StatAlgo

Computational Statistics, Machine Learning, et. al.

Subscribe to feed

StatAlgo's blog
StatAlgo's connections' blogs
All site blogs

December 2015
June 2014
May 2014
January 2014
October 2012
August 2012
July 2012
May 2012

Bookmark & Share

statalgo

Computational Statistics, Machine Learning, et. al.

Reinforcement Learning in R: Markov Decision Process (MDP) and Value Iteration

Mon, 06 Jan 2014 20:07:37 GMT

How can we find the best long-term plan?

In the last post, we looked at the idea of dynamic programming, where a problem is divided into overlapping parts, each part is solved independently, and then aggregated back up into a global solution. This is one of the most basic and widely used methods for solving problems that involve *optimal long-term planning*. I am emphasizing the *long-term* part because typical machine learning or statistical models consider the optimal solution to the current state. This short-term solution can lead to sub-optimal behaviors over many steps when there is a *dependency* between each observation. Many statistical models assume *independent* observations, but this is not always a valid assumption.

Reinforcement Learning (RL) is an important but sometimes overlooked method for solving complex planning problems. Many machine learning textbooks ignore this subject entirely, instead focusing strictly on supervised and unsupervised learning. RL grew independently within the Artificial Intelligence (AI) and Operations Research (OR) communities (and is increasingly applied in other fields such as finance and economics).

- AI often deals with problems to do with playing games (e.g. checkers, chess) and robots. These are examples of problems that require taking actions over time to find a long-term optimal solution to a dynamic problem. They are dynamic in the sense that the conditions are constantly changing, sometimes in response to another agent who can be adversarial.
- OR has a subfield known as *optimal control theory*, which covers *dynamic programming* (DP) and *approximate dynamic programming* (ADP). ADP is a very specialized topic which aims to solve problems that cannot be solved directly by DP because of the *curse of dimensionality*.

As with almost all academic disciplines, the lines between these fields has become increasingly fuzzy over time and there is much more interdisciplinary work being done.

WOW – RESEARCHERS TAUGHT A COMPUTER TO BEAT THE WORLD'S BEST HUMANS AT YET ANOTHER TASK. DOES OUR SPECIES HAVE ANYTHING LEFT TO BE PROUD OF?
WELL, IT SOUNDS LIKE WE'RE PRETTY AWESOME AT TEACHING.
HUH? WHAT GOOD IS THAT?

- Environment (E): There is an environment which the agent inhabits. In mathematics, we think about problems existing within a domain. The environment provides the same kind of context to an RL problem, by limiting the scope in some clear way, by defining what states are available, what actions can be taken, and other attributes of the problem.
- State (S): The agent have a state value which changes over time. This is often a *finite discrete variable*, meaning that it can only vary over a few different values, although it is also important that it will sometimes be less constrained.
- Action (A): Given a current state, the agent is allowed to choose an action. Actions are also often defined as *finite discrete variable* as well (a limited range of actions). This limitation is often realistic. For instance, a robot may be limited to only moving north, south, east, or west.
- Reward (R): After taking an action, the agent receives a reward and moves into a new state. The reward value can be anything, including positive or negative values, so it can be used to encourage or discourage behaviors. The reward is thus a function of the state and action chosen.

We will start by limiting both the actions and states to finite discrete values to avoid issues to do with dimensionality.

We add a few more terms to provide more structure to the problem. We will start by considering the most common simplifying case: the markov decision process (MDP). MDP's follow the markov property that the next state only depends on the current state and action. MDP are an extension of markov chains, with the addition of actions and rewards.

- Policy (π): A policy consists of a set of state/action pairs. We may start off in state 1, take action 1, and move to state 2, etc.
- Value (V): the value function, depends on the initial state and policy that is followed. The value of starting in some state s_0 and following a specific policy can thus be represented as $V^\pi(s_0)$. Every state will have an associated value.
- Transition probability matrix (P): This gives the probability of transitioning from one state to another given a certain action. In some cases, this is provided along with the problem definition, while in other cases this is learned.
- Discount (γ): The discount factor can be set to one, but it is generally included in order to limit the number of steps in the policy. This is of great practical importance. Imagine a problem where a robot can take a short path to its target by walking on one path along the edge of a cliff, or take an extremely long path on very safe ground. There is a trade-off in this scenario, and the discount factor allows you to add a time value to the ultimate objective.

The basic MDP is a 5-tuple S, A, P, R, γ , meaning that we need to keep track of these five terms at all times. In the basic setup, P is provided, although alternative versions also allows for this to be a learned value as well.

The fundamental objective of an RL model is to maximize the reward (or value, or utility). This is usually done using the Bellman equation (here shown as a recursion relation):

$$V^*(s) = R(s) + \max_a \gamma \sum_{s'} P(s'|s,a)V^*(s').$$

Where the * represents the optimal policy π .

Lastly, we should touch on one crucial feature of RL problems: exploration vs. exploitation. This is sometimes discussed in the context of a problem known as the multi-armed bandit. A similar statement of this problem is the Feynman Restaurant Problem:

"Richard Feynman and Ralph Leighton had a hard time at restaurants deciding whether to order the best dish they had tried so far or something new, which - who knows - might be even better."

Assume that we start off with no knowledge of the environment (in terms of what rewards are available, and what policies to follow). We start to explore the environment by moving from one state to another, and in the process, we start to learn about what we think are more optimal policies. The question is: at what point should we stop exploring and start to exploit our knowledge. Keep in mind that exploring is costly: we may know a good policy, but choose to ignore it in case we might be missing something better.

Example: Grid World

Let us start by considering one of the most canonical examples: grid world. This is a very simple example, but it can easily be extended and it serves to explore all the building blocks.

Consider a 3x4 grid. An agent (e.g. robot, rat) starts in some state and moves through the grid until it reaches a terminal state, provided by a positive reward (+1) or a negative reward (-1). We can add a discount factor to encourage the robot to reach the objective more quickly.

The robot can move 1 step at a time, and is able to move north, south, east, or west. Unfortunately, the robot isn't perfect, so there is a probability transition function such that any time it wants to move in a particular direction (e.g. north), then there is some small chance that it will instead move to the left or right instead (probability = 0.1 respectively).

Solving an MDP with Value Iteration

Now that we have defined the building blocks of a MDP, let us consider how we might go about explicitly solving the example problem. Following Sutton and Barto (1998), there are three different basic approaches for solving RL problems: dynamic programming, monte carlo, and temporal-difference learning. One of the simplest algorithms for dynamic programming is known as **value iteration**.

Value iteration is very straight forward: it simply repeatedly tries to update the value based on the Bellman equation until it

First, we set up the grid, the possible actions, rewards, and states. (NOTE: This is included as a github gist, which may be viewed here if it doesn't show up embedded in the post.)

```r
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))

# 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
```

Next, we run through the value iteration algorithm (see gist here if not shown below). Here I simply iterate for a set number of

```
    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.st
    }
    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=
            values[state["y"], state["x"]] <- max(q.values)
        }
    }
    return(values)
}

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

> 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
```

gistfile1.txt hosted with ♡ by GitHub view raw

We are now able to get the optimal policy based on these values for a given discount rate.

In future posts, we will study the convergence properties of different RL algorithms, considering alternative methods for solving reinforcement learning problems, including the temporal difference method. we will also walk through approximate methods and solve problems that are less constrained (continuous, infinite). Lastly, we should spend at least one post considering RL models within the PAC framework (Probably Approximately Correct).

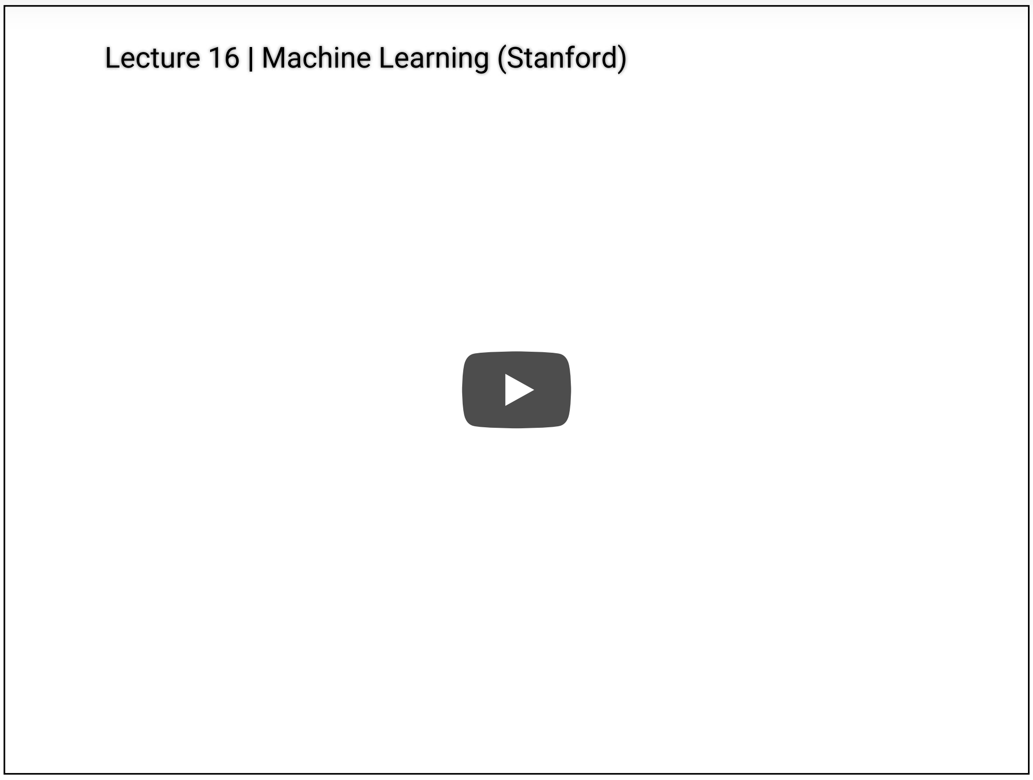
Resources

There are a number of excellent books that provide good introductions to this subject:

- My primary reference is Sutton and Barto (1998) "Reinforcement Learning: An Introduction" (book website, includes free HTML version). This is one of my favorite textbooks; it provides an extremely clear, well structured, and thorough overview of the subject.
- For general machine learning texts, both Tom Mitchell "Machine Learning" and Stephen Marsland "Machine Learning: An Algorithmic Perspective" have chapters on the subject.
- Lastly, Stuart Russell and Peter Norvig "Artificial Intelligence: A Modern Approach" (book site) is the most widely used introductory AI textbook, and it has fairly extensive material on reinforcement learning. This may also have been the first place to introduce this specific example of grid world.

There are also a number of good online courses on this subject. The best that I have seen is Berkeley's CS188 "Artificial Intelligence", which is available as a MOOC from edX.

Also, Andrew Ng's Stanford CS229 "Machine Learning" has a lecture on the subject, including some very good lecture notes.



html, artificial intelligence, rl algorithms, machine learning, value iteration algorithm, simplest algorithms, online courses, finance, short-term solution, iteration algorithm, value iteration algorithm, book site, long-term optimal solution, stuart russell, peter norvig, andrew ng, tom mitchell, ralph leighton, machine learning, statistics, dynamic programming, markov processes, mathematical optimization, stochastic control, markov models, reinforcement learning, markov decision process, bellman equation, temporal difference learning, machine learning, straight forward

Read source article

Related content

Blog: Postdoctoral fellow in machine learning, artificial and collective intelligence
Complexity Digest 1380 days ago

Event: Data Science in Finance - The Final Frontier?
Starting on: Thursday 18th of January, 2018. Posted by: MoneyScience.
In recent years the topics of **Data Science**, **Artificial Intelligence, Machine Learning & Big Data** have become increasingly popular. This growth has been fueled by the collection & availability of new data; continually increasing processing power & storage and the open source movement making tools more widely available. As a result, we have already witnessed profound changes to how we work, rest and play. And this trend will only increase ...But how has the world of finance been impacted and investment managers in particular?Join us to explore what Data Science means for finance professionals.

Event: Frontiers of Factor Investing
Starting on: Monday 23rd of April, 2018. Posted by: MoneyScience.
The Centre for Financial Econometrics, Asset Markets and Macroeconomic Policy (EMP) at Lancaster University Management School and Invesco Quantitative Strategies invite the submission of papers in the field of factor investing and related research areas including:

News: Weekly Top 5 Papers – April 17, 2015
1515 days ago - The SSRN Blog | The SSRN Blog
1. China's Ideological Spectrum by Jennifer Pan (Harvard University – Graduate School of Arts and Sciences) and Yiqing Xu (Massachusetts Institute of Technolog...

Event: 11th Annual Conference on Asia-Pacific Financial Markets (CAFM)
Starting on: Friday 2nd of December, 2016. Posted by: MoneyScience.
The Korean Securities Association (KSA) is pleased to announce its 11th Annual Conference on Asia-Pacific Financial Markets (CAFM) to be held at the Westin Chosun Hotel in Seoul, Korea, from December 2 to December 3, 2016. We welcome interested scholars, practitioners, and policy-makers to submit research papers in all areas of finance for presentation at the conference.