```lua
--                                                  __
--                                                 /\_ \
--    _ __    __   ____     ___     ___            \//\ \    __  __     __
--   /\`'__\/'__`\/\_ ,`\  / __`\ /' _ `\            \ \ \  /\ \/\ \  /'__`\
--   \ \ \//\  __/\/_/  /_/\ \L\ \/\ \/\ \       __    \_\ \_\ \ \_\ \/\ \L\.\_
--    \ \_\\ \____\ /\____\ \____/\ \_\ \_\     /\_\   /\____\\ \____/\ \__/.\_\
--     \/_/ \/____/ \/____/\/___/  \/_/\/_/     \/_/   \/____/ \/___/  \/__/\/_/
--
-- (c)2021 Tim Menzies. Permission is hereby granted, free of charge,
-- to any person obtaining a copy of this software and associated
-- documentation files (the "Software"), to deal in the Software without
-- restriction, including without limitation the rights to use, copy,
-- modify, merge, publish, distribute, sublicense, and/or sell copies
-- of the Software, and to permit persons to whom the Software is
-- furnished to do so, subject to the following conditions:
--
-- The above copyright notice and this permission notice shall be included in all
-- copies or substantial portions of the Software.
--
-- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
-- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
-- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
-- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
-- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
-- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
-- SOFTWARE
local help = [[
lua rezon.lua [OPTIONS]
Tree learner (binary splits on numerics using Gaussian approximation)
(c)2021 Tim Menzies <timm@ieee.org> MIT license.

OPTIONS:
  -best      X   Best examples are in 1..best*size(all)    = .2
  -debug     X   run one test, show stackdumps on fail     = pass
  -epsilon   X   ignore differences under epsilon*stdev    = .35
  -Far       X   How far to look for remove items          = .9
  -file      X   Where to read data                        = ../../data/auto93.csv
  -h             Show help                                 = false
  -little    X   size of subset of a list                  = 256
  -p         X   distance calc coefficient                 = 2
  -round     X   Control for rounding numbers              = 2
  -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      = pass
                 If "X=all", then run all.
                 If "X=ls" then list all.

Data read from "-file" is a csv file whose first row contains column
names (and the other row contain data.  If a name contains ":",
that column will get ignored.  Otherwise, names starting with upper
case denote numerics (and the other columns are symbolic).  Names
containing "!" are class columns and names containing "+" or "-"
are goals to be maximized or minimized. --]] --[[

Internally,  columns names are read by a COLS object where numeric,
symbolic, and ignored columns generate NUM, SYM, and SKIP instances
(respectively).  After row1, all the other rows are examples ('EG')
which are stored in a SAMPLE. As each example is added to a sample,
they are summarized in the COLS' objects.

Note that SAMPLEs can be created from disk data, or at runtimes from
lists of examples (see SAMPLE:clone()). --]]

local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
local THE = {} -- The THE global stores the global config for this software.
-- any line of help text startling with "  -" has flag,default as first,last word
help:gsub("\n [-]([^%s]+)[^\n]*%s([^%s]+)",
  function(flag,x)
    for n,word in ipairs(arg) do -- check for any updated to "flag" on command line
      -- use any command line "word" that matches the start of "flag"
      if flag:match("^"..word:sub(2)..".*") then
        -- command line "word"s for booleans flip the default value
        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 else x=tonumber(x) or x end
    THE[flag] = x end)

THE.seed = THE.seed or 10019
if THE.h then return print(help) end
```

```lua
--                __   __
--      |\/| |   (_   /
--      |  | |   __)  \__
--
-- meta
local same
function same(x,...) return x end

-- sorting
local push,sort,ones
function push(t,x) table.insert(t,x); return x end
function sort(t,f) table.sort(t,f);   return t end
function ones(a,b) return a[1] < b[1] end

-- tables
local copy,keys,map,sum
function copy(t,   u) u={};for k,v in pairs(t) do u[k]=v            end; return u       end
function keys(t,   u) u={};for k,_ in pairs(t) do u[1+#u]=k         end; return sort(u) end
function map(t,f,  u) u={};for k,v in pairs(t) do u[1+#u] =f(k,v)   end; return u       end
function sum(t,f,  n) n=0 ;for _,v in pairs(t) do n=n+(f or same)(v) end;return n       end

-- printing utils
local hue,shout,out,say,fmt
fmt  = string.format
function say(...) print(string.format(...)) end
function hue(n,s) return string.format("\27[1m\27[%sm%s\27[0m",n,s) end
function shout(x) print(out(x)) end
function out(t,   u,key,val) -- convert nested tables to a string
  function key(_,k) return string.format(":%s %s", k, out(t[k])) end
  function val(_,v) return out(v) end
  if type(t) ~= "table" then return tostring(t) end
  u = #t>0 and map(t, val) or map(keys(t), key)
  return "{"..table.concat(u," ").."}" end

-- reading from file
local coerce,csv
function coerce(x)
  if x=="true" then return true elseif x=="false" then return false end
  return tonumber(x) or x end

function 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 push(t,coerce(y)) end
      if #t>0 then return t end
    else io.close(file) end end end

-- maths
local log,sqrt,rnd,rnds,roots
log = math.log
sqrt= math.sqrt
function rnd(x,d,  n) n=10^(d or THE.round); return math.floor(x*n+0.5) / n end
function rnds(t,d)    return map(t, function(_,x) return rnd(x,d) end) end

function roots(m1,m2,std1,std2,      a,b,c)
  if std1==std2 then return (m1+m2)/2 end
  a = 1/(2*std1^2) - 1/(2*std2^2) -- 1/(2*1^1)
  b = m2/(std2^2)  - m1/(std1^2)
  c = m1^2 /(2*std1^2) - m2^2 / (2*std2^2) - log(std2/std1)
  return ((-b - sqrt(b*b - 4*a*c))/(2*a)), ((-b + sqrt(b*b - 4*a*c))/(2*a)) end

-- random stuff (LUA's built-in randoms give different results on different platfors)
local randi,rand,any,some,shuffle
function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
function 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 any(t)       return t[randi(1,#t)] end
function some(t,n,    u)
  if n >= #t then return shuffle(copy(t)) end
  u={}; for i=1,n do push(u,any(t)) end; return u end

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

-- objects
local ako,has,obj
ako= getmetatable
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
```

```lua
--
--     |\ | /  \ |\/|
--     | \| \__/ |  |
--
local NUM=obj"NUM"
function NUM.new(inits,at,txt,     self)
  self = has(NUM,{n=0, at=at or 0, txt=txt or "",
                  w=(txt or ""):find"-" and -1 or 1,
                  mu=0, m2=0, lo=math.huge, hi=-math.huge})
  for _,x in pairs(inits or {}) do self:add(x) end
  return self end

-- summarizing
function NUM:mid()    return self.mu end
function NUM:spread() return (self.m2/(self.n-1))^0.5 end

-- updating
function NUM:add(x,  d)
  if x ~= "?" then
    self.n  = self.n  + 1
    d       = x       - self.mu
    self.mu = self.mu + d/self.n
    self.m2 = self.m2 + d*(x-self.mu)
    self.lo = math.min(x, self.lo)
    self.hi = math.max(x, self.hi) end
  return x end

-- querying
function NUM:norm(x)
  local lo,hi = self.lo,self.hi
  return math.abs(hi - lo) < 1E-9 and 0 or (x-lo)/(hi-lo) end

function NUM:dist(x,y)
  if     x=="?" then y=self:norm(y); x=y>0.5 and 0 or 1
  elseif y=="?" then x=self:norm(x); y=x>0.5 and 0 or 1
  else   x, y = self:norm(x), self:norm(y) end
  return maths.abs(x-y) end

-- discretization
function NUM:splits(other)
  local function cuts(x,s,at) return {
    {val=x,at=at,txt=fmt("%s <= %s",s,rnd(x)),when=function(z) return z<=x end},
    {val=x,at=at,txt=fmt("%s > %s" ,s,rnd(x)),when=function(z) return z >x end}}
  end
  local root1,root2 = roots(self:mid(), other:mid(), self:spread(), other:spread())
  if   self.mu<=root1 and root1<=other.mu
  then return cuts(root1,self.txt,self.at)
  else return cuts(root2,self.txt,self.at) end end
--
--      __
--     (_  \_/ |\/|
--     __)  |  |  |
--
local SYM=obj"SYM"
function SYM.new(inits,at,txt,sample,     self)
  self=  has(SYM,{n=0, at=at or 0, txt=txt or "", sample=sample,
                  seen={}, mode=nil, most=0})
  for _,x in pairs(inits or {}) do self:add(x) end
  return self end

-- Summarizing
function SYM:mid() return self.mode end
function SYM:spread()
  return sum(self.seen, function(n) return -n/self.n*log(n/self.n,2) end) end

-- update
function SYM:add(x)
  if x ~= "?" then
    self.n = 1 + self.n
    self.seen[x] = (self.seen[x] or 0) + 1
    if self.seen[x] > self.most then self.mode, self.most = x, self.seen[x] end
    return x end end

-- querying
function SYM:dist(x,y) return  x==y and 0 or 1 end

-- discretization
function SYM:splits(other)
  local function cut(_,x) return
    {val=x, at=self.at, txt=fmt("%s==%s",self.txt,x),
     when = function(z) return z==x end} end
  local out={}
  for k,_ in pairs(self.seen)  do out[k]=k end
  for k,_ in pairs(other.seen) do out[k]=k end
  return map(sort(out),cut) end

```

```lua
--      __        __
--     (_  |_/ | |__)
--     __) | \ | |
--
-- Columns for values we want to ignore.
local SKIP=obj"SKIP"
function SKIP.new(inits,at,txt)
  return has(SKIP,{n=0, at=at or 0, txt=txt or ""}) end

function SKIP:mid()     return "?" end
function SKIP:spread()  return 0   end
function SKIP:add(x)    return x   end
function SKIP:splits(_) return {}  end
--      __  __
--     |_  / _
--     |__ \__)
--
-- One example
local EG=obj"EG"

function EG.new(cells) self.cells = cells end

-- Sumamrizing
function EG:mid(cols)    return map(cols, function(_,c) return c:mid()    end) end
function EG:spread(cols) return map(cols, function(_,c) return c:spread() end) end

-- Queries
function EG:dist(other,cols,   a,b,d,n,inc)
  d,n = 0,0
  for _,col in pairs(cols) do
    a,b = self.cells[col.at], other.cells[col.at]
    inc = a=="?" and b=="?" and 1 or col:dist(a,b)
    d   = d + inc^THE.p
    n   = n + 1 end
  return (d/n)^(1/THE.p) end

-- Sorting
function EG:better(other,cols,     e,n,a,b,s1,s2)
  n,s1,s2,e = #cols, 0, 0, 2.71828
  for _,num in pairs(cols) do
    a  = num:norm(self.cells[ num.at])
    b  = num:norm(other.cells[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
--      __  __       __
--     /   /  \ |   (_
--     \__ \__/ |__ __)
--
-- Convert column headers into NUMs and SYMs, etc.
local COLS=obj"COLS"
function COLS.new(names,    self, new,what)
  self = has(COLS, {names=names, xs={}, all={}, ys={}})
  for n,x in pairs(names) do
    new = (x:find":" and SKIP or x:match"^[A-Z]" and NUM or SYM)({},n,x)
    push(self.all, new)
    if not x:find":" then
      if x:find"!" then self.klass = new
      what = (x:find"-" or x:find"+") and self.ys or self.xs
      push(what, new) end end end
  return self end

-- Updates
function COLS:add(eg)
  return map(eg, function(n,x) self.all[n]:add(x); return x end) end

```

```lua
--      __            __       __
--     (_   /\  |\/| |__) |   |_
--     __) /--\ |  | |    |__ |__
--
-- SAMPLEs hold many examples
local SAMPLE=obj"SAMPLE"
function SAMPLE.new(inits,    self)
  self = has(SAMPLE, {cols=nil, egs={}})
  if type(inits)=="string" then for eg in csv(inits)   do self:add(eg) end end
  if type(inits)=="table"  then for eg in pairs(inits) do self:add(eg) end end
  return self end

-- Create a new sample with the same structure as this one
function SAMPLE:clone(inits,   out)
  out = SAMPLE:new{self.cols.names}
  for _,eg in pairs(inits or {}) do out:add(eg) end
  return out end

-- Updates
function SAMPLE:add(eg)
  eg = eg.cells and eg.cells or eg
  if   self.cols
  then push(self.egs, eg); self.cols:add(eg)
  else self.cols = COLS(eg) end end

-- Distance queries
function SAMPLE:neighbors(eg1,egs,cols)
  local dist_eg2 = function(_,eg2) return {eg1:dist(eg2,cols or self.xs),eg2} end
  return sort(map(egs or self.egs,dist_eg2),firsts) end

function SAMPLE:distance_farExample(eg1,egs,cols,    tmp)
  tmp = self:neighbors(eg1, egs, cols)
  return table.unpack(tmp[#tmp*self.Far//1]) end

-- Discretization
function SAMPLE:twain(egs,cols)
  local egs, north, south, a,b,c, lo,hi
  egs      = some(egs or self.egs, self.little)
  _,north = self:distance_farExample(any(self.egs), egs, cols)
  c,south = self:distance_farExample(north,         egs, cols)
  for _,eg in pairs(self.egs) do
    a = eg:dist(north, cols)
    b = eg:dist(south, cols)
    eg.x = (a^2 + c^2 - b^2)/(2*c) end
  lo, ho = self:clone(), self:clone()
  for n,eg in pairs(sort(self.egs, function(a,b) return a.x < b.x end)) do
    if n < .5*#eg then lo:add(eg) else hi:add(eg) end end
  return lo, hi end

function SAMPLE:mid(cols)
  return map(cols or self.cols.all,function(_,col) return col:mid() end) end
function SAMPLE:spread(cols)
  return map(cols or self.cols.all,function(_,col) return col:spread() end) end

```

```lua
--      __            __       __ ___  __   __ __
--     (_   /\  |\/| |__) |   |_   |  |__) |_ |_
--     __) /--\ |  | |    |__ |__  |  | \  |__|__
--
-- need to sort first

-- how to score
function SAMPLE:splits(other,both,    cuts,unplaced,place,score)
  function guess(todos,cuts)
    for _,todo in pairs(todos) do
      local f=function(_,cut)
                 return {Row(cut.has:mid()):dist(todo, both.cols.xs),cut} end
      sort(map(cuts,f),firsts)[1][2].has:add(todo) end
    return cuts end
  function divide(cuts,    todos,placed)
    todos = {}
    for _,eg in pairs(both.egs) do
      placed = false
      for _,cut in pairs(cuts) do
        if   cut.what(eg.cells[cut.at])
        then cut.has = cut.has or self.clone()
             cut.has:add(eg)
             placed = true
             break end end
      if not placed then push(todos, eg) end end
    return guess(todos,cuts) end
  function score(cut,     m,n)
    m,n = #cut.has.egs,both.egs; return -m/n*log(m/n,2) end
  local best, cutsx, tmp = math.huge
  for pos,col in pairs(both.cols.xs) do
    cutsx = col:splits(other.cols.xs[pos])
    tmp   = sum(divide(cutsx),score)
    if tmp < best then best,cuts = tmp,cutsx end end
  return cuts end

function SAMPLE:tree(top)
  top = top or self
  one,two = self:twain(self.egs, top.cols.xs)
  for _,cut in pairs(one:splits(two,self)) do
    if cut.stats.n > (#top.egs)^THE.Tiny then
      cut.sub= cut.has:tree(top) end end end

function SAMPLE:show(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.pass() return true end
function go.the() shout(THE) end
function go.bad(  s) assert(false) end

function go.sort(   u,t)
  t={}; for i=100,1,-1 do push(t,i) end
  t=sort(t,function(x,y)
      if x+y<20 then return x>y else return x<y end end)
  assert(sum(t,function(x) return x*100 end)==505000)
  assert(t[1] == 10)
  assert(t[#t]==100)
  u=copy(t)
  t[1] = 99
  assert(u[1] ~= 99) end

function go.out( s)
  assert("{:age 21 :milestones {1 2 3 4} :name tim}"==out(
         {name='tim', age=21, milestones={1,2,3,4}}))end

function go.file( n)
  for _,t in pairs{{"true",true,"boolean"}, {"false",false,"boolean"},
                   {"42.1",42.1,"number"},  {"32zz","32zz","string"},
                   {"nil","nil","string"}} do
    assert(coerce(t[1])==t[2])
    assert(type(coerce(t[1]))==t[3]) end
  n =0
  for row in csv(THE.file) do
    n = n + 1
    assert(#row==8)
    assert(n==1 or type(row[1])=="number")
    assert(n==1 or type(row[8])=="number") end end

function go.rand( t,u)
  t,u={},{}; for i=1,20 do push(u,push(t,100*rand())) end
  t= sort(rnds(t,0))
  assert(t[1]==3 and t[#t]==88)
  t= sort(some(t,4))
  assert(#t==4)
  assert(t[1]==7)
  assert(79.5 == rnds(shuffle(u))[1])
end

function go.num(    cut,min)
  local z = NUM{9,2,5,4,12,7,8,11,9,3,7,4,12,5,4,10,9,6,9,4}
  assert(7 ==  z:mid(), 3.06 == rnd(z:spread(),2))
  local r1,r2 = roots(2.5, 5, 1.1, .9)
  assert(rnd(r2,2)==3.8)
  local x, y =  NUM(), NUM()
  for i=1,20 do x:add(rand(1,5)) end
  for i=1,20 do y:add(randi(20,30)) end
  for _,cut in pairs(x:splits(y)) do shout(cut) end end

function go.sym(    cut,min)
  local w = SYM{"m","m","m","m","b","b","c"}
  local z = SYM{"a","a","a","a","b","b","c"}
  assert(1.38 == rnd(z:spread(),2))
  for _,cut in pairs(w:splits(z)) do shout(cut) end
  end

function go.sample(   s)
  SAMPLE(THE.file) end

function go.kordered(  s,n)
  s = ordered(slurp())
  n = #s.egs
  shout(s.heads)
  for i=1,15 do shout(s.egs[i].cells) end
  print("#")
  for i=n,n-15,-1 do shout(s.egs[i].cells) end
end

function go.ksymcuts(  s,xpect,cuts)
  s=ordered(slurp())
  print(out(s.xs),out(s.ys))
  xpect,cuts = symcuts(7,s.egs, "origin")
  for _,cut in pairs(cuts) do print(xpect, out(cut)) end end

function go.knumcuts(  s,xpect,cuts)
  s=ordered(slurp())
  xpect,cuts = numcuts(s,2,s.egs,"Dsiplcment")
```

```lua
  if xpect then
    for _,cut in pairs(cuts) do print(xpect, out(cut)) end end end

function  go.katcuts(s,cuts,at,ynum)
  s=ordered(slurp())
  ynum=NUM(a); map(s.egs,function(_,eg) add(ynum, eg.klass) end)
  at,cuts = at_cuts(s,egs,sd(ynum)*THE.epsilon, (#s.egs)^THE.Tiny)
  for _,cut in pairs(cuts) do print(at, out(cut)) end end
```

```
--       __ ___       __ ___      __
--      (_   |   /\  |__) |  __ /  \|__)
--      __)  |  /--\ | \  |     \__/|
--
local fails, defaults = 0, copy(THE)
go[ THE.debug ]()
local todos = THE.todo == "all" and keys(go) or {THE.todo}
for _,todo in pairs(todos) do
  THE = copy(defaults)
  local ok,msg = pcall( go[todo] )
  if ok then io.write(hue(32,"PASS ")..todo.."\n")
        else io.write(hue(31,"FAIL ")..todo.." "..msg.."\n")
             fails=fails+1 end end

for k,v in pairs(_ENV) do if not b4[k] then print("?:",k,type(v)) end end
os.exit(fails)
```