```lua
local _=require("lib")
local the=_.settings[[
TINY: a lean little learning library, in LUA
(c) 2022 Tim Menzies <timm@ieee.org> BSD-2 license

USAGE: lua l5.lua [OPTIONS]

OPTIONS:
 -b  --bins    max number of bins                     = 8
 -d  --dump    on test failure, exit with stack dump  = false
 -f  --file    file with csv data                     = ../data/auto93.csv
 -F  --Far     how far to look for poles (max=1)      = .95
 -g  --go      start-up example                       = nothing
 -h  --help    show help                              = false
 -m  --min     min size. If<1 then t^min else min.    = 10
 -n  --nums    number of nums to keep                 = 512
 -p  --p       distance calculation coefficient       = 2
 -r  --rest    size of "rest" set                     = 3
 -s  --seed    random number seed                     = 10019
 -S  --Sample  how many numbers to keep               = 10000]]

local any,cli,copy,csv,lt,many,map= _.any,_.cli,_.copy,_.csv,_.lt,_.many,_.map
local o,obj,oo,per,push,rnd,rogues= _.o,_.obj,_.oo,_.per,_.push,_.rnd,_.rogues
local shallowCopy,shuffle,sort  = _.shallowCopy,_.shuffle,_.sort
local Egs,Num,Row,Some,Sym = obj"Egs",obj"Num",obj"Row",obj"Some",obj"Sym"
-- ---------------------------------------------------------------------------
function Sym:new(c,x) return {at=c or 0,txt=x or "",n=0,has={}} end
function Sym:add(x)
  if x~="?" then self.n =1+self.n;self.has[x]=1+(self.has[x] or 0) end end
function Sym:dist(v1,v2)
  return  v1=="?" and v2=="?" and 1 or v1==v2 and 0 or 1 end

function Sym:entropy(     e,fun)
  function fun(p) return p*math.log(p,2) end
  e=0; for _,n in pairs(self.has) do if n>0 then e=e-fun(n/self.n) end end
  return e end

-- ---------------------------------------------------------------------------
function Some:new(c,x)
  return {at=c or 0, txt=x or "",n=0,isSorted=true, _has={}} end
function Some:nums()
  if not self.isSorted then table.sort(self._has) end
  self.isSorted=true
  return self._has end

function Some:add(v,    pos)
  if v~="?" then
    self.n=self.n+1
    if #self._has < the.Sample then pos=1+(#self._has)
    elseif math.random()<the.Sample/self.n then pos=math.rand(#self._has) end
    if pos then self.isSorted=false
                self._has[pos]= v end end end

-- ---------------------------------------------------------------------------
function Num:new(c,x)
  return {at=c or 0,txt=x or "",lo=1E32,hi=-1E32, n=0, has=Some(),
          w=(x or ""):find"-$" and -1 or 1} end
function Num:add(x)
  if x~="?" then self.n = self.n+1
                 self.lo = math.min(x,self.lo)
                 self.hi = math.max(x,self.hi)
                 self.has:add(x)  end end
function Num:norm(n,   lo,hi)
  lo,hi=self.lo,self.hi
  return n=="?" and n or (hi-lo < 1E-0 and 0 or  (n-lo)/(hi-lo + 1E-32)) end
function Num:pers(t,    a)
  a=self.has:nums()
  return map(t,function(p) return per(a,p) end) end

function Num:dist(v1,v2)
  if   v1=="?" and v2=="?" then return 1 end
  v1,v2 = self:norm(v1), self:norm(v2)
  if v1=="?" then v1 = v2<.5 and 1 or 0 end
  if v2=="?" then v2 = v1<.5 and 1 or 0 end
  return math.abs(v1-v2) end
-- ---------------------------------------------------------------------------
function Egs:new(src) -- constructor
  self.rows, self.cols = {}, {all={},x={},y={}}
  if   type(src)=="string"
  then csv(src,       function(row) self:add(row) end)
  else map(src or {}, function(row) self:add(row) end) end  end

function Egs:clone(  src,    out) -- copy structure
  out= Egs( {map(self.all, function(col) return col.txt end)} )
  map(src or {}, function (row) out:add(row) end)
  return out end

function Egs:add(row) -- the new row is either a header, or a data row
  if #self.cols.all==0 then self:header(row) else self:body(row) end end

function Egs:header(row) -- build the column headers
  for c,x in pairs(row) do
    local col = push(self.cols.all, (x:find"^[A-Z]" and Num or Sym)(c,x))
    if not x:find":$" then
      push(x:find"[!+-]" and self.cols.y or self.cols.x, col) end end end

function Egs:body(row)
  push(self.rows, row)
  for _,cols in pairs{self.cols.x, self.cols.y} do
    for _,col in pairs(cols) do
      col:add(row[col.at]) end end end

function Egs:better(row1,row2) -- is row1 better than row2
  local s1,s2,d,n,x,y,ys=0,0,0,0
  ys = self.cols.y
  for _,col in pairs(ys) do
    x,y= row1[col.at], row2[col.at]
    x,y= col:norm(x), col:norm(y)
    s1 = s1 - 2.71828^(col.w * (x-y)/#ys)
    s2 = s2 - 2.71828^(col.w * (y-x)/#ys) end
  return s1/#ys < s2/#ys end

function Egs:betters(rows) -- sort a set of rows
  return sort(rows or self.rows,
              function(r1,r2) return self:better(r1,r2) end) end

function Egs:cheat(   ranks) -- return percentile ranks for rows
  ranks={}
  for i,row in pairs(self:betters()) do
    ranks[row[1]] = math.floor(.5+ 100*i/#self.rows) end
  return self.rows,ranks end

function Egs:dist(row1,row2,    d,n,d1) -- distance between rows
  d,n = 0,0; for i,col in pairs(self.cols.x) do
               d1   = col:dist(row1[col.at], row2[col.at])
               n, d = n + 1,  d + d1^the.p end
  return (d/n)^(1/the.p) end

function Egs:around(r1,rows) -- sort `rows` by distance to `r11.
  return sort(map(rows,
                  function(r2) return {r=r2,d=self:dist(r1,r2)} end),lt"d") end

function Egs:far(row,rows) return per(self:around(row,rows),the.far).r end

function Egs:half(  above, -- split data by distance to two distant points
                  some,x,y,c,rxs,xs,ys)
  some = many(self.rows, the.Sample)
     x = above or self:far(any(some),some)
     y = self:far(x,some)
     c = self:dist(x,y)
   rxs = function(r) return
                  {r=r, x=(self:dist(r,x)^2 + c^2 - self:dist(r,y)^2)/(2*c)} end
  xs,ys= self:clone(), self:clone()
  for j,rx in pairs(sort(map(self.rows,rxs),lt"x")) do
    if j<=#self.rows/2 then xs:add(rx.r) else ys:add(rx.r) end end
  return {xs=xs, ys=ys, x=x, y=y, c=c} end

function Egs:best(  above,stop,evals) --recursively divide, looking 4 best leaf
  stop = stop or (the.min >=1 and the.min or (#self.rows)^the.min)
  evals= evals or 2
  if   #self.rows < stop
  then return self,evals
  else local node = self:half(above)
       if    self:better(node.x,node.y)
       then  return node.xs:best(node.x, stop, evals+1)
       else  return node.ys:best(node.y, stop, evals+1) end end end

function Egs:fours( rows,stop,evals,above, four)
  local pop,bests
  stop = stop or the.min >=1 and the.min or (#self.rows)^the.min
  evals= evals or {}
  pop  = table.remove
  print(#rows)
  four = self:betters{above or pop(rows), pop(rows), pop(rows), pop(rows)}
  map(four,oo)
  --for _,row in pairs(four) do evals[row[1]] = true end
  bests ={}
  for _,row in pairs(rows) do
    if four[1][1] == self:around(row,four)[1].r[1] then push(bests,row) end
  end
  print("::",four[1][1],stop,#bests,#rows)
  if   #bests >= stop and #bests < #rows
  then return self:fours(bests,stop,evals,four[1])
  else return bests,evals end   end
-- ---------------------------------------------------------------------------
local go = {}
local function goes(    fails,old)
  the = cli(the)
  fails=0
  old = copy(the)
  for k,fun in pairs(go) do
    if the.go == "all" or the.go == k then
      for k,v in pairs(old) do the[k]=v end
      math.randomseed(the.seed)
      print("\n>>>>>",k)
      if not fun() then fails = fails+1 end end end
  rogues()
  os.exit(fails) end

function go.the() oo(the); return true end

function go.num(  z)
  z=Num(); for i=1,100 do z:add(i) end; print(z); return true end

function go.sym(  z)
  z=Sym(); for _,x in pairs{1,1,1,1,2,2,3} do z:add(x) end;
  print(z); return true end

function go.eg( d)
  d=Egs(the.file);  map(d.cols.x,print) return true end

function go.dist(    num,d,r1,r2,r3)
  d=Egs(the.file)
  num=Num()
  for i=1,20 do
    r1= any(d.rows)
    r2= any(d.rows)
    r3= d:far(r1, d.rows)
    io.write(rnd(d:dist(r1,r3))," ")
    num:add(rnd(d:dist(r1,r2))) end
  oo(sort(num.has:nums()))
  print(#d.rows)
  return true end

function go.sort(     d,rows,ranks)
  d = Egs(the.file)
  rows,ranks = d:cheat()
  for i=1,#d.rows,32 do print(i,ranks[rows[i][1]],o(rows[i])) end end

function go.clone(    d1,d2)
  d1 = Egs(the.file)
  d2 = d1:clone(d1.rows)
  oo(d1.cols.x[2])
  oo(d2.cols.x[2]) end

function go.half( d,node)
  d=Egs(the.file)
  node = d:half()
  print(#node.xs.rows, #node.ys.rows, d:dist(node.x, node.y))end

function go.best(     num)
  num=Num()
  for i=1,20 do
    local d=Egs(the.file)
    local _,ranks = d:cheat()
    shuffle(d.rows)
    local leaf,evals = d:best()
    for _,row in pairs(leaf.rows) do num:add(ranks[ row[1] ]) end end
  print(o(num:pers{.1,.3,.5,.7,.9}))
end

function go.bests(     num,tmp)
  num=Num()
  for i=1,20 do
    local d = Egs(the.file)
    d:cheat()
    shuffle(d.rows)
    tmp=d:best()
    map(tmp,function(row) num:add(row.rank) end) end
  print(#tmp,o(num:pers{.1,.3,.5,.7,.9}))
  return end

function go.discretize(   d)
  d=Egs(the.file)
  print(d:xentropy()); return true end

function go.four(    d,ranks,rows,evals)
  d=Egs(the.file)
  --_,ranks= d:cheat()
  rows,evals=d:fours(shuffle(d.rows))
  --print(#rows)
--oo(map(rows,function(row) io.write(ranks[row[1]] ," ") end)) end
end

goes()
```