoInfo", "ClueType", "ClueIndirect","TimesOnMap","IgnoreHistory",
"DetainHistory"))),stringsAsFactors=F)
for (i in 1:Parameters$pop)
{
Person$ID <-sprintf("%03d",i)
Person$Type <- sample(0:1, 1, replace=T,prob=c(1-Parameters$perctype,Parameters$perctype))
Person$Target <- sample(0:1, 1, replace=T,prob=c(1-Parameters$perctarget,Parameters$perctarg
Person$Random<- sample(0:5, 1, replace=T)
if (Person$Type==0)
  {Person$Indirect<- sample(0:1, 1, replace=T,prob=c(1-Parameters$indirect0,Parameters$indir
if (Person$Type==1)
{Person$Indirect<- sample(0:1, 1, replace=T,prob=c(1-Parameters$indirect1,1-Parameters$indir
  Person$ClueNoInfo <-3*Person$Target+Person$Random
  Person$ClueType <-3*Person$Target+Person$Type+Person$Random
  Person$ClueIndirect <-3*Person$Target+Person$Indirect+Person$Random
  Person$TimesOnMap<-0
  Person$IgnoreHistory<-0
  Person$DetainHistory<-0
  Population<-rbind(Population,Person)
}

save(Population, file = paste0("Population.",Parameters$seed,".rda"))
write.csv(Population, file = paste0("Population.",Parameters$seed,".csv"),row.names=FALSE)
```

# Reweighting Suspicious. This chunk creates a version of population where the only
#change is that type 2 is rated more suspicious.

#```{r}
#PopulationY<-PopulationN
#  for (j in 1:Parameters$pop)
#{
#    if (PopulationY$Type[j]=="2")
#PopulationY$Suspicious[j] <- PopulationY$Suspicious[j]*(1+Parameters$bias)
#    Person$SuspWeight <- 0
~~~

```
id_num<-sample(100:140, 40, replace=F)
for(i in 1: nrow(simplepop))
{
  simplepop$ID[i]<-id_num[i]
  simplepop$Code[i]<-paste0(simplepop$ID[i],'.',simplepop$Type[i],'.',simplepop$Susp[i])
}
id_num0<-sample(1:40, 40, replace=F)
Left<-NA
Right<-NA
LeftCode<-NA
RightCode<-NA
LeftReward<-NA
RightReward<-NA
State<-NA
NextState<-NA
for(i in 1: 20)
{
  id_num1<-id_num0[i]
  id_num2<-id_num0[i+20]
  Left[i]<-id_num1
  Right[i]<-id_num2
  LeftCode[i]<-simplepop$Code[id_num1]
  RightCode[i]<-simplepop$Code[id_num2]
  LeftReward[i]<-simplepop$Crim[id_num1]
  RightReward[i]<-simplepop$Crim[id_num2]
  State[i]<-paste0(LeftCode[i],'.',RightCode[i])
  NextState[i-1]<-paste0(LeftCode[i],'.',RightCode[i])
  NextState[i]<-"End"
}
createsample<-data.frame(Left, Right,LeftCode, RightCode, LeftReward, RightReward, State, Ne
return(createsample)
}
statediagramfunction <- function(createsample, ...) {
time = 20
detain = 3
move = 1
statemap<-data.frame("State"=paste0(time,".",createsample$State[1]), "Action"="Left",
"Reward"=createsample$LeftReward[1], "NextState"=paste0(time-detain,".",createsample$NextSta
nextrow<-data.frame("State"=paste0(time,".",createsample$State[1]), "Action"="Right",
"Reward"=createsample$RightReward[1], "NextState"=paste0(time-detain,".",createsample$NextSt
statemap<-rbind(statemap,nextrow)
nextrow<-data.frame("State"=paste0(time,".",createsample$State[1]), "Action"="None", "Rewarc
"NextState"=paste0(time-move,".",createsample$NextState[1]))
statemap<-rbind(statemap,nextrow)

for(i in 2:1000)
{
  statedummy<-as.character(statemap$NextState[i])
    flag<-0
    if (is.na(statedummy))
{}
      else if (statedummy=="End")
```

~~~
  {
    openstate<-unlist(stri_split_fixed(as.character(statedummy),".", fixed = TRUE, n=2))
        for (k in 1:nrow(createsample))
    {
      if (openstate[2]==as.character(createsample$State[k]))
      {
        timedet<-as.numeric(openstate[1])-detain
        timemove<-as.numeric(openstate[1])-move
if(as.numeric(openstate[1])>=detain)
{
                nextrow<-data.frame("State"=statedummy, "Action"="Left",
"Reward"=createsample$LeftReward[k], "NextState"=paste0(timedet,".",createsample$State[k+1])
        nextrow2<-data.frame("State"=statedummy, "Action"="Right", "Reward"=createsample$Rig
 "NextState"=paste0(timedet,".",createsample$State[k+1]))
}
        else if(as.numeric(openstate[1])<detain)
{
                nextrow<-data.frame("State"=statedummy, "Action"="Left",
"Reward"=createsample$LeftReward[k], "NextState"="End")
        nextrow2<-data.frame("State"=statedummy, "Action"="Right", "Reward"=createsample$Rig
 "NextState"="End")
}
        if(as.numeric(openstate[1])>=move)
{
        nextrow3<-data.frame("State"=statedummy, "Action"="None", "Reward"=0,
"NextState"=paste0(timemove,".",createsample$State[k+1]))
        statemap<-rbind(statemap, nextrow, nextrow2, nextrow3)
        }
        else if (as.numeric(openstate[1])<move)
{
        nextrow3<-data.frame("State"=statedummy, "Action"="None", "Reward"=0, "NextState"="E
        statemap<-rbind(statemap, nextrow, nextrow2, nextrow3)
      }
    }
  }
  }
}
}
return(statemap)
}
```

```{r}
runRL<-function(simpledat, trainmodelold)
{
# Load dataset
simpledat$State<-as.character(simpledat$State)
simpledat$NextState<-as.character(simpledat$NextState)
simpledat$Action<-as.character(simpledat$Action)
# Define reinforcement learning parameters
control <- list(alpha = 0.2, gamma = 0.4, epsilon = 0.1)
~~~

~~~
{
# Load dataset
simpledat$State<-as.character(simpledat$State)
simpledat$NextState<-as.character(simpledat$NextState)
simpledat$Action<-as.character(simpledat$Action)
# Define reinforcement learning parameters
control <- list(alpha = 0.2, gamma = 0.4, epsilon = 0.1)

# Perform reinforcement learning
trainmodelnew <- ReinforcementLearning(simpledat, s = "State", a = "Action", r =
                                 "Reward",
                               s_new = "NextState", iter = 1000, control = control)
# Print optimal policy
return(trainmodelnew)
}

```
```{r}
library(stringi)
library(dplyr)
library(ReinforcementLearning)

setwd("~/Alethea")
simplepop <- read.csv("~/Alethea/simplepop.csv")
createsample<-createsamplefunction(simplepop)
train<-statediagramfunction(createsample)
trainmodel<-runRLinit(train)

for (m in 100:199)
{
createsample<-createsamplefunction(simplepop)
train<-statediagramfunction(createsample)
trainmodel<-runRL(train, trainmodel)
policytrain<-computePolicy(trainmodel)
}
View(policytrain)

createsample<-createsamplefunction(simplepop)
test<-statediagramfunction(createsample)
testmodel<-runRL(test,trainmodel)
```
```{r}
policytest<-computePolicy(testmodel)

policytest<- data.frame(unlist(policytest))
policytest<-cbind(policytest,State=rownames(policytest))
policytest$State<-as.character(policytest$State)
     finalpolicy<-NA
for (n in 1:nrow(policytest))
{
 policytest$State[n]<-sub('X','', policytest$State[n])
}
~~~

```
    { finalpolicy2$State<-policytest$State[n]
   finalpolicy2$Action<-policytest$unlist.policytest.[n]
   finalpolicy<-rbind(finalpolicy, finalpolicy2)
 }
}
 }
finalpolicy<-unique(finalpolicy)
 View(finalpolicy)
```