```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 push,sort
push= function(t,x) table.insert(t,x); return x end
sort= function(t)   table.sort(t);     return t end

local copy,keys,map
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

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 has,obj
has = function(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

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


--  __.      .
-- (__  _ ._ _ ._ | _
-- .__) (_) [ | ) |_) | (/,
--                |
-- [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]= {hi=-math.huge,lo=math.huge} 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]
    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
  --------------
  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.rank=j end
  return i end                                   -- [14]
```



```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 push(xy,{x,y}) 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

local Num=obj"Num"
function Num.new() return has(Num,{n=0,mu=0,m2=0}) end

function Num:add (x,    d)
  self.n  = self.n + 1
  d       = x - self.mu
  self.mu = self.mu + d / self.n
  self.m2 = self.m2 + d * (x - self.mu) end

function Num:sd()
  return self.n < 2 and 0 or self.m2 <0 and 0 or (self.m2/(self.n - 1))^.5 end

function Num:sub (x,     d)
  self.n  = self.n - 1
  d       = x - self.mu
  self.mu = self.mu - d / self.n
  self.m2 = self.m2 - d * (x - self.mu) end

function split(xy, epsilon, tiny)
   local min,xy,cut = math.huge, sort(xy,firsts)
   local yright,yleft = Num(),  Num()
   for _,v in pairs(xy) do yright:add(v[2]) end
   xy = sort(xy,firsts)
   xhi, xlo= xy[#xy][1], xy[1][1]
   for k,v in pairs(xy) do
     x,y = v[1],v[2]
     yleft:add(y);
     yright:sub(y)
     if k > tiny and k < #xy-tiny and x ~= v[k+1][1] and
        x-xlo > epsilon and xhi-x>epsilon
     then xpect = (yleft.n*yleft:sd()+ yright.n*yright:sd())/#xy
          if xpect < min then
            cut, min = k,xpect end end end
   return  cut,min 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

--  __. ,        ,
-- (__  -+- _.._.-+- ___ . ..._
-- .__) | (_)[    |       (_|[_)
--                           |
help:gsub("^.*OPTIONS:","\n"):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]
```