```lua
local the,help = {}, [[
lua rezon.lua [OPTIONS]

Tree learner (binary splits on numerics using Gaussian approximation)
(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
  -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
  -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 b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
----------------------------------------------------------------------------------------
local same                                                                --  _  _ _ ____ ____
function same(x,...) return x end                                         --  |\/| | [__  |
                                                                          --  |  | | ___] |___
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

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

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 shoud(x) print(out(x)) end
function out(t,   u,key,val)
  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

local coerce,csv
function coerce(x)
  if x=="true"  then return true  end
  if 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

local log,sqrt,randi,rand,rnd,rnds,any,some
log = math.log
sqrt= math.sqrt
function rnd(x,d,  n) n=10^(d or 0); return math.floor(x*n+0.5) / n end
function rnds(t,d)    return map(t, function(_,x) return rnd(x,d or 2) end) end
function and(t)       return t[randi(1,#t)] end
function some(t,n,    u)
  if n >= #t then return copy(t) end
  u={};for i=1,n do push(u,any(t)) end; return u end

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

--------------------------------------------------------------------------------
```



```lua
local Eg=obj"Eg"

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

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

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

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

--------------------------------------------------------------------------------
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

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

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

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 (x-y) end

function Num:splits(other)
  function cuts(x,s,at) return {
    {val=x, at=at, txt=fmt("%s<=$s",s,x), when=function(z) return z<=x end},
    {val=x, at=at, txt=fmt("%s >$s",s,x), when=function(z) return z >x end}}
  end
  local i, j, e, a, b, c, x1, x2 = self, other, 2.71828
  a = 1/(2*sd(i)^2) - 1/(2*sd(j)^2)
  b = j.mu/(sd(j)^2) - i.mu/(sd(i)^2)
  c = i.mu^2 /(2*sd(i)^2) - j.mu^2 / (2*sd(j)^2) - mat
  x1 = (-b - sqrt(b*b - 4*a*c) )/2*a
  x2 = (-b + sqrt(b*b - 4*a*c) )/2*a
  if  i.mu<=x1 and x1<=j.mu
  then return cuts(x1,self.txt,self.at)
  else return cuts(x2,self.txt,self.at) end end

--------------------------------------------------------------------------------
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

--------------------------------------------------------------------------------
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

```

```
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
function Sym:add(x)
  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

function Sym:dist(x,y) return  x==y and 0 or 1 end

function Sym:split(other)
  local out={}
  for k,_ in pairs(self.seen)  do push(out,k) end
  for k,_ in pairs(other.seen) do push(out,k) end
  return out end

-------------------------------------------------------------------------------
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
      what = (x:find"-" or x:find"+") and self.ys or self.xs
      push(what, new) end end
  return self end

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

-------------------------------------------------------------------------------
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

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

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

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

function Sample:twain(egs,cols)
  local egs, north, south, a,b,c, lo,hi
  egs      = nany(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.tmpx = (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.tmpx < b.tmpx 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:nearest(eg, one,two)
  eg         = eg.cells and eg or Row(eg)
  mid1, mid2 = Row(one:mid()), Row(two;mid())
  d1, d2     = eg:dist(mid1,self.xs), eg:dist(mid2,self.xs)
  return d1 < d2 and one or two end

upto = function(x,y) return y<=x end
over = function(x,y) return y>x  end
eq   = function(x,y) return x==y end

function Sample:splits(other)
  todo = {}
  for pos,col in pairs(self.cols.xs) do
    cut = col:splits(other.cols.xs[pos])
    if   isa(col) == Num
```



```
      then lo,hi = {txt=fmt("self:clone(), self:clone)
    else cuts  = map(end
    for _,eg in pairs(i.egs) do
      x = eg.cells[col.at]
      if x=="?" then push(todo, eg) else
        if isa(col) == Num

    if isa(col)==Num then

end
map(self.cols.all
local slurp,sample,ordered
function slurp(out)
  for eg in csv(the.file) do out=sample(out,eg) end --[2]
  return ordered(out) end

function sample(i,eg)
  local head,datum
  function head(n,x)
    if not x:find":" then  -- [10]
      if x:match"^[A-Z]" then i.num[n]= Num() end -- [6]]
      if x:find"-" or x:find"+"
      then i.ys[n]   = x
           i.nys     = i.nys+1
           i.num[n].w = x:find"-" and -1 or 1     -- [9]
      else i.xs[n] = x end end
    return x  end
  function datum(n,x) --- [4]
    local num=i.num[n]
    if num and x ~= "?" then add(num,x) end
    return x end
  ---------------
  if  i
  then push(i.egs, {cells = map(eg,datum)})           -- [4]
  else 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.cells[n])      -- [13]
      b  = norm(num.lo, num.hi, eg2.cells[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 eg.klass=j end
  return i end                                 -- [14]
```

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

-- utility to take a list of {{x,y},..} pairs to return a cut on
-- x that most minimizes expected value of variance of y
local minXpect,upto,over,eq,symcuts,numcuts,at_cuts
function minXpect(xy,ynum,eps,tiny,    x,y,xlo,xhi,cut,min,left,right,xpect)
  xy  = sort(xy, ones)
  min, xlo, xhi = sd(ynum), xy[1][1], xy[#xy][1]
  if xhi - xlo > 2*tiny then
    left, right = Num(), copy(ynum)
    for k,z in  pairs(xy) do
      x,y = z[1], z[2]
      sub(right,add(left,y))
      if k>=tiny and k<=#xy-tiny and x~=xy[k+1][1] and x-xlo>=eps and xhi-x>=eps
      then xpect = left.n/(#xy)*sd(left) + right.n/(#xy)*sd(right)
           if min-xpect > 0.01 then cut,min = x,xpect end end end end
  return cut,min end

function numcuts(i,at,egs,txt,epsilon,tiny)
  local xy,x,xpect,ynum,cut
  xy, ynum = {}, Num()
  for _,eg in pairs(egs) do
    x = eg.cells[at]
    if x ~= "?" then
      add(ynum, x)
      push(xy, {x, eg.klass}) end end
  cut,xpect = minXpect(xy,ynum, epsilon,tiny)
  if cut then return xpect, {
                {txt=fmt("%s<=%s",txt,cut),at=at,op=upto,val=cut},
                {txt=fmt("%s>%s",txt,cut), at=at,op=over,val=cut}} end end

function symcuts(at,egs,txt)
  local xy,x,xpect,n
  xy,n = {},0,0
  for _,eg in pairs(egs) do
    x=eg.cells[at]
    if  x ~= "?" then
      n = n + 1
      xy[x] = xy[x] or Num()
      add(xy[x], eg.klass) end  end
  if #(keys(xy)) > 1 then
    xpect = sum(xy, function(num) return num.n/n*sd(num) end)
    return xpect,map(keys(xy), function(x) return
                  {txt=fmt("%s=%s",txt,x),at=at,op=eq,val=x} end) end end

function at_cuts(i,egs,epsilon,tiny)
  local min,at, cuts, cuts0, xpect
  min = 1E9
  for at0,txt in pairs(i.xs) do
    if i.num[at0]
    then xpect,cuts0 = numcuts(i,at0,egs,txt,epsilon,tiny)
    else xpect,cuts0 = symcuts(at,egs,txt) end
    if xpect and xpect < min then at,min,cuts = at0,xpect,cuts0 end end
  return at, cuts end



local function tree(i)
  local here,at,splits,counts
  eps = sd(ynum)*the.epsilon
  tiny= (#s.egs)^the.Tinyx)
  lvl=lvl or ""
  return tree1(i, eps,epsilon,tiny,lvl) end

  epsilon=epsilon
  for _,eg in pairs(egs) do counts=count(counts,eg.klass) end
  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
```



```
-- .___.      ,
-- |  _ __-+- __
-- |  (/,_) | _)
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].cells) end
  print("#")
  for i=n,n-15,-1 do shout(s.egs[i].cells) end
end

function go.num(    cut,min)
  local xy, xnum, ynum = {}, Num(), Num()
  for i=1,400   do push(xy, {add(xnum,i), add(ynum, rand()^3  )}) end
  for i=401,500 do push(xy, {add(xnum,i), add(ynum, rand()^.25)})  end
  cut,min= minXpect(xy, ynum, .35*sd(xnum), (#xy)^the.Tiny)
  shout{cut=cut, min=min} end

function go.symcuts(  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.numcuts(  s,xpect,cuts)
  s=ordered(slurp())
  xpect,cuts = numcuts(s,2,s.egs,"Dsiplcment")
  if xpect then
    for _,cut in pairs(cuts) do print(xpect, out(cut)) end end end

function  go.atcuts(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
-- __. ,     ,
-- (__ -+- _.._.-+- ___ . .._
-- .__) | (_)[  |    (_|[_)
--                |
help: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
    the[flag] = coerce(x) end)        -- [1]

if the.h     then return print(help) end         -- [2]
if the.debug then go[the.debug]() end            -- [3]

local fails, defaults = 0, copy(the)             -- [1]
for _,todo in pairs(the.todo == "all" and keys(go) or {the.todo}) do
  the = copy(defaults)
  the.seed = the.seed or 10019                   -- [5]
  local ok,msg = pcall( go[todo] )               -- [6]
  if ok then print(hue(32,"PASS ")..todo)
        else print(hue(31,"FAIL ")..todo,msg)
             fails=fails+1 end end -- [7]

for k,v in pairs(_ENV) do if not b4[k] then print("?:",k,type(v)) end end
os.exit(fails) -- [8]
```