```lua
#!/usr/bin/env lua
local b4={}; for k,v in pairs(_ENV) do b4[k]=v end; --[[
         _ _ _   _   _        _ _ _       
  __ _  | (_) |_| |_| | ___  | (_) |_ ___ 
 / _` | | | | __| __| |/ _ \ | | | __/ _ \
| (_| | | | | |_| |_| |  __/ | | | ||  __/
 \__,_| |_|_|\__|\__|_|\___| |_|_|\__\___|
   __              _        _                       _             
  / /  /\ /\      /_\      | | ___  __ _ _ __ _ __ (_)_ __   __ _ 
 / /  / / \ \    //_\\     | |/ _ \/ _` | '__| '_ \| | '_ \ / _` |
/ /___\ \_/ /   /  _  \    | |  __/ (_| | |  | | | | | | | | (_| |
\____/ \___/    \_/ \_/    |_|\___|\__,_|_|  |_| |_|_|_| |_|\__, |
 _ _ _                                                      |___/ 
| (_) |__  _ __ __ _ _ __ _   _ 
| | | '_ \| '__/ _` | '__| | | |
| | | |_) | | | (_| | |  | |_| |
|_|_|_.__/|_|  \__,_|_|   \__, |
                          |___/     --]] local options={

what = "Small sample multi-objective optimizer.",
usage= "(c) 2021 Tim Menzies <timm@ieee.org> unlicense.org",
about= [[
Sort N examples on multi-goals using a handful of 'hints'; i.e.

- Evaluate and rank, a few examples (on their y-values);
- Sort other examples by x-distance to the ranked ones;
- Recurse on the better half (so we sample more and more
  from the better half, then quarter, then eighth...).

A regression tree learner then explores the examples (sorted
left to right, worst to best).  By finding branches that
reduce the variance of the index of those examples, this
tree reports what attribute ranges select for the better (or
worse) examples.  ]],

how= {{"file",     "-f",  "../../data/auto93.csv",  "read data from file"},
      {"cull",     "-c",   .5     ,"cuts per repeat"},
      {"help",     "-h",   false   ,"show help"                    },
      {"hints",    "-H",   4       ,"hints per generation"       },
      {"p",        "-p",   2       ,"distance calc exponent"     },
      {"small",    "-s",   .5      ,"div list t into t^small"    },
      {"seed",     "-S",   10019   ,"random number seed"          },
      {"train",    "-t",   .5      ,"size of training set"       },
      {"trivial",  "-T",   .35     ,"small delta=trivial*sd"     },
      {"todo",     "-T",   "all"   ,"run unit test, or 'all'"    },
      {"wild",     "-W",   false   ,"run tests, no protection"   }}}

local Seed,cli,the
Seed=10019
-- If `-x X` appears on command line and `-x default` is in `how`
-- then update default from the command  line (and if `default`)
-- is false, then set it to true. Also, maybe
-- set random number seed and maybe show help string.
function cli(opt,   u)
  u={}
  for _,t in pairs(opt.how) do
    u[t[1]] = t[3]
    for n,word in ipairs(arg) do if word==t[2] then
      u[t[1]] = t[3] and (tonumber(arg[n+1]) or arg[n+1]) or true end end end
  if   u.help
  then print(string.format("\n%s [OPTIONS]\n%s\n%s\n\nOPTIONS:\n",
             arg[0],opt.usage,opt.what))
       for _,t in pairs(opt.how) do print(string.format("%4s %-9s%s\t%s %s",
         t[2], t[3] and t[1] or"", t[4], t[3] and"=" or"", t[3] or "")) end
       print("\n"..opt.about)
       os.exit() end
  if u.seed then Seed = u.seed end
  return u end

-- Make a global for our options e.g. the = {seed=10019, help=false, p=2...}
the = cli(options)
```

```lua
--         _                 _   _ _    
--  _ __  (_)____ __   _  _| |_(_) |___
-- | '  \ | (_-</ _| | || |  _| | (_-<
-- |_|_|_||_/__/\__|  \_,_|\__|_|_/__/
------------------------------------------------------------------------------
-- ## Table Stuff
local randi -- defined later, needed now in "shuffle"
local cat,map,lap,keys, last,copy,pop,push,sort,firsts,first,second,shuffle,bchop
-- Table to string.
cat     = table.concat
-- Return a sorted table.
sort    = function(t,f) table.sort(t,f); return t end

-- Add to end, pull from end.
push    = table.insert
pop     = table.remove

-- Return first,second, last  item.
first   = function(t) return t[1] end
second  = function(t) return t[2] end
last    = function(t) return t[#t] end

-- Function for sorting pairs of items.
firsts  = function(a,b) return first(a) < first(b) end

-- Random order of items in a list (sort in place).
function shuffle(t,   j)
  for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end

-- Collect values, passed through 'f'.
function lap(t,f)  return map(t,f,1) end
-- Collect key,values, passed through 'f'.
-- If `f` returns two values, store as key,value.
-- If `f` returns one values, store at index value.
-- If `f' return nil then add nothing (so `map` is also `select`).
function map(t,f,one,     u)
  u={}; for x,y in pairs(t) do
    if one then x,y=f(y) else x,y=f(x,y) end
    if x ~= nil then
      if y then u[x]=y else u[1+#u]=x end end end
  return u end

-- Return a table's keys (sorted).
function keys(t,u)
  u={}
  for k,_ in pairs(t) do if tostring(k):sub(1,1)~="_" then push(u,k) end end
  return sort(u)
end

-- Binary chop (assumes sorted lists)
function bchop(t,val,lt,lo,hi,     mid)
  lt = lt or function(x,y) return x < y end
  lo,hi = lo or 1, hi or #t
  while lo <= hi do
    mid =(lo+hi) // 2
    if lt(t[mid],val) then lo=mid+1 else hi= mid-1 end end
  return math.min(lo,#t)  end

------------------------------------------------------------------------------
-- ## Maths Stuff
local abs,norm,sum,rnd,rnds,Seed,rand
abs = math.abs

-- Round `x` to `d` decimal places.
function rnd(x,d,  n) n=10^(d or 0); return math.floor(x*n+0.5) / n end

-- Round list of items to  `d` decimal places.
function rnds(t,d) return lap(t, function(x) return rnd(x,d or 2) end) end

-- Sum items, filtered through `f`.
function sum(t,f)
  f= f or function(x) return x end
  out=0; for _,x in pairs(f) do out = out + f(x) end; return out end

Seed=937162211
function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
function rand(lo,hi,     mult,mod)
  lo, hi = lo or 0, hi or 1
  Seed = (16807 * Seed) % 2147483647
  return lo + (hi-lo) * Seed / 2147483647 end

------------------------------------------------------------------------------
-- ## Printing Stuff
local out,shout,red,green,yellow,blue,color,fmt
fmt = string.format

-- Print as red, green, yellow, blue.
function color(s,n) return fmt("\27[1m\27[%sm%s\27[0m",n,s) end
function red(s)     return color(s,31) end
function green(s)   return color(s,32) end
function yellow(s)  return color(s,34) end
function blue(s)    return color(s,36) end

-- Printed string from a nested structure.
shout= function(x) print(out(x)) end

-- Generate string from a nested structures
-- (and don't print any contents more than once).
function out(t,seen,    u,key,value,public)
  function key(k)   return fmt(":%s %s",blue(k),out(t[k],seen)) end
  function value(v) return out(v,seen) end
  if type(t) == "function" then return "(...)" end
  if type(t) ~= "table"    then return tostring(t) end
  seen = seen or {}
  if seen[t] then return "..." else seen[t] = t end
  u = #t>0 and lap(t, value) or lap(keys(t), key)
  return red((t._is or"").."{")..cat(u," ")..red("}") end

------------------------------------------------------------------------------
-- ## File i/o Stuff

-- Return one table per line, split on commans.
local csv
function csv(file,   line)
  file = io.input(file)
  line = io.read()
  return function(   t,tmp)
    if line then
      t={}
      for cell in line:gsub("[\t\r ]*",""):gsub("#.*",""):gmatch("([^,]+)") do
        push(t, tonumber(cell) or cell) end
      line = io.read()
      if #t>0 then return t end
    else io.close(file) end end end

------------------------------------------------------------------------------
-- ## OO Stuff
local has,obj

-- Create an instance
function has(mt,x) return setmetatable(x,mt) end

-- Create a clss
function obj(s, o,new)
   o = {_is=s, __tostring=out}
   o.__index = o
   return setmetatable(o,{__call = function(_,...) return o.new(...) end}) end
```

```lua
-------------------------------------------------------------------------------
-- ## Stuff for tracking `Sym`bol Counts.

-- `Sym`s track symbol counts and the `mode` (most frequent symbol).
local Sym=obj"Sym"
function Sym.new(inits,     self)
  self= has(Sym,{has={}, n=0, mode=nil, most=0})
  for _,one in pairs(inits or {}) do self:add(one) end
  return self end

function Sym:add(x)
  self.n = self.n + 1
  self.has[x] = 1 + (self.has[x] or 0)
  if self.has[x] > self.most then self.most, self.mode = self.has[x], x end end

function Sym:dist(a,b) return a==b and 0 or 1 end

function Sym:mid() return self.mode end

-------------------------------------------------------------------------------
-- ## Stuff for tracking `Num`bers.

-- `Num`s track a list of number, and can report  it sorted.
local Num=obj"Num"
function Num.new(inits,     self)
  self= has(Num,{has={}, n=0, lo=1E32, hi =1E-32, ready=true})
  for _,one in pairs(inits or {}) do self:add(one) end
  return self end

function Num:add(x)
  if     x>self.hi then self.hi = x
  elseif x<self.lo then self.lo = x end
  push(self.has,x); self.n=self.n+1; self.ready=false end

-- Ensure that the returned list of numbers is sorted.
function Num:all(x)
  if not self.ready then table.sort(self.has) end
  self.ready = true
  return self.has end

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

-- Combine two `num`s.
function Num:merge(other,    new)
  new = Num.new(self.has)
  for _,x in pairs(other.has) do new:add(x) end
  return new end

-- Return a merged item if that combination
-- is simpler than its parts.
function Num:mergeable(other,    new,b4)
  new = self:merge(other)
  b4  = (self.n*self:sd() + other.n*other:sd()) / new.n
  if b4 >= new:sd() then return new end end

-- The `mid` is the 50th percentile.
function Num:mid() return self:per(.5) end

-- Return `x` normalized 0..1, lo..hi.
function Num:norm(x,     lo,hi)
  if x=="?" then return x end
  lo,hi = self.lo, self.hi
  return abs(hi - lo) < 1E-32 and 0 or (x - lo)/(hi - lo) end

-- Return the `p`-th percentile number.
function Num:per(p,    t)
  t = self:all()
  p = p*#t//1
  return #t<2 and t[1] or t[p < 1 and 1 or p>#t and #t or p] end

-- The 10th to 90th percentile is 2.56 times the standard deviation.
function Num:sd() return (self:per(.9) - self:per(.1))/ 2.56 end

-------------------------------------------------------------------------------
-- discretization tricks
local splits={}
function splits.best(sample,    best,tmp,xpect,out)
  best = maths.huge
  for _,x in pairs(sample.xs) do
    tmp, xpect = splits.whatif(x.at,self)
    if   xpect < best
    then out,best = tmp,xpect end end
  return out end

function splits.whatif(col,sample,     out)
  out   = splits.spans(col,sample)
  xpect = sum(out, function(x) return x.has.n*x:sd() end)/#sample.egs
  out   = map(out, function(_,x) x.has=x.has:all(); x.col= col end)
  return out, xpect end

function splits.spans(col,sample,      xs, symbolic,x)
  xys,xs,  symbolic ={}, Num(), sample.nums[col]
  for rank,eg in pairs(sample.egs) do
    x = eg[col]
    if x ~= "?" then
      xs:add(x)
      if   symbolic
      then -- in symbolic columns, xys are the indexes seen with each symbol
        xys[x] = xys[x] or {}
        push(xys[x], rank)
      else -- in numeric columns,  xys are each number paired with its row id
        push(xys, {x=x,y=rank}) end end
  end
  if   symbolic
  then return map(xys, function(x,t) return {lo=x, hi=x, has=Num(t)} end)
  else return splits.merge(
                splits.div(xys, #xs^the.small, sd(sort(xs))*the.trivial)) end end

-- Generate a new range when
-- 1. there is enough left for at least one more range; and
-- 2. the lo,hi delta in current range is not boringly small; and
-- 3. there are enough x values in this range; and
-- 4. there is natural split here
-- Fuse adjacent ranges when:
-- 5. the combined class distribution of two adjacent ranges
--    is just as simple as the parts.
function splits.div(xys, tiny, dull,           now,out,x,y)
  xys = sort(xys, function(a,b) return a.x < b.x end)
  now = {lo=xys[1].x, hi=xys[1].x, has=Num()}
  out = {now}
  for j,xy in pairs(xys) do
    x, y = xy.x, xy.y
    if   j<#xys-tiny and x~=xys[j+1].x and now.has.n>tiny and now.hi-now.lo>dull
    then now = {lo=x, hi=x, has=Num()}
         push(out, now) end
    now.hi = x
    now.has:add(y) end
  return out end

function splits.merge(b4,       j,tmp,a,n,simpler)
  j, n, tmp = 0, #b4, {}
  while j<n do
    j = j + 1
    a = b4[j]
    if j < n-1 then
      simpler = a.has:mergeable(b4[j+1].has)
      if simpler then
        j = j + 1
        a = {lo=a.lo, hi= b4[j+1].hi, has=simpler} end end
    push(tmp,a) end
  return #tmp==#b4 and b4 or merge(tmp) end


```

```lua
-------------------------------------------------------------------------------
-- Samples store examples. Samples know about
-- (a) lo,hi ranges on the numerics
-- and (b) what  are independent `x` or dependent `y` columns.
local Sample=obj"Sample"
function Sample.new(     src,self)
  self = has(Sample,{names=nil, all={}, ys={}, xs={}, egs={}})
  if src then
    if type(src)=="string" then for x   in csv(src) do self:add(x)   end end
    if type(src)=="table" then for _,x in pairs(src) do self:add(x) end end end
  return self end

function Sample:clone(      inits,out)
  out = Sample.new():add(self.names)
  for _,eg in pairs(inits or {}) do out:add(eg) end
  return out end

function Sample:add(eg,      name,datum)
  function name(col,new,    weight, where, what)
    if new:find":" then return end
    weight= new:find"-" and -1 or 1
    what  = {col=col, w=weight, seen=(new:match("^[A-Z]",x) and Num() or Sym())}
    where = (new:find("+") or new:find("-")) and self.ys or self.xs
    push(self.all, what)
    push(where,    what)
  end -----------------
  function datum(one,new)
    if new ~= "?" then one.seen:add(new) end
  end -----------------
  if   not self.names
  then self.names = eg
       map(eg, function(col,x) name(col,x) end)
  else push(self.egs, eg)
       map(self.all, function(_,col) datum(col,eg[col.col]) end)
  end
  return self end

function Sample:better(eg1,eg2,     e,n,a,b,s1,s2)
  n,s1,s2,e = #self.ys, 0, 0, 2.71828
  for _,num in pairs(self.ys) do
    a  = num.seen:norm(eg1[num.col])
    b  = num.seen:norm(eg2[num.col])
    s1 = s1 - e^(num.w * (a-b)/n)
    s2 = s2 - e^(num.w * (b-a)/n) end
  return s1/n < s2/n end

function Sample:betters(egs)
  return sort(egs or self.egs,function(a,b) return self:better(a,b) end) end

function Sample:dist(eg1,eg2,     a,b,d,n,inc)
  d,n = 0,0
  for _,x in pairs(self.xs) do
    a,b = eg1[x.col], eg2[x.col]
    inc = a=="?" and b=="?" and 1 or x.seen:dist(a,b)
    d   = d + inc^the.p
    n   = n + 1 end
  return (d/n)^(1/the.p) end

function Sample:stats(cols)
  return lap(cols or self.ys,function(col) return col.seen:mid() end) end
-- bins his
-- bins sorts

function Sample:tree(min,      node,min,sub)
  node = {node=self, kids={}}
  min = min  or (#self.egs)^the.small
  if #self.egs >= 2*min then
    --- here
    for _,span in pairs(splits.best(sample)) do
      sub = self:clone()
      for _,at in pairs(span.has) do sub:add(self.egs[at]) end
      push(node.kids, span)
      span.has = sub:tree(min) end end
  return node end

-- at node
function Sample:where(tree,eg,    max,x,default)
  if #kid.has==0 then return tree end
  max = 0
  for _,kid in pairs(tree.node) do
    if #kid.has > max then default,max = kid,#kid.has end
    x = eg[kid.col]
    if x ~= "?" then
      if x <= kid.hi and x >= kid.lo then
        return self:where(kid.has.eg) end end end
  return self:where(default, eg) end


```

```lua

----------------------------------------------------------------------------------
-- sample sample sorting
local hints={}
function hints.default(eg) return eg end

function hints.sort(sample,score,    test,train,evals)
  sample = Sample.new(the.file)
  train,test = {}, {}
  for i,eg in pairs(shuffle(sample.egs)) do
     push(i<= the.train*#sample.egs and train or test, eg) end
  evals,train = hints.recurse(sample, train,0,
                          score or hints.default, {}, (#train)^the.small)
  return evals,sample:clone(train), sample:clone(test) end

function hints.recurse(sample, egs, evals, scorefun, out, small, worker)
  if #egs < small then
    for i=1, #egs do push(out, pop(egs)) end
    return evals,out
  end
  local scoreds = {}
  function worker(eg) return hints.locate(scoreds,eg,sample) end
  for j=1,the.hints do evals=evals+1;
                     push(scoreds, scorefun(pop(egs))) end
  scoreds = sample:betters(scoreds)
  egs     = lap(sort(lap(egs, worker),firsts),second)
  print(the.cull*#egs)
  for i=1,the.cull*#egs//1 do push(out, pop(egs)) end
  return hints.recurse(sample, egs,evals, scorefun, out, small) end

function hints.locate(scoreds,eg,sample,        closest,rank,tmp)
  closest, rank, tmp = 1E32, 1E32, nil
  for rank0, scored in pairs(scoreds) do
    tmp = sample:dist(eg, scored)
    if tmp < closest then closest,rank = tmp,rank0 end end
  return {rank, eg} end
  --return {rank+closest/10^6, eg} end


```

```lua
--         _
--      __| |   ___   _ __    ___   ___
--     / _` |  / -_) | '  \  / _ \ (_-<
--     \__,_|  \___| |_|_|_| \___/ /__/
----------------------------------------------------------------------------------
local eg,fail,example={},0
function example(k,      f,ok,msg)
  f= eg[k]; assert(f,"unknown action "..k)
  the=cli(options)
  if the.wild then return f() end
  ok,msg = pcall(f)
  if ok then print(green("PASS"),k)
  else       print(red("FAIL"),  k,msg); fail=fail+1 end end

function eg.shuffle(   t)
  t={}
  for i=1,32 do push(t,i) end
  assert(#t == #shuffle(t) and t[1] ~= shuffle(t)[1]) end

function eg.lap()
  assert(3==lap({1,2},function(x) return x+1 end)[2]) end

function eg.map()
  assert(3==map({1,2},function(_,x) return x+1 end)[2]) end

function eg.tables()
  assert(20==sort(shuffle({{10,20},{30,40},{40,50}}),firsts)[1][2]) end

function eg.csv(   n,z)
  n=0
  for eg in csv(the.file) do n=n+1; z=eg end
  assert(n==399 and z[#z]==50) end

function eg.rnds(    t)
  assert(10.2 == first(rnds({10.22,81.22,22.33},1))) end

function eg.sym(    s)
  s=Sym{"a","a","a","a","b","b","c"}
  assert("a"==s.mode) end

function eg.num1(    n)
  n=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(.375 == n:norm(25))
  assert(15.625 == n:sd()) end

function eg.num2(    n1,n2,n3,n4)
  n1=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n2=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(n1:mergeable(n2)~=nil)
  n3=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n4=Num{100,200,300,400,500,100,200,300,400,500,100,200,300,400,500}
  assert(n3:mergeable(n4)==nil) end

function eg.sample(    s,tmp,d1,d2,n)
  s=Sample(the.file)
  assert(2110 == last(s.egs)[s.all[3].col])
  local sort1= s:betters(s.egs)
  local lo, hi = s:clone(), s:clone()
  for i=1,20                 do lo:add(sort1[i]) end
  for i=#sort1,#sort1-30,-1 do hi:add(sort1[i]) end
  shout(s:stats())
  shout(lo:stats())
  shout(hi:stats())
  for m,eg in pairs(sort1) do
    n = bchop(sort1, eg,function(a,b) return s:better(a,b) end)
    assert(m-n <=2) end end

function eg.dists(    s,tmp,d1,d2,n)
  s=Sample(the.file)
  tmp = sort(lap(shuffle(s.egs),
                     function(eg2) return {s:dist(eg2,s.egs[1]), eg2} end),
               firsts)
   d1=s:dist(tmp[1][2], tmp[10][2])
   d2=s:dist(tmp[1][2], tmp[#tmp][2])
   assert(d1*10<d2)
end

function eg.hints(    s,_,__,evals,sort1,train)
  s=Sample(the.file)
  --for _,eg in pairs(sort1) do lap(s.ys, function(col) return eg[col.col] end ) end
  -- assert(s.ys[4].lo==1613)
  evals, train,test = hints.sort(s)
  print("=",evals)
  test.egs = test:betters()
  for m,eg in pairs(test.egs) do
     n = bchop(train.egs, eg,function(a,b) return s:better(a,b) end)
     print(m,n) end
  end

if the.todo=="all" then lap(keys(eg),example) else example(the.todo) end

----------------------------------------------------------------------------------
-- trick for checking for rogues.
for k,v in pairs(_ENV) do if not b4[k] then print("?rogue: ",k,type(v)) end end
os.exit(fail)



```

```
--[[
--  seems to be  a revers that i  need to do .... but dont

teaching:
- sample is v.useful


--]]
```