```lua
local the,help = {}, [[
lua 2tree.lua [OPTIONS]

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

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

local copy,keys,map,sum
copy=function(t,    u) u={};for k,v in pairs(t) do u[k]=v          end; return u       end
keys=function(t,    u) u={};for k,_ in pairs(t) do u[1+#u]=k       end; return sort(u) end
map =function(t,f,  u) u={};for k,v in pairs(t) do u[1+#u] =f(k,v) end; return u       end
sum =function(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
hue  = function(n,s) return string.format("\27[1m\27[%sm%s\27[0m",n,s) end
shout= function(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 Num,sd,sub,add
Num= function(i) return {n=0, mu=0, m2=0, lo=math.huge, hi= -math.huge} end
sd = function(i) return i.n<2 and 0 or (i.m2/(i.n-1))^0.5 end

function sub(i,x,  d) i.n=i.n-1; d=x-i.mu; i.mu=i.mu-d/i.n; i.m2=i.m2-d*(x-i.mu) end
function add(i,x,  d) 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

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

--  __.           .
-- (__  _.._ _ ._ | _
-- .__) (_)[ | ) |_) |(/,
--                |
-- [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
function slurp(out)
  for eg in csv(the.file) do out=sample(eg,out) end --[2]
  return ordered(out) end

function sample(eg,i)
  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 end -- [9]
      else i.xs[n] = x 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]
```



```lua
-- .___.
--    |  ._. _  _  __
--    |  [  (/,(/,_)
-- 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,xeps,tiny,    xy,xlo,xhi,min,left,right,x,y,xpect)
  xy  = sort(xy,firsts)
  xlo = xy[  1][1]
  xhi = xy[#xy][1]
  min = sd(ynum)
  if ynum.hi - ynum.lo > 2*tiny then
    left, right = Num(), copy(ynum)
    for k,z in  pairs(xy) do
      x,y = z[1], z[2]
      add(left, y)
      sub(right,y)
      if k>=tiny and k<=#xy-tiny and x~=xy[k+1][1] and x-xlo>=xeps and xhi-x>=xeps
      then xpect = left.n/#xy*sd(left) + right.n/#xy*sd(right)
           if tmp < xpect then cut,min = x,xpect end end end end
  return cut,min 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

-- Divide a column of symbols into one row per symbol. Return the
-- cuts and expecte
function symcuts(at,egs,txt,    xy,n,x)
  function cuts(     xpect,cuts,size)
    size  = 0
    xpect = sum(xy, function(num) size=size+1; return num.n/n*sd(num) end)
    if size > 1 then
      return xpect,map(keys(xy),function(x)
                       return {txt=fmt("%s=%s",txt,x),at=at,op=eq,val=x} end) end end
  -----------
  xy,n = {},0
  for _,eg in pairs(egs) do
    local x=eg.cell[at]
    if  x ~= "?" then
      n = n + 1
      xy[x] = xy[x] or Num()
      add(xy[x], eg.klass) end  end
  return cuts() end

function numcuts(i,at,txt,     argmin,cuts)
  -------------------
  function cuts(xy,ynum,    xepsilon, tiny)
    xespilon  = sd(i.num[at])*the.epsilon
    tiny      = (#i.egs)*the.Tiny
    cut,xpect  = minXpect(xy,ynum,xepsilon, 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
  --------------------
  local xy, ynum = {}, Num()
  for _,eg in pairs(egs) do
    local x = eg.cell[at]
    if x ~= "?" then
      add(ynum, x)
      push(xy, {x, eg.klass}) end end
  return cuts(xy,ynum) end

function at_cuts(i)
  local at,cuts
  for at,txt in pairs(i.xs) do
    if i.num[at]
    then xpect,cuts0 = nums(i,at,txt)
    else xpect,cuts0 = syms(i,at,txt) end
    if xpect and xpect < min then out,min,cuts = at,xpect,cuts0 end end
  return at, cuts end

local 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].cells) end
  print("#")
  for i=n,n-15,-1 do shout(s.egs[i].cells) end
end

function go.num(    num)
  num=Num()
  for i=1,1000 do add(num,rand()^2) end
  print(sd(num), out(num)) 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]
```