```lua
#!/usr/bin/env lua
local lib=require"lib"
local the=lib.init[[

./duo.lua [OPTIONS]
(c)2022 Tim Menzies, MIT license

Data miners using/used by optimizers.
Understand N items after log(N) probes, or less.

OPTIONS:
  -ample   when enough is enough =  512
  -enough  use (#t)^enough       =  .5
  -far     how far to go         =  .9
  -file    read data from file   =  data/auto93.csv
  -help    show help             =  false
  -p       distance coefficient  =   2
  -seed    random number seed    =  10019
  -task    start up actions      =  donothing]]

local _=lib
local map, mapp, fmt, new, sort, push = _.map, _.mmap, _.fmt, _.new, _.sort
local push, o,   oo,  asserts         = _.push, _.o,  _.oo,   _.asserts
local EGS, NUM, RANGE, SYM            = {},{},{},{}
-- ---------------------------------------------------------------------------
function RANGE.new(k,col,lo,hi,b,B,r,R)
  return new(k,{col=col,lo=lo,hi=hi or lo,b=b,B=B,r=r,R=R}) end

function RANGE.__lt(i,j) return i:val() < j:val() end
function RANGE.merge(i,j,k,   lo,hi)
  lo = math.min(i.lo, j.lo)
  hi = math.max(i.hi, j.lhi)
  k = RANGE:new(i.col,lo,hi,i.b+j.b,i.B+j.B,i.r+j.r, i.R+j.R)
  if k:val() > i:val() and j:val() then return k end end

function RANGE.__tostring(i)
  if i.lo == i.hi       then return fmt("%s == %s", i.col.txt, i.lo) end
  if i.lo == -math.huge then return fmt("%s < %s",  i.col.txt, i.hi) end
  if i.hi ==  math.huge then return fmt("%s >= %s", i.col.txt, i.lo) end
  return fmt("%s <= %s < %s", i.lo, i.col.txt, i.hi) end

function RANGE.val(i,   z,B,R)
  z=1E-31; B,R = i.B+z, i.R+z; return (i.b/B)^2/( i.b/B + i.r/R) end

function RANGE.selects(i,row,    x)
  x=row.has[col.at]; return x=="?" or i.lo<=x and x<i.hi end
-- ---------------------------------------------------------------------------
function NUM.new(k,at,s)
  return new(k,{at=at,txt=s,w=s:find"-" and -1 or 1,_has={},
                   ok=false, lo=math.huge, hi=-math.huge}) end

function NUM.add(i,x)
  if x ~= "?" then
    i.ok = false
    push(i._has, x)
    if x < i.lo then i.lo = x end
    if x > i.hi then i.hi = x end end
  return x end

function NUM.dist(i,a,b)
  if     a=="?" and  b=="?" then a,b=1,0
  elseif a=="?" then b   = i:norm(b); a=b>.5 and 0 or 1
  elseif b=="?" then a   = i:norm(a); b=a>.5 and 0 or 1
  else               a, b= i:norm(a), i:norm(b) end
  return math.abs(a-b) end

function NUM.has(i)
  if not i.ok then sort(i._has); i.ok=true end; return i._has end

function NUM.norm(i,x)
  return i.hi - i.lo<1E-9 and 0 or (x - i.lo)/(i.hi - i.lo) end

-- compare to old above
function NUM.ranges(i,j,lo,hi)
  local z,is,js,lo,hi,m0,m1,m2,n0,n1,n2,step,most,best,r1,r2
  is,js   = i:has(), j:has()
  lo,hi   = lo or is[1], hi or is[#is]
  gap,max = (hi - lo)/16, -1
  if hi-lo < 2*gap then
    z       = 1E-32
    m0, m2 = lib.search(is, lo),lib.bsearch(is, hi+z)
    n0, n2 =lib.bsearch(js, lo),lib.bsearch(js, hi+z)
    --                  col,lo hi,b     B   r     R
    best    = nil
    for mid in lo,hi,gap do
      if mid > lo and k < hi then
        m1 = bsearch(is, mid+z)
        n1 = bsearch(js, mid+z)
        --             col,  lo hi, b     B   r         R
        r1 = RANGE:new(i,    lo,mid,m1-m0,i.n,m2-(m1+1),j.n)
        r2 = RANGE:new(i, mid+z,hi, n1-n0,i.n,n2-(n1+1),j.n)
        if r1:val() > max then best, max = r1, r1:val() end
        if r2:val() > max then best, max = r2, r2:val() end end end end
  if   best
  then return i:ranges(j, best.lo, best.hi)
  else return RANGE:new(i,  lo,hi,m2-m0,i.n,n2-n0,j.n) end end

-- ---------------------------------------------------------------------------
function SYM.new(k,at,s)
  return new(k,{at=at,txt=s,_has={}}) end

function SYM.add(i,x)
  if x ~= "?" then i._has[x] = 1+(i._has[x] or 0) end
  return x end

function SYM.dist(i,a,b)
  return  a=="?" and b=="?" and 1 or a==b and 0 or 1 end

function SYM.has(i)  return i.has end

function SYM.ranges(i,j)
  return mapp(i._has,
      function(x,n) return RANGE:new(i,x,x,n,i.n,(j._has[k] or 0),j.n) end) end
-- ---------------------------------------------------------------------------
function EGS.new(k,file,   i)
  i= new(k,{_rows={}, cols=nil, x={},  y={}})
  if file then for row in lib.rows(file) do i:add(row) end end
  return i end

function EGS.add(i,t)
  local add,now,where = function(col) return col:add(t[col.at]) end
  if   i.cols
  then push(i._rows, map(i.cols, add))
  else i.cols = {}
       for n,x in pairs(t) do
         now = (x:find"^[A-Z]" and NUM or SYM):new(n,x)
         push(i.cols, now)
         if not x:find":" then
           where = (x:find"+" or x:find"-") and i.y or i.x
           push(where, now) end end end end

function EGS.clone(i,inits,    j)
  j = EGS:new()
  j:add(map(i.cols, function(col) return col.txt end))
  for _,row in pairs(inits or {}) do j = j:add(row) end
  return j end

function EGS.cluster(i,top,lvl,         tmp1,tmp2,left,right)
  top = top or i
  lvl = lvl or 0
  print(fmt("%s%s", string.rep(".",lvl),#i._rows))
  if #i._rows >= 2*(#top._rows)^the.enough then
    tmp1, tmp2 = top:half(i._rows)
    if #tmp1._rows < #i._rows then left  = tmp1:cluster(top,lvl+1) end
    if #tmp2._rows < #i._rows then right = tmp2:cluster(top,lvl+1) end
  end
  return {here=i, left=left, right=right} end

function EGS.dist(i,r1,r2)
  local d,n,inc = 0, (#i.x)+1E-31
  for _,col in pairs(i.x) do
    inc = col:dist(r1[col.at], r2[col.at])
    d   = d + inc^the.p end
  return (d/n)^(1/the.p) end

function EGS.far(i,r1,rows,        fun,tmp)
  fun = function(r2) return {r2, i:dist(r1,r2)} end
  tmp = sort(map(rows,fun), seconds)
  return table.unpack(tmp[#tmp*the.far//1] ) end

function EGS.half(i,rows)
  print(11)
  local some,left,right,c,cosine,lefts,rights
  rows    = rows or i._rows
  print(#rows)
  some    = #rows > the.ample and lib.many(rows, the.ample) or rows
  left    = i:far(lib.any(rows), some)
  right,c = i:far(left,          some)
  function cosine(r,     a,b)
    a, b = i:dist(r,left), i:dist(r,right); return {(a^2+c^2-b^2)/(2*c),r} end
  lefts,rights = i:clone(), i:clone()
  for n,pair in pairs(sort(map(rows,cosine), firsts)) do
    (n <= #rows/2 and lefts or rights):add( pair[2] ) end
  return lefts,rights,left,right,c end
-- ---------------------------------------------------------------------------
local no,go={},{}

function go.any(   t,x,n)
  t={}; for i=1,10 do t[1+#t] = i end
  n=0; for i=1,5000 do x=lib.any(t); n= 1 <= x and x <=10 and n+1 or 0 end
  asserts(n==5000,"any")  end

function no.bsearch(   t,z)
  --          1  2  3  4  5  6  7  8  9  10
  z,t=1E-16, {10,10,10,20,20,30,30,40,50,200}
  print(brange(t,200)) end

function go.oo(  u)      oo{10,20,30} end
function go.rows()       for row in lib.rows(the.file) do oo(row) end end
function go.egs(   i)    i=EGS:new(the.file); map(i.y,oo) end
function go.half(  a,b)  a,b=EGS:new(the.file):half() end

the(go)
```