```lua
local help= [[
NB:
(c)2022 Tim Menzies, timm@ieee.org

OPTIONS:
  -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 =  5

OPTIONS (other):
  -f  --file   file           = ../../data/auto93.csv
  -g  --go     start-up goal  = nothing
  -h  --help   show help      = false
  -s  --seed   seed           = 10019]]
--        ._    _.   ._ _     _    _   ------------
--        | |  (_|   | | |   (/_  _>

local lib = require"lib"
local argmax,big                = lib.argmax, lib.big
local cli,csv,demos,is,normpdf  = lib.cli, lib.csv, lib.demos, lib.is, lib.normpdf
local oo,push,read,rnd          = lib.oo, lib.push, lib.read, lib.rnd
local same,str                  = lib.same,lib.str

local THE={}
help:gsub(" [-][-]([^%s]+)[^\n]*%s([^%s]+)",function(key,x) THE[key] = read(x) end)

local NB,NUM,SYM,COLS,ROW,ROWS= is"NB",is"NUM",is"SYM",is"COLS",is"ROW",is"ROWS"
local RANGE, TREE = is"RANGE", is"TREE"
--        _    _    |         ._ _    ._   ---------
--       (_   (_)   |   |_|   | | |   | |

function NUM.new(i)        i.n,i.mu,i.m2,i.mu,i.lo,i.hi = 0,0,0,0,big,-big end
function NUM.mid(i,p)      return rnd(i.mu,p) end
function NUM.like(i,x,...) return normpdf(x, i.mu, i.sd) end
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

function NUM.add(i,v,   d)
  if v=="?" then return v end
  i.n  = i.n + 1
  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(x, i.lo)
  i.hi = math.max(x, i.hi) end

function SYM.new(i)          i.n,i.syms,i.most,i.mode = 0,{},0,nil end
function SYM.mid(i,...)      return i.mode end
function SYM.like(i,x,prior) return ((i.syms[x] or 0)+THE.m*prior)/(i.n+THE.m) end
function SYM.bin(x)          return x end

function SYM.sub(i,v) return i:add(v, -1) end
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

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
--        _    _    |    _   --------------------
--       (_   (_)   |   _>
--
local function usep(x)   return not x:find":$" end
local function nump(x)   return x:find"^[A-Z]" end
local function goalp(x)  return x:find"[!+-]$" end
local function klassp(x) return x:find"!$"     end
local function new(at,txt,          i)
  txt = txt or ""
  i = (nump(txt) and NUM or SYM)()
  i.txt, i.usep, i.at, i.w = txt, usep(txt), at or 0, txt:find"-$" and -1 or 1
  return i  end

function COLS.new(i,t,     col)
  i.all, i.xs, i.ys, i.names = {},{},{},t
  for at,x in pairs(t) do
    col = push(i.all, new(at,x))
    if col.usep  then
      if klassp(col.txt) then i.klass=col end
      push(goalp(col.txt) and i.ys or i.xs, col) end end end

function COLS.add(i,t)
  for _,cols in pairs{i.xs,i.ys} do
    for _,col in pairs(cols) do col:add(t[col.at]) end end
  return t end
--       ._    _          ----------------------
--       |    (_)  \/\/

function ROW.new(i,of,cells) i.of,i.cells,i.evaled=of,cells,false end
function ROW.klass(i) return i.cells[i.of.cols.klass.at] end
function ROW.within(i,range,         lo,hi,at,v)
   lo, hi, at = range.xlo, range.xhi, range.ys.at
   v = i.cells[at]
   return  v=="?" or (lo==hi and v==lo) or (lo<v and v<=hi) end

function ROW.b4(i,j,at,   x,y)
  x, y = i.cells[at], j.cells[at]
  x    = x=="?" and -big or x
  y    = y=="?" and -big or y
  return x < y end
--       ._    _          _  ------------------
--       |    (_)  \/\/  _>

local function load(src, fun)
  if type(src)~="string" then for _,t in pairs(src) do fun(t) end
                          else for   t in csv(src)   do fun(t) end end end

function ROWS.new(i,t) i.cols=COLS(t); i.rows={} end
function ROWS.add(i,t)
  t=t.cells and t or ROW(i,t)
  i.cols:add(t.cells)
  return push(i.rows, t) end

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

function ROWS.clone(i,t,  j)
  j= ROWS(i.cols.names);for _,row in pairs(t or {}) do j:add(row) end; return j end

function ROWS.like(i,t, 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 = t.cells[col.at]
    if x and x ~= "?" then
      inc  = col:like(x,prior)
      like = like + math.log(inc) end end
  return like end
--       ._    |_   --------------------------
--       | |   |_)
--
-- (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,      guess)
  report = report or print
  i.overall, i.dict, i.list  = nil, {}, {}
  load(src, function(row,   k)
    if not i.overall then i.overall = ROWS(row) else  -- (0) eat row1
      row = i.overall:add(row)                      -- (1) add to overall
      if #i.overall.rows > THE.wait then
         report(row:klass(), i:guess(row)) end        -- (3) classify before updating
      k = row:klass()                               -- what klass is this?
      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 end) end                -- (2) add to this row's klass

function NB.guess(i,row)
    return argmax(i.dict,
      function(klass) return klass:like(row,#i.list,#i.overall.rows) end) end
--       ._    _.  ._    _    _   -------------
--       |    (_|  | |  (_|  (/_
--                       _|
function RANGE.new(i, xlo, xhi, ys) i.xlo, i.xhi, i.ys = 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

-- 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 = {}
--              for label,rows in pairs(listOfRows) do
--                 for _,row in pairs(rows) do
--                   if row:selects(range) then push(listOfRows[label],
-- end
--
function TREE.bins(i,col,listOfRows)
  local function merge(b4,min)
    local t,j, a,b,c,A,B,C = {},1
    while j <= #b4 do
      a, b = b4[j], b4[j+1]
      if b then
        A,B = a.ys, b.ys
        C   = A:merge(B)
        if A.n<min or B.n<min or C:div() <= (A.n*A:div() + B.n*B:div())/C.n then
          j = j + 1
          a = RANGE(a.xlo, b.xhi, C) end end
      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 ---------------------------------
  local out, ranges, n = {}, {}, 0
  for label,rows in pairs(listOfRows) do
    for _,row in pairs(rows) do
      local v = row.cells[col.at]
      if v ~= "?" then
        n = n + 1
        local  r  = col:bin(v)
        ranges[r] = ranges[r] or push(out, RANGE(v,v, SYM(col.at, col.txt)))
        ranges[r]:add(v,label) end end end
  out = sort(out, lt"xlo")
  out = col.is=="NUM" and merge(out, n^THE.min) or out
  return {col    = col,
          ranges = #out<2 and {} or out,
          div    = sum(out, function(one) return one.n/n*one:div() end)} end
--      _|_   _    _  _|_   _   --------------
--       |_  (/_  _>   |_  _>

local no,go = {},{}
function go.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 i=1,100 do n:add(i) 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)
  load(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
--        _   _|_   _.  ._  _|_  ---------------
--       _>    |_  (_|  |    |_

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

--         .---------.
--         |         |
--       -= _________ =-
--          ___   ___
--         |   )=(   |
--          ---   ---
--             ###
--           #  =  #    "This ain't chemistry.
--           #######     This is art."
```