Reinforcement Learning

This project is a competition to find the best policy for three different Markov decision processes given sampled transitions, each consisting of a state (s), action (a), reward (r), and next state (sp). The best policy is the one that maximizes the total expected reward.

Three csv-formatted datasets are provided.

Software requirements

The project runs on Julia 0.6. For full technical rquirements, see:

```
Juliacommand.jl
```

Created Libraries

The following original library was created to solve MDP/OMDP problems:

```
imarkov-solve-grid-maxlikelihood-ONENOW.jl
imarkov-solve-car-ONENOW.jl
imarkov-solve-mystery-ONENOW.jl
```

Addiitionally, the following library was created to experiment with solving MDP/OMDP problems using the POMDPs.jl library:

```
imarkov-solve-grid-probabilistic-POMDP.jl
imarkov-solve-grid-maxlikelihood-POMDP.jl
imarkov-solve-car-POMDP.jl
imarkov-solve-mystery-POMDP.jl
```

The following simulations library was created to run simulations on MDP/POMDP problems:

```
imarkov-simulate-grid-probabilistic.jl
imarkov-simulate-grid-maxlikelihood.jl
imarkov-simulate-car.jl
imarkov-simulate-mystery.jl
```

For file input/outout and dataframe manipulation:

```
imarkov-io.jl
```

Health check

```
julia imarkov-grid-probabilistic-test.jl
julia imarkov-grid-maxlikelihood-test.jl
julia imarkov-car-test.jl
julia imarkov-mystery-test.jl
```

How to run

```
julia imarkov-rl.jl
```

1. Grid World

1a) BONUS -- Completely Observable World: Explicity given probabilistic Transition() and Reward()

Description

Given Transition probability distribution.

```
* 0.7 of moving in the intended direction
* 0.3 divided among all other valid transitions, within grid
```

Given Reward() probability distribution:

```
* a list of reward states
* a list of reward values
```

Thus, no dataset is given.

Approach: Solved with POMDPs.jl

Implementation extends fom http://juliapomdp.github.io/POMDPs.jl/latest/api/

In POMDPs.jl, an MDP is defined by creating a subtype of the MDP abstract type. The types of the states and actions for the MDP are declared as parameters of the MDP type. For example, if our states and actions are both represented by integers we can define our MDP type in the following way:

```
type MyMDP <: MDP{Int64, Int64} # MDP{StateType, ActionType}
```

where MyMDP is a subtype from an abstract MDP type defined in POMDPs.jl.

For a full tutorial see: http://nbviewer.jupyter.org/github/JuliaPOMDP/POMDPs.jl/blob/master/examples/GridWorld.ipynb

Chose to implement distributions as SparseCat

```
SparseCat(values,probabilities)
```

Initially developed GridWold with defined Transition and Rewards distributions

```
SIMULATION: startingState is: BuenosAiresState(7, 1, false)....Action: right
==> AT BuenosAiresState(7, 1, false) ACTION right HAS:
         DISTRIBUTION OF STATES IS BuenosAiresState[BuenosAiresState(7, 2, false), BuenosAiresState(6, 1,
false), BuenosAiresState(8, 1, false)]
         PROBABILITY DISTRIBUTION IS [0.15, 0.15, 0.7]
```

Identified the methods necessary to implement in order to leverage POMDPs.jl as a library

```
INFO: POMDPs.jl requirements for solve(::ValueIterationSolver, ::Union{POMDPs.MDP,POMDPs.POMDP}) and
dependencies. ([✔] = implemented correctly; [X] = missing)

For solve(::ValueIterationSolver, ::Union{POMDPs.MDP,POMDPs.POMDP}):
  [✔] discount(::ProbabilisticGrid)
  [✔] n_states(::ProbabilisticGrid)
  [✔] n_actions(::ProbabilisticGrid)
  [✔] transition(::ProbabilisticGrid, ::ProbabilisticState, ::Symbol)
  [✔] reward(::ProbabilisticGrid, ::ProbabilisticState, ::Symbol, ::ProbabilisticState)
  [✔] state_index(::ProbabilisticGrid, ::ProbabilisticState)
  [✔] action_index(::ProbabilisticGrid, ::Symbol)
  [✔] actions(::ProbabilisticGrid, ::ProbabilisticState)
  [✔] iterator(::Array)
  [✔] iterator(::Array)
  [✔] iterator(::SparseCat)
  [✔] pdf(::SparseCat, ::ProbabilisticState)
For ordered_states(::Union{POMDPs.MDP,POMDPs.POMDP}) (in solve(::ValueIterationSolver,
::Union{POMDPs.MDP,POMDPs.POMDP})):
  [✔] states(::ProbabilisticGrid)
For ordered_actions(::Union{POMDPs.MDP,POMDPs.POMDP}) (in solve(::ValueIterationSolver,
::Union{POMDPs.MDP,POMDPs.POMDP})):
  [✔] actions(::ProbabilisticGrid)
```

Performance of the ValueIterationSolver() with explicit Transition() and Reward()

```
[Iteration 24  ] residual:   0.000934 | iteration runtime:      0.261 ms, (    0.0135 s total)
```

Determined total utility

```
TOTAL discounted reward: 3.8742048900000015
```

Determined policy from utility

```
FROM POLICY => s:BuenosAiresState(9, 2, false) BEST a:up
FROM POLICY => s:BuenosAiresState(9, 2, false) BEST a:up
```

Implemented history replay

```
REPLAY s: BuenosAiresState(8, 2, false)  a: right   s': BuenosAiresState(9, 2, false)
REPLAY s: BuenosAiresState(9, 2, false)  a: up      s': BuenosAiresState(9, 3, false)
```

1b) Uncertain GRID: Estimable Transition() and Reward()

Description

- Grid world (100 states: 10x10) with 4 actions. Use sub2ind((10, 10), x, y) to find the integer state from the x and y coordinates.
- Actions are 1:left, 2:right, 3:up, 4:down.
- The discount factor is 0.95.
- There are no terminal states in the small problem.

Dataset

Generated or sampled data:

```
small.csv
```

Sample:

```
"s","a","r","sp"
12,3,0,22
22,4,0,12
12,2,0,11
11,3,0,21
21,1,0,21
21,4,0,11
11,3,0,1
1,3,0,11
11,4,0,12
```

The Ro Distribution has high rewards in areas of the grid:

```
0.0 0.0 0.0 0.0; 378.0 339.0 366.0 402.0; 0.0 0.0..
```

And rather even distributed visit counts:

```
130.0 142.0 139.0 128.0; 130.0 111.0 131.0 128.0;....
```

Approach: Max Likelihood Model-based Transition() and Reward()

Model-based methods tend to be more sample efficient. In other words, given a small number of observed transitions, model-based methods tend to perform better.

Chose to implement distributions like SparseCat in the Solver

At every step, the solver updates visit counts N(s,a) and Ro(s,a)

```
SparseCat(N, Ro)

where N[s,a,s']

and Ro[si,ai] <- Ro[si,ai] + r
```

And thus every step it updates the qValue table:

```
qValue(s,ai) <- Ro(s,ai)*T(s'|s,ai)

where T(s'|s,ai) <- N(s,a,s')/N(s,a)
```

The qTable tells the story of where to go next:

```
[641.0 587.0 545.0 630.0; 584.0 570.0 565.0 444.0; 720.0 669.0 611.0 610.0; 797.0 590.0 736.0 735.0; 663.0
542.0 573.0 604.0; 581.0 620.0 543.0 541.0; 456.0 592.0 594.0 604.0; 525.0 655.0 546.0 579.0; 520.0 626.0 638.0
597.0; 595.0 685.0 622.0 660.0; 476.0 556.0 494.0 535.0; 550.0 518.0 656.0 490.0; 599.0 621.0 590.0 555.0;
614.0 583.0 638.0 705.0; 535.0 570.0 618.0 520.0; 565.0 637.0 662.0 530.0; 707.0 591.0 608.0 536.0; 600.0 693.0
575.0 695.0; 618.0 658.0 604.0 604.0; 615.0 576.0 619.0 709.0; 569.0 531.0 487.0 471.0; 551.0 649.0 631.0
672.0; 638.0 561.0 618.0 662.0; 604.0 608.0 540.0 685.0; 618.0 654.0 552.0 590.0; 646.0 633.0 613.0 601.0;
630.0 616.0 679.0 634.0; 720.0 595.0 590.0 685.0; 541.0 781.0 694.0 546.0; 730.0 676.0 682.0 625.0; 530.0 551.0
532.0 469.0; 601.0 667.0 635.0 642.0; 626.0 764.0 598.0 649.0; 687.0 661.0 605.0 566.0; 621.0 642.0 645.0
643.0; 608.0 716.0 591.0 758.0; 646.0 718.0 699.0 632.0; 654.0 563.0 655.0 640.0; 643.0 601.0 695.0 702.0;
657.0 574.0 710.0 714.0; 516.0 503.0 601.0 502.0; 475.0 603.0 607.0 630.0; 555.0 717.0 582.0 641.0; 726.0 567.0
656.0 611.0; 567.0 681.0 654.0 570.0; 639.0 566.0 695.0 649.0; 606.0 579.0 660.0 654.0; 548.0 653.0 675.0
597.0; 611.0 693.0 635.0 676.0; 771.0 777.0 812.0 611.0; 703.0 598.0 602.0 620.0; 586.0 689.0 675.0 610.0;
614.0 650.0 721.0 682.0; 696.0 691.0 633.0 786.0; 669.0 593.0 631.0 599.0; 663.0 573.0 608.0 718.0; 642.0 650.0
556.0 641.0; 618.0 631.0 680.0 669.0; 624.0 592.0 676.0 655.0; 687.0 732.0 541.0 726.0; 551.0 667.0 592.0
614.0; 583.0 550.0 639.0 661.0; 570.0 501.0 604.0 649.0; 655.0 646.0 604.0 666.0; 715.0 643.0 625.0 621.0;
700.0 765.0 551.0 608.0; 599.0 676.0 590.0 620.0; 577.0 624.0 641.0 566.0; 568.0 609.0 612.0 706.0; 541.0 677.0
579.0 714.0; 671.0 639.0 639.0 570.0; 695.0 610.0 669.0 567.0; 623.0 649.0 631.0 591.0; 591.0 611.0 638.0
608.0; 612.0 596.0 597.0 642.0; 644.0 631.0 628.0 720.0; 720.0 646.0 656.0 629.0; 606.0 548.0 614.0 625.0;
628.0 529.0 581.0 615.0; 491.0 524.0 581.0 617.0; 646.0 675.0 639.0 600.0; 720.0 682.0 630.0 713.0; 686.0 682.0
677.0 681.0; 674.0 665.0 649.0 648.0; 748.0 626.0 625.0 644.0; 698.0 596.0 622.0 672.0; 695.0 645.0 595.0
647.0; 673.0 584.0 615.0 712.0; 593.0 616.0 693.0 589.0; 544.0 525.0 642.0 486.0; 666.0 597.0 686.0 715.0;
679.0 669.0 671.0 609.0; 676.0 620.0 667.0 659.0; 612.0 676.0 568.0 654.0; 672.0 720.0 698.0 649.0; 611.0 719.0
691.0 617.0; 630.0 592.0 641.0 644.0; 558.0 516.0 574.0 664.0; 598.0 561.0 536.0 613.0; 659.0 590.0 652.0
514.0]
```

Then at policy time, for a given state, the full range of available actions is evaluated. The one with the highest qValue is chosen:

Performance of created Solver

Wote a MaxLikelihoodSolver on the ideas of DAU 5.2. It in seconds since it is a small dataset.

Note: T(s'|s,ai) in that algorithm is over-estimating reward from transitioning to a valued state.

Sample recommendations:

```
1 ACTION RECOMMENDED FOR s=61
```

```
2 ACTION RECOMMENDED FOR s=62
1 ACTION RECOMMENDED FOR s=63
1 ACTION RECOMMENDED FOR s=64
1 ACTION RECOMMENDED FOR s=65
1 ACTION RECOMMENDED FOR s=66
1 ACTION RECOMMENDED FOR s=67
3 ACTION RECOMMENDED FOR s=68
1 ACTION RECOMMENDED FOR s=69
1 ACTION RECOMMENDED FOR s=70
1 ACTION RECOMMENDED FOR s=71
1 ACTION RECOMMENDED FOR s=72
1 ACTION RECOMMENDED FOR s=73
2 ACTION RECOMMENDED FOR s=74
1 ACTION RECOMMENDED FOR s=75
1 ACTION RECOMMENDED FOR s=76
1 ACTION RECOMMENDED FOR s=77
4 ACTION RECOMMENDED FOR s=78
1 ACTION RECOMMENDED FOR s=79
1 ACTION RECOMMENDED FOR s=80
1 ACTION RECOMMENDED FOR s=81
1 ACTION RECOMMENDED FOR s=82
1 ACTION RECOMMENDED FOR s=83
1 ACTION RECOMMENDED FOR s=84
1 ACTION RECOMMENDED FOR s=85
1 ACTION RECOMMENDED FOR s=86
1 ACTION RECOMMENDED FOR s=87
4 ACTION RECOMMENDED FOR s=88
```

References: Grid World

```
https://github.com/JuliaStats/Distributions.jl
https://github.com/JuliaPOMDP/POMDPModels.jl/blob/master/notebooks/GridWorld%20Visualization.ipynb
http://nbviewer.jupyter.org/github/sisl/POMDPs.jl/blob/master/examples/DeterministicGrid.ipynb
https://github.com/JuliaPOMDP/POMDPs.jl/blob/master/examples/DeterministicGrid.ipynb
https://github.com/JuliaPOMDP/POMDPModels.jl/blob/master/src/GridWorlds.jl
```

2) Mountain Car

Description

Like MountainCarContinuous-v0:

```
* See Open AI Gym https://github.com/openai/gym/wiki/MountainCarContinuous-v0 with altered parameters.
* Discretized State measurements are given by integers with 500 possible position values and 100 possible
velocity values (50,000 possible state measurements).
* 1+pos+500*vel is the formula that was used to discretize to integer states, utilizing position pos and
velocity vel.
* There are 7 actions that represent different amounts of acceleration.
* This problem is undiscounted, but ends when the goal (the flag) is reached. Note that, since the discrete
state measurements are calculated after the simulation, the data in medium.csv does not quite satisfy the
Markov property.
```

Dataset

Generated or sampled data:

```
medium.csv
```

Sample:

```
"s","a","r","sp"
24715,1,-225,24214
24214,1,-225,23713
23713,1,-225,23212
23212,1,-225,22711
22711,1,-225,22709
22709,1,-225,22207
22207,1,-225,21705
21705,1,-225,21703
21703,1,-225,21200
```

Glitch: Rewards are not stochastic, i.e. for a given (s, a, sp), r is deterministic. But in medium.csv for (s, a, sp) = (30457, 5, 30461), there are two different rewards: -25 and 99975. You can see this in lines 370, 5829, and 21591.

Approach: Model-Free, operating without Transition() and Reward()

Since his problem is is high-dimensional with sparse-data, I choose instead to apply model-free Reinforcement Learning.

In addition, since Reward() is very sparse, for faster back-propagation of rewards, the best algorithm choice was using Eligibility Traces: with SARSA Lambda.

Wrote specific model-free solver that feeds on the MDP's dataset:

```
* SARSALambdaSolverONENOW(mdp, learning_rate=0.1, lambda=0.9, n_episodes=5000, max_episode_length=50,
eval_every=50, n_eval_traj=100)
```

Followed the same interface as POMDPs.jl:

```
policy = solve(solver, mdp)
```

Also sketched out implementations to:

```
* QLearningSolverONENOW(mdp, learning_rate=0.1, n_episodes=5000, max_episode_length=50, eval_every=50,
n_eval_traj=100)
* SARSASolverONENOW(mdp, learning_rate=0.1, n_episodes=5000, max_episode_length=50, eval_every=50,
n_eval_traj=100)
```

Say Hello to My New Baby: SarsaLambda

In the middle of doing its elligibility traces for Mountain car:

```
BACKPROPAGATING: esi=23215 eai=1 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.76
BACKPROPAGATING: esi=23216 eai=4 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.76
BACKPROPAGATING: esi=23218 eai=2 r=99775 delta=99775.0 Q[esi,eai]=849507.2884951112
ecounts[esi,eai]=8.591599328603509e-8 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23218 eai=5 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
```

```
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23219 eai=1 r=99775 delta=99775.0 Q[esi,eai]=849507.2884951112
ecounts[esi,eai]=8.591599328603509e-8 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23219 eai=5 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23220 eai=4 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23220 eai=5 r=99775 delta=99775.0 Q[esi,eai]=849507.2884951112
ecounts[esi,eai]=8.591599328603509e-8 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23221 eai=3 r=99775 delta=99775.0 Q[esi,eai]=1.0990076707841489e6
ecounts[esi,eai]=2.6961349066755114e-8 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23221 eai=6 r=99775 delta=99775.0 Q[esi,eai]=508270.2604738388
ecounts[esi,eai]=9.116688771198934e-11 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23222 eai=1 r=99775 delta=99775.0 Q[esi,eai]=813367.7940710045
ecounts[esi,eai]=1.2699142395579657e-9 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23223 eai=2 r=99775 delta=99775.0 Q[esi,eai]=803357.966736839
ecounts[esi,eai]=1.0286305340419522e-9 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23223 eai=6 r=99775 delta=99775.0 Q[esi,eai]=849507.2884951112
ecounts[esi,eai]=8.591599328603509e-8 2017-11-06T06:51:29.761
BACKPROPAGATING: esi=23224 eai=2 r=99775 delta=99775.0 Q[esi,eai]=917782.7190920939
ecounts[esi,eai]=1.160597671060218e-8 2017-11-06T06:51:29.762
BACKPROPAGATING: esi=23225 eai=3 r=99775 delta=99775.0 Q[esi,eai]=813367.7940710045
ecounts[esi,eai]=1.2699142395579657e-9 2017-11-06T06:51:29.762
BACKPROPAGATING: esi=23225 eai=5 r=99775 delta=99775.0 Q[esi,eai]=917782.7190920939
ecounts[esi,eai]=1.160597671060218e-8 2017-11-06T06:51:29.762
BACKPROPAGATING: esi=23226 eai=1 r=99775 delta=99775.0 Q[esi,eai]=381824.98805100546
ecounts[esi,eai]=1.5204067513659465e-11 2017-11-06T06:51:29.765
```

Performance of the Solver

With these parameters, I stopped the learning process after an hour, it had not finished:

```
function getSarsaLambdaSolver(mdp::MountainCar;
            learning_rate::Float64=0.1,
            lambda::Float64=0.9,
            n_episodes::Int64=5000,
            max_episode_length::Int64=50,
            eval_every::Int64=50,
            n_eval_traj::Int64=100,
            Q_vals::Matrix{Float64}=qTable,                         # qTable
            eligibility::Matrix{Float64}=visitCount)                # visit count N
    return SARSALambdaSolverONENOW(learning_rate, lambda, n_episodes, max_episode_length, eval_every,
n_eval_traj, qTable, visitCount)
end
```

With the following changes, learning of the medium dataset had finished within 5 minutes:

```
0)  Backpropagated large qValue only
    minPropagationThreshold = 1000
    # minPropagationThreshold = 2
1)  Fewer episodes:
    numberOfEpisodes = solver.n_episodes
    # numberOfEpisodes = length(dataset[stateKey])
2)  Shorter episodes
    maxEpisodeLengthTest = 5
    #max_episode_length::Int64=50,
```

Policy evaluation for the entire state space takes a few minutes, due to the large number of unseen states. Pending of course approximation for unseen states. Sample policy recommendations:

```
1 ACTION RECOMMENDED FOR s=35334
6 ACTION RECOMMENDED FOR s=35335
1 ACTION RECOMMENDED FOR s=35336
1 ACTION RECOMMENDED FOR s=35337
1 ACTION RECOMMENDED FOR s=35338
1 ACTION RECOMMENDED FOR s=35339
1 ACTION RECOMMENDED FOR s=35340
1 ACTION RECOMMENDED FOR s=35341
1 ACTION RECOMMENDED FOR s=35342
1 ACTION RECOMMENDED FOR s=35343
1 ACTION RECOMMENDED FOR s=35344
1 ACTION RECOMMENDED FOR s=35345
1 ACTION RECOMMENDED FOR s=35346
1 ACTION RECOMMENDED FOR s=35347
7 ACTION RECOMMENDED FOR s=35348
1 ACTION RECOMMENDED FOR s=35349
1 ACTION RECOMMENDED FOR s=35350
1 ACTION RECOMMENDED FOR s=35351
3 ACTION RECOMMENDED FOR s=35352
```

Note for the future: try a logarithmic search over different values (i.e. 10−1, 10−2, etc.) to find a good learning rate.

Data in the qTable is very space, additional compute time for backpropagation would vastly improve the policy's performance:

```
503533.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0
0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0
0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0
0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0
0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.0 0.0 1.2825e5 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0
0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 138912.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
0.0 0.0 0.0 0.0 0.0; 0.0 0.0 1.2825e5 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0
0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0
0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 138912.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0
0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 1.2825e5 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0
0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0
0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0
0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 1.2825e5 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0
0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 138912.0 0.0 0.0; 0.0 0.0 0.0
0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0
0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0
0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0 138912.0
```

I found Julia on this task to be CPU bound, consuming a whole processor, and not parallelizing beyond. Memory consumption seemed capped at 20gb for MysteryWorld.

Generalization--with known problem structure: estimation for states unseen in the dataset

Given that the discretized state of in this problem embodies two dimensions, position and velocity, it should be possible to find k-nearest possible neighbors to any given state.

It seems reasonable to apply Local Approximation, where the key assumption is that states close to each other enjoy similar utility--over continous space.

The discretized mountain car values:

```
* as formulated, state: 1+position+500*velocity
* position: 1 to 500, contributing 1 to 500 to the state
* velocity: 1 to 100, contributing 1 to 50000 to the state
* thus, velocity is a bigger determinant o state by 100x
```

Given the weight of velocity, for simplicity a single-dimmensional approximation may be attempted, for any unseen state like 23095.

The following alogorithm sketch could be applied for Exact Local Approximation to the utility of any unseen target_state:

```
stateAction[]                                       # all discrete (state,action) tuples under
consideration
for velocity 1:100                                  # all possible discrete velocities
    for action 1:7                                  # There are 7 actions that represent different amounts
of acceleration
        (v,a) <- (velocity,action)                  # approximateState: 500*velocity.  Position ignored per
above: 1+position
        stateAction.append[(v,a)]
qtable <- qlearning of (s,a,s',r) dataset
stateDelta{}                                        # stores distance from target_state to every other
state
for (s,a) in qtable                                 # map over all observed in the dataset
    deltaBetweenStates <- s-500*v                   # eucledian distance from actual state to velocity
contribution: 500*velocity
    stateDelta[(s,a)] <- deltaBetweenStates         # between target_state and every other state
sortedDetta <- sort(stateDelta)                     # states sorted by lower deltaBetweenStates
return sortedDetta[1]                               # nearest neghbor value in qtable
```

A lighter computation for Estimated Local Approximation:

```
1) Search the qTable for the nearest state by id that has non-zero qValues.
2) Act according to those qValues: argmax(a), choosing the action estimated to lead to the highest value.
```

References: Mountain Car

```
* https://github.com/JuliaStats/Distributions.jl
* https://github.com/JuliaPOMDP/POMDPs.jl#reinforcement-learning
* https://github.com/JuliaPOMDP/POMDPModels.jl/blob/master/src/MountainCar.jl
* https://github.com/openai/gym/blob/master/gym/envs/classic_control/continuous_mountain_car.py
```

3) Mystery: It's a secret!

Meant to be a challenge, blindly applying reinforcement learning:

```
* MDP with 10101010 states
* 125 actions
* discount factor of 0.95
* Each line in the CSV file represents a transition from state (s) to (sp) (along with the associated reward
(r)), when taking an action (a)
* You might not have data for every state. Neither are all states equally important. Use your best judgement in
handling this
```

Applying q-learning or Sarsa is sufficient. But that is not likely to yield a good policy. Look at the state indices and their transitions. There's a structure to the large problem.

Dataset

Generated or sampled data:

```
large.csv
```

Sample:

```
"s","a","r","sp"
6494120,52,0,6394027
6394027,62,0,6493027
6493027,104,0,6493036
6493036,72,0,6593035
6593035,100,0,6693031
6693031,13,0,7692027
7692027,31,0,6682127
6682127,73,0,6582115
6582115,63,0,7581117
```

Approach: Model-Free, operating without Transition() and Reward()

Since his problem appears to be is high-dimensional with sparse-data, I initially choose instead to apply model-free Reinforcement Learning.

Wrote model-free solver optimized for fast back-propagation:

```
* SARSALambdaSolverONENOW(mdp, learning_rate=0.1, lambda=0.9, n_episodes=5000, max_episode_length=50,
eval_every=50, n_eval_traj=100)
```

Followed the same interface as POMDPs.jl:

```
policy = solve(solver, mdp)
```

Used a semi-Bayesian approach with Laplace smoothing, using 1 as initial pseudocount for N parameters.

The standard approach is to treat it as 0, which feels a bit weird since P(s'|s,a) would not sum to 1. Looking at some of the classic texts (e.g., Kaelbling, Littman, and Moore), they don't mention this issue. Of course 0/0 is undefined, but this is treated as 0 in this context.

Once you have determined your transition and reward functions, you can just run value iteration to find your policy.

Performance of the Solver

Because of the large number of states, and the large number of possible actions, Sarsa Lambda's most critical parameter is the backpropagating threshood:

```
minPropagationThreshold = 50     # 5000
if abs(delta)>minPropagationThreshold        # something to propagate
```

When that threshold is changed from 50 to 10, the large dataset goes from finishing in minutes, to hours.

After all, Sarsa Lambda backpropagates for every state and every action:

```
for es in 1:10101010   # TODO: getNumMysteryStates(mdp)
    for ea in 1:125     # TODO: getNumMysteryActions
        esi, eai = es, ea
        #esi, eai = getMysteryStateIndex(mdp, es), getCarActionIndex(mdp, ea)
        Q[esi,eai] += solver.learning_rate * delta * ecounts[esi,eai]
        ecounts[esi,eai] *= mdp.discount * solver.lambda                    # exponential decay
        if abs(Q[esi,eai])>0
            print("BACKPROPAGATING2: ", "esi=", esi, " eai=", eai, " r=", r, " delta=", delta, " Q[esi,eai]=",
Q[esi,eai], " ecounts[esi,eai]=", ecounts[esi,eai], " ", now(), "\n")
        end
    end
end
```

Policy evaluation for the entire state space takes a several minutes. Pending of course approximation for unseen states.

NOTE As of the printing of this writeup, I put the Samza Lambda in full back propagation mode, and it has not finished generating the updated policy:

```
BACKPROPAGATING2: esi=6416903 eai=84 r=-15 delta=-15.0 Q[esi,eai]=-8.13 ecounts[esi,eai]=1.4580000000000002
2017-11-06T13:19:49.066
BACKPROPAGATING2: esi=6417027 eai=119 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.071
BACKPROPAGATING2: esi=6418036 eai=36 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.126
BACKPROPAGATING2: esi=6419017 eai=37 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.178
BACKPROPAGATING2: esi=6425165 eai=66 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.467
BACKPROPAGATING2: esi=6426257 eai=39 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.514
BACKPROPAGATING2: esi=6428337 eai=81 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:49.617
BACKPROPAGATING2: esi=6431206 eai=90 r=-15 delta=-15.0 Q[esi,eai]=-11.781319669797732
ecounts[esi,eai]=0.4117822641892982 2017-11-06T13:19:49.744
BACKPROPAGATING2: esi=6433093 eai=15 r=-15 delta=-15.0 Q[esi,eai]=-8.13 ecounts[esi,eai]=1.4580000000000002
2017-11-06T13:19:49.834
BACKPROPAGATING2: esi=6433205 eai=34 r=-15 delta=-15.0 Q[esi,eai]=-8.13 ecounts[esi,eai]=1.4580000000000002
```

```
2017-11-06T13:19:49.839
BACKPROPAGATING2: esi=6434987 eai=87 r=-15 delta=-15.0 Q[esi,eai]=-11.781319669797732
ecounts[esi,eai]=0.4117822641892982 2017-11-06T13:19:49.937
BACKPROPAGATING2: esi=6437267 eai=12 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:50.084
BACKPROPAGATING2: esi=6441190 eai=103 r=-15 delta=-15.0 Q[esi,eai]=-11.781319669797732
ecounts[esi,eai]=0.4117822641892982 2017-11-06T13:19:50.264
BACKPROPAGATING2: esi=6444147 eai=105 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:50.402
BACKPROPAGATING2: esi=6446783 eai=30 r=-15 delta=-15.0 Q[esi,eai]=-8.13 ecounts[esi,eai]=1.4580000000000002
2017-11-06T13:19:50.521
BACKPROPAGATING2: esi=6448042 eai=5 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:50.579
BACKPROPAGATING2: esi=6454142 eai=47 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:50.868
BACKPROPAGATING2: esi=6457207 eai=31 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.014
BACKPROPAGATING2: esi=6458045 eai=110 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.052
BACKPROPAGATING2: esi=6459103 eai=37 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.106
BACKPROPAGATING2: esi=6459147 eai=23 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.108
BACKPROPAGATING2: esi=6461492 eai=118 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.232
BACKPROPAGATING2: esi=6462051 eai=61 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.258
BACKPROPAGATING2: esi=6462097 eai=67 r=-15 delta=-15.0 Q[esi,eai]=-8.13 ecounts[esi,eai]=1.4580000000000002
2017-11-06T13:19:51.26
BACKPROPAGATING2: esi=6462486 eai=96 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.279
BACKPROPAGATING2: esi=6470057 eai=62 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.659
BACKPROPAGATING2: esi=6471493 eai=110 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.734
BACKPROPAGATING2: esi=6472057 eai=32 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.766
BACKPROPAGATING2: esi=6472093 eai=27 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.768
BACKPROPAGATING2: esi=6475787 eai=23 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:51.973
BACKPROPAGATING2: esi=6477871 eai=51 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:52.086
BACKPROPAGATING2: esi=6484207 eai=12 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:52.385
BACKPROPAGATING2: esi=6485107 eai=101 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:52.437
BACKPROPAGATING2: esi=6485887 eai=33 r=-15 delta=-15.0 Q[esi,eai]=5.098425470000001
ecounts[esi,eai]=0.7748409780000003 2017-11-06T13:19:52.482
```

Generalization--with unknown problem structure: estimation for states unseen in the dataset

Given that this is a mystery problem, it wasn't immediately evident how/if to consider (state,action) tuples to be similar. One possible approach is to plot Q over (s,a) and to visualize the structure of the problem. It is conceivable that linear regression over (s,a) provides practicable Global Approximation. In the limit, a multi-stage perceptron, potentially constructing a deep neural network may prove to be the best approach.

In the interim, for the unobserved states, the policy was set to a random value from the populated Q table, reflecting some uniformity of distribution.

Upon a further at the state indices, there is perhaps structure to this problem. Perhaps another approach would be to model the Transitions()

References: Mystery

```
* Decisions Under Uncertainty, by Mykel J. Kochenderfer, MIT Press
```

Future work: addressing slowness of Reinforcement Learning rate

Especially in sparcely-rewarded worlds, Q-Lerning and Sarsa can be slow to discover reward. And then take extensive further iterations to back-propagate rewards.

That is the case in my final project, for which I intend to implement eligibility traces with Sarsa Lambda.

Also, take a look at Frobenius norm of change in Q values, with respect to reinforcement learning iterations.

And POMPD.jl for Partially Observable Markov Decision Problems!

```
http://nbviewer.jupyter.org/github/sisl/POMDPs.jl/blob/master/examples/Tiger.ipynb
```

References

- Further references are mentioned in each Julia code file.

References: General

These ones are broadly applicable:

```
* http://juliapomdp.github.io/POMDPs.jl/latest/
* https://github.com/JuliaStats/Distributions.jl
* https://github.com/JuliaPOMDP/POMDPs.jl#mdp-solvers
* https://github.com/JuliaPOMDP/POMDPToolbox.jl#simulators
```

Rules

You can use any programming language but you cannot use any package directly related to reinforcement learning. You can use general optimization packages and libraries for things like function approximation and ODE solving so long as you discuss what you use in your writeup and make it clear how it is used in your code.

You can use different approaches and algorithms for the different problems. It will probably help to look at the data carefully and use the structure of the problem to your advantage.

Discussion is encouraged but you must write your own code. Otherwise, it violates the honor code.

You may look at the code for the MountainCarContinuous-v0 environment (Links to an external site.)Links to an external site., but note that we do not use exactly the same parameters.

Submission

Use the button at the bottom which says Load Project 2 - Reinforcement Learning to go to Vocareum.

You should find the three CSV files: small.csv, medium.csv and large.csv in your workspace.

Either use Vocareum's editor to work on the project. Or upload your code along with the following files to your workspace:

Required items

Input Size, including heather row:

```
$ wc -l small.csv
   50001 small.csv
$ wc -l medium.csv
  100001 medium.csv
$ wc -l large.csv
 1000001 large.csv
```

Output:

Your program should output a file with the same name as the input filename, but with a .policy extension, e.g., "small.policy".

Each line in your policy file represents the argmax() action to take at any of all the valid states.

Each output file should contain an action for every possible state in the problem, wether it is visible in the dataset or not.

The ith row contains the action to be taken from the ith state. small.policy should contain 100 lines, medium.policy should contain 50000 lines, and large.policy should contain 10101010 lines. If using Linux/MAC, you can check the number of lines using wc -l e.g. wc -l small.policy.

```
* writeup.pdf: The writeup.pdf should briefly describe your strategy. This should not be more than 1 or 2 pages
with description of your algorithm and their performance characteristics. Remember to include running/training
time for each policy.
* small.policy should be 100 rows
* medium.policy should be 50000 rows
* large.policy should be 10101010 rows
```

Make sure the policy file length is correct:

```
print the output of `head -n 20 large.policy` and `tail -n 20 large.policy`
```

Click on the Submit button. It will complain if any files are missing and tell you your score for each .policy file. It will also submit your .policyfiles for evaluation. It takes a while to evaluate the policies so unless some urgent issue with the leaderboard, wait at least 20 minutes for the leaderboard to update.

FAQ

What does the score on the scoreboard represent? For each problem, the score is the expected value for a state drawn from initial distribution D, that is E[s]~D[U(s)]

For all problems, we evaluate this this expectation using simulations. If the scores are too close in the end we'll increase the number of Monte Carlo simulations. What's the reward? All we can say is that the reward is a function of both the states and actions. What packages can we use?

For the computational part sparse matrix libraries (builtin in numpy and Julia) could be useful. Feel free to use any DeepLearning and Machine Learning package as well (as long as you are not using any RL implementations therein.) But a lot of effort would go into building a model for the data. So data munging libraries like DataFrames.jl, Pandas for python, graphing libraries like StatPlots.jl might be useful. This tool was successfully loaded in a new browser window. Reload the page to access the tool again.