```lua
local R = require
local _,the,COLS,BIN,NUM          = R"lib", R"the", R"cols", R"bin", R"num"
local o,oo,down1,map,push,sort,powerset = _.o,_.oo,_.down1,_.map,_.push,_.sort,_.p
owerset
local slice,merge,slots,fmt,rnds =  _.slice, _.merge,_.slots,_.fmt,_.rnds
local class,OBJ                  = _.class, _.OBJ

local RULE = class("RULE",OBJ)

function RULE.best(bins,h)
   local function score1(b1,b2) return RULE({b1},h).score > RULE({b2},h).score end
   return slice(sort(bins, score1), 1, the.beam) end

function RULE.fromBins(bins,all,h,bests,rests,    n,out,rule,sizes,scores)
  out={}
  sizes=NUM()
  scores=NUM()
  for _,some in pairs(powerset(RULE.best(bins,h))) do
    if #some>0 then
      rule = RULE(some,h)
      sizes:add(#some)
      scores:add(rule.score)
      push(out, {size=#some,score=rule.score,rule=rule}) end end
  local function order(one)
    return ((0 - sizes:norm(one.size))^2 + (1 - scores:norm(one.score))^2)^.5 end
  local n = 0
  for _,three in pairs(sort(out, function(a,b) return order(a) < order(b) end)) do
    local selected1= three.rule:selects(bests)
    local cover1   = 100*#selected1/#bests//1
    local selected2= three.rule:selects(rests)
    local cover2   = 100*#selected2/#rests//1
    local some     = all:clone(three.rule:selects(all.rows))
    if cover1 < 100 or cover2 < 100 then
      print(o(rnds(some:mid())),
            o(rnds(some:div())),
            fmt("%5.3f %4u %4u %s", three.score, cover1, cover2, three.rule)
            )
      n=n+1
      if n > the.beam then return end  end end
  return out end

function RULE:new(bins,h,   t)
  self.seen={}
  self.bins = {}
  for _,bin in pairs(bins) do
    self.bins[bin.at] = self.bins[bin.at] or {}
    push(self.bins[bin.at],  bin) end
  for _,one in pairs(self.bins) do sort(one, function(a,b) return a.lo < b.lo end)
 end
  self.score = self:scored(h)
  end
function RULE:__tostring() return self:show(self.bins)  end --return self:show(sel
f.bins) end

function RULE:like(klass,h) -- h={"true"=100, "false"=40) n=100+40
  local n=0; for _,v in pairs(h) do n = n + v end
  local fs = {}
  for at,bins in pairs(self.bins) do
    fs[at] = 0
    for _,bin in pairs(bins) do
      fs[at] = fs[at] + (bin.ys.has[klass] or 0) end end
  self.seen[klass] = fs
  local prior = ((h[klass] or 0) + the.K) / (n + the.K * 2)
  local out   = math.log(prior)
  for at,v in pairs(fs) do
    local inc = (v+the.M*prior)/(h[klass]+the.M)
    out=out + math.log( inc)
    end
  return out end

RULE.bias = {}
local bias = RULE.bias
function bias.optimize(b,r) return b+r==0 and 0 or b^2/(b+r) end
function bias.monitor( b,r) return b+r==0 and 0 or r^2/(b+r) end
function bias.tabu(    b,r) return b+r==0 and 0 or 1/(b+r) end

function RULE:scored(h)
  return self.bias[the.rule](self:like("left",h), self:like("right",h)) end

function RULE:selects(rows)
  return map(rows, function(row) if self:select(row) then return row end end) end

function RULE:select(row)
  local function ors(bins)
    for _,bin in pairs(bins) do if bin:select(row) then return true end end
    return false end
  for at,bins in pairs(self.bins) do if not ors(bins) then return false end end
  return true end

function RULE:show(ands)
  local cat, order, sortor, sortand
  cat    = function(t,sep) return table.concat(t,sep) end
  sortand= function(t) return map(slots(t) ,function(k) return t[k] end) end
  sortor = function(a,b)  return a.lo < b.lo end
  return cat(map(sortand(ands),
           function(and1)
            return "("..cat(map(sort(and1,sortor),
                    function(or1) return tostring(or1) end),"or")..")" end),"and")
end

-- print has to wipe out fullranges and print selected items
  --sort(bins,order)
  -- ors=   function(bins)
  --          return cat(map(merge(sort(bins,order),BIN.mergeNext)),"or") end
  -- return cat(map(bins, ors),"and") end

return RULE
```