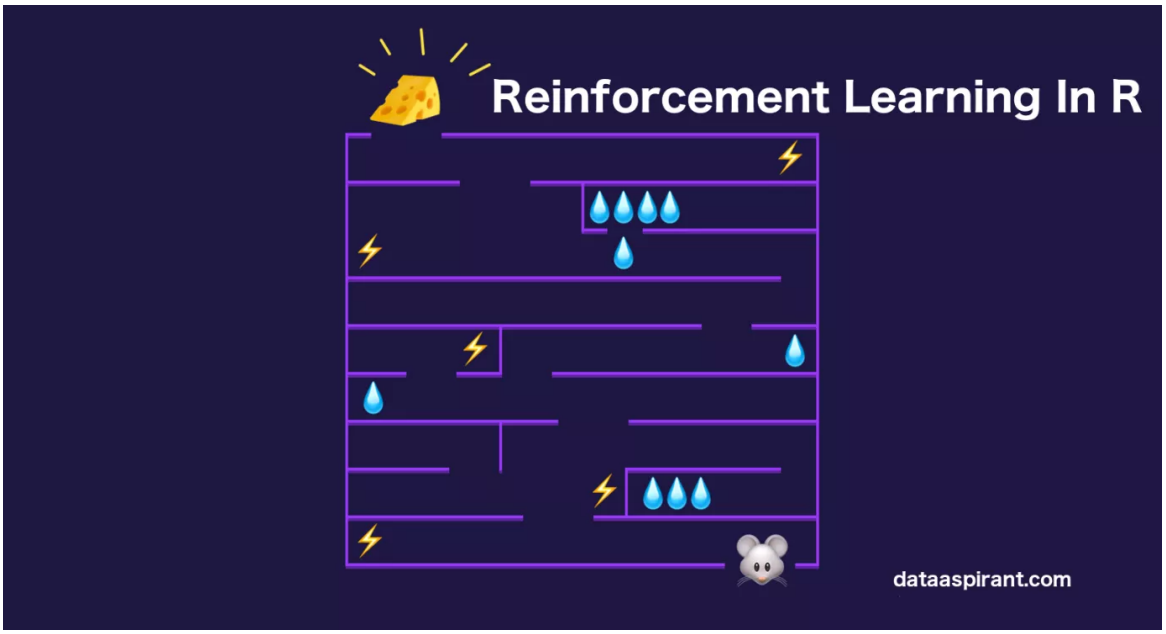
HOW TO PERFORM REINFORCEMENT LEARNING WITH R

February 5, 2018 Chaitanya Sagar 1 Comment Machine Learning, R



Reinforcement learning in R

Reinforcement Learning with R

Machine learning algorithms were mainly divided into **three** main categories.

- Supervised learning algorithms
  - Classification and regression algorithms
- Unsupervised learning algorithms
  - Clustering algorithms
- Reinforcement learning algorithms

We have covered supervised learning and unsupervised learning algorithms couple of times in our blog articles. In this article, you are going to learn about

the third category of machine learning algorithms. Which are reinforcement learning algorithms.

Before we drive further let quickly look at the table of contents.

Table of contents:

Reinforcement Learning with R CLICK TO TWEET

Reinforcement learning real-life example

The modern education system follows a standard pattern of teaching students. The teacher goes over the concepts need to be covered and reinforces them through some example questions. After explaining the topic and the process with a few solved examples, students are expected to solve similar questions from their exercise book themselves.

This mode of learning is also adopted in machine learning algorithms as a separate class known as reinforcement learning. Though it is easy to know and understand how reinforcement works, the concept is hard to implement.

Typical reinforcement process

In a typical reinforcement process, the machine acts as the 'student' trying to learn the concept.

To learn, the machine interacts with a 'teacher' to know the classes of specific data points and learns it. This learning is guided by assigning rewards and penalties to correct and incorrect decisions respectively. Along the way, the machine makes mistakes and corrects itself so as to maximize the reward and minimize the penalty.

As it learns through trial and error and continuous interaction, a framework is built by the algorithm. Since it is so human-like, it has used in specific facets in the industry where a predefined training data is not available. Some examples include puzzle navigation and tic-tac-toe games.

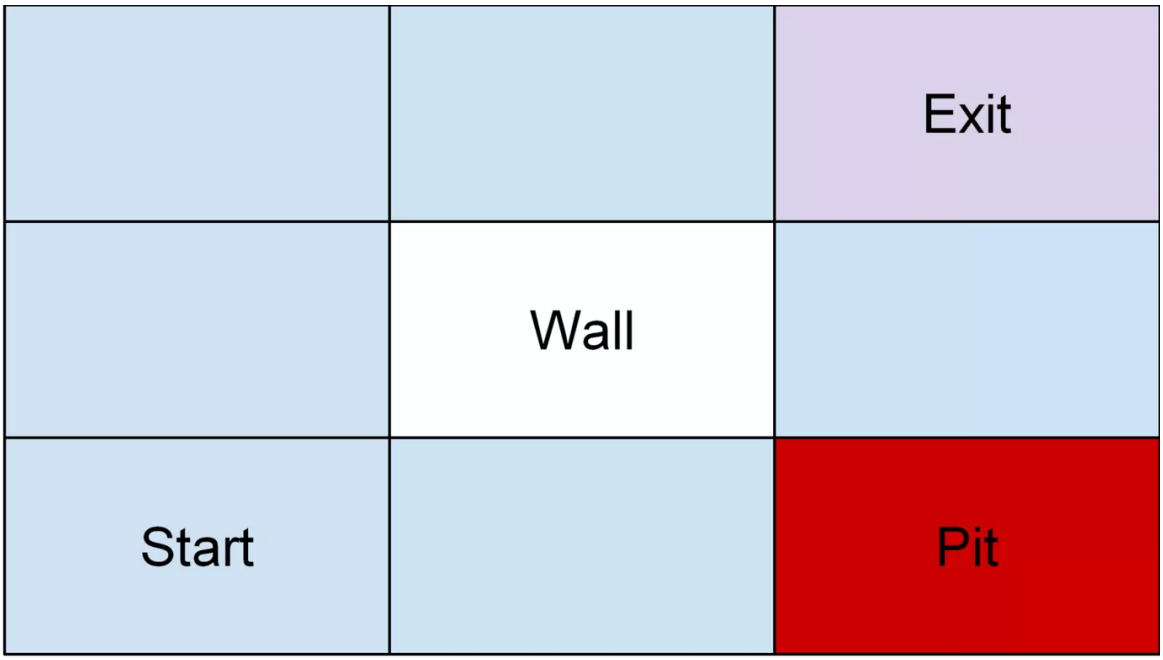
Reinforcement Learning process

Before developing Reinforcement learning algorithm using R, one needs to break down the process into smaller tasks. In programming terminology Divide and Rule.

Divide and Rule: Breaking down reinforcement learning process

Following a step-wise approach, one needs a set of 'policies' laid down for the machine to follow. A set of reward and penalty rules for the machine to assess how it is performing. The training limit specifying the trial and error experiences which the machine uses to train itself.

Now let's start with a toy example: Navigating to the exit in a **3 by 3** matrix. Let's say we have the following matrix.



Reinforcement Learning: Image01

In this example, the machine can navigate in **4 directions**.

- UP
- DOWN
- LEFT
- RIGHT

From the **'Start',** the aim is to reach the **'Exit'** without going through the 'Pit'. The only path to reach **Exit** from **Start** is the below sequence.

1. UP
2. UP
3. LEFT
4. LEFT

But how does the machine learn it?

Here the policies are the set of actions (UP, DOWN, LEFT, RIGHT) with rules that an action is not available if choosing it takes you out of the boundary or to the block named 'Wall'.

Then we have the reward matrix where taking each step is a small penalty, falling into the pit is a big penalty and reaching the exit has a reward. The final

piece is the way experience is calculated.

In this case, the sum of all the actions. Assigning a small penalty to each step will be instrumental for the machine to minimize the number of steps. Assigning a big penalty to the pit should make the machine avoid it and the reward to the goal will attract the machine towards it. This is how the machine trains.

Let's now understand the same from a coding perspective before we try it using R!

Reinforcement learning implementation in R

Before we straightway implementing the reinforcement learning in R programming language, Let's understand about some background implementation concepts.

Reinforcing yourself – Learning the background before the actual implementation

To make the navigation possible, the machine will continuously interact with the puzzle and try to learn the optimal path. Over time, it will start seeking the reward and avoiding the pit. When the optimal path is obtained, the output is provided in the form of a set of actions performed and the rewards associated with each of them.

While learning, the machine iterates by taking each of the possible actions and the change in reward after each action. This is usually followed using the **'Markov Process'** which implies that the decision the machine makes at any given state is independent of the decisions the machine has made at the previous states.

As a result, the machine arrives with the following five elements of reinforcement learning.

- Possible set of states, **s**
- Set of possible actions, **A** – Defined for the algorithm
- Rewards and Penalties – **R**
- Policy, π; and

- Value, **v**

In defined terms, we want to explore the set of possible states,**s**, by taking actions, **A** and come up with an optimal policy π^* which maximizes the value, **v** based on rewards and penalties, **R**.

Now that we have understood the concept, let's try a few examples using R.

Teaching the child to walk – MDP toolbox package

The 'MDPtoolbox' package in R is a simple Markov decision process package which uses the Markov process to learn reinforcement. It is a good package for solving problems such as the toy example demonstrated in this article earlier.

Let's load the package first.

```
# Teaching the child to walk - MDPtoolbox package

# Installing and loading the package

# install.packages("MDPtoolbox")

library(MDPtoolbox)
```

To define the elements of reinforced learning. We need to assign a label to each of the states in the navigation matrix. For the sake of simplicity, we will take a shot-down 2*2 version of the navigation matrix which looks like this:

Reinforcement Learning: Image 02

I have labeled each block as a state from S1 to S4. S1 is the start point and S4 is the endpoint. One cannot go directly from S1 to S4 due to the wall. In S1, we see that there is no way to reach S4. One can only move to S2 or remain in S1.

Hence, the down matrix will have the probabilities only for S1 and S2 in the first row. We can similarly define the probabilities for every action in each state.

Let's define the actions now.

```
# 1. Defining the Set of Actions - Left, Right, Up and Down for 2*2 matrix
# Remember! This will be a probability matrix, so we will use the matrix() func

#Up Action
up=matrix(c( 1, 0, 0, 0,
         0.7, 0.2, 0.1, 0,
         0, 0.1, 0.2, 0.7,
         0, 0, 0, 1),
       nrow=4,ncol=4,byrow=TRUE)

#Down Action
down=matrix(c(0.3, 0.7, 0, 0,
           0, 0.9, 0.1, 0,
           0, 0.1, 0.9, 0,
           0, 0, 0.7, 0.3),
         nrow=4,ncol=4,byrow=TRUE)

#Left Action
left=matrix(c( 0.9, 0.1, 0, 0,
         0.1, 0.9, 0, 0,
         0, 0.7, 0.2, 0.1,
         0, 0, 0.1, 0.9),
       nrow=4,ncol=4,byrow=TRUE)

#Right Action
right=matrix(c( 0.9, 0.1, 0, 0,
           0.1, 0.2, 0.7, 0,
           0, 0, 0.9, 0.1,
           0, 0, 0.1, 0.9),
         nrow=4,ncol=4,byrow=TRUE)

#Combined Actions matrix
Actions=list(up=up, down=down, left=left, right=right)
```

The second element is the rewards and penalties function. The only penalty is the small penalty on every additional step. Let's keep it -1.

The reward is obtained on reaching state S4. Let's keep the weight to be +10. Hence our Rewards matrix R can be obtained

```
#2. Defining the rewards and penalties
Rewards=matrix(c( -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              10, 10, 10, 10),
            nrow=4,ncol=4,byrow=TRUE)
```

That's it! Now it is up to the algorithm to come up with the optimal policy and its value.

The **mdp_policy_iteration()** function is used to solve the problem in R. The function requires actions, rewards, and discount as inputs to calculate the results.

Discount is used to decrease the value of the current reward or penalty as each of the steps are taken.

Let's see if the defined problem can be solved correctly by the package.

```
#3. Solving the navigation
solver=mdp_policy_iteration(P=Actions, R=Rewards, discount = 0.1)
```

The result gives us the policy, the value at each step and additionally, the number of iterations and time taken. As we know, the policy should dictate the correct path to reach the final state S4. We use the policy function to know the matrices used for defining the policy and then the names from the actions list.

```
#4. Getting the policy
solver$policy #2 4 1 1
names(Actions)[solver$policy] #"down"  "right" "up" "up"
```

The values are contained in V and show the reward at each step.

```
#5. Getting the Values at each step. These values can be different in each run
solver$V #58.25663  69.09102  83.19292 100.00000
```

iter and time can be used to know the iterations and time to keep track of the complexity.

```
#6. Additional information: Number of iterations
solver$iter #2
```

```
#7. Additional information: Time taken. This time can be different in each run
solver$time #Time difference of 0.009523869 secs
```

Using Github reinforcement learning package

Cran provides documentation to 'ReinforcementLearning' package which can partly perform reinforcement learning and solve a few simple problems.

However, since the package is experimental, it has to be installed after installing **'devtools'** package first and then installing from GitHub as it is not available in cran repository.

Getting into rough games (Reinforcement learning GitHub package)

```
# Getting into rough games - ReinforcementLearning github package
# install.packages("devtools")
library(devtools)

# Option 1: download and install latest version from GitHub
install_github("nproellochs/ReinforcementLearning")
```

```
library(ReinforcementLearning)
```

If we attempt the same problem using this package, we have to first define a function of actions and states to indicate the possible actions in each state. We also define the reward associated in each state.

This package has this toy example pre-built hence, we just look at the function which should have otherwise been defined.

```
# Viewing the pre-built function for each state, action and reward

print(gridworldEnvironment)

function (state, action)
{
    next_state <- state
    if (state == state("s1") && action == "down")
        next_state <- state("s2")
    if (state == state("s2") && action == "up")
        next_state <- state("s1")
    if (state == state("s2") && action == "right")
        next_state <- state("s3")
    if (state == state("s3") && action == "left")
        next_state <- state("s2")
    if (state == state("s3") && action == "up")
        next_state <- state("s4")
    if (next_state == state("s4") && state != state("s4")) {
        reward <- 10
    }
    else {
        reward <- -1
    }
    out <- list(NextState = next_state, Reward = reward)
    return(out)
}
<environment: namespace:ReinforcementLearning>
```

We now define the names of the states and actions and start solving using the **sampleExperience()** function right away.

```
# Define names for state and action
states <- c("s1", "s2", "s3", "s4")
actions <- c("up", "down", "left", "right")

# Generate 1000 iterations
sequences <- sampleExperience(N = 1000, env = gridworldEnvironment, states = st

#Solve the problem
solver_rl <- ReinforcementLearning(sequences, s = "State", a = "Action", r = "R

#Getting the policy; this may be different for each run
solver_rl$Policy
s1       s2       s3       s4
 "down" "right"    "up"  "down"

#Getting the Reward; this may be different for each run
solver_rl$Reward #-351
```

Here we see that the first three steps are always the same and correct to reach s4. The fourth action is random and can be different for each run

Adapting to the changing environment

The package also has the tic-tac-toe game data generated in it's pre-built library. The data contains about 4 lac rows of steps for tic-tac-toe.

We can directly load the data and perform reinforcement learning on the data.


```
# Conclusion: Adapting to the changing environment
# Load dataset
data("tictactoe")

# Perform reinforcement learning on tictactoe data
model_tic_tac <- ReinforcementLearning(tictactoe, s = "State", a = "Action", r

Since the data is very large, it will take some time to learn. We can then see

# Optimal policy; this may be different for each run
model_tic_tac$Policy #This will print a very large matrix of the possible step

# Reward; this may be different for each run
model_tic_tac$Reward #5449
```

Complete code used in this article

```
# Teaching the child to walk - MDPtoolbox package
# Installing and loading the package
# install.packages("MDPtoolbox")

library(MDPtoolbox)

# 1. Defining the Set of Actions - Left, Right, Up and Down for 2*2 matrix
# Remember! This will be a probability matrix, so we will use the matrix() fur

# Up Action
up=matrix(c( 1, 0, 0, 0,
         0.7, 0.2, 0.1, 0,
         0, 0.1, 0.2, 0.7,
         0, 0, 0, 1),
       nrow=4,ncol=4,byrow=TRUE)

# Down Action
down=matrix(c(0.3, 0.7, 0, 0,
           0, 0.9, 0.1, 0,
           0, 0.1, 0.9, 0,
           0, 0, 0.7, 0.3),
         nrow=4,ncol=4,byrow=TRUE)

# Left Action
left=matrix(c( 0.9, 0.1, 0, 0,
         0.1, 0.9, 0, 0,
         0, 0.7, 0.2, 0.1,
         0, 0, 0.1, 0.9),
       nrow=4,ncol=4,byrow=TRUE)

```

```r
# Right Action
right=matrix(c( 0.9, 0.1, 0, 0,
           0.1, 0.2, 0.7, 0,
           0, 0, 0.9, 0.1,
           0, 0, 0.1, 0.9),
         nrow=4,ncol=4,byrow=TRUE)

# Combined Actions matrix
Actions=list(up=up, down=down, left=left, right=right)

# 2. Defining the rewards and penalties
Rewards=matrix(c( -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              10, 10, 10, 10),
            nrow=4,ncol=4,byrow=TRUE)

# 3. Solving the navigation
solver=mdp_policy_iteration(P=Actions, R=Rewards, discount = 0.1)

# 4. Getting the policy
solver$policy #2 4 1 1
names(Actions)[solver$policy] #"down"  "right" "up" "up"

# 5. Getting the Values at each step. These values can be different in each ru
solver$V #58.25663  69.09102  83.19292 100.00000

# 6. Additional information: Number of iterations
solver$iter #2

# 7. Additional information: Time taken. This time can be different in each ru
solver$time #Time difference of 0.009523869 secs

# Getting into rough games - ReinforcementLearning github package
# install.packages("devtools")

library(devtools)

# Option 1: download and install latest version from GitHub
install_github("nproellochs/ReinforcementLearning")
library(ReinforcementLearning)

# Viewing the pre-built function for each state, action and reward
print(gridworldEnvironment)

# Define names for state and action
states <- c("s1", "s2", "s3", "s4")
actions <- c("up", "down", "left", "right")

# Generate 1000 iterations
sequences <- sampleExperience(N = 1000, env = gridworldEnvironment, states = s

# Solve the problem
solver_rl <- ReinforcementLearning(sequences, s = "State", a = "Action", r = "

# Getting the policy; this may be different for each run
solver_rl$Policy

# Getting the Reward; this may be different for each run
solver_rl$Reward #-351

# Conclusion: Adapting to the changing environment
# Load dataset
data("tictactoe")

```

```
# Perform reinforcement learning on tictactoe data
model_tic_tac <- ReinforcementLearning(tictactoe, s = "State", a = "Action", r

# Optimal policy; this may be different for each run
model_tic_tac$Policy #This will print a very large matrix of the possible step

# Reward; this may be different for each run
model_tic_tac$Reward #5449
```

You can clone this article code in our GitHub.

Reinforcement learning has picked up the pace in the recent times due to its ability to solve problems in interesting human-like situations such as games. Recently, Google's Alpha-Go program beat the best Go players by learning the game and iterating the rewards and penalties in the possible states of the board.

Being human-like makes it associated with behavioral psychology and thus, it gives the opportunity to add human behavior and artificial intelligence to machine learning and include it in one's arsenal of newest technologies.

Conclusion

The field of data science is changing rapidly with so many new methods and algorithms being developed in every field for all purposes. Reinforcement learning is one such technique, though experimental and incomplete, it can solve the problem of completing simple tasks easily.

At present, machines are adept at performing repetitive tasks and solve complex problems easily but cannot solve easy tasks without getting into complexity. This is why, making machines perform simple tasks such as walking, moving hands or even playing tic-tac-toe is very difficult though we, as humans, perform this every day without much effort. With reinforcement learning, these tasks can be trained with an order of complexity.

This article is aimed at explaining the same process of reinforcement learning to data science enthusiasts and open the gates of a new set of learning opportunities with reinforcement.

Follow us:

FACEBOOK| QUORA |TWITTER| GOOGLE+ | LINKEDIN| REDDIT | FLIPBOARD | MEDIUM | GI

I hope you like this post. If you have any questions, then feel free to comment below. If you want me to write on one particular topic, then do tell it to me in the comments below.

Related Courses:

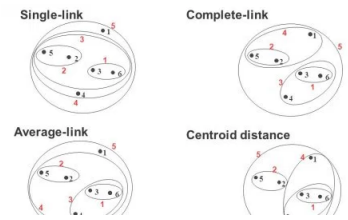
- Practical Reinforcement learning

Author Bio:

This article was contributed by Perceptive Analytics. Madhur Modi, Chaitanya Sagar, Vishnu Reddy and Saneesh Veetil contributed to this article.

Perceptive Analytics provides data analytics, data visualization, business intelligence and reporting services to e-commerce, retail, healthcare and pharmaceutical industries. Our client roster includes Fortune 500 and NYSE listed companies in the USA and India.

Related



How to perform hierarchical clustering in R
January 8, 2018
In "Data Science"

How Q learning can be used in reinforcement learning
February 8, 2018
In "Data Science"

Decision Tree Classifier implementation in R
February 3, 2017
In "Data Science"

R Programming Reinforcement Learning