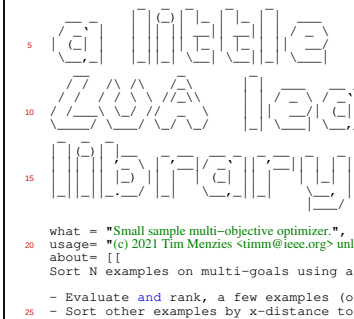
```
local b4={}; for k,v in pairs(_ENV) do b4[k]=v end --[[
```



```
]] 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"},
      {"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 fmt,help,cli,the
fmt = string.format
function help(opt)
  print(fmt("lua %s [ARGS]\n%s\n%s\n\nARGS:",arg[0],opt.usage,opt.what))
    for _,t in pairs(opt.how) do print(fmt("%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

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 help(opt) end
  math.randomseed(u.seed or 100019)
  return u end

the = cli(options) -- e.g. the = {seed=10019, help=false, p=2...}

-------------------------------------------------------------------------------
-- table tricks
local cat,map,lap,keys, copy,pop,push,sort,firsts,first,second,shuffle,bchop
cat     = table.concat
sort    = function(t,f) table.sort(t,f); return t end
push    = table.insert
pop     = table.remove
first   = function(t) return t[1] end
second  = function(t) return t[2] end
firsts  = function(a,b) return first(a) < first(b) end

function shuffle(t,   j)
  for i=#t,2,-1 do j=math.random(1,i); t[i],t[j]=t[j],t[i] end; return t end

function lap(t,f)  return map(t,f,1) end

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

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 tricks
local abs,norm,sum,rnd,rnds
abs = math.abs

function rnd(x,d,  n)
  n=10^(d or 0); return math.floor(x*n+0.5) / n end

function rnds(t,d)
  return lap(t, function(x) return rnd(x,d) end ) end

function norm(x,lo,hi)
  if x=="?" then return x end
  return abs(hi - lo) < 1E-32 and 0 or (x - lo)/(hi - lo) end

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

-------------------------------------------------------------------------------
-- printing tricks
local out,shout,red,green,yellow,blue
function red(s)    return "\27[1m\27[31m"..s.."\27[0m" end
function green(s)  return "\27[1m\27[32m"..s.."\27[0m" end
function yellow(s) return "\27[1m\27[33m"..s.."\27[0m" end
function blue(s)   return "\27[1m\27[36m"..s.."\27[0m" end

shout= function(x) print(out(x)) end

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 tricks
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 tricks
local has,obj
function has(mt,x) return setmetatable(x,mt) end
function obj(s, o,new)
   o = {_is=s, __tostring=out}
   o.__index = o
   return setmetatable(o,{__call = function(_,...) return o.new(...) end}) end

local Nums=obj"Nums"
function Nums.new(inits,     self)
  self= has(Nums,{has={}, n=0, ready=true})
  for _,one in pairs(inits or {}) do self:add(one) end
  return self end

function Nums:add(x)
  push(self.has,x); self.n=self.n+1; self.ready=false end

function Nums:all(x)
  if not self.ready then table.sort(self.has) end
  self.ready = true
  return self.has end

function Nums:per(p,    here,t)
  function here(x) x=x*#t//1; return x < 1 and 1 or x>#t and #t or x end
  t=self:all()
  return #t <2 and  t[1] or t[ here(p or .5) ] end

function Nums:sd() return (self:per(.9) - self:per(.1))/ 2.56 end

function Nums:merge(other,    new)
  new = Nums.new(self.has)
  for _,x in pairs(other.has) do new:add(x) end
  return new end

function Nums: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

-------------------------------------------------------------------------------
-- doscretization 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,xys, symbolic,x)
  xys,xs,  symbolic ={}, Nums(), 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=Nums(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=Nums()}
  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=Nums()}
         push(out, now) end
    now.hi = x
    now.has:add(y) end
  return out end

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

-------------------------------------------------------------------------------
-- 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, nums={}, 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,    tmp)
    if new:find":" then return end
    if not (new:find"+") or new:find"-") then self.xs[col]=col end
    if new:match"^[A-Z]" then
      tmp = {col=col, w=0, lo=1E32, hi=-1E22}
      self.nums[col] = tmp
      if new:find"-" then tmp.w=-1; self.ys[col] = tmp end
      if new:find"+" then tmp.w= 1; self.ys[col] = tmp end end
  end ----------------
  function datum(col,new)
    if self.nums[col] and new ~= "?" then
```

```
        self.nums[col].lo = math.min(new, self.nums[col].lo)
        self.nums[col].hi = math.max(new, self.nums[col].hi) 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(eg, function(col,x) datum(col,x) end) end
    return self 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

-- ordered object
-- per sd add sort here. mergabe

--------------------------------------------------------------------------------
-- geometry tricks
-- y column rankings
local dist, better,betters
function dist(eg1,eg2,sample,     a,b,d,n,inc,dist1)
  function dist1(num,a,b)
    if not num then return a==b and 0 or 1 end
    if     a=="?" then b=norm(b, num.lo, num,hi); a = b>.5 and 0 or 1
    elseif b=="?" then a=norm(a, num.lo, num.hi); b = a>.5 and 0 or 1
    else   a,b = norm(a, num.lo, num.hi), norm(b, num.lo, num.hi)
    end
    return abs(a-b)
  end ------------------------
  d,n=0,0
  for col,_ in pairs(sample.xs) do
    a,b = eg1[col], eg2[col]
    inc = a=="?" and b=="?" and 1 or dist1(sample.nums[col],a,b)
    d   = d + inc^the.p
    n   = n + 1 end
  return (d/n)^(1/the.p) end

function betters(egs,sample)
  return sort(egs,function(a,b) return better(a,b,sample) end) end

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

--------------------------------------------------------------------------------
-- 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 = betters(scoreds, sample)
  egs     = lap(sort(lap(egs, worker),firsts),second)
  for i=1,#egs//2 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 = dist(eg, scored, sample)
    if tmp < closest then closest,rank = tmp,rank0 end end
  return {rank+closest/10^6, eg} end

--------------------------------------------------------------------------------
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.norm()
  assert(norm(5,0,10)==.5,"small") 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.nums(    n)
  n=Nums{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(15.625 == n:sd()) end

function eg.nums(    n1,n2,n3,n4)
  n1=Nums{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n2=Nums{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(n1:mergeable(n2)~=nil)
  n3=Nums{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n4=Nums{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)
  s=Sample(the.file)
  assert(s.ys[4].lo==1613)
  tmp = sort(map(shuffle(s.egs),
                 function(_,eg2) return {dist(eg2,s.egs[1],s), eg2} end),
             firsts)
  d1=dist(tmp[1][2], tmp[10][2], s)
  d2=dist(tmp[1][2], tmp[#tmp][2], s)
  assert(d1*10<d2)
end

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

```
--[[
needs stats on samples

teaching:
- sample is v.useful


--]]
```