```lua
local the =require"tiny0"[[
lua hint.lua [OPTIONS]

A small sample multi-objective optimizer / data miner.
(c)2021 Tim Menzies <timm@ieee.org> unlicense.org

OPTIONS:
  -best     X   Best examples are in 1..best*size(all)    = .2
  -debug    X   run one test, show stackdumps on fail     = ing
  -epsilon  X   ignore differences under epsilon*stdev    = .35
  -file     X   Where to read data                        = ../../data/auto93.csv
  -h            Show help                                 = false
  -seed     X   Random number seed;                       = 10019
  -Stop     X   Create subtrees while at least 2*stop egs =  4
  -Tiny     X   Min range size = size(egs)^tiny           = .5
  -todo     X   Pass/fail tests to run at start time      = ing
                If "X=all", then run all.
                If "X=ls" then list all. ]]
--  .  .
-- |\/|*  __ _.
-- |  ||_) (_.
local _=require"tinylib"
local say,fmt,color,out,shout= _.say, _.fmt,_.color,_.out,_.shout,_.csv -- strings
local map,copy,keys,push      = _.map,_.copy, _.keys, _.push -- tables
local sort, firsts, seconds   = _.sort, _.firsts, _.seconds  -- sorting
local norm, sum               = _.norm,  _.sum               -- maths
local randi,rand              = _.randi, _,rand              -- randoms
local same                    = _.same                       -- meta
local csv                     = _.csv -- files

local ent,mode
function ent(t,    n,e)
  n=0; for _,n1 in pairs(t) do n = n + n1 end
  e=0; for _,n1 in pairs(t) do e = e - n1/n*math.log(n1/n,2) end
  return e,n  end

function mode(t,     most,out)
  most = 0
  for x,n in pairs(t) do if n > most then most,out = n,x end end
  return out end

--  __.             .
-- (__  _.._ _ ._ | _
-- .__)(_][ | )[_)|(/,
--             |
-- [5]  Returns a sample, initialized, updated
-- [1]  Self initialize (if nil, then create).
-- [2]  Read from disc file
-- [3]  First item is special (contains names of columns)
-- [4]  Other rows are the actual examples. Use these to update column headers
-- [6]  Numeric columns have an "num[n]" entry that tracks the
--      "num[n].lo" and "num[n].hi" range for each variable.
-- [7]  Columns to be minimized or maximized are dependent (listed in "ys")
-- [8]  All other columns are the independent (listed in "xs")
-- [9]  Dependent variables are minimized,maximized at weights -1,1
--      if their name contains "-" or "+". The number of dependents ins "nys"
-- [10] Columns contain ":" are ignored
-- [11] Each example will be discretized (later) so each example holds the
--      "raw" values (not discretized) and the "cooked" examples (discretized).
local slurp,sample,ordered,clone
function slurp(out)
  for eg in csv(the.file) do out=sample(eg,out) end --[2]
  return out end

function clone(i, inits,     out)
  out = sample(i.heads)
  for _,eg in pairs(inits or {}) do out = sample(eg,out) end
  return out end

function sample(eg,i)
  local numeric,independent,dependent,head,data,datum
  function head(n,x)
    function numeric()     i.num[n]= {hi=-math.huge,lo=math.huge} end -- [6]]
    function independent() i.xs[n]= x end  -- [8]
    function dependent()                    -- [7]
      i.num[n].w  = x:find"-" and -1 or 1   -- [9]
      i.ys[n] = x
      i.nys   = i.nys+1 end
    if not x:find":" then  -- [10]
      if x:match"^[A-Z]" then numeric() end
      if x:find"-" or x:find"+" then dependent() else independent() end end --[7,8]
    return x end
  function data(eg) return {raw=eg, cooked=copy(eg)} end --[11]
  function datum(n,x) -- [4]
    if x ~= "?" then
      local num=i.num[n]
      if num then
        num.lo = math.min(num.lo,x)          -- [6]
        num.hi = math.max(num.hi,x) end end  -- [6]
    return x end
  eg = eg.raw and eg.raw or eg
  if i then push(i.egs, data(map(eg,datum))) else            -- [4]
     i = {xs={},nys=0,ys={},num={},egs={},divs={},heads={}}  -- [1] [3]
     i.heads = map(eg,head) end                              -- [3]
  return i end                                               -- [5]

-- [14] Returns the sample, examples sorted by their goals, each example
--      tagged with "eg.klass=best" or "eg.klass=rest" if "eg" is in the top
--     "the.best" in the sort.
-- [12] Sort each example by exploring all goals (dependent variables).
-- [15] The direction that losses the most points to best example.
--      e.g. a.b=.7,.6 and a-b is .1 (small loss) and b-a is -.1
--      (much smaller than a or b) so a is more important than b.
-- [13] Goal differences are amplified by raining them to a power (so normalize
--      the goals first so you that calculation does not explode.
function ordered(i)
  local function better(eg1,eg2,     a,b,s1,s2)
    s1,s2=0,0
    for n,_ in pairs(i.ys) do                        -- [12]
      local num = i.num[n]
      a  = norm(num.lo, num.hi, eg1.raw[n])          -- [13]
      b  = norm(num.lo, num.hi, eg2.raw[n])          -- [13]
      s1 = s1 - 2.71828^(num.w * (a-b)/i.nys)        -- [13] [15]
      s2 = s2 - 2.71828^(num.w * (b-a)/i.nys) end -- [13] [15]
    return s1/i.nys < s2/i.nys end                   -- [15]
  for j,eg in pairs(sort(i.egs,better)) do
    if j < the.best*#i.egs then eg.klass="best" else eg.klass="rest" end end
  return i end                                       -- [14]

```

```lua
-- .__
-- [__)*._  __
-- [__)|[ )_)
local discretize, xys_sd, bin, div
function bin(z,divs)
  if z=="?" then return "?" end
  for n,x in pairs(divs) do
    if x.lo<= z and z<= x.hi then return n end end end

function discretize(i)
  function xys_sd(col,egs,    out,p)
    out={}
    for _,eg in pairs(egs) do
      local x=eg.raw[col]
      if x~="?" then push(out, {x=x,  y=eg.klass}) end end
    out = sort(out, function(a,b) return a.x < b.x end)
    p   = function(z) return out[z*#out//10].x end
    return out, math.abs(p(.9) - p(.1))/2.56
  end -----------------------
  for col,name in pairs(i.xs) do
    if i.num[col] then
      local xys,sd = xys_sd(col, i.egs)
      i.divs[col]  = div(col,name,xys, (#xys)^the.Tiny, the.epsilon*sd)
      for _,eg in pairs(i.egs) do
        eg.cooked[col]= bin(eg.raw[col], i.divs[col]) end end end
  return i end

local function showDiv(i,at,val,      out)
  out="??"
  if i.num[at] then
    for k,div in pairs(i.divs[at]) do
      if k==val then out =fmt("%s <= %s <= %s",div.lo, i.xs[at], div.hi) end end
  else out= fmt("%s = %s", i.xs[at], val) end
  return out end

function div(col,name,xys,tiny,epsilon,     one,all,merged,merge)
  function merged(a,b,an,bn,      c)
    c={}
    for x,v in pairs(a) do c[x] = v end
    for x,v in pairs(b) do c[x] = v + (c[x] or 0) end
    if ent(c)*.99 <= (an*ent(a) + bn*ent(b))/(an+bn) then return c end
  end ------------------------
  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 = merged(now.has,after.has, now.n,after.n)
        if simpler then
          now = {col=col,name=name, lo=now.lo, hi=after.hi,
                   n=now.n+after.n, has=simpler}
          j = j + 1 end end
      push(tmp,now) end
    return #tmp==#b4 and b4 or merge(tmp) -- recurse until nothing merged
  end ------------------------
  one = {col=col,name=name,lo=xys[1].x, hi=xys[1].x, n=0, has={}}
  all = {one}
  for j,xy in pairs(xys) do
    local x,y = xy.x, xy.y
    if   j< #xys-tiny and x~= xys[j+1].x and one.n> tiny and one.hi-one.lo>epsilon
    then one = push(all, {col=col,name=name,lo=one.hi, hi=x, n=0, has={}})
    end
    one.n  = 1 + one.n
    one.hi = x
    one.has[y] = 1 + (one.has[y] or 0); end
  return merge(all) end

```

```lua
-- .___.
--   |    ._. _  _  __
--   |    [  (/,(/,_)
local splitter,worth,tree,count,keep,tree

function count(t,at)  t=t or {}; t[at]=1+(t[at] or 0); return t  end
function keep(t,at,x) t=t or {}; t[at]=t[at] or {}; push(t[at],x); return t  end

function splitter(xs, egs)
  function worth(at,_,    xy,n,x,xpect)
    xy,n = {}, 0
    for _,eg in pairs(egs) do
      x = eg.cooked[at]
      if x ~= "?" then
        n=n+1
        xy[x] = count(xy[x] or {},eg.klass) end end
    return {at, sum(xy, function(t)  local e,n1=ent(t); return n1/n* e end)} end
  return sort(map(xs,worth),seconds)[1][1] end

function tree(xs, egs,lvl)
  local here,at,splits,counts
  for _,eg in pairs(egs) do counts=count(counts,eg.klass) end
  here = {mode=mode(counts), n=#egs, kids={}}
  if #egs > the.Stop then
    splits,at = {},splitter(xs,egs)
    for _,eg in pairs(egs) do  splits=keep(splits,eg.cooked[at],eg) end
    for val,split in pairs(splits) do
      if #split < #egs and #split > the.Stop then
        push(here.kids, {at=at,val=val,
                         sub=tree(xs,split,(lvl or "").."|.. ")}) end end end
  return here end

local function show(i,tree)
  local vals=function(a,b) return a.val < b.val end
  local function show1(tree,pre)
    if #tree.kids==0 then io.write(fmt(" ==> %s [%s]",tree.mode, tree.n)) end
    for _,kid in pairs(sort(tree.kids,vals))  do
        io.write("\n"..fmt("%s%s",pre, showDiv(i, kid.at, kid.val)))
         show1(kid.sub, pre.."|.. ") end
  end -----------------------
  show1(tree,""); print("") end

```

```lua
-- .___.       ,
--   |    _  __-+- __
--   |   (/,_)  | _)
local go={}
function go.ls()
  print("\nlua "..arg[0].." -todo ACTION\n\nACTIONS:")
  for _,k in pairs(keys(go)) do  print(" -todo",k) end end
function go.the() shout(the) end
function go.bad(  s) assert(false) end
function go.ing() return true end
function go.ordered(  s,n)
  s = ordered(slurp())
  n = #s.egs
  shout(s.heads)
  for i=1,15 do shout(s.egs[i].raw) end
  print("#")
  for i=n,n-15,-1 do shout(s.egs[i].raw) end
  n={}; for _,eg in pairs(s.egs) do n=count(n,eg.klass) end
  shout(n)
end

function go.bins(    s)
  s= discretize(ordered(slurp()))
  for m,div in pairs(s.divs) do
    print("")
    for n,div1 in pairs(div) do print(m, n,out(div1)) end end
  end

function go.tree(  s,t)
  s = discretize(ordered(slurp()))
  show(s,tree(s.xs, s.egs))
end

--  __. ,         ,
-- (__ -+- _.._.-+- ___ . .._
-- .__) |  (_][   |      (_|[_)
--                         |
the(go)
```

```lua
local lib={}

--   __. ,
-- (__ -+-._.*._  _  __
-- .__) |  [   |[ ) (_]_)
--                 ._|
lib.fmt = string.format
function lib.say(...)  print(lib.fmt(...)) end
function lib.color(n,s)    return lib.fmt("\27[1m\27[%sm%s\27[0m",n,s) end
function lib.shout(x) print(lib.out(x)) end

function lib.out(t,     u,key,val)
  function key(_,k) return string.format(":%s %s", k, lib.out(t[k])) end
  function val(_,v) return lib.out(v) end
  if type(t) ~= "table" then return tostring(t) end
  u = #t>0 and lib.map(t, val) or lib.map(lib.keys(t), key)
  return "{"..table.concat(u," ").."}" end

-- .___.    .   .
--   |    _.|_  |  _  __
--   |   (_][_) | (/,_)
function lib.push(t,x)     t[ 1+#t ]=x; return x end
function lib.copy(t,  u)   u={};for k,v in pairs(t) do u[k]=v end; return u end

function lib.map(t,f,   u)
  u,f={},f or same; for k,v in pairs(t) do u[1+#u] = f(k,v) end; return u end

function lib.keys(t,u)
  u={}; for k,_ in pairs(t) do u[1+#u]=k end;return lib.sort(u);end

--   __.          ,
-- (__   _  ._.-+-*._  _
-- .__) (_) [    | |[ ) (_]
--                     ._|
function lib.sort(t,f)     table.sort(t,f); return t end
function lib.firsts(x,y)   return x[1] < y[1] end
function lib.seconds(x,y)  return x[2] < y[2] end

-- .  .     , .
-- |\/| _.-+-|_  __
-- |  |(_]  |  [ )_)
function lib.norm(lo,hi,x)
  return math.abs(lo-hi)<1E-32 and 0 or (x-lo)/(hi-lo) end

function lib.sum(t,f,   n)
  n,f=0,f or same; for _,v in pairs(t) do n = n + f(v)  end; return n end

-- .__              .
-- [__)  _.._   _|  _ ._ _
-- |  \(_][ ) (_] (_)[ | )
function lib.randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end

function lib.rand(lo,hi)
  lo, hi = lo or 0, hi or 1
  the.seed = (16807 * the.seed) % 2147483647
  return lo + (hi-lo) * the.seed / 2147483647 end

-- .  .     ,
-- |\/| _  -+- _.
-- |  |(/,  |  (_]
function lib.same(x,...)   return x end

-- .___ .
-- [__ *|  _  __
-- |   ||(/,_)
function lib.csv(file,   x)
  file = io.input(file)
  return function(   t,tmp)
    x  = io.read()
    if x then
      t={}
      for y in x:gsub("[\t ]*",""):gmatch"([^,]+)" do t[1+#t]=tonumber(y) or y end
      if #t>0 then return t end
    else io.close(file) end end end

-- .__      ,
-- [__) _ -+-. ..._.._
-- |  \(/, |  (_|[  [ )
return lib
```

```lua
-- standard load and  start functions
-- first line of code should be a help string (e.g. see tiny.lua)
-- last line  of code should call this code, pass in table of actions
-- e.g
--       the(go)

-- .__
-- [__) _  _ . . _  __
-- |  \(_) (_](_|(/,_)
--          ._|
-- at load time, remember the current globals
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
-- after start time, complain if code has created  rogue globals
local function rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?:",k,type(v)) end end end

-- .  .
-- |\/|* __ _.
-- |  ||_) (_.
-- Table keys, in sorted order
local function keys(t,u)
  u={}; for k,_ in pairs(t) do u[1+#u]=k end;  table.sort(u); return u end

-- pretty colors, n={31,32},={red,green}
local function color(n,s) return string.format("\27[1m\27[%sm%s\27[0m",n,s) end

-- shallow copy of a list
local function copy(t,  u)
     u={}; for k,v in pairs(t) do u[k]=v end ; return u end

--   __. ,        ,
-- (__ -+- _.._.-+- ___ .  ._
-- .__) | (_][    |      (_|[_)
local help = ""

-- All the start-up actions:
-- [1] keep a copy of the options as "defaults"
-- [2] maybe just show the  help text
-- [3] maybe run an  action in verbose mode (show stackdump; halt on error)
-- [4] before actions, reset options to detaults
-- [5] before actions, reset random number seed
-- [6] maybe  run an  action in fast mode (no stackdumps; no halts one errors)
-- [7] for fast mode, count the number of failures
-- [8] return to the operating system the count of failures
-- [9] lint the code (right now, we just print rogue globals)
local function what2doAtLastLine(options, actions)
  local fails, defaults = 0, copy(options)             -- [1]
  if options.h     then return print(help) end         -- [2]
  if options.debug then actions[ options.debug ]() end -- [3]
  local todos = options.todo =="all" and keys(actions) or {options.todo}
  for _,todo in pairs(todos) do
    if   type(actions[todo]) ~= "function"
    then print(color(31,"NOFUN:"),todo)
    else for k,v in pairs(defaults) do options[k]=v end -- [4]
         options.seed = options.seed or 10019           -- [5]
         local ok,msg = pcall( actions[todo] )          -- [6]
         if ok then print(color(32,"PASS ")..todo)
               else print(color(31,"FAIL ")..todo,msg)
                    fails=fails+1 end end                -- [7]
  end
  rogues()           -- [9]
  os.exit(fails) end -- [8]

-- .         .__.
-- |    *._  _   |  |._  _
-- |___|[ )(/,  |__|[ )(/,
-- In paragraph of the text that starts with "Options", all lines that start with
-- "-flag" have a default value as the last word on that line.
-- [1] Build the "options" array from those flags and defaults
-- [2] Check if we can update those defaults from command line arguments).
-- [3] Anything on the command line is a string. Check if these can become nums
-- For the sake of brevity:
-- [4] command line flags need only match the start of the flag;
-- [5] for boolean values, -flag flips the default boolean
-- [6] add in the ability to call "what2doAtLastLine"
local function what2doAtFirstLine(txt)
  local options={}
  help = txt
  txt:gsub("^.*OPTIONS:",""):gsub("\n%s*-([^%s]+)[^\n]*%s([^%s]+)",
    function(flag,x)
      for n,word in ipairs(arg) do                     -- [2]
        if flag:match("^"..word:sub(2)..".*") then  -- [4]
          x=(x=="false" and "true") or (x=="true" and "false") or arg[n+1] end end
      if     x=="true"  then x=true
      elseif x=="false" then x=false -- [4]
      else   x= tonumber(x) or x     -- [3]
      end
      options[flag] = x end)         -- [1]
  return setmetatable(options,{__call=what2doAtLastLine}) end -- [6]

-- .__     ,
-- [__) _ -+-. .._.._
-- |  \(/, | (_|[  [ )
return what2doAtFirstLine
```