```
NB:
(c)2022 Tim Menzies, timm@ieee.org
      OFTIONS:
-b --Bins max number of bins = 16
-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
   local _ = require"lib"
     local argmax,big = _argmax, _.big
local cli,csv, demos,is,normpdf = _.cli, _.csv, _.demos, _.is, _.normpdf
local oo,push,read,rnd,same,str= _.oo, _.push, _.read, _.rnd,_same,_.str
    local NB,NUM,SYM,COLS,ROW,ROWS= is"NB",is"NUM",is"SYM",is"COLS",is"ROW",is"ROWS"
local FEW,RANGE, TREE = is"FEW", is"RANGE", is"TREE"
             1,- CI, 1,I CI (\(\sigma\)
     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 > 1.xhi then i.xli = x end -- works for string or num
        i.vs:add(v) end
     function BANCE. Lostring(i) local x, lo, fi = i, ys txt, i, xlo, i, xhi if lo = hi then return fmt("%= = %s", x, lo) elseif li = big then return fmt("%= > %s", x, lo) elseif lo == -big then return fmt("%> > %s", x, x, x) else = -big then return fmt("%> > %s", x, x, x) else = -big then return fmt("%> > %s", x, x, x) else = -big then return fmt("%> > %s", x, x, x, x) else = -big then return fmt("%> > %s = > %s", lo, x, x, x) end end
                  C (3) | |_| | | | | | | | | | | | |
     function FEW.new(i) i.n,i.t,i.ok=0,{},true end
    function FEW.add(i,x)

i.t=i.ok and i.t or sort(i.t); i.ok=true; return i.t end
function FEW.add(i,x)
       if x=="?" then return x end
        i.n=1.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</pre>
     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_NUM, v_number)
  if v=="?" then return v end
  i.few:add(v)
       i.few: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
    function NUM.merge(i,j, k)
local k = NUM(i.at, i.txt)
for _,n in pairs(i.few.t) do k:add(x) end
for _,n in pairs(j.few.t) do k:add(x) end
return k end
     function NUM.mergeRanges(i,b4,min)
local t, j, a,b,c,A,B,C = {},1
while j c = $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</pre>
                  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 i:mergeRanges(t,min) end</pre>
        for j=2,#t do t[j].xlo = t[j-1].xhi end
t[1].xlo, t[#t].xhi = -big, big
                                                       i.n,i.syms,i.most,i.mode = 0,{},0,nil end
     function SYM.new(i)
     Tunction SiM.new(i) i.h.i.symms,i.mose = U,(j,v,hii end function SiM.nid(i, v, prior) return (in.symms(x) or 0)+THE.m*prior)/(i.n+THE.m) end function SiM.hin(x) return (ii.symms(x) or 0)+THE.m*prior)/(i.n+THE.m) end function SiM.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
function SYM.mergeRanges(i,t,...) return t end
```

125 -- (_ (_) _> verturn not x:find".\$" end return not x:find".\$" end return x:find".\$" local function hassp(x) events within 's' and 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 -l 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 = range.xlo, range.xhi, range.ys.at v = i.cells[at] return v=="7" or (lo=hi and v==lo) or (lov and v<=hi) end</pre> 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</pre> local function data(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.w) 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 rowl 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) Aunction NB.new(i,src,report, report = report or print i.overall, i.dict, i.list = nil, {}, {} data(src, function(row, k)) if not i.overall then i.overall = ROWS(row) else -- (0) eat rowl -- add to overall -- add to overall if #i.overall.rows > THE.wait then report(row:klass(), i:guess(row)) end i:train(row) end end) end -- add tp rows's klass return i.dict[k] end function NB quess(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 -- function Tree.new(i,rowss,gaurd) -- i.gaurd, i.kids, labels = gaurd, {},{} -- xcols,rows = nil,{} -- local function labeller(row) return labels[row.id] end) end end -- local function ranges -- for label,rows0 in pairs(rows0) do -- for _rrow in pairs(rows0) do -- labels[row.id] = label -- xcols = push(rows,row).of.cols.xs end end -- ranges= sort(map(of.cols.xs, function(xcol) return i:bins(rows, xcol, SYM, labell er) end), er) end), lt"div")[1].ranges end end 244 -- function TREE.bins(i,rows,xcol,yklass,y) 244 -- Tunction IREE.DINS(1,rowS,RCOL,y 245 -- local n,list, dict = 0,{1,{1}} 246 -- for __row in pairs(rows) do 247 -- local v = row.cells[xcol.at] 248 -- if v ~= "?" then

page 3

```
268 -- n = n + 1

260 -- local pos = xcol:bin(v)

261 -- dict[pos] = dict[pos] or push(list, RANGE(v,v, yklass(xcol.at, xcol.txt)))

262 -- dict[pos]:add(v, y(row)) end end

263 -- list = xcol:merqeRanges(sort(list, lt*xlo*),n^THE.min)

264 -- return {ranges=list,

265 -- div = sum(list, function(z) return z.ys:div()*z.ys.n/n end)} end
```