```lua
#!/usr/bin/env lua
-- vim : filetype=lua ts=2 sw=2 et :
--
--     ___ ___      __  __    ____     __
--   /' __` __`\   /\ \/\ \  /',__\  /'__`\
--   /\ \/\ \/\ \  \ \ \_\ \/\__, `\/\  __/
--   \ \_\ \_\ \_\  \ \____/\/\____/\ \____\
--    \/_/\/_/\/_/   \/___/  \/___/  \/____/
--
-- (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 = [[
muse [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                  = 1024
  -more     X   Use more*#best for rest                   = 3.5
  -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
--            __   __
--     |\/| | (_  /
--     |  | | __) \__
--
-- meta
local function same(x,...) return x end
local function upto(x,y)   return x < y end
local function over(x,y)   return not(upto(x,y)) end

-- sorting
local function push(t,x) table.insert(t,x); return x end
local function sort(t,f) table.sort(t,f);   return t end
local 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 _,v in pairs(t) do u[1+#u] =f(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 fmt  = string.format
local function say(...) print(string.format(...)) end
local function btw(...) io.stderr:write(fmt(...).."\n") end
local function hue(n,s) return string.format("\27[1m\27[%sm%s\27[0m",n,s) end

local o
local function out(x) print(o(x)) end
function o(t,   u,f) -- convert nested tables to a string
  local function f(k) return fmt(":%s %s", hue(33,k), o(t[k])) end
  if type(t) ~= "table" then return tostring(t) end
  u = #t>0 and map(t, o) or map(keys(t), f)
  return hue(32,(t._is or "")).."{"..table.concat(u," ").."}" end

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

local function csv(file,   x,line)
  function line(x,  t)
    t={}; for y in x:gsub("[\t ]*",""):gmatch"([^,]+)" do push(t,coerce(y)) end
    return t end
  file = io.input(file)
  return function(   x)
    x = io.read()
    if x then return line(x) else io.close(file) end end end

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

-- random stuff (LUA's built-in randoms give different results on different platfors)
local rand
local 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

local function any(t)  return t[randi(1,#t)] end
local 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
local 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

-- objects
local function as(mt,x) return setmetatable(x,mt) end
local function is(s, obj)
  obj = {_is=s, __tostring=o}
  obj.__index = obj
  return as({__call=function(_,...) return obj.new(...) end},obj) end
```

```lua
--
--     |\ | /  \ |\/|
--     | \| \__/ |  |
--
local NUM=is"NUM"
function NUM.new(inits,at,txt,     i)
  i = as(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 i:add(x) end
  return i end

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

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

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

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

-- discretization
local _roots
function NUM.splits(i,j,tiny,       cuts,cut)
  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 n1,n2,mu1,mu2 = i.n, j.n, i.mu, other.mu
  print("mu", math.abs(mu1-mu2))
  if math.abs(mu1 - mu2) < tiny then return {} end
  cut = _roots(i:mid(), j:mid(), i.n, j.n, i:spread(), other:spread())
  out{m1=rnd(i.mu), n1=i.n, cut=rnd(cut),m2=rnd(j.mu), n2=other.n}
  return cuts(cut,i.txt,i.at) end

--      __
--     |  \ | \  /
--     |__/ |  \/
--
-- 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`.
local div={}
function div.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 div.merge(spans) end

function div.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 div.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 = div.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 div.merge(tmp) -- recurse until nothing merged
end
```

```lua
--      __
--     (_  \_/ |\/|
--     __)  |  |  |
--
local SYM=is"SYM"
function SYM.new(inits,at,txt,     i)
  i=  as(SYM,{n=0, at=at or 0, txt=txt or "",
                  seen={}, mode=nil, most=0})
  for _,x in pairs(inits or {}) do i:add(x) end
  return i end

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

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

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

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

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

function SKIP.mid(i)       return "?" end
function SKIP.spread(i)    return 0   end
function SKIP.add(i,x)     return x   end
function SKIP.splits(i,_) return {}  end

--      __  __        __
--     /   /  \ |    (_
--     \__ \__/ |__  __)
--
-- Convert column headers into NUMs and SYMs, etc.
local COLS=is"COLS"
function COLS.new(names,     i, new,what)
  i = as(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(i.all, new)
    if not x:find":" then
      if x:find"!" then i.klass = new end
      what = (x:find"-" or x:find"+") and "ys" or "xs"
      push(i[what], new) end end
  return i end

-- Updates
function COLS.add(i,eg)
  return map(i.all, function(col) col:add(eg[col.at]); return x end) end

```

```lua
--      __  __
--     |_  / _
--     |__ \__)
--
-- One example
local EG=is"EG"
function EG.new(cells) return as(EG,{cells=cells}) end

-- Sumamrizing
function EG.cols(i,all)
  return map(all,function(c) return i.cells[c.at] end) end

-- Queries
function EG.dist(i,j,cols,   a,b,d,n,inc)
  d,n = 0,0
  for _,col in pairs(cols) do
    a,b = i.cells[col.at], j.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(i,j,cols,     e,n,a,b,s1,s2)
  n,s1,s2,e = #cols, 0, 0, 2.71828
  for _,col in pairs(cols) do
    a  = col:norm(i.cells[col.at])
    b  = col:norm(j.cells[col.at])
    s1 = s1 - e^(col.w * (a-b)/n)
    s2 = s2 - e^(col.w * (b-a)/n) end
  return s1/n < s2/n end

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

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

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

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

function SAMPLE.distance_farEg(i,eg1,egs,cols,    tmp)
  tmp = i:neighbors(eg1, egs, cols)
  tmp = tmp[#tmp*THE.Far//1]
  return tmp[2], tmp[1] end

-- Unsupervised discretization
function SAMPLE.best(i)
  local rest,div = {}
  function div(egs, lvl, one,         tmp,a,b,c,two,want,low,good)
    tmp = i:clone(egs)
    say("%s%s\t%s",string.rep("|.. ",lvl),#egs,o(rnds(tmp:mid(tmp.cols.ys),1)))
    if #egs < 2*(#i.egs)^THE.epsilon then
      return i:clone(egs), i:clone(some(rest,THE.more*#egs)) end
    one   = one or i:distance_farEg(any(egs), egs, i.cols.xs)
    two,c = i:distance_farEg(one,             egs, i.cols.xs)
    for _,eg in pairs(egs) do
      a = eg:dist(one, i.cols.xs)
      b = eg:dist(two, i.cols.xs)
      eg.x = (a^2 + c^2 - b^2)/(2*c) end
    low  = one:better(two,i.cols.ys)
    good = {}
    for n,eg in pairs(sort(egs,function(a,b) return a.x < b.x end)) do
      if n < .5*#egs then push(low and good or rest, eg)
                     else push(low and rest or good, eg) end end
    return div(good, lvl+1,two) end
  return div(same(i.egs,THE.little), 0) end

```

```
function SAMPLE.mid(i,cols)
  return map(cols or i.cols.all,function(col) return col:mid() end) end

function SAMPLE.spread(i,cols)
  return map(cols or i.cols.all,function(col) return col:spread() end) end

function SAMPLE.sorted(i)
  i.egs= sort(i.egs, function(eg1,eg2) return eg1:better(eg2,i.cols.ys) end)
  return i.egs end

--     __                 __         __   ___  __   __   __
--    (_    /\   |\/|  |__)  |    |_     |   |__)  |_   |_
--    __)  /--\  |  |  |     |__  |__    |   | \   |__  |__
--
function SAMPLE:splits(other,both,    place,score)
  function place(eg,cuts,   x)
    for _,cut in pairs(cuts) do
      cut.has = cut.has or self:clone()
      x = eg.cells[cut.at]
      if x ~= "?" and cut.when(x) then   return cut.has:add(eg) end end end
  function score(cut,     m,n)
    m,n = #(cut.has.egs), #both.egs; print(m,n); return -m/n*log(m/n,2) end
  local best, cutsx, cuts, tmp = math.huge
  for pos,col in pairs(both.cols.xs) do
    print("eps", col.at, col:spread()*THE.epsilon)
    cutsx = col:splits(other.cols.xs[pos], col:spread()*THE.epsilon)
    for _,eg in pairs(both.egs) do place(eg, cutsx) end
    tmp  = sum(cutsx, score)
    if tmp < best then best,cuts = tmp,cutsx end end
  return cuts 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() out(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.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, z,r1,r2,x,y)
  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))
  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 end

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

function go.sample(   s,egs,xs,ys,scopy)
  s=SAMPLE(THE.file)
  scopy=s:clone(s.egs)
  print(s.cols.all[1]:spread(), scopy.cols.all[1]:spread())
  xs,ys= s.cols.xs, s.cols.ys
  assert(4 == #xs)
  assert(3 == #ys)
  egs=s:sorted()
  out(rnds(s:mid(ys),1));
  out(rnds(map(s:spread(ys),function(x) return .35*x end), 1)); print("")
  for i=1,10 do out(rnds(egs[i]:cols(ys),1)) end; print("")
  for i=#egs,#egs-10,-1 do out(rnds(egs[i]:cols(ys),1)) end end

function go.dist(   s,xs,sorted, show )
  s=SAMPLE(THE.file)
  xs= s.cols.xs
  sorted = s:neighbors(s.egs[1], s.egs,xs)
  show=function(i) print(rnd(sorted[i][1],2), o(sorted[i][2]:cols(xs))) end
  for i=1,10               do show(i) end; print("")
  for i=#sorted-10,#sorted do show(i) end end

function go.far(   s,xs,d,eg2)
  s  = SAMPLE(THE.file)
  xs = s.cols.xs
  for k,eg1 in pairs(shuffle(s.egs)) do
    if k > 10 then break end
    eg2,d = s:distance_farEg(eg1, s.egs, xs)
    print(rnd(d), o(eg1:cols(xs)), o(eg2:cols(xs))) end end

function go.best(  all,best,rest)
  all = SAMPLE(THE.file)
  best,rest = all:best()
  print(#best.egs, #rest.egs)
  for _,cut in pairs(best:splits(rest,all)) do print(100,cut.txt) end end
--     __ ___     __  ___         __
--    (_   |  /\ |__)  |   __ /  \|__)
--    __)  |  /--\| \   |      \__/|
--
local fails,defaults,todos,ok,msg
fails, defaults = 0, copy(THE)
go[ THE.debug ]()

todos = THE.todo == "all" and keys(go) or {THE.todo}
for _,todo in pairs(todos) do
  THE = copy(defaults)
  ok,msg = pcall( go[todo] )
  if ok then btw("%s%s",hue(32,"-- PASS "),todo)
        else btw("%s%s %s",hue(31,"-- FAIL "),todo,msg); fails=fails+1 end end

btw(hue(33,"-- %s errors"),fails)
for k,v in pairs(_ENV) do
  if not b4[k] then btw(hue(31,"-- rogue? %s %s"),k,type(v)) end end
os.exit(fails)
```