Function_Description

Monkie

4/12/2022

0. Requirement

0. Install Rgui and Rstudio
1. Line 3: setwd("~/<directory path>/Rasr_submit") Set the directory to where the folder is located. For :

- Prelim.R
- main.R
- Evaluator.R
- RiskOfEval.R
- Barplot.R
- Table.R
- Histogram.R

2. Make sure your R has all the libraries that is required. Use **install.packages("<package_name>")** to install missing packages.

1. Prelim.R

Installation of benchmark data is required

- Run the code and auto-install the benchmark data
- Read description of each dataset.

2. Basic_Utils.R

Basic Utils consist of basic utility function that we need for pre-processing, risk measure and so on.

MDP Preprocessing

The function df2mdp takes in an mdp.data.frame and return a list of MDP information MDP = list$[O, S, A, |O|, |S|, |A|, P, R, \bar{P}, \bar{R}, \gamma, S_0]$. prep_MDP is a helper function convert each outcome R, P from mdp.data.frame to individual matrix.

```
# TURN single outcome mdp.data.frame into it's respective reward,transition (S,A,S') matrix.
prep_MDP = function(mdp.df,MDP){
  P <- array(0, c(MDP$lSl,MDP$lAl,MDP$lSl)) # transition prob dim
  R = array(0, c(MDP$lSl,MDP$lAl,MDP$lSl)) # reward dim
  dimnames(P) = list(MDP$S,MDP$A,MDP$S)

  n_rows = nrow(mdp.df)
  # Assign transition prob and reward matrix w.r.t(S,A,S')
  for (r in 1:n_rows){
    R[MDP$S == mdp.df$idstatefrom[r],MDP$A == mdp.df$idaction[r],MDP$S == mdp.df$idstateto[r]] = mdp.df$
    P[MDP$S == mdp.df$idstatefrom[r],MDP$A == mdp.df$idaction[r],MDP$S == mdp.df$idstateto[r]] = mdp.df$
  }
  return(list(P=P,R=R))
}

df2mdp = function(folder_name,file){
  mdp.df = read.csv( paste0(folder_name,file)  ,header = TRUE)
  MDP = list()
  # Parse in Basic parameter States, Actions, Outcomes space and their length.
  MDP$O = sort(unique(mdp.df$idoutcome))
  MDP$S = sort(unique(unique(c(mdp.df$idstatefrom,mdp.df$idstateto))))
  MDP$A = sort(unique(mdp.df$idaction))
  MDP$lOl = length(MDP$O)
  MDP$lSl = length(MDP$S)
  MDP$lAl = length(MDP$A)
  # Extract each outcomes Reward and Transition Matrix.
  temp = lapply(MDP$O, function(o) prep_MDP(mdp.df[mdp.df$idoutcome==o,],MDP))
  MDP$P = lapply(MDP$O+1, function(o) temp[[o]]$P)
  MDP$R = lapply(MDP$O+1, function(o) temp[[o]]$R)
  # Solve or Parse in other MDP parameters.
  MDP$Pbar = Reduce("+",MDP$P)/MDP$lOl
  MDP$Rbar = Reduce("+",MDP$R)/MDP$lOl
  MDP$gamma = read.csv(paste0(folder_name,"/parameters.csv"),header = TRUE)$value
  MDP$S_0 = read.csv(paste0(folder_name,"/initial.csv"),header = TRUE)
  return(MDP)
}
```

Risk Measures

Expected Value

Expectation function which calculate the mean for vector-reward X and probability of occurrence. E(X,prob = NULL), takes in distribution X with probability. If prob is not given, then consider uniformly distributed/weighted X.

$$\mathbb{E}(X,P) = \sum_i \left(\; P(X_i) \cdot X_i \; \right) = P^T X$$

```
E = function(X, prob = NULL){
  if (!is.null(prob)){
    if (length(prob) != length(X)){
      stop("Mismatch Dimensions of prob and value")
    }
    return(sum(X*prob))
  }
  return(mean(X))
}
```

Entropic Risk Measure

Entropic Risk Measure for a variable X with a risk aversion parameter α is define as

$$\mathrm{ERM}^{\alpha}[X] = -\frac{1}{\alpha}\log(\mathbb{E}[^{-\alpha X}]) = -\frac{1}{\alpha}\log\big(\sum_i (p(X_i)\cdot{}^{-\alpha X_i})\big)$$

To establish computational stability log-sum-exp trick is used, we use $\hat{X} = \{x \in X : P(x) > 0\}$. Let $\hat{C} = \max(-\alpha\hat{X})$ the entropic risk is computed as

$$\mathrm{ERM}^{\alpha}[X] = -\frac{1}{\alpha}\big(\hat{C} + \log(\mathbb{E}[^{-\alpha\hat{X}-\hat{C}}])\big)$$

α refers to the risk level $\lim_{\alpha\to 0}\mathrm{ERM}^{\alpha}[X] = E[X]$ and $\lim_{\alpha\to\infty}\mathrm{ERM}^{\alpha}[X] = \min[X]$

```
# Entropic Risk Measure with Log-Sum-Exp Trick with Prob
ERM = function(X, alpha=0.9, prob = NULL){
  if (alpha == 0){
    return(E(X,prob))
  }
  Y = -alpha*X
  C = max(Y)+1
  if (!is.null(prob)){
    if (length(prob) != length(X)){
      stop("Mismatch Dimensions of prob and value")
    }
    # The hat is avoid underflow, we need to shift the value to max of non-zero prob
    Yhat = Y[prob!=0]
    Chat = max(Yhat)
    probhat = prob[prob!=0]
    return(-(Chat+log(sum(exp(Yhat-Chat)*probhat)))/alpha)
  }
  return(-(C+log(mean(exp(Y-C))))/alpha)
}
```

EVaR

EVaR takes in a random variable (X) and discrete risk levels($\Lambda \subset \mathbb{R}$) and a confident level (β) that is comparable to VaR and CVaR, note that $(\mathrm{EVaR}^0[X] \geq \mathrm{EVaR}^{\beta}[X] \geq \mathrm{EVaR}^1[X])$.

$$\mathrm{EVaR}^{\beta}[X] = \sup_{\alpha>0}\{-\frac{1}{\alpha}\log(\frac{\mathbb{E}[^{-\alpha X}]}{1-\beta})\} = \sup_{\alpha>0}\{\mathrm{ERM}^{\alpha}[X] + \frac{\log(1-\beta)}{\alpha}\} \approx \max_{\alpha\in\Lambda}\{\mathrm{ERM}^{\alpha}[X] + \frac{\log(1-\beta)}{\alpha}\}$$

Note that it is in-feasible to sample throughout the whole real number space $\alpha > 0$. Therefore, the levels Λ has to be specified. This method could be improved via bisection.

```r
EVAR = function(X,levels,risk=0.95, prob=NULL){
  return(max(sapply(levels, function(z) ERM(X,alpha = z,prob = prob) + log(1-risk)/z )))
}
```

If the Λ and set $\eta = \{\mathrm{ERM}^\alpha[X] : \alpha \in \Lambda\}$ is given. We can directly choose from the maximum over the values.

which.Erm2Evar

which.Erm2Evar takes a vector of Scores for Erm (Eta), their respective alphas (L) and a confident level (beta) that is comparable to VaR and CVaR. Erm2Evar return the optimal index of the optimal alpha in L.

```r
# Eta is ERM^alpha[X]
which.Erm2Evar = function(L, Eta, beta = 0.9){
  if (length(Eta) != length(L)){
    warning("Risk and return dimension does not match.")
  }
  return( which.max( Eta + log(1-beta)/L ) )
}
```

CVaR

$$\mathrm{CVaR}^\theta[X] = \frac{1}{\theta}\int_0^\theta F_X^{-1}[t]\ dt = \frac{1}{\theta}(\mathbb{E}[X \cdot \mathbb{1}_{X\le x_\theta}] + x_\theta(\theta - \mathbb{P}[X \le x_\theta]))$$

where $x_\theta = \inf\{x \in \mathbb{R} : \mathbb{P}[X \le x] \ge \theta\}$ is the lower θ quantile and $\mathbb{1}_A(x) = \begin{cases} 1 & \text{if} x \in A \\ 0 & \text{otherwise} \end{cases}$ is an indicator function.

Note that the risk parameter θ is refer to as significant level $\mathrm{CVaR}^1[X] = \mathbb{E}[X]$ and $\mathrm{CVaR}^0[X] = \min[X]$. Therefore $\mathrm{CVaR}^1[X] \ge \mathrm{CVaR}^\theta[X] \ge \mathrm{CVaR}^0[X]$ and $\mathrm{VaR}^1[X] \ge \mathrm{VaR}^\theta[X] \ge \mathrm{VaR}^0[X]$.

```r
CVAR = function(X,thetas = 0.05,prob = NULL){
  n = length(X)
  if (is.null(prob)){
    prob = rep(1/n,n)
  } else if (length(prob) != length(X)){
    stop("Mismatch Dimensions of prob and value")
  } else if (abs(sum(prob) - 1) > 1e-8){
    stop("Distribution probability does not sum to one (1)")
  }

  # sort value and its probability
  ord = order(X)
  X = X[ord]
  prob = prob[ord]

  # force thetas to be a vector
  thetas = c(thetas)
  if (max(thetas)>1 || min(thetas)<0){
    stop("Undefined confident level. (thetas) should be an array between 0 and 1.")
  }
  lLl = length(thetas)
  v = thetas*0
  names(v) = thetas

  # Initialize parameter for loop
  Psum = 0
  Vsum = 0
  index = 1

  for (l in 1:lLl){
    k = thetas[l]
    if (k == 0){
      v[l] = min(X[prob>0])
    } else {
      while (k > (Psum + prob[index] +1e-10) ){
        Psum = (Psum + prob[index])
        Vsum = Vsum + prob[index]*X[index]
        index = index + 1
      }
      v[l] = (Vsum + (k-Psum)*X[index])/k  # This is Piecewise Linear
    }
  }
  return(v)
}
```

Other function

Total Discounted Return (TDR)

TDR take in vector of returns V[t=0,t=1,...] and discount factor γ.

$$\text{TDR}(V,\gamma) = \sum_t V_t \cdot \gamma^t$$

```
TDR = function(V, discount=0.9){
  D = sapply(1:length(V),function(t) discount^(t-1))
  return(sum(D * V))
}
```

wdir

check and create an directory if does not exist.

```
wdir = function(directory_name){
  if (!dir.exists(directory_name)){
    cat("Directory",directory_name,"Not Exist. Creating Directory...\n")
    dir.create(directory_name)
  }
  return(directory_name)
}
```

generate_sample

Generate Model and sampling instances for every time step.This methods also generate initial state distribution but we did not use this initial state distribution since we have either single initial state or uniformly distributed initial state.

```
generate_sample = function(MDP,n,t,folder_name){
  set.seed(1)
  O = 1:MDP$lOl
  for (i in 1:n){
    write.csv(data.frame(R_outcome = sample(O,t,replace=TRUE),
                         T_outcome = sample(O,t,replace=TRUE),
                         S_ = runif(t)),paste0(folder_name,"/instance_",i,".csv"))
  }
  write.csv(sample(1:MDP$lSl,prob = MDP$S_0$probability,n,replace = TRUE),
            paste0(folder_name,"/S0.csv"))
}
```

drawS_

This function takes in a probability vector and choice (a randomly generated number between 0 and 1), return the index of the states it transition to.

```
drawS_ = function(weights,choice){
  choiceIndex = 1
  for (w in weights){
    choice = choice - w
    if (choice <= 1e-10){
      return(choiceIndex)
    }
    choiceIndex = choiceIndex + 1
  }
  return(choiceIndex)
}
```

EvalMarkovPi

This function evaluate Markovian [stationary (1 row) and time dependent (n rows)] policy. Pass in initial state, policy, instance_index(i), folder of instance.

```
EvalMarkovPi = function(i,s0,Pi,MDP,folder_name,Time = NULL){
  cur_df = read.csv(paste0(folder_name,"/instance_",i,".csv"))
  if (is.null(Time)){Time = nrow(cur_df)} # Sample Time Horizon
  lLl = nrow(Pi)       # Risk Levels Depth
  TDR = 0
  gamma_T = 1
  cur_s = s0
  for (t in 1:Time){
    lvl = min(t,lLl)
    s_next = drawS_(MDP$P[[cur_df$T_outcome[t]]][cur_s,Pi[ lvl, cur_s],],choice = cur_df$S_[t])
    TDR = TDR + gamma_T*MDP$R[[cur_df$R_outcome[t]]][cur_s,Pi[lvl , cur_s],s_next]
    gamma_T = MDP$gamma*gamma_T
    cur_s = s_next
  }
  TDR = TDR/(1-gamma_T)
  return(TDR)
}
```

EvalHistPi

This function evaluate History Dependent policy. Pass in initial state, policy, instance_index(i), folder of instance, augmented state_mapping for both direction and T_cnt that tells which augmented state it transition to. In every time step it require solving the transition in the augmented space.

```
EvalHistPi = function(i,s0,Pi,MDP,folder_name, S_Aug,S_map,T_cnt,Time = NULL){
  cur_df = read.csv(paste0(folder_name,"/instance_",i,".csv"))
  if (is.null(Time)){Time = nrow(cur_df)} # Sample Time Horizon
  lLl = nrow(Pi)      # Risk Levels Depth
  TDR = 0
  gamma_T = 1
  # Cur_s and s_next is a reference to the augmented States
  cur_s = s0
  # s_ori,s_new and l_ori,l_new refer to the initial state and level correspond the the augment state
  for (t in 1:Time){
    c(s_ori,l_ori) %<-% as.numeric(unlist(str_split(S_Aug[cur_s],"-")))
    P_Aug = rep(0,length(S_Aug))
    for (s2 in 1:MDP$lSl){
      P_Aug[S_map[[paste(s2,floor(T_cnt[l_ori,s_ori,s2]),sep="-")]]] = MDP$P[[cur_df$T_outcome[t]]][s_o
      P_Aug[S_map[[paste(s2,floor(T_cnt[l_ori,s_ori,s2]) + 1,sep="-")]]] = MDP$P[[cur_df$T_outcome[t]]]
    }
    if (abs(1-sum(P_Aug))>1e-10){
      warning("Transition does not sum to one")
    }
    s_next = drawS_(P_Aug,choice = cur_df$S_[t])
    c(s_new,l_new) %<-% as.numeric(unlist(str_split(S_Aug[s_next],"-")))
    TDR = TDR + gamma_T*MDP$R[[cur_df$R_outcome[t]]][s_ori,Pi[l_ori , s_ori],s_new]
    gamma_T = MDP$gamma*gamma_T
    cur_s = s_next
  }
  TDR = TDR/(1-gamma_T)
  return(TDR)
}
```

3. RASR_code

ERM Preprocessing

prep_ERM takes in a MDP object, objectives α_0 and ϵ, output the required time or level horizon $|T|$, the vector of levels L and the vector of $r^\alpha \in \mathbb{R}^{|S||A||S|} = \text{ERM}^\alpha[R]$, $\forall \alpha \in L$. r^α is used when reward model is independent from transition and can be calculated independently in preprocess.

```r
prep_ERM = function(MDP,alpha_0 = exp(10),epsilon = exp(-15) ,Rtemp = NULL){
  # Generate approximation horizon (T) and required levels (L)
  lTl = ceiling(log(alpha_0/epsilon)/(1-MDP$gamma))
  L = c(sapply(1:(lTl - 1), function(t) alpha_0*MDP$gamma^(t-1) ),0)

  if (is.null(Rtemp)){
    Rtemp = array(unlist(MDP$R), dim=c(MDP$lSl,MDP$lAl,MDP$lSl,MDP$lOl))
  }
  registerDoParallel(cores=detectCores())
  # Save the final score in combined score
  rAlp <- foreach (t = 1:lTl) %dopar% {
    array( sapply(1:MDP$lSl,function(s2) sapply(1:MDP$lAl,function(a) sapply(1:MDP$lSl,function(s)
      ERM(Rtemp[s,a,s2,],alpha = L[t])
    ))) , dim=c(MDP$lSl,MDP$lAl,MDP$lSl))
  }
  registerDoParallel(cores=1)
  stopImplicitCluster()

  return(list(lTl = lTl, L = L, rAlp = rAlp))
}
```

ErmMat2List is used for NaiveErm and EpisErm calculation due to their matrix output. This function converts the ERM matrix output to its respective list type.

```r
ErmMat2List = function(stats,levels,S,lSl){
  Scores = stats[,2*lSl+1]
  V = stats[,(1:lSl)+lSl]
  rownames(V) = levels
  colnames(V) = S

  Pi = stats[,(1:lSl)]
  rownames(Pi) = levels
  colnames(Pi) = S
  return(list(Pi = Pi, V = V,Scores = Scores))
}
```

Erm algorithm

Note that the algorithm or functions introduce in this section evaluate the policy if policy is given, otherwise it would greedily solve for the optimal policy $\pi^\star$.

RASR - Entropic Risk Measure (Ours)

The RasrErm takes in the MDP parameter and the Erm parameter. Note that Rasr can be used to solve uncertainty causes by different parameters simultaneously (R, P) with an assumption : The RasrErm assume the uncertainties of different parameters are independent. Which might not be the case for most Epistemic Uncertainty paper.

First, solve for standard MDP value function which refer to risk neutral. We evaluated the standard value function with 1,000 iterations of standard bellman update. Assumption : MDP converges within 1,000 value iteration.

$$v^\star_{|T|}(s) = \max_{a\in A} \mathbb{E}\left[\bar{R}(s,a,S') + \gamma \cdot v^\star_{|T|}(S')\right] ,$$

Once we have the standard MDP value function $v^\star_{|T|}$ we can use it to update the value function $v^\star_t$ for $t \in 1 : (|T| - 1)$. Note that (the coding language : R) is one-indexed instead of zero-indexed that is why the ERM level is $\alpha\gamma^{t-1}$ instead of $\alpha\gamma^t$.

$$v^\star_t(s) = \max_{a \in A} \mathrm{ERM}^{\alpha \cdot \gamma^{t-1}} \left[r^\alpha(s, a, S') + \gamma \cdot v^\star_{t+1}(S') \right] ,$$

```
RasrErm = function(MDP, rAlp, L, lTl=length(L) ,Pi = NULL,v_nominal = NULL){
  V = matrix( 0 , nrow = lTl , ncol = MDP$lSl )
  solvePi = is.null(Pi)
  # Solve for Pi if not given otherwise only solve for V.
  if (solvePi){ Pi = V*0 }
  # don't need to solve nominal if passed in
  n_iter = ifelse(is.null(v_nominal),1000,1)
  if (!is.null(v_nominal)){V[lTl,] = v_nominal}
  for (i in 1:n_iter){ # Here we assume the nominal MDP converge in less than 1000 steps
    if (solvePi){ Pi[lTl,] = sapply(1:MDP$lSl, function(s)  which.max(sapply(1:MDP$lAl, function(a)  MDI
    V[lTl,] = sapply(1:MDP$lSl, function(s) MDP$Pbar[s,Pi[lTl,s],] %*% (MDP$Rbar[s,Pi[lTl,s],] + MDP$gar
  }
  for (t in (lTl - 1):1){
    if (solvePi){ Pi[t, ] = sapply(1:MDP$lSl, function(s)  which.max(sapply(1:MDP$lAl, function(a)  ERM
    V[t, ] = sapply(1:MDP$lSl, function(s) ERM( c(rAlp[[t]][s,Pi[t,s],] + MDP$gamma * V[t+1,]),alpha = I
  }
  # Evaluate the ERM for each risk level.
  Scores = sapply(1:lTl,function (t) ERM(V[t,],alpha = L[t],prob = MDP$S_0$probability) )
  return(list(Pi = Pi, V = V, Scores = Scores))
}
```

Naive Entropic Risk Measure

The NaiveERM is way more computational expensive then the RasrERM and also less accurate. The benefit of NaiveERM is that it is beneficial if we wanted to solve for a single ERM risk level α. However, solving for a huge amount of α for ERM is more expensive due to the fact that no-value function is shared, we will required solving $|T|$ many MDP instead of 1 single MDP for RasrERM. The differences between NaiveERM and the RasrERM is that NaiveERM uses the same level of risk α over time which causes an overly pessimistic value function approximation.

$$v^\star(s) = \max_{a \in A} \mathrm{ERM}^{\alpha} \left[r^\alpha(s, a, S') + \gamma \cdot v^\star(S') \right] ,$$

We exploit all the computer core and compute parallel-ly to calculate the NaiveERM for all $\alpha \in L$.

```
# Solve the NaiveErm for single level.
NaiveErm1L = function(alpha,V_cur,R_cur,Pbar,Pi_cur,lSl,lAl,discount,S_0 ){
  solvePi = is.null(Pi_cur)
  for (i in 1:1000){
    if (solvePi) { Pi_cur = sapply(1:lSl, function(s)  which.max(sapply(1:lAl, function(a) ERM( c(R_cur
    V_cur = sapply(1:lSl, function(s) ERM( c(R_cur[s,Pi_cur[s],] + discount * V_cur),alpha = alpha,prob=
  }
  Scores_cur = ERM(V_cur,alpha = alpha,prob=S_0$probability)
  return(c(Pi_cur,V_cur,Scores_cur))
}
# NaiveErm is RasrErm without changing risk-level for bellman update.
NaiveErm = function(MDP , rAlp, levels, Pi = NULL){
  registerDoParallel(cores=detectCores())
  # Save the final score in combined score
  statistics <- foreach (t = 1:length(levels),.combine = 'rbind') %dopar% {
    NaiveErm1L(levels[t] , rep(0,MDP$lSl) , rAlp[[t]] , MDP$Pbar , Pi[t,] , MDP$lSl , MDP$lAl , MDP$gamm
  }
  registerDoParallel(cores=1)
  stopImplicitCluster()
  return(ErmMat2List(statistics , levels , MDP$S , MDP$lSl))
}
```

Epistemic Entropic Risk Measure

The EpistemicERM is similar to NaiveERM, with a slight differences that Epistemic-ERM does not consider aleatory uncertainty, only epistemic uncertainty Model $= (P, R)$. Unlike RasrErm and NaiveErm, in EpistemicErm the risk for (P, R) is connected in tuple/pair and considered as a model.

Similar to NaiveERM solving ERM for all $\alpha \in L$ is computationally expensive as the set L increase in size because this require solving an independent MDP for each α.

$$v^\star(s) = \max_{a\in A} \mathrm{ERM}^\alpha\left[\mathbb{E}\big[R(s,a,S') + \gamma \cdot v^\star(S') \mid (P,R)\big]\right],$$

```
# Solve the EpisErm for single level.
EpisErm1L = function(alpha,outcomes,V_cur,Rew,P,Pi_cur,lSl,lAl,discount,S_0){
  solvePi = is.null(Pi_cur)
  for (i in 1:1000){
    if (solvePi){ Pi_cur = sapply(1:lSl, function(s)  which.max(sapply(1:lAl, function(a) ERM( sapply(ou
    V_cur = sapply(1:lSl, function(s) ERM( sapply(outcomes+1, function(o) P[[o]][s,Pi_cur[s],] %*% (Rew
  }
  Scores_cur = ERM(V_cur,alpha = alpha,prob=S_0$probability)
  return(c(Pi_cur,V_cur,Scores_cur))
}
# EpisErm is the NaiveErm which consider epistemic uncertainty only.
EpisErm = function(MDP, levels, Pi = NULL){
  registerDoParallel(cores=detectCores())
  # Save the final score in combined score
  statistics <- foreach (t = 1:length(levels),.combine = 'rbind') %dopar% {
    EpisErm1L(levels[t] , MDP$O , rep(0,MDP$lSl) , MDP$R , MDP$P , Pi[t,] , MDP$lSl , MDP$lAl , MDP$gamm
  }
  registerDoParallel(cores=1)
  stopImplicitCluster()
  return(ErmMat2List(statistics , levels , MDP$S , MDP$lSl))
}
```