```lua
#!/usr/bin/env lua
--                _   _ _   _   _        _ _ _       
--        __ _  | (_) |_| |_| | ___   | (_) |_ ___ 
--       / _` | | | | __| __| |/ _ \  | | | __/ _ \
--      | (_| | | | | |_| |_| |  __/  | | | ||  __/
--       \__,_| |_|_|\__|\__|_|\___|  |_|_|\__\___|
--         __    _   _    _      _                      _             
--        / /   / \ / \  /_\    | | ___  __ _ _ __ _ __ (_)_ __   __ _ 
--       / /   / / \/ / //_\\   | |/ _ \/ _` | '__| '_ \| | '_ \ / _` |
--      / /___ \ \_/ / /  _  \  | |  __/ (_| | |  | | | | | | | | (_| |
--      \_____/ \___/ \_/ \_/   |_|\___|\__,_|_|  |_| |_|_|_| |_|\__, |
--       _ _ _                                                   |___/ 
--      | (_) |__  _ __ __ _ _ __ _   _ 
--      | | | '_ \| '__/ _` | '__| | | |
--      | | | |_) | | | (_| | |  | |_| |
--      |_|_|_.__/|_|  \__,_|_|   \__, |
--                                |___/ 
local it=require"options"{
what  = "Small sample multi–objective optimizer.",
who   = "(c) 2021 Tim Menzies <timm@ieee.org> unlicense.org",
why   = [[
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 generation"        },
     {"HELP",      "–h",   false  , "show help"                  },
     {"HINTS",     "–H",    4     ,"hints per generation"       },
     {"P",         "–p",    2     ,"distance calc exponent"     },
     {"TINY",      "–s",   .5     ,"div list t into t^small"    },
     {"SEED",      "–S",   10019  ,"random number seed"         },
     {"TRAIN",     "–t",   .5     ,"size of training set"       },
     {"TODO",      "–T",   "all"  ,"run unit test, or 'all'"    },
     {"TRIVIAL",   "–v",   .35    ,"small delta=trivial*sd"     },
     {"WILD",      "–W",   false  ,"run tests, no protection"   }}}

local _=require"lib"
local abs,bchop,cat,copy        = _.abs,     _.bchop, _.cat,     _.copy
local csv,first,firsts,fmt,has  = _.csv,     _.first, _.firsts,  _.fmt,    _.has
local keys,last,lap,map,obj     = _.keys,    _.last,  _.lap,     _.map,    _.obj
local out,pop,push,rand,shout   = _.out,     _.pop,   _.push,    _.rand,   _.shout
local rnd,rnds,rogues,second    = _.rnd,     _.rnds,  _.rogues,  _.second
local shuffle,sort,sum,top      = _.shuffle, _.sort,  _.sum,     _.top

--[[
Spans
 Little languages:
   - options
   - data language

Lesson plan
-- w1: ssytems: github. github workplaces. unit tests. doco tools.
-- w2: num,sym
-- W3: sample
-- w4: eval, knn, unfarinessness
-- W5:
--]]

```

```lua
-- NUM ----------------------------------------------------------------------
--
-- ## 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,at, txt,     self)
  self= has(Num,{at=at or 0, txt=txt or"", w=(txt or""):find"–" and -1 or 1,
                 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.at, new.txt = self.at, self.txt
  for _,x in pairs(self.has)  do new:add(x) end
  for _,x in pairs(other.has) do new:add(x) end
  return new 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 range is 2.56 times the standard deviation.
function Num:sd() return math.abs(self:per(.9) - self:per(.1))/ 2.56 end
function Num:spread() return self:sd() end

-- Create one span (each has the  row indexes of the rows)
-- where each span has at least `tiny` items and  span is more than
-- `tirvial`ly small.
local div -- defined below
function Num:spans(sample,tiny,trivial)
  local xys = {}
  for _,eg in pairs(sample.egs) do
    local x = eg[self.at]
    if x ~= "?" then push(xys, {col=col, x=x, y=eg[sample.klass.at]}) end end
  return div(xys, tiny, trivial,  self, getmetatable(sample.klass)) end

-- SYM ----------------------------------------------------------------------
--
-- 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,at,txt,     self)
  self= has(Sym,{at=at or 0, txt=txt or "", 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,n)
  n = n or 1
  self.n = self.n + n
  self.has[x] = n + (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:merge(other)
  new=Sym()
  new.at, new,txt = self.at, self.txt
  for k,n in pairs(self.has)  do new:add(k,n) end
  for k,n in pairs(other.has) do new:add(k,n) end
  return new end

function Sym:mid() return self.mode end

-- Create one span holding  row indexes associated with each symbol
function Sym:spans(sample,...)
  local xys,yklass = {}, getmetatable(sample.klass)
  for pos,eg in pairs(sample.egs) do
    local x = eg[self.at]
    if x ~= "?" then
      xys[x] = xys[x] or yklass()
      xys[x]:add( eg[sample.klass.at] )  end end
  return map(xys, function(x,ys) return {col=self, lo=x, hi=x, has=ys} end) end

function Sym:spread()
  return sum(self.has,
             function(n1) return  -n1/self.n * math.log(n1/self.n,2) end) end

-- SKIP ---------------------------------------------------------------------
--
-- ## Stuff for skipping all things sent to a column
local Skip=obj"Skip"
function Skip.new(_,at,txt) return has(Skip,{at=at or 0, txt=txt or"", n=0}) end
function Skip:add(x) self.n = self.n + 1; return  x end
function Skip:mid() return "?" end

```

```lua
-- SAMPLE ----------------------------------------------------------------------
--
-- 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, klass=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:add(eg,      ako,what,xy)
  if not self.names
  then -- create the column headers
    self.names = eg
    for at,x in pairs(eg) do
      ako  = (x:find":"       and Skip) or
             (x:match"^[A-Z]" and Num ) or
             Sym
      what = push(self.all, ako({}, at, x))
      if not x:find":" then
        if x:find"!" then self.klass = what end
        xy = (x:find("+") or x:find("-") or x:find"!") and self.ys or self.xs
        push(xy, what) end end
  else -- store another example; update column headers
    push(self.egs, eg)
    for at,x in pairs(eg) do if x ~= "?" then self.all[at]:add(x) end 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:norm(eg1[num.at])
    b  = num:norm(eg2[num.at])
    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: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:dist(eg1,eg2,     a,b,d,n,inc)
  d,n = 0,0
  for _,col in pairs(self.xs) do
    a,b = eg1[col.at], eg2[col.at]
    inc = a=="?" and b=="?" and 1 or col:dist(a,b)
    d   = d + inc^it.P
    n   = n + 1 end
  return (d/n)^(1/it.P) end

-- Report mid of the columns
function Sample:mid(cols)
  return lap(cols or self.ys,function(col) return col:mid() end) end

-- Return spans of the column that most reduces variance
function Sample:bestSplits(tiny, trivials)
  local function column1(col,   total,xpect,spans,total,xpect)
    local function xpect1(span) return span.has.n/total * span.has:spread() end
    spans = col:spans(self, tiny,trivials[col.at])
    total = sum(spans,function(span) return span.has.n end)
    xpect = sum(spans, xpect1)
    return {xpect, spans}
  end -------------------------------
  return first(sort(lap(self.xs, column1), firsts))[2]  end

-- Split on column with best span, recurse on each split.
function Sample:tree(tiny,trivials,pre,      node,new,x)
  pre=pre or ""
  print(pre ..":"..#self.egs)
  tiny     = tiny     or (#self.egs)^it.TINY
  trivials = trivials or map(self.xs,
                         function(_,x)
                           return x.at,it.TRIVIAL*x:spread() end)
  node     = {node=self, kids={}}
  if #self.egs <= 2*tiny then print(333333);return node end
  for _,span in pairs(self:bestSplits(tiny,trivials)) do
    new = self:clone()
    for _,eg in pairs(self.egs) do
      x = eg[span.col.at]
      if x=="?" or (span.lo <= x and x <= span.hi) then new:add(eg) end end
    if #new.egs < #self.egs then
      push(node.kids, {txt = span.col.txt, txt= span.col.at,
                       lo  = span.lo,      hi = span.hi,
                       sub = new:tree(tiny,trivials,pre.."|.. ")}) end end
    --os.exit()
    --end end
  return node end

-- Find which leaf best matches an example `eg`.:w

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.at]
    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
-- DISCRIMINATION --------------------------------------------------------------
--
-- Input a list of {{x,y}..} values. Return spans that divide the `x` values
-- to minimize variance on the `y` values.
-- local div -- do not uncomment. `div` was declared local above for `Num:spans`.
local mergeable,merge

-- Return a list of `spans` {lo=,hi=,col=col}.
-- Sort the list of pairs `xys` then split it into `spans` of cardinally at
-- least `tiny`. Ensure that the max-min of each span is more that `trivial`.
function div(xys, tiny, trivial,col,yklass)
  xys    = sort(xys, function(a,b) return a.x < b.x end)
  local tenth=#xys//10
  trvial = trivial or it.TRIVIAL*math.abs(xys[9*tenth][1] - xys[tenth][1])/2.56
  tiny   = tiny    or it.TINY*#xys
  yklass = yklass  or Num
  local spans,span
  span   = {col=col,lo=xys[1].x, hi=xys[1].x, has=yklass()}
  spans = {span}
  for j,xy in pairs(xys) do
    local x, y = xy.x, xy.y
    if    j < #xys - tiny   and     -- enough items remaining after split
          x ~= xys[j+1].x   and     -- next item is different (so can split here)
          span.has.n > tiny and     -- span has enough items
          span.hi - span.lo > trivial -- span is not trivially small
    then span = push(spans, {col=col, lo=span.hi, hi=x, has=yklass()})  -- then new span
    end
    span.hi = x
    span.has:add(y) end
  first(spans).lo = -math.huge
  last(spans).hi  =  math.huge
  return merge(spans) end

function mergeable(a,b,   new,b4)
  new = a:merge(b)
  b4  = (a.n*a:spread() + b.n*b:sd()) / new.n
  if new:spread() <= b4 then return new end
end

--  Merge adjacent spans if the combo is simpler than the parts.
function merge(b4)
  local j, tmp = 0, {}
  while j < #b4 do
    j = j + 1
    local now, after = b4[j], b4[j+1]
    if after then
      local simpler = mergeable(now.has, after.has)
      if simpler then
        now = {col=col, lo=now.lo, hi= after.hi, has=simpler}
        j = j + 1 end end
    push(tmp,now) end
  return #tmp==#b4 and b4 or merge(tmp) -- recurse until nothing merged
end


```

```lua
-- HINTING ---------------------------------------------------------------------
--
-- Sorting on a few y values
local hints={}
function hints.default(eg) return eg end

function hints.sort(sample,scorefun,    test,train,egs,scored,small)
  sample = Sample.new(it.FILE)
  train,test = {}, {}
  for i,eg in pairs(shuffle(sample.egs)) do
     push(i<= it.TRAIN*#sample.egs and train or test, eg) end
  egs = copy(train)
  small = (#egs)^it.TINY
  local i=0
  scored = {}
  while #egs >= small do
    local tmp ={}
    i = i + 1
    io.stderr:write(fmt("%s",string.char(96+i)))
    for j=1,it.HINTS do
      egs[j] = (scorefun or hints.default)(egs[j])
      push(tmp, push(scored, egs[j]))
    end
    egs = hints.ranked(scored,egs,sample)
    for i=1,it.CULL*#egs//1 do pop(egs) end
  end
  io.stderr:write("\n")
  train=hints.ranked(scored, train, sample)
  return #scored, sample:clone(train), sample:clone(test) end

function hints.ranked(scored,egs,sample,worker,  some)
  function worker(eg) return {hints.rankOfClosest(scored,eg,sample),eg} end
  scored = sample:betters(scored)
  return  lap(sort(lap(egs, worker),firsts),second) end

function hints.rankOfClosest(scored,eg1,sample,        worker,closest)
  function worker(rank,eg2) return {sample:dist(eg1,eg2),rank} end
  closest = first(sort(map(scored, worker),firsts))
  return  closest[2] end --+ closest[1]/10^8 end

```

```lua
-- demos -----------------------------------------------------------------------
--
it._eg={}
it._no={}
function it._eg.shuffle(   t,u,v)
  t={}
  for i=1,32 do push(t,i) end
  u = shuffle(copy(t))
  v = shuffle(copy(t))
  assert(#t == #u and u[1] ~= v[1]) end

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

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

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

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

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

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

function it._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 it._eg.sample(    s,tmp,d1,d2,n)
  s=Sample(it.FILE)
  assert(2110 == last(s.egs)[s.all[4].at])
  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,#sort1-20,-1 do hi:add(sort1[i]) end
  shout(s:mid())
  shout(lo:mid())
  shout(hi:mid())
  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 it._eg.dists(    s,tmp,d1,d2,n)
  s=Sample(it.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 it._eg.hints(    s,_,__,evals,sort1,train,test,n)
  s = Sample(it.FILE)
  evals, train,test = hints.sort(s)
  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); end end

function it._eg.dump()
  shout(it) end
function it._eg.tree(    s,t,u,eg1,evals,ordered,rest)
  s = Sample(it.FILE)
  t = copy(s.names)
  push(t,"Rank!")
  u = Sample.new():add(t)
  evals, ordered,rest = hints.sort(s)
  for m,eg in pairs(ordered.egs) do
    eg1 = copy(eg)
    push(eg1,m)
    u:add(eg1) end
  print(1)
  u:tree() end

-- START-UP --------------------------------------------------------------------
--
it{demos=it._eg, nervous=true}

```

```
--[[
    _|_ _    _| _
     | (_)  (_|(_)


Spans
 Little languages:
   - options
   - data language

Lesson plan
- w1: ssytems: github. github workplaces. unit tests. doco tools.

- w2: num,sym
- W3: sample
- w4: eval, knn, unfariness
- W5:

-  seems to be  a revers that i  need to do .... but dont
- check if shuffle is working

teaching:
- sample is v.useful
--]]
```