```lua
local help= [[

SHORTR: semi-supervised multi-objective optimization
(c) 2022 Tim Menzies <timm@ieee.org> BSD2 license

Explore N points via  O(log2(N)) evaluations. Generate a
human-readable summary of that space.  In pass1, find and
eval  two distant points using multi-objective criteria.
Everything nearest the  worst is pruned and we recurse on
the rest.  This algorithm is only approximate so, in pass2,
we do it all again, starting with the better items seen in
pass1.  Explain the final results by  a decision tree that
recursively discretizes numerics via their ability to
distinguish the best/worst things found in pass2.

USAGE:
  lua shortr.lua [OPTIONS]

OPTIONS:

OPTIONS:
  -M  --Min    min size of space                  =  .5
  -b  --bins   max number of bins                 =  16
  -k  --k      NB hack, low attribute frequency   =  2
  -m  --m      NB back, low class frequency       =  2
  -s  --some   max number of nums to keep         =  256
  -w  --wait   wait this number before testing    =  10

OPTIONS (other):
  -f  --file   file            =  data/auto93.csv
  -g  --go     start-up goal   =  nothing
  -h  --help   show help       =  false
  -s  --seed   seed            =  10019]]

--------------------------------------------------------------------------------
-- ## Names
--
-- `the` stores settings for this code. <p>As to the classes used by this system:
--
-- - `Row` hold the `cells` or record and a pointer (`of`) back to the
--   container that made them.
-- - `Col` summarizes columns. One `Col` can be for
--   numerics or symbolic columns (denoted with ` aCol.nums`).
-- - `Data` holds many `Row`s, summarized in a table `aData.cols`
--   (where `aData.cols.x` holds independent columns and
--    `aData.cols.y` holds dependent columns).
-- - `Bin` is a helper class that summarizes what dependent `ys` values are
--   found between `lo` and `hi` of an independent column.
-- - `NB` is an application class that implements a Naive Bayes classifier.
local b4={}; for x,_ in pairs(_ENV) do b4[x]=x end
local _    = require"lib1"
local Abcd = require"abcd"

local argmax,atom,big,cli,csv,demos = _.argmax,_.atom,_.big,_.cli,_.csv,_.demos
local fmt,lt,map,o,oo,per,push      = _.fmt,_.lt,_.map,_.o,_.oo,_.per,_.push
local R,sort,splice,sum             = _.R, _.sort, _.splice, _.sum

local the={}
help:gsub("[-][-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x) the[key] = atom(x) end)

local Col,Data,Row,Bin,NB = {},{},{},{},{}

--------------------------------------------------------------------------------
-- ## class Col
-- Summaries a column of data. Uses different types for numeric or other data.

--> .NEW(at:?int, txt:?str) :Col -> constructor of columns.
-- `.ok` is set to false after every update then set back
-- to true if ever we update the columns (see `Col.ok`).
function Col.NEW(at,txt)
  return {n  =0,     at=at or 0, txt=txt or "",
          ok =false, kept={},
          div=0,     mid=0} end

--> .NUM(at:?int, txt:?str) :Col -> constructor, specialized for numerics.
-- Numbers have a weight (-1,1) as well as the manddate to keep
-- no more than  `aNum.nums` samples.
function Col.NUM(at,txt,some,   i)
   i     = Col.NEW(at,txt) -- numerics are an extension to general columns.
   i.w   = Col.WEIGHT(txt)
   i.nums= some or the.some -- if non-nil the i.nums is a numeric
   return i end
-- ### Factory to make Cols

--> .GOAL(x:[str]) :bool ->
--> .NUMP(x:[str]) :bool ->
--> .KLASS(x:[str]) :bool ->
--> .SKIP(x:[str]) :bool -> recognize different column types
function Col.GOAL(x)   return (x or ""):find"[!+-]$" end
function Col.NUMP(x)   return (x or ""):find"^[A-Z]" end
function Col.KLASS(x)  return (x or ""):find"!$"  end
function Col.SKIP(x)   return (x or ""):find":$"  end

--> .WEIGHT(x:[str]) :(-1|1) -> assign column weight.e.g. "-1" means "minimize",
function Col.WEIGHT(x) return (x or ""):find"-$" and -1 or 1 end

--> .COLS(names:[str]) :tab -> constructor (builds `Col`s from list of `names`).
-- Returns a table that stores dependents in `.y`, independents in `.x`,
-- the klass (if it exists)in `.klass`. Caveat:
-- only if we are not `.SKIP()`ping them.
function Col.COLS(names)
  local i={x={}, y={}, names=names, klass=nil}
  for at,txt in pairs(names) do
    local new = Col.NUMP(txt) and Col.NUM(at,txt) or Col.NEW(at,txt)
    if not Col.SKIP(txt) then
      push(Col.GOAL(txt) and i.y or i.x, new)
      if Col.KLASS(txt) then i.klass=new end end end
  return i end
-- ### Update

--> .add(i:Col, v:any, inc:?int) :Col ->  update `i` with  `v ` ( inc  times)
-- Numeric columns keep a sample of the numbers while other columns track the
-- frequency of symbols seen so far.  The larger the sample, the less often
-- we update the numerics.
function Col.add(i,v,inc)
 inc = inc or 1
 if   v ~= "?"
 then i.n = i.n + inc
      if i.nums
      then for __=1,inc do
             if     #i.kept < i.nums then i.ok=false;push(i.kept,v)
             elseif R() < i.nums/i.n then i.ok=false;i.kept[R(#i.kept)]=v end end
      else i.ok = false
           i.kept[v] = inc + (i.kept[v] or 0) end end
  return i end
```

```lua
-- ### Computing derived properties

--> .ok(i:Col) -> ensure that the current contents are up to date. Returns `kept`.
-- E.g. update `mid`dle and `div`ersity (_median_ and _standard
-- deviation_ for numerics; and _mode_ and _entropy_ for others).<p>
-- This code uses the idiom "(per(.9) - per(.1))/2.56" to find
-- standard deviation. To grok that,
-- recall that &pm;1 and &pm;2
-- standard deviations marks out 66 to  95% of the mass. Somewhere in
-- between (at &pm;1.28), we get to 90% of the mass.  So to find one
-- standard deviation, divide the 90th minus 10th percentile by twice 1.28 (2.56).
function Col.ok(i)
  if   not i.ok
  then i.div, i.mid = 0, 0
       if   i.nums
       then i.kept = sort(i.kept) -- very fast since "kept" is small
            i.mid  = per(i.kept, .5) -- median
            i.div  = (per(i.kept, .9) - per(i.kept, .1)) / 2.56 -- stdev
       else local most = -1 -- find the mode and ent
            for x,n in pairs(i.kept) do
              if n > most then most, i.mid = n, x end
              if n > 0 then i.div=i.div - n/i.n*math.log(n/i.n,2) end end end end
  i.ok = true
  return i.kept end
-- ### Querying
-- Most of these need to call `Col.ok()` first (to ensure column is up to date).

--> .lo(i:Col) :num ->
--> .hi(i:Col) :num ->
--> .div(i:Col) :num ->
--> .mid(i:Col) :any ->  `lo`west number, `hi`ghest number, `div`ersity, `mid`dle number.
function Col.lo(i)   Col.ok(i); return i.kept[1] end
function Col.hi(i)   Col.ok(i); return i.kept[#i.kept] end
function Col.div(i)  Col.ok(i); return i.div end
function Col.mid(i)  Col.ok(i); return i.mid end

--> .norm(i:Col,x:num) :0..1 -> normalize `x` 0..1 for lo..hi.
function Col.norm(i,x)
  local a=Col.ok(i); return a[#a]-a[1] < 1E-9 and 0 or (x-a[1])/(a[#a]-a[1]) end
--- ### For Discretization

--> .bin(i:Col,x:any) :any -> round numeric `x` to nearest `(hi-lo)/the.bins`
-- (and for non-numerics, just return `x`).
function Col.bin(i,x)
  if   i.nums then
    local lo,hi = Col.lo(i), Col.hi(i)
    local b=(hi - lo)/the.bins
    x = lo==hi and 1 or math.floor(x/b+.5)*b end
  return x end

--> .bin(i:Col,j:Col) :Col -> returns a combination of two columns.
function Col.merge(i,j,     k)
  k = (i.nums and Col.NUM or Col.NEW)(i.at, i.txt)
  for _,kept in pairs{i.kept, j.kept} do
    for v,inc in pairs(kept) do Col.add(k,v,inc) end end
  return k end

-->.simpler(i:col,this:col,that:col):bool->am `i` simpler than `this` and `that`?
function Col.simpler(i,this,that)
  return Col.div(i) <= (this.n*Col.div(this) + that.n*Col.div(that)) / i.n end
--- ### For Naive Bayes

function Col.like(i,x,prior)
  if   i.nums
  then local sd,mu=Col.div(i), Col.mid(i)
       return sd==0 and (x==mu and 1 or 0) or
             math.exp(-1*(x - mu)^2/(2*sd^2)) / (sd*((2*math.pi)^0.5))
  else return ((i.kept[x] or 0)+the.m*prior)/(i.n+the.m) end end
```

```lua
--------------------------------------------------------------------------------
-- ## Row
function Row.NEW(of,cells) return {of=of,cells=cells,evaled=false} end

function Row.better(i,j)
  local s1, s2, ys = 0, 0, i.of.cols.y
  for _,c in pairs(ys) do
    local x,y =  i.cells[c.at], j.cells[c.at]
    x,y = Col.norm(c, x), Col.norm(c, y)
    s1  = s1 - 2.7183^(c.w * (x-y)/#ys)
    s2  = s2 - 2.7183^(c.w * (y-x)/#ys) end
  return s1/#ys < s2/#ys  end

function Row.klass(i) return i.cells[i.of.cols.klass.at] end
--------------------------------------------------------------------------------
-- ## Data
function Data.NEW(t) return {rows={}, cols=Col.COLS(t)} end

function Data.ROWS(src,fun)
  if type(src)=="table" then for  _,t in pairs(src) do fun(t) end
                        else for    t in csv(src)   do fun(t) end end end

function Data.LOAD(src,    i)
  Data.ROWS(src,function(t)
    if i then Data.add(i,t) else i=Data.NEW(t) end end); return i end

function Data.clone(i,inits)
  local j=Data.NEW(i.cols.names)
  for _,t in pairs(inits or {}) do Data.add(j,t) end; return j end

function Data.add(i,t)
  t = t.cells and t or Row.NEW(i,t)
  push(i.rows, t)
  for _,cols in pairs{i.cols.x, i.cols.y} do
    for _,c in pairs(cols) do Col.add(c, t.cells[c.at]) end end
  return t end

function Data.mids(i,cols)
  local t={}
  for _,c in pairs(cols or i.cols.y) do t[c.txt] = Col.mid(c) end;return t end

function Data.like(i,row, nklasses, nrows)
  local prior,like,inc,x
  prior = (#i.rows + the.k) / (nrows + the.k * nklasses)
  like  = math.log(prior)
  for _,col in pairs(i.cols.x) do
    x = row.cells[col.at]
    if x and x ~= "?" then
      inc  = Col.like(col,x,prior)
      like = like + math.log(inc) end end
  return like end
--------------------------------------------------------------------------------
-- ## NB
function NB.NEW(src,report)
  local i  = {overall=nil, dict={}, list={}}
  report = report or print
  Data.ROWS(src, function(row)
    if not i.overall then i.overall = Data.NEW(row)  else -- (0) eat row1
      row = Data.add(i.overall, row)  -- XX add to overall
      if #i.overall.rows > the.wait then report(Row.klass(row), NB.guess(i,row)) end
      NB.train(i,row) end end)                -- add tp rows’s klass
  return i end

function NB.train(i,row)
  local kl = Row.klass(row)
  i.dict[kl] = i.dict[kl] or push(i.list, Data.clone(i.overall)) -- klass is known
  i.dict[kl].txt = kl                                   -- each klass knows its name
  Data.add(i.dict[kl],row) end                         -- update klass with row

function NB.guess(i,row)
  return argmax(i.dict,
    function(klass) return Data.like(klass,row,#i.list,#i.overall.rows) end) end
--------------------------------------------------------------------------------
-- ## Bin
function Bin.NEW(xlo, xhi, ys) return {lo=xlo, hi=xhi, ys=ys} end
function Bin.add(i,x,y)
  i.lo = math.min(i.lo, x)
  i.hi = math.max(i.hi, x)
  Col.add(i.ys, y) end

function Bin.merge(i,j, min)
  local k = Col.merge(i,j)
  if i.n < min or j.n<min or Col.simpler(k,i,j)  then return k end end

function Bin.BINS(listOfRows,col,y)
  local n,list, dict = 0,{}, {}
  for label,rows in pairs(listOfRows) do
    for _,row in pairs(rows) do
      local v = row[col.at]
      if v ~= "?" then
        n = n + 1
        local pos = Col.bin(col,v)
        dict[pos] = dict[pos] or push(list, Bin.new(v,v,Col.new(col.at,col.txt)))
        Bin.add(dict[pos], v, label) end end end
    list = sort(list, lt"lo")
    list = col.nums and Bin.MERGES(list, n^the.min) or list
    return {bins= list,
            div = sum(list,function(z) return Col.div(z.ys)*z.ys.n/n end)} end

function Bin.MERGES(b4, min)
  local n,now = 1,{}
  while n <= #b4 do
    local merged = n<#b4 and Bin.merge(b4[n], b4[n+1], min)
    now[#now+1]  = merged or b4[n]
    n            = n + (merged and 2 or 1)  end
  if   #now < #b4
  then return Bin.MERGES(now,min) -- loop to look for other merges
  else -- stretch the bins to cover any gaps from minus infinity to plus infinity
       for n=2,#now do now[n].lo = now[n-1].hi end
       now[1].lo, now[#now].hi = -big, big
       return now end end
```

```
--------------------------------------------------------------------------------
-- To disable a test, relabel it from `Go` to `No`.
local Go,No = {},{}

function Go.THE() oo(the); return true end

function Go.ROWS(  d)
  Data.ROWS(the.file,function(row)
    if not d then d=Data.NEW(row) else
       Data.add(d,row) end end)
  oo(Data.mids(d))
  return true end

function Go.STATS()
  oo(Data.mids(Data.LOAD(the.file) )); return true end

function Go.ORDER(  i,t,m,left,right)
  i= Data.LOAD(the.file)
  t= sort(i.rows,Row.better)
  m= (#t)^.5
  left = Data.clone(i,splice(t,1,m))
  right= Data.clone(i,splice(i.rows,#t - m))
  print("all",  o(Data.mids(i)))
  print("best", o(Data.mids(left)))
  print("rest", o(Data.mids(right)))
  return true end

function Go.DIABETES(f,  i,t,a)
  a = Abcd.NEW()
  NB.NEW(f or "data/diabetes.csv",function(x,y) Abcd.add(a,x,y) end)
  Abcd.pretty(a,Abcd.report(a))
  return true end

function Go.SOYBEAN()  return Go.DIABETES("data/soybean.csv") end

--------------------------------------------------------------------------------
if    pcall(debug.getlocal, 4, 1)
then  return {DATA=DATA,ROW=ROW, COL=COL, the=the,lib=lib}
else  the = cli(the,help)
      demos(the,Go) end
```