Cognitive Modeling LC4

Harm Manders, 10677186

```
## Loading required package: proto
```

```r
# Q1
q_learn <- function(Q, means, std, action, a) {
  # Vrije variablelen alpha, std, means
  r = rnorm(1, means[action], std)
  d = r - Q[action]
  Q[action] = Q[action] + a * d
  return(list(Q=Q, pnts=r))
}

q_learn_ee <- function(a, e, trials, decay=FALSE, dec_fac=0.9) {
  Q = rep(0,4)
  M = c(20, 30, 50, 70)
  std = 4
  total = 0
  for (t in 1:trials) {
    # In slides is random > e maar in opdracht q6 gaan ze er vanuit random < e
    if (runif(1) < e) {
      action = sample(1:4,1)
    }
    else {
      action = which.max(Q)
    }
    list[Q,pnts] = q_learn(Q, M, std, action, a)
    total = total + pnts
    if (decay) {
      e = e * dec_fac
    }
  }
  return(list(Q=Q, pnts=total, avg=total/trials))
}

q_learn_score <- function(a, e, eps, trials, decay=FALSE, dec_fac=0.9) {

  total = 0
  tot_avg = 0
  for (i in 1:eps) {
      list[Q, pnts, avg] = q_learn_ee(a, e, trials, decay, dec_fac)
      total = total + pnts
      tot_avg = tot_avg + avg
  }
  return(list(tot=total, avg=tot_avg/eps))
}
```

```r
q_learn_smax <- function(a, t, trials) {
  Q = rep(0,4)
  M = c(20, 30, 50, 70)
```

```
  std = 4
  P = rep(1,4)
  tot = 0
  for (i in 1:trials) {
    P = exp(Q * t) / sum(exp(Q * t))

    action = which.max(rmultinom(1, 1, P))
    list[Q, pnts] = q_learn(Q, M, std, action, a)
    tot = tot + pnts
  }
  return(list(Q=Q, pnts=tot, avg=tot/trials))
}

q_learn_score_smax <- function(a, t, eps, trials) {

  total = 0
  tot_avg = 0
  for (i in 1:eps) {
      list[Q, pnts, avg] = q_learn_smax(a, t, trials)
      total = total + pnts
      tot_avg = tot_avg + avg
  }
  return(list(tot=total, avg=tot_avg/eps))
}
```

```
# Q2
Q = rep(0,4)
# 2 munten schatkist 1
M = c(2, 0, 0, 0)
std = 0
action = 1

alpha = 0.5
q_learn(Q, M, std, action, alpha)
```

```
## $Q
## [1] 1 0 0 0
##
## $pnts
## [1] 2
```

```
alpha = 0.2
q_learn(Q, M, std, action, alpha)
```

```
## $Q
## [1] 0.4 0.0 0.0 0.0
##
## $pnts
## [1] 2
```

```
# Q3
Q1 = q_learn_ee(0.1,0.1,200)
Q5 = q_learn_ee(0.5,0.1,200)
Q1$pnts
```

```
## [1] 5621.662
```

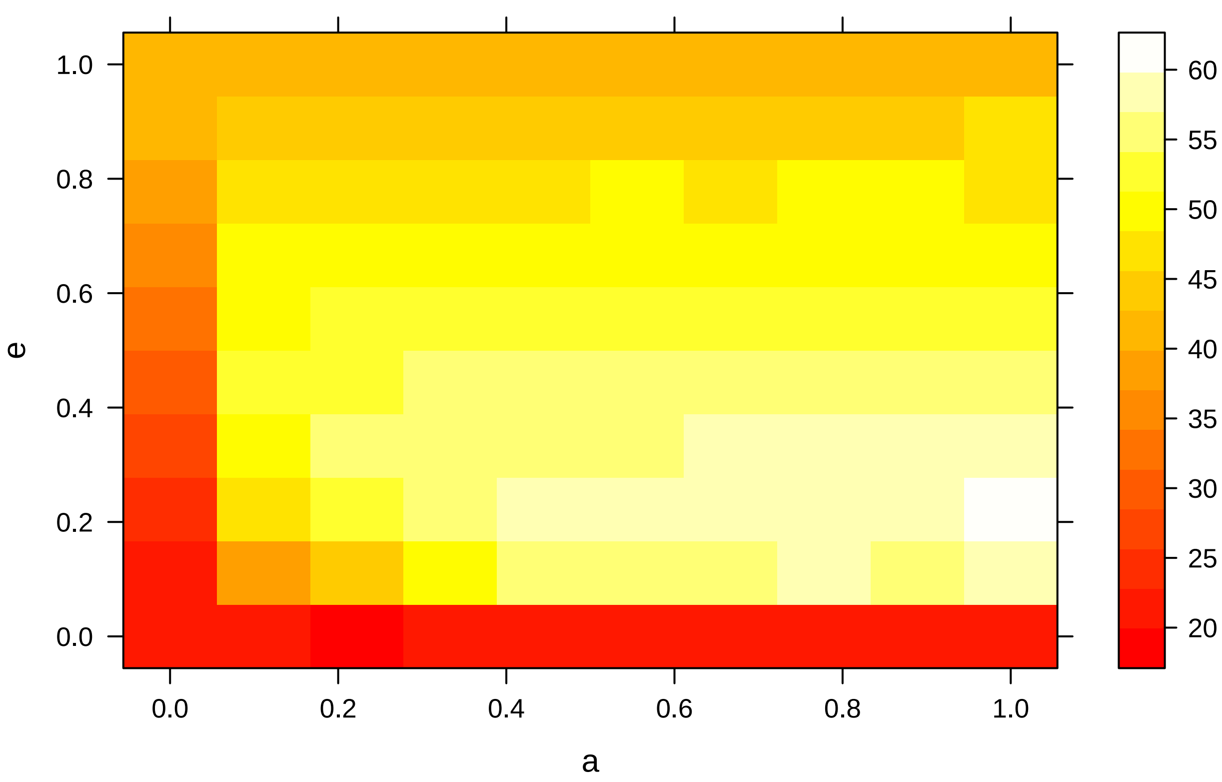
```
Q5$pnts
```

```
## [1] 13212.88
```

```
#  a = 0.1 is beter over het algemeen
```

```
# Q4 exploration of parameters
M = c(20,30,50,70)
x <- seq(0, 1,length.out = 10)
y <- seq(0, 1, length.out = 10)
data <- expand.grid(X=x, Y=y)
data$tot <- rep(0,length(x) * length(y))
data$avg <- data$Z
n = 1
for (e in y) {
  for (a in x) {
    total = 0
    # hoge episodes en trials kan r studio niet aan.
    list[tot, avg] = q_learn_score(a,e,50,100)
    data$tot[n] = tot
    data$avg[n] = avg
    n = n + 1
  }
}

levelplot(avg ~ X*Y, data=data, xlab='a', ylab='e', col.regions = heat.colors(100))
```



```r
# Q4
param1 = q_learn_score(0.1, 0.1, 500, 200)
param2 = q_learn_score(0.3, 0.1, 500, 200)
param3 = q_learn_score(0.8, 0.1, 500, 200)
param1$avg
```

```
## [1] 41.65376
```

```r
param2$avg
```

```
## [1] 56.71169
```

```r
param3$avg
```

```
## [1] 61.37408
```

```r
# Hogere alpha zorgt voor hogere score
```

```r
# Q5
# Optimum ligt rond e=0.2
param1 = q_learn_score(0.3, 0.05, 500, 200)
param2 = q_learn_score(0.3, 0.2, 500, 200)
param3 = q_learn_score(0.3, 0.6, 500, 200)
param1$avg
```

```
## [1] 51.08365
```

```r
param2$avg
```

```
## [1] 59.02702
```

```r
param3$avg
```

```
## [1] 52.81574
```

```r
# Q6
q_learn_ee(0.8,0.5,200, decay=TRUE)
```

```
## $Q
## [1] 21.43029  0.00000  0.00000 69.52661
##
## $pnts
## [1] 13514.32
##
## $avg
## [1] 67.57161
```

```r
# Q7
param1 = q_learn_score(0.9,0.2,500,200,decay=TRUE)
param2 = q_learn_score(0.9,0.4,500,200,decay=TRUE)
param3 = q_learn_score(0.9,0.6,500,200,decay=TRUE)
param4 = q_learn_score(0.9,0.8,500,200,decay=TRUE)
param5 = q_learn_score(0.9,1,500,200,decay=TRUE)
param1$avg
```

```
## [1] 45.79271
```

```r
param2$avg
```

```
## [1] 58.54742
```

```r
param3$avg
```

```
## [1] 63.80443
```

```r
param4$avg
```

```
## [1] 66.21194
```

```r
param5$avg
```

```
## [1] 66.8115
```

```r
# Q8
q_learn_score_smax(0.3, .01, 500, 200)$avg
```

```
## [1] 46.05465
```

```r
q_learn_score_smax(0.3, .05, 500, 200)$avg
```

```
## [1] 59.75811
```

```r
q_learn_score_smax(0.3, .1, 500, 200)$avg
```

```
## [1] 63.27398
```

```r
q_learn_score_smax(0.3, .15, 500, 200)$avg
```

```
## [1] 59.93282
```

```r
q_learn_score_smax(0.3, .3, 500, 200)$avg
```

```
## [1] 48.51707
```

```r
#random moves zou gemiddeld score van 42.5 opleveren
```

```r
# Q9
data<-read.delim("L4_data_1.txt")

Q_LEARN_SMAX_FIT<-function(par){
  nRounds<-nrow(data)
  alpha<- par[1] #learning rate from wins
  theta<- par[2] #exploratoin
  if (length(par) >= 3) {
    init = par[3]
  } else {
    init = 30
  }

  M = c(50,30,20,80)
  std = 4
  # int weights
  Q_list<-rep(init,4)

  likes<-c() # empty list for the LL

  # print('hoi')

  for (i in 1:nRounds) {

    ## Use softmax to calculate probability of chosing each option:
```

```
    P = exp(Q_list*theta) / sum(exp(Q_list*theta))

    # find choice
    action <- data$choice[i]
    outcome <- data$outcome[i]

    # now update Q value of chosen option based on feedback! Here we simulate the subjects    learning f
    Q_list <- q_learn(Q_list, M, 0, action, alpha)$Q

    ## store the probability of only chosen option in the likes() list you made before, we only need tha
    likes[i] = P[action]

  }
  ## determine summed log like
  LL<-sum(log(likes))
  #transform to G2 (we minimize function so it should return a negative number)
  G2=-2*LL
  return(G2)
}

# to fit

optim(c(.5,.5,30), fn = Q_LEARN_SMAX_FIT, method = 'L-BFGS-B',lower = c(0,0,0),upper = c(1,10,100))
```

```
## $par
## [1] 0.51352773 0.02559758 0.00000000
##
## $value
## [1] 170.3217
##
## $counts
## function gradient
##       33       33
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
Q_LEARN_HUMAN<-function(a, t, init){
  nRounds<-nrow(data)

  M = c(50,30,20,80)
  std = 4

  Q_list<-rep(init,4)

  for (i in 1:nRounds) {

    ## Use softmax to calculate probability of chosing each option:
    P = exp(Q_list*t) / sum(exp(Q_list*t))

    # find choice
```

```
    action <- data$choice[i]
    outcome <- data$outcome[i]

    # now update Q value of chosen option based on feedback! Here we simulate the subjects    learning p
    Q_list <- q_learn(Q_list, M, std, action, a)$Q

    ## store the probability of only chosen option in the likes() list you made before, we only need tha

  }
  ## determine summed log like

  return(Q_list)
}

Q_LEARN_HUMAN(0.51352773, 0.02559758, 0.00000000)
```

```
## [1] 50.76522 14.39815 10.19429 77.89772
```