Mdpmf Documentation

Version: 1.5

Updated On: June 18, 2018

Author: Michelle M. LaMar

Mdpmf is a program that will generate simulated play records, estimate model parameters and estimate person parameters for MDP-modeled performance data. The program is run from command line and has a number of different options that configure its behavior.

Allowed options:

-h [ --help ] produce help message

--genData generate simulated play data from specified models

--estModelParams estimate the model parameters, taking person

parameters as random

--estPersonParams estimate the person parameters, taking the

model parameters as fixed

--compileModel initialize and compile a model for faster

loading on later runs

-t [ --actionEffectFile ] arg the table of actions and their effects, csv

format with headers

-w [ --worldEffectFile ] arg the table of actions and their effects, csv

format with headers

-r [ --rewardFile ] arg the table of reward parameters and the

conditions that trigger the rewards, csv

format, no headers

--compModelFile arg a compiled model file, output from the

--compileModel command

-d [ --discount ] arg set the discount parameter. Must be between 0

and 1 [default 1.0]

-o [ --dataOutputFile ] arg the file to which the generated data will be written

-i [ --dataInputFile ] arg the play records from which to estimate parameters

-p [ --personParamFile ] arg the file containing specifications or estimates

of the person parameters

-m [ --modelParamFile ] arg the file containing specifications or estimates

of the model parameters

-c [ --recActionChoice ] record action choice probabilities

-v [ --recRewardValues ] output the rewards received for each action

(used with genData)

--useMLE use Maximum Likelihood estimation for the

person parameters

--useMAP use Maximum A-Posteriori estimation for the

person parameters

--maxIters arg set the maximum iterations in the policy

optimization

--noPruneStateSpace do not attempt to prune the state space to only

reachable states

--stopAction arg specify stop action [default: ‘End Mission’]

--paramEstSpecsFile arg file containing parameter estimation specifications

--modelSpecConfFile arg file containing model specification configuration information

--fast run quickly, sacrificing some accuracy and skip

error calculations; good for testing

--verbose give per record updates on processing

Two configuration files are needed to specify the MDP model. The “action effect table” defines the available action set, the state variables, and the probabilistic effect of the available actions on the state variables (transitions). The “rewards table” specifies the reward structure in terms of which actions and state conditions trigger rewards or costs and the quantity of those rewards. For data simulation an optional “real world” action effect table may be provided if the cognitive model differs from the actual model. In this document, we will first describe how these files define the MDP model. Then we will describe how each different mode works: data simulation, model parameter estimation, and person capability estimation. Finally we will conclude with some technical notes.

# Defining the MDP Model

An MDP model is defined by the state space (S), action set (A), transition function (T) and reward structure (R). For the *mdpmf* MDP, we define the state space based on a set of finite, discrete state variables. The full state space consists of all possible combinations of the state variable values. The action set is a named list of possible actions. Both the state space and the action set are defined in the ActionEffect table.

As a running example, we will describe a toy problem called “Fishing” (available in the examples directory). In this example an agent is given a limited time in which they may choose at each time step to fish or study. Each *Fish* action gives a chance of catching a fish, while a *Study* action results in a gain of knowledge, but consumes a fish. The agent can also choose to sell their fish. At the end of the fixed period, if they have 5 knowledge points they can get a good job (with high reward). Selling the fish does not provide as much reward as getting the job.

## ActionEffect Table

The ActionEffect table is a csv (comma separated value) file that describes the state space, the action set, and the transition function. The ActionEffect table for the Fishing MDP model is shown below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Prob | Fish | Knowledge | TimeLeft |  | State  Variable Description |
| varMin |  | 0 | 0 | 0 |  |
| varMax |  | 5 | 5 | 20 |  |
| startVal |  | 0 | 0 | 20 |  |
| Fish | 0.5 | 1 | 0 | -1 |  | Actions and their effects on the State Variables |
| Fish | 0.5 | 0 | 0 | -1 |  |
| SellOne | 1 | -1 | 0 | -1 |  |
| SellFive | 1 | -5 | 0 | -1 |  |
| Study | 1 | -1 | 1 | -1 |  |
| EndMission | 1 | 0 | 0 | 0 |  |
|  |  |  |  |  |  |  |
|  | Action Effect  Probability | State Variables and  the effects actions have on them | | |  |  |

The first four rows of the ActionEffect table describe the state variables. Starting from column 3, these rows list the state variable name, the minimum value for the variable, the maximum value, and the starting value. Note that the “Prob” column is not a state variable. In our example, there are three state variables: “Fish” for number of fish you currently have, “Knowledge” for the number of knowledge points you have, and “TimeLeft” which tracks how long before the episode is over. As all variables are integer values ranging from varMin to varMax, the total state space size is: .

The actions are listed in the first column starting on row 5. An action may be listed more than once if its effects on the state variables are stochastic. In that case, the *Prob* column will denote the probability of each possible outcome. If the values in *Prob* do not sum to 1.0 over all rows for a particular action, the values will be normalized to sum to 1.0. The cells to the right of the *Prob* column describe the effect that action will have on the various state variables. The effects are relative by default, though absolute values can be set using “->” at the start of the cell. The cell values can also reference any state variable by name, including the one to be set. In all cases, the value of the state variable substituted is the value of that variable **before** the action was taken. In our example, the action “Fish” has a 50% chance of increasing your number of fish by 1, has no effect on Knowledge, and will always decrease time by 1. All other actions are deterministic in that they have a single outcome which is attained with 100% certainty when the action is performed. Note that any action which would take any variable out of its range is not allowed. Thus you cannot Study unless you have a fish to eat. Also note that when TimeLeft reaches zero, the only allowable action is EndMission. End Mission is a special action which terminates the episode.

## Reward Configuration File

To complete the definition of the MDP model, the reward structure must be specified in the Reward table. The reward table for Fishing is shown below. There are no column headers for this file [should we add some?], but the columns are: Reward Label, Reward Amount, Action Required, State Variable Condition. Note that the reward amount can be any numeric value, the action required must be one of the actions in the actions set or the wild card character ‘\*’ signifying all actions, and the state variable condition must evaluate as a Boolean expression to either true (1) or false (0).

|  |  |  |  |
| --- | --- | --- | --- |
| GetJob | 25 | EndMission | Knowledge >= 5 |
| TimeCost | -1 | \* | 1 |
| StudyCost | -1 | Study | 1 |
| SellOne | 1.5 | SellOne | 1 |
| SellFive | 10 | SellFive | 1 |
|  |  |  |  |
| Reward  Label | Reward  Amount | Action Required | State Variable Conditions |

Each row specifies a reward (which can be a cost if negative) and the conditions under which the reward is produced. For *Fishing*, the large end-goal reward, GetJob, is listed first. The agent will get a reward of 25 when the *EndMission* action is taken and the state variable *Knowledge* is greater than or equal to 5. The next two rows indicate costs. Generally an MDP requires a time-cost or movement cost to ensure that the agent acts efficiently. Our TimeCost is -1 and it applies to all actions (designated by the action wildcard character: \*) and all conditions (designated by the Boolean value for TRUE: 1). An additional cost for studying is also included to signify that studying is not as enjoyable as the other available actions. Selling fish produce value, with a bulk sale being more valuable than a single fish sale.

## Real World ActionEffect File

The real world ActionEffect file has the exact same format as the model ActionEffect file, but presents an alternative set of transitions. It can differ in the probabilities and the action’s effects on the state variables, but must contain the same state variables and the same action set as the cognitive model file. The fishing example does not include a real-world file, but could, for example, if the agent’s beliefs about the probability of catching a fish are different from the true probabilities of catching a fish.

# Generating Simulated Data

Once an MDP model is defined, performance data can be simulated from the model. Simulating data can be used for simulation studies, but it also is extremely helpful for testing and revising the MDP model definitions.

## Required Files

Four files are used for simulating data: the ActionEffect table, the Rewards table, the RealWorld-ActionEffect table (optional), and the Person Parameter file. The first three are described above; the Person Parameter file consists of a list of the simulated population and includes a person-ID and a capability value (the beta parameter) for each simulated person. Capability ranges from 0 to infinity, where 0 indicates that the person chooses uniformly at random from the available actions in each state.

The example person parameter file from the Fishing example is shown here:

|  |  |
| --- | --- |
| 1 | 0.3 |
| 2 | 0.7 |
| 3 | 1.0 |
| 4 | 1.5 |
| 5 | 2.0 |
| 6 | 20.0 |

Note that there are no column headers. The first column is the person-ID and the second column is the capability (beta) value. In our example file, we would expect the first simulated person to behave fairly randomly, which the 6th one will display a near ideal planned performance.

## Command line options

Command line options that pertain to data generation are listed in the table below.

|  |  |  |
| --- | --- | --- |
| Option | Description | Default |
| --genData | generate simulated play data from specified MDP | Required |
| --actionEffectFile arg | the table of actions and their effects | <none> |
| --worldEffectFile arg | the table of actions and their effects for the real world | actionEffectFile argument |
| --rewardFile arg | the table of rewards | <none> |
| --discount arg | the discount parameter, range (0-1) | 1.0 |
| --dataOutputFile arg | the file to which the generated data will be written | <none> |
| --personParamFile arg | the file containing person capabilities | <none> |
| --modelParamFile arg | the file containing specifications of the model parameters | <none> |
| --recActionChoice | record action choice probabilities | No probability output |
| --recRewardValues | output the rewards received for each action | No reward output |
| --maxIters arg | set the maximum iterations in the policy optimization | 100 (check) |
| --noPruneStateSpace | do not attempt to prune the state space to only reachable states | Prune state space |

An example minimum command line for generating data:

$ mdpmf.exe --genData --personParamFile persBetaData.txt --actionEffectFile FishingAE.csv --rewardFile FishingSVR.csv --dataOutputFile Fishing\_genDat.out

After running (should take less than a minute), the file Fishing\_genDat.out should contain play records

StudID Action Fish Knowledge TimeLeft

STD001 StartMission 0.0 0.0 20.0

STD001 Fish 1.0 0.0 19.0

STD001 SellOne 0.0 0.0 18.0

STD001 EndMission 0.0 0.0 18.0

STD002 StartMission 0.0 0.0 20.0

STD002 Fish 1.0 0.0 19.0

STD002 Fish 1.0 0.0 18.0

STD002 Fish 2.0 0.0 17.0

STD002 Study 1.0 1.0 16.0

STD002 Fish 2.0 1.0 15.0

STD002 SellOne 1.0 1.0 14.0

STD002 Fish 2.0 1.0 13.0

STD002 Study 1.0 2.0 12.0

STD002 Fish 1.0 2.0 11.0

STD002 Fish 1.0 2.0 10.0

STD002 Study 0.0 3.0 9.0

STD002 Fish 0.0 3.0 8.0

STD002 Fish 1.0 3.0 7.0

STD002 Study 0.0 4.0 6.0

STD002 Fish 0.0 4.0 5.0

STD002 Fish 1.0 4.0 4.0

STD002 Fish 1.0 4.0 3.0

STD002 Study 0.0 5.0 2.0

STD002 Fish 1.0 5.0 1.0

STD002 SellOne 0.0 5.0 0.0

STD002 EndMission 0.0 5.0 0.0

STD003 StartMission 0.0 0.0 20.0

Note that this generation process is stochastic, so repeated runs of the same command will produce different play records. If you want to see the probabilities for each action choice, include the *recActionChoice* option. The play records will be generated as usual and in addition a file with the same base names as *dataOutputFile* but with an extension of .acp will be generated. The beginning of the action choice record that corresponds to the simulated play above is shown below:

PersonID Action NumOptions Fish SellOne SellFive Study EndMission

STD001 Fish 2 0.39978 0.00000 0.00000 0.00000 0.60022

STD001 SellOne 4 0.23774 0.30304 0.00000 0.19383 0.26538

STD001 EndMission 2 0.39617 0.00000 0.00000 0.00000 0.60383

STD002 Fish 2 0.71448 0.00000 0.00000 0.00000 0.28552

STD002 Fish 4 0.39579 0.06998 0.00000 0.50970 0.02453

STD002 Fish 4 0.39951 0.06569 0.00000 0.50684 0.02795

STD002 Study 4 0.39877 0.07383 0.00000 0.52236 0.00504

STD002 Fish 4 0.40374 0.07798 0.00000 0.51733 0.00095

STD002 SellOne 4 0.36491 0.14109 0.00000 0.49380 0.00020

STD002 Fish 4 0.44277 0.02615 0.00000 0.52980 0.00128

STD002 Study 4 0.40672 0.07120 0.00000 0.52185 0.00022

STD002 Fish 4 0.41392 0.06286 0.00000 0.52317 0.00004

STD002 Fish 4 0.43972 0.02951 0.00000 0.53071 0.00005

STD002 Study 4 0.46167 0.01018 0.00000 0.52808 0.00007

STD002 Fish 2 0.99997 0.00000 0.00000 0.00000 0.00003

STD002 Fish 2 0.99995 0.00000 0.00000 0.00000 0.00005

STD002 Study 4 0.45083 0.00775 0.00000 0.54142 0.00001

STD002 Fish 2 1.00000 0.00000 0.00000 0.00000 0.00001

STD002 Fish 2 1.00000 0.00000 0.00000 0.00000 0.00001

STD002 Fish 4 0.34748 0.00789 0.00000 0.64463 0.00001

STD002 Study 4 0.37982 0.00022 0.00000 0.61996 0.00001

STD002 Fish 2 0.33965 0.00000 0.00000 0.00000 0.66035

STD002 SellOne 3 0.14146 0.57366 0.00000 0.00000 0.28487

STD002 EndMission 1 0.00000 0.00000 0.00000 0.00000 1.00000

The first column is the student id and the second column is the action they chose. The 3rd column contains the number of actions they had to choose from. Finally the remaining columns list, for each action in the ActionSet, the probability the model assigned to that action being chosen. Thus, STD001 first chose to Fish, even though they had a larger likelihood to quit (given their low beta of 0.3). As they were successful in catching a fish their next choice includes four options, from which they choose to Sell-One. Finally they choose to quit (EndMission) as their third action. STD002 does better with a good run of fishing and studying. STD002 at beta = 0.7 has action probabilities shifted toward the theoretically correct actions.

# Estimating Model Parameters

This allows the estimation of selected model parameters without estimating person capabilities. Note that while different reward parameters can be identified for estimation, the model has, at minimum, two parameters: the mean and standard deviation of the population capability parameter, beta.

The program can be run to model level parameters for the simulated data produced above as follows:

$ mdpmf.exe --estModelParams --modelParamFile ModEst\_out.txt --actionEffectFile FishingAE.csv --rewardFile FishingSVR.csv --dataInputFile Fishing\_genDat.out

The model estimate file will include a row for each model-level variable along with the final log-likelihood for the full data given the estimated parameters. For the example data generated above, the model parameters are only the beta population parameters mu and sigma. A result of the *estModelParams* call might be:

betaMu 0.5593

betaSigma 4.8605

log\_likelihood -51.8882

Note that the optimization functions used to estimate model parameters are stochastic and different runs over the same data will produce slightly different results.

## Parameter Configuration File

The parameters to be estimated are specified in the parameter specification file using the –paramEstSpecsFile command line argument. This file is formatted as a table

# Estimating Person Capabilities

Given play records in the same format as those produced by *genData*, the program will estimate the person capability parameters. Right now the play records are expected to include only the actions that are defined in the action effect table. In the future the program will skip over action rows that it does not recognize.

The program can be run to estimate person capabilities for the simulated data produced above as follows:

$ mdpmf.exe --estPersonParams --personParamFile StudEst\_out.txt --actionEffectFile FishingAE.csv --rewardFile FishingSVR.csv --dataInputFile Fishing\_genDat.out

This run will estimate the needed model parameters and then estimate the person parameters. If you already have the model parameters from a run using *estModelParams,* or if you are fixing the model parameters, you can estimate person parameters only by specifying the model parameter file:

$ mdpmf.exe --estPersonParams --personParamFile StudEst\_out.txt --actionEffectFile FishingAE.csv --rewardFile FishingSVR.csv --dataInputFile Fishing\_genDat.out --modelParamFile ModEst\_out.txt

The file StudEst\_out.txt will contain the beta parameter estimates, along with person fit metrics. Each row lists the student id and the log of the capability parameter in the first two columns. (This really should be consistent with the file for generating data, i.e., they should both be in log format or neither.) If estimation errors were calculated, the LogBeta estimate will be followed by a standard error and confidence interval. The remaining three columns report the log likelihood of the data record, the expected log-likelihood of the data, and the variance in the log-likelihood of the data. These statistics can be used to compute a standardized log-likelihood which serves as a person-fit metric.

The estimation of person parameters may take a while (1-15 minutes). If a model parameter file has been supplied, but the needed model parameters are not defined in the file, the program will write out the estimated model parameters. I need to confirm that this actually works.

# Technical Details

The parameter estimation (for both the model parameters and the person capability) are done through numerical approximation, maximizing the data likelihood. Note that there is no closed form for the likelihood equation and so no gradient can be provided to guide the optimization. The library nlopt is used to perform the maximization. There are many settings in the use of nlopt including the exact optimization algorithm, the maximum number of iterations and the stopping criteria. All of these could be tweaked to possibly improve performance.

For the estimation of the model parameters, we use marginal maximum likelihood, marginalizing over the person capability parameters. The capability parameters are drawn from a log-normal distribution. Gaussian quadrature is used for the integrals involved with nine quadrature points.

# Work Remaining

While there all of the red text above indicate features that need to be added or tested/fixed, there remain a couple of other improvements that should be considered in the near term:

* The program is currently not robust to model misspecification and formatting errors in the input files. This should be fixed before allowing many others to use it. Estimation for full implementation (will included, coding, testing and debugging): 20 hours.