```
-- If you understand "it", can you write "it" shorter?  Lets try.
-- B.G. how short can we write a multi-objective semi-supervised learner?<hr>
-- <span id="forkongithub"><a href="https://github.com/timm/l5">Fork me on GitHub</a><
/span>
--
-- _Mother Teresa:_  The more you have, the more you are occupied.
-- The less you have, the more free you are.<p>
-- _Ken Thompson:_  One of my most productive days was throwing
-- away 1,000 lines of code.<p>
-- _William of Occam:_  It is vain to do with more what can be done with less.  <p>
-- <img width=200 align=left src=cup.png>
-- _Leonardo da Vinci:_  Simplicity is the ultimate sophistication.<p>
-- _Edsger D. Dijkstra:_  Simplicity is prerequisite for reliability.<p>
-- _Dieter Ram:_  Less, but better.<p>
-- _timm:_  less, plz <p>
-- My heroes: [Jack Diederich](https://www.youtube.com/watch?v=o9pEzgHorH0)
-- | [Hilary Mason](https://boingboing.net/2017/06/30/next-level-regexp.html)<p>
local help= [[
shorter.lua : a multi-objective semi-supervised learner.
(c)2022 Tim Menzies, timm@ieee.org

OPTIONS:
  -b  --Bins    max number of bins         = 16
  -F  --Few     only keep a "Few" numbers = 256
  -k  --k       handle rare classes        =  1
  -m  --m       handle rare attributes     =  2
  -p  --p       distance coefficient       =  2
  -S  --Small   small leaf size            = .5
  -w  --wait    wait before classifying    =

OPTIONS (other):
  -f  --file    file           = ../../data/auto93.csv
  -g  --go      start-up goal  = nothing
  -h  --help    show help      = false
  -s  --seed    seed           = 10019]]
-------------------------------------------------------------------------
-- ## Names
local _ = require"lib"
local argmax,big                  = _.argmax, _.big
local cli,csv,demos,klass,normpdf = _.cli,    _.csv,  _.demos,_.klass, _.normpdf
local oo,push,read,rnd,same,str   = _.oo,     _.push, _.read, _.rnd,_same,_.str

-- `THE` settings is parsed from `help`
-- string lines that contain two dashes "`--`".
local THE={}
help:gsub("[-][-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x) THE[key] = read(x) end)
-- ### Our classes

-- - ROWS use COLS to make either NUMs or SYMs.
-- - ROWS holds data in ROWs, and summarizes columns in NUMs and SYMs.
-- - NUMs use SOMEs to store at most `THE.Few` samples per numeric columns.
-- - RANGE objects track what `y` values are seen between `xlo` and `xhi`.
local ROWS, COLS, NUM, SYM = klass"ROWS",  klass"COLS",  klass"NUM", klass"SYM"
local ROW = klass"ROW"
local SOME = klass"SOME"
local RANGE = klass"RANGE"
-------------------------------------------------------------------------
-- ## class COLS

-- The system uses COLS to make lists of NUMs or SYMs.
-- Independent and dependent columns are stored in `xs` or `ys` lists.<p>
-- This code reads data from csv files or lists of tables.
-- The first row of data is a list of column names.
-- Those can contain some special symbols.
local  is={}
-- For example, we want to ignore columns whose name ends with "`:`".
-- Number columns start with an upper case letter.
-- Dependent variables end in "`!`", "`+`", or "`-`"
-- for klasses or goals to be minimized or maximized.
function is.use(x)     return not x:find"$" end
function is.num(x)     return x:find"^[A-Z]" end
function is.goal(x)    return x:find"[!+-]$" end
function is.klass(x)   return x:find"!$"     end
function is.dislike(x) return x:find"-$"     end

-- For example, `COLS` would read linel of this data, skipping column #2, making
-- NUMerics out of columns #1,#2, and a SYMbolic out of columni #4. Further,
-- any goal columns (in this case, "`Lbs-`")  get stored in a `ys` list
-- and all other columns are stored in `xs`.
--
--      Clndrs,  Hp:,  Lbs-,  origin
--      8,       193,  4732,  1
--      8,       215,  4615,  1
--      8,       200,  4376,  1
--      ...

-- __COLS(  `t`  :[string] )__<br>constructor.
function COLS.new(i,t,      new,is)
  i.xs, i.ys, i.names = {},{},{},t
  for at,txt in pairs(t) do
    new = (is.nump(txt) and NUM or SYM)(at,txt)
    new.usep,  new.w = is.use(txt), is.dislike(txt) and -1 or 1
    if new.usep  then
      if is.klass(new.txt) then i.klass=new end
      push(is.goal(new.txt) and i.ys or i.xs, new) end end end

-- __.add(  `t`  :ROW  )  :S__<br>update column summaries.
function COLS.add(i,t)
  for _,cols in pairs{i.xs,i.ys} do
    for _,col in pairs(cols) do col:add(t.cells[col.at]) end end
  return t end
-------------------------------------------------------------------------
-- ## class SOME
-- NUMs use SOMEs to store at most `THE.Few` samples per numeric columns.

-- __SOME()__<br>constructor.
function SOME.new(i) i.n,i.t,i.ok=0,{},true end
-- __.add(  x:any  )__<br>
-- Technically, this is a reservoir sampler; i.e. given a small cache,
-- sample at an equal (but low) probability across  a much larger  space.
-- Note one trick: If we full, make space by replacing anything at all.
function SOME.add(i,x)
  if x~="?" then return x end
  i.n=i.n+1
  if     #i.t    < THE.some     then i.ok=false; push(i.t,x)
  elseif rand() < THE.some/i.n then i.ok=false; i.t[rand(#i.t)]=x end end
-- __.all()  :table__<br>If we ask for `.all()`, then make sure they are sorted.
function SOME.all(i) i.t=i.ok and i.t or sort(i.t); i.ok=true; return i.t end
-------------------------------------------------------------------------
-- ## class NUM
-- Summarize a stream of numbers, maintaining its `lo`, `hi`, `mu` (mean).

-- __NUM()__<br>constructor.
function NUM.new(i) i.n,i.mu,i.m2,i.w,i.lo,i.hi,i.some=0,0,0,1,big,-big,SOME() end
--  __.add(  `v`  :number  )__<br>add `v` to the summary, updating `mu,sd,lo,hi`.
function NUM.add(i, v)
  if v~="?" then return v end
  i.some:add(v)
  i.n  = i.n + 1
  local d    = v - i.mu
  i.mu = i.mu + d/i.n
  i.m2 = i.m2 + d*(v - i.mu)
  i.sd = i.n<2 and 0 or (i.m2/(i.n-1))^0.5
  i.lo = math.min(v, i.lo)
  i.hi = math.max(v, i.hi) end
--  __.bin(  `x`  :number  )  :number__<br>return x, rounded into `THE.bins` values.
function NUM.bin(x)
  b=(i.hi - i.lo)/THE.bins; return i.lo==i.hi and 1 or math.floor(x/b+.5)*b end
--  __.like(  `x`  :number  )  :number__<br>return likelihood.
function NUM.like(i,x,...) return normpdf(x, i.mu, i.sd) end
--  __.merge(  `j`  :NUM  )  :NUM__<br>Return a NUM that combines self and `j`.
function NUM.merge(i,j,     k)
  local k = NUM(i.at, i.txt)
  for _,n in pairs(i.some.t) do k:add(x) end
  for _,n in pairs(j.some.t) do k:add(x) end
  return k end
--  __.mid(  ?`p`=2  )  :number__<br>report rounded middle.
function NUM.mid(i,p) return rnd(i.mu,p) end
-------------------------------------------------------------------------
-- ## class SYM

-- Summarize a stream of symbols, maintaining the `mode` and frequencies counts
-- of symbols seen so far.

-- __SYM()__<br>constructor.
function SYM.new(i)           i.n,i.syms,i.most,i.mode = 0,{},0,nil end
--  __.add(  `v`  :any,  ?`inc`=1  )__<br>add `v`, `inc` number of times.
function SYM.add(i,v,inc)
  if v~="?" then return v end
  inc=inc or 1
  i.n = i.n + inc
  i.syms[v] = inc + (i.syms[v] or 0)
  if i.syms[v] > i.most then i.most,i.mode = i.syms[v],v end end
--  __.bin(  `x`  :any  )  :any__<br>discrediting a symbol just returns that symbol.
function SYM.bin(x) return x end
--  __.like(  `x`  :any  )  :number__<br>report likelihood.
function SYM.like(i,x,prior) return ((i.syms[x] or 0)+THE.m*prior)/(i.n+THE.m) end
--  __.merge(  `j`  :SYM  )  :SYM__<br>Return a SYM that combines self and `j`.
function SYM.merge(i,j,     k)
  local k = SYM(i.at, i.txt)
  for x,n in pairs(i.has) do k:add(x,n) end
  for x,n in pairs(j.has) do k:add(x,n) end
  return k end
--  __.mid()  :any__<br>return the  most commonly seen symbol.
function SYM.mid(i,...)       return i.mode end
-------------------------------------------------------------------------
-- ## class ROW
-- Store data from one record. Track if we ever use this ROW's dependent variables.
-- Hold a pointer back to a ROWS (so we can access attribute positions and
-- weightings).

-- __ROW()__<br>constructor.
function ROW.new(i,of,t) i.of,i.cells,i.evaled  = of,t,false end
--  __.klass()__<br>Return the klass variable of this ROW.
function ROW.klass(i)    return i.cells[i.of.cols.klass.at] end
--  __.within(  `range`  :RANGE  )  :boolean__<br>True if ROW is in `range`.
function ROW.within(i,range)
   local lo, hi, at = range.xlo, range.xhi, range.ys.at
   local y = i.cells[at]
   return  v==y or (lo==hi and v==lo) or (lo<v and v<=hi) end
-------------------------------------------------------------------------
-- ## class ROWS

-- Store some ROWs. Keep a summary of the columns inside `cols`. Keep the
-- dependent and independent summaries seepage in `cols.ys` and `cols.xs`.

-- __ROWS(  `t`  :[string]  )__<br>constructor.
function ROWS.new(i,t) i.cols=COLS(t); i.rows={} end
--  __.add(  `t`  :(table|ROW)  )  :ROW__<br>update with a table or ROW.
function ROWS.add(i,t) return push(i.rows, i.cols:add(t.cells and t or ROW(i,t))) end
--  __.mid(  `cols`  :[NUM|SYM],  `p`=2  )  :table__
-- returns `mid`s of some columns; round numerics to `p` decimal places.
function ROWS.mid(i, cols, p,     t)
  t={};for _,col in pairs(cols or i.cols.ys) do t[col.txt]=col:mid(p) end;return t end
--  __.clone(  ?`data`  :(table|[ROW])  )  :ROWS__<br>copy this structure, maybe add da
ta.
function ROWS.clone(i,data,  j)
  j= ROWS(i.cols.names);for _,row in pairs(data or {}) do j:add(row) end; return j end
--  __.like(  `row`  :ROWS,  `nklasses`  :int;  `nrows`  :int  )  :number__
-- how likely is it that `row` could live here?
function ROWS.like(i,row, nklasses, nrows,    prior,like,inc,x)
  prior = (#i.rows + THE.k) / (nrows + THE.k * nklasses)
  like  = math.log(prior)
  for _,col in pairs(i.cols.xs) do
    x = row.cells[col.at]
    if x and x ~= "?" then
      inc  = col:like(x,prior)
      like = like + math.log(inc) end end
  return like end

-- __doRows(  ?`src`  :(string|table),  `fun`  :function( table|ROW )   )__
-- helper function for reading from tables or files. Case argl of ...
-- ... _table_  : call  function for all items in table.
-- ... _string_ :  call function on all rows from a file.
-- ... _nil_    :  call function of all rows from standard input.
local function doRows(src, fun)
  if type(src)=="table" then for  _,t in pairs(src) do fun(t) end
                        else for   t in csv(src)   do fun(t) end end end

-------------------------------------------------------------------------
-- ## class NB
-- (0) Use row1 to initial our `overall` knowledge of all rows.
-- After that (1) add row to `overall` and (2) ROWS about this row's klass.
-- (3) After `wait` rows, classify row BEFORE updating training knowledge
function NB.new(i,src,report,             row)
  report = report or print
  i.overall, i.dict, i.list  = nil, {}, {}
  doRows(src,  function(row,   k)
    if not i.overall then i.overall = ROWS(row) else  -- (0) eat row1
      row = i.overall:add(row)                        -- add to overall
      if #i.overall.rows > THE.wait then report(row:klass(), i:guess(row)) end
      i:train(row) end end)                           -- add tp rows's klass

function NB.train(i,row,      k)
  k=row:klass()
  i.dict[k] = i.dict[k] or push(i.list,i.overall:clone()) -- klass is known
  i.dict[k].txt = k                                       -- each klass knows its name
  i.dict[k]:add(row) end                                  -- update klass with row

function NB.guess(i,row)
    return argmax(i.dict,
      function(klass) return klass:like(row,#i.list,#i.overall.rows) end) end
-- function TREE.new(i,listOfRows,gaurd)
--   i.gaurd, i.kids = gaurd, {}
--   of   = listOfRows[1][1].of
--   best = sort(map(of.cols.x,
--                   function(col) i:bins(col,listOfRows) end),lt"div")[1]
--   i.kids = map(best.ranges, function(range)
--              listOfRows1 = {}
--    -- local function within(row)       return row:within(best) end
--    -- local function withins(rows)     return map(rows, within) end
--    -- map(listOrRanges, function(rows) return withins(rows) end) end
--      --   tmp= map(rows,withins)
--      --   if #tmp > stop then
--      --    end)
--      --
-------------------------------------------------------------------------
-- ## class TREE
-- function decisionTree(listOfRows)
-- -- function tree(rows, xols, yklass,y, gaurd)
--    local function xranges(xcol) return i:ranges(rows,xcol,yklass,y) end
--    i.gaurd = gaurd
--    ranges = sort(map(xcols, xranges),lt"div")[1].ranges
--    for _,row in pairs(rows) do
--      for _,range in pairs(ranges) do
--        if row:within(range) then push(range.rows,row) end; break end end
--    i.kids = map(ranges,
--                 function(range) return TREE(range.rows,xcols,yklass,y,range) end)
--  end
-- --
--   labels , all, xcols = {},{}
--   for label,rows in pairs(listofRows) do
--     for _,row in pairs(rows) do
--       xcols = row.of.cols.xs
--       labels[ push(all,row).id ] = label end end
--   return TREE(all, xcols, SYM, function(row) return labels[row.id] end) end
--
local _ranges, _merge
function _ranges(i,rows,xcol,yklass,y)
  local n,list, dict = 0,{}, {}
  for _,row in pairs(rows) do
    local v = row.cells[xcol.at]
    if v ~= "?" then
      n = n + 1
      local pos = xcol:bin(v)
      dict[pos] = dict[pos] or push(list, RANGE(v,v, yklass(xcol.at, xcol.txt)))
      dict[pos]:add(v, y(row)) end end
  list = sort(list, lt"xlo")
  list = xcol.is=="NUM" and _merge(list, n^THE.min) or list
  return {ranges  = list,
          div   = sum(list,function(z) return z.ys:div()*z.ys.n/n end)} end

function _merge(b4,min)
  local j,t a,b,c,ay,by,cy = 1,{}
  while j <= #b4 do
    a, b = b4[j], b4[j+1]
    if b then
      ay,by,cy = a.ys, b.ys, a.ys:merge(b.ys)
      if ay.n<min or by.n<min or cy:div() <= (ay.n*ay:div()+by.n*by:div())/cy.n
      then a = raNGE(a.xlo, b.xhi, cy)
           j = j + 1 end end -- skip one, since it has just been merged
    t[#t+1] = a
    j = j + 1 end
  if #t < #b4 then return _merge(t,min) end
  for j=2,#t do t[j].xlo = t[j-1].xhi end
  t[1].xlo, t[#t].xhi = -big, big
  return t end
-------------------------------------------------------------------------
-- ## class RANGE
function RANGE.new(i, xlo, xhi, ys) i.xlo,i.xhi,i.ys,i.rows = xlo,xhi,ys,{} end
function RANGE.add(i,x,y)
  if x < i.xlo then i.xlo = x end -- works for string or num
  if x > i.xhi then i.xhi = x end -- works for string or num
  i.ys:add(y) end

function RANGE.__tostring(i)
  local x, lo, hi = i.ys.txt, i.xlo, i.xhi
  if     lo ==  hi  then return fmt("%s == %s",x, lo)
  elseif hi ==  big then return fmt("%s > %s",x, lo)
  elseif lo == -big then return fmt("%s <= %s", x, hi)
  else                   return fmt("%s < %s <= %s",lo,x,hi) end end
```

```lua
-------------------------------------------------------------------------------
-- ## Tests
local no,go = {},{}
function go.the()  print(l); print(THE); return type(THE.p)=="number" and THE.p==2 end

function go.argmax(  t,fun)
  fun=function(x) return -x end
  t={50,40,0,40,50}
  return 3 == argmax(t,fun) end

function go.num(n) n=NUM(); for x=1,100 do n:add(x) end; return n.mu==50.5 end

function go.sym(s)
  s=SYM(); for _,x in pairs{"a","a","a","a","b","b","c"} do s:add(x) end
  return s.mode=="a" end

function go.csv(    n,s)
  n,s=0,0; for row in csv(THE.file)  do n=n+1; if n>1 then s=s+row[1] end end
  return rnd(s/n,3) == 5.441  end

function go.rows(    rows)
  doRows(THE.file,function(t) if rows then rows:add(t)  else rows=ROWS(t) end end)
  return rnd(rows.cols.ys[1].sd,0)==847 end

function go.nb()
  return 268 == #NB("../../data/diabetes.csv").dict["positive"].rows  end

local function _classify(file)
  local Abcd=require"abcd"
  local abcd=Abcd()
  NB(file, function(got,want) abcd:add(got,want) end)
  abcd:pretty(abcd:report())
  return true end

function go.soybean() return _classify("../../data/soybean.csv") end
function go.diabetes() return _classify("../../data/diabetes.csv") end
-------------------------------------------------------------------------------
-- ## Start
if    pcall(debug.getlocal, 4, 1)
then  return {ROW=ROW, ROWS=ROWS, NUM=NUM, SYM=SYM, THE=THE,lib=lib}
else  THE = cli(THE,help)
      demos(THE,go) end
```