

```

1  #!/usr/bin/env lua
2  -- vim : filetype=lua ts=2 sw=2 et :
3  --
4  --
5  --
6  --
7  --
8  --
9  -- (c)2021 Tim Menzies. Permission is hereby granted, free of charge,
10 -- to any person obtaining a copy of this software and associated
11 -- documentation files (the "Software"), to deal in the Software without
12 -- restriction, including without limitation the rights to use, copy,
13 -- modify, merge, publish, distribute, sublicense, and/or sell copies
14 -- of the Software, and to permit persons to whom the Software is
15 -- furnished to do so, subject to the following conditions:
16 --
17 -- The above copyright notice and this permission notice shall be included in all
18 -- copies or substantial portions of the Software.
19 --
20 -- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
21 -- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
22 -- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
23 -- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
24 -- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
25 -- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
26 -- SOFTWARE
27 local help = [[
28 lua rezon.lua [OPTIONS]
29
30 Tree learner (binary splits on numerics using Gaussian approximation)
31 (c)2021 Tim Menzies <timmm@ieee.org> MIT license.
32
33 OPTIONS:
34 -best      X   Best examples are in 1..best*size(all)      = .2
35 -debug     X   run one test, show stackdumps on fail      = pass
36 -epsilon   X   ignore differences under epsilon*stdev     = .35
37 -Far       X   How far to look for remove items          = .9
38 -file      X   Where to read data                        = ../data/auto93.csv
39 -h         X   Show help                                  = false
40 -little    X   size of subset of a list                   = 256
41 -p         X   distance calc coefficient                  = 2
42 -round     X   Control for rounding numbers               = 2
43 -seed      X   Random number seed;                       = 10019
44 -Stop      X   Create subtrees while at least 2*stop eggs = 4
45 -Tiny      X   Min range size = size(eggs)^tiny           = .5
46 -todo      X   Pass/fail tests to run at start time      = pass
47               If "X=all", then run all.
48               If "X=k" then list all.
49
50 Data read from "-file" is a csv file whose first row contains column
51 names (and the other row contain data. If a name contains ":",
52 that column will get ignored. Otherwise, names starting with upper
53 case denote numerics (and the other columns are symbolic). Names
54 containing "!" are class columns and names containing "+" or "-"
55 are goals to be maximized or minimized.]] --[[
56
57 Internally, columns names are read by a COLS object where numeric,
58 symbolic, and ignored columns generate NUM, SYM, and SKIP instances
59 (respectively). After row1, all the other rows are examples ('EG')
60 which are stored in a SAMPLE. As each example is added to a sample,
61 they are summarized in the COLS' objects.
62
63 Note that SAMPLEs can be created from disk data, or at runtimes from
64 lists of examples (see SAMPLE:clone()). --]]
65
66 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
67 local THE = {} -- The THE global stores the global config for this software.
68 -- any line of help text starting with " -" has flag,default as first,last word
69 help:gsub("\n [-]([^\s]+)[^\n]*%s([^\s]+)",
70 function(flag,x)
71     for n,word in ipairs(arg) do -- check for any updated to "flag" on command line
72         -- use any command line "word" that matches the start of "flag"
73         if flag:match("^"..word:sub(2).."*") then
74             -- command line "word"s for booleans flip the default value
75             x=(x=="false" and "true") or (x=="true" and "false") or arg[n+1] end end
76         if x=="true" then x=true elseif x=="false" then x=false else x=tonumber(x) or x end
77     end THE[flag] = x end)
78
79 THE.seed = THE.seed or 10019
80 if THE.h then return print(help) end

```

```

81 --
82 --
83 --
84 --
85 -- meta
86 local same
87 function same(x,...) return x end
88
89 -- sorting
90 local push,sort,ones
91 function push(t,x) table.insert(t,x); return x end
92 function sort(t,f) table.sort(t,f); return t end
93 function ones(a,b) return a[1] < b[1] end
94
95 -- tables
96 local copy,keys,map,sum
97 function copy(t, u) u={};for k,v in pairs(t) do u[k]=v end; return u end
98 function keys(t, u) u={};for k,_ in pairs(t) do u[1+#u]=k end; return sort(u) end
99 function map(t,f, u) u={};for k,v in pairs(t) do u[1+#u] =f(k,v) end; return u end
100 function sum(t,f, n) n=0 ;for _,v in pairs(t) do n=n+(f or same)(v) end;return n end
101
102 -- printing utils
103 local hue,shout,out,say,fmt
104 fmt = string.format
105 function say(...) print(string.format(...)) end
106 function hue(n,s) return string.format("%27[1m27[%sm%27[0m",n,s) end
107 function shout(x) print(out(x)) end
108 function out(t, u,key,val) -- convert nested tables to a string
109     function key(_,k) return string.format("%.5s %s", k, out(t[k])) end
110     function val(_,v) return out(v) end
111     if type(t) ~= "table" then return tostring(t) end
112     u = #t>0 and map(t, val) or map(keys(t), key)
113     return (t._is or "").."{"..table.concat(u,"").."}" end
114
115 -- reading from file
116 local coerce,csv
117 function coerce(x)
118     if x=="true" then return true elseif x=="false" then return false end
119     return tonumber(x) or x end
120
121 function csv(file, x)
122     file = io.input(file)
123     return function() t,tmp)
124         x = io.read()
125         if x then -- kill space, split on ",", return non-empty lines
126             t={};for y in x:gsub("[\t]*",""):gmatch("[^\t,]+") do push(t,coerce(y)) end
127             if #t>0 then return t end
128             else io.close(file) end end end
129
130 -- maths
131 local log,sqrt,rnd,rnds,roots
132 log = math.log
133 sqrt = math.sqrt
134 function rnd(x,d, n) n=10^(d or THE.round); return math.floor(x*n+0.5) / n end
135 function rnds(t,d)
136     return map(t,function(_,x) return type(x)=="number" and rnd(x,d) or x end) end
137
138 function roots(m1,m2,std1,std2, a,b,c)
139     if std1==std2 then return (m1+m2)/2 end
140     a = 1/(2*std1^2) - 1/(2*std2^2) -- 1/(2*1^1)
141     b = m2/(std2^2) - m1/(std1^2)
142     c = m1^2/(2*std1^2) - m2^2/(2*std2^2) - log(std2/std1)
143     return ((-b - sqrt(b*b - 4*a*c))/(2*a)), ((-b + sqrt(b*b - 4*a*c))/(2*a)) end
144
145 -- random stuff (LUA's built-in randoms give different results on different platforms)
146 local randi,rand,any,some,shuffle
147 function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
148 function rand(lo,hi)
149     lo, hi = lo or 0, hi or 1
150     THE.seed = (16807 * THE.seed) % 2147483647
151     return lo + (hi-lo) * THE.seed / 2147483647 end
152
153 function any(t) return t[randi(1,#t)] end
154 function some(t,n, u)
155     if n >= #t then return shuffle(copy(t)) end
156     u={}; for i=1,n do push(u,any(t)) end; return u end
157
158 function shuffle(t, j)
159     for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end
160
161 -- objects
162 local ako,has,obj
163 ako= getmetatable
164 function has(mt,x) return setmetatable(x,mt) end
165 function obj(s, o,new)
166     o = {_is=s, __tostring=out}
167     o.__index = o
168     return setmetatable(o,{__call=function(_,...) return o.new(...) end}) end

```

```

169 --
170 -- NUM
171 --
172 --
173 local NUM=obj"NUM"
174 function NUM.new(inits,at,txt, self)
175   self = has(NUM,{n=0, at=at or 0, txt=txt or "",
176     w=(txt or ""):find("-" and -1 or 1,
177     mu=0, m2=0, lo=math.huge, hi=-math.huge)})
178   for _,x in pairs(inits or {}) do self:add(x) end
179   return self end
180
181 -- summarizing
182 function NUM:mid() return self.mu end
183 function NUM:spread() return (self.m2/(self.n-1))^.5 end
184
185 -- updating
186 function NUM:add(x, d)
187   if x ~= "?" then
188     self.n = self.n + 1
189     d = x - self.mu
190     self.mu = self.mu + d/self.n
191     self.m2 = self.m2 + d*(x-self.mu)
192     self.lo = math.min(x, self.lo)
193     self.hi = math.max(x, self.hi) end
194   return x end
195
196 -- querying
197 function NUM:norm(x)
198   local lo,hi = self.lo,self.hi
199   return math.abs(hi - lo) < 1E-9 and 0 or (x-lo)/(hi-lo) end
200
201 function NUM:dist(x,y)
202   if x=="?" then y=self:norm(y); x=y>0.5 and 0 or 1
203   elseif y=="?" then x=self:norm(x); y=x>0.5 and 0 or 1
204   else x, y = self:norm(x), self:norm(y) end
205   return maths.abs(x-y) end
206
207 -- discretization
208 function NUM:splits(other)
209   local function cuts(x,s,at) return {
210     {val=x,at=at,txt=fmt("%s <= %s",s,rnd(x)),when=function(z) return z<=x end},
211     {val=x,at=at,txt=fmt("%s > %s",s,rnd(x)),when=function(z) return z > x end}}
212   end
213   local root1,root2 = roots(self:mid(), other:mid(), self:spread(), other:spread())
214   if self.mu<=root1 and root1<=other.mu
215   then return cuts(root1,self.txt,self.at)
216   else return cuts(root2,self.txt,self.at) end end
217
218 --
219 -- SYM
220 --
221 local SYM=obj"SYM"
222 function SYM.new(inits,at,txt, self)
223   self= has(SYM,{n=0, at=at or 0, txt=txt or "",
224     seen={}, mode=nil, most=0})
225   for _,x in pairs(inits or {}) do self:add(x) end
226   return self end
227
228 -- Summarizing
229 function SYM:mid() return self.mode end
230 function SYM:spread()
231   return sum(self.seen, function(n) return -n/self.n*log(n/self.n,2) end) end
232
233 -- update
234 function SYM:add(x)
235   if x ~= "?" then
236     self.n = 1 + self.n
237     self.seen[x] = (self.seen[x] or 0) + 1
238     if self.seen[x] > self.most then self.mode, self.most = x, self.seen[x] end
239     return x end end
240
241 -- querying
242 function SYM:dist(x,y) return x==y and 0 or 1 end
243
244 -- discretization
245 function SYM:splits(other)
246   local function cut(_,x) return
247     {val=x, at=self.at, txt=fmt("%s==%s",self.txt,x),
248     when = function(z) return z==x end} end
249   local out={}
250   for k,_ in pairs(self.seen) do out[k]=k end
251   for k,_ in pairs(other.seen) do out[k]=k end
252   return map(sort(out),cut) end
253

```

```

254 --
255 -- SKIP
256 --
257 --
258 -- Columns for values we want to ignore.
259 local SKIP=obj"SKIP"
260 function SKIP.new(inits,at,txt)
261   return has(SKIP,{n=0, at=at or 0, txt=txt or ""}) end
262
263 function SKIP:mid() return "?" end
264 function SKIP:spread() return 0 end
265 function SKIP:add(x) return x end
266 function SKIP:splits(_) return {} end
267
268 -- EG
269 --
270 --
271 -- One example
272 local EG=obj"EG"
273 function EG.new(cells) return has(EG,{cells=cells}) end
274
275 -- Sumamrizing
276 function EG:mid(cols)
277   return map(cols,function(_,c) return self.cells[c.at] end) end
278
279 -- Queries
280 function EG:dist(other,cols, a,b,d,n,inc)
281   d,n = 0,0
282   for _,col in pairs(cols) do
283     a,b = self.cells[col.at], other.cells[col.at]
284     inc = a=="?" and b=="?" and 1 or col:dist(a,b)
285     d = d + inc^THE.p
286     n = n + 1 end
287   return (d/n)^(1/THE.p) end
288
289 -- Sorting
290 function EG:better(other,cols, e,n,a,b,s1,s2)
291   n,s1,s2,e = #cols, 0, 0, 2.71828
292   for _,num in pairs(cols) do
293     a = num:norm(self.cells[num.at])
294     b = num:norm(other.cells[num.at])
295     s1 = s1 - e^(num.w * (a-b)/n)
296     s2 = s2 - e^(num.w * (b-a)/n) end
297   return s1/n < s2/n end
298
299 -- COLS
300 --
301 --
302 -- Convert column headers into NUMs and SYMs, etc.
303 local COLS=obj"COLS"
304 function COLS.new(names, self, new,what)
305   self = has(COLS, {names=names, xs={}, all={}, ys={}})
306   for n,x in pairs(names) do
307     new = (x:find":" and SKIP or x:match"^[A-Z]" and NUM or SYM)({},n,x)
308     push(self.all, new)
309     if not x:find":" then
310       if x:find"!" then self.klass = new end
311       what = (x:find"-" or x:find"+") and "ys" or "xs"
312       push(self[what], new) end end
313   return self end
314
315 -- Updates
316 function COLS:add(eg)
317   return map(eg, function(n,x) self.all[n]:add(x); return x end) end
318

```

```

319 --
320 -- SAMPLE
321 --
322 --
323 -- SAMPLEs hold many examples
324 local SAMPLE=obj"SAMPLE"
325 function SAMPLE.new(inits, self)
326   self = has(SAMPLE, {cols=nil, eggs={}})
327   if type(inits)=="string" then for eg in csv(inits) do self:add(eg) end end
328   if type(inits)=="table" then for eg in pairs(inits) do self:add(eg) end end
329   return self end
330
331 -- Create a new sample with the same structure as this one
332 function SAMPLE:clone(inits, out)
333   out = SAMPLE:new(self.cols.names)
334   for _,eg in pairs(inits or {}) do out:add(eg) end
335   return out end
336
337 -- Updates
338 function SAMPLE:add(eg)
339   eg = eg.cells and eg.cells or eg
340   if self.cols
341   then push(self.egs, EG(eg)); self.cols:add(eg)
342   else self.cols = COLS(eg) end end
343
344 -- Distance queries
345 function SAMPLE:neighbors(eg1,egs,cols)
346   local dist_eg2 = function(_,eg2) return {eg1:dist(eg2,cols or self.xs),eg2} end
347   return sort(map(egs or self.egs,dist_eg2),firsts) end
348
349 function SAMPLE:distance_farExample(eg1,egs,cols, tmp)
350   tmp = self:neighbors(eg1, egs, cols)
351   return table.unpack(tmp[#tmp*self.Far//1]) end
352
353 -- Discretization
354 function SAMPLE:twain(egs,cols)
355   local egs, north, south, a,b,c, lo,hi
356   egs = some(egs or self.egs, self.little)
357   _,north = self:distance_farExample(any(self.egs), egs, cols)
358   c,south = self:distance_farExample(north, egs, cols)
359   for _,eg in pairs(self.egs) do
360     a = eg:dist(north, cols)
361     b = eg:dist(south, cols)
362     eg.x = (a^2 + c^2 - b^2)/(2*c) end
363   lo, hi = self:clone(), self:clone()
364   for n,eg in pairs(sort(self.egs, function(a,b) return a.x < b.x end)) do
365     if n < .5*#eg then lo:add(eg) else hi:add(eg) end end
366   return lo, hi end
367
368 function SAMPLE:mid(cols)
369   return map(cols or self.cols.all,function(_,col) return col:mid() end) end
370
371 function SAMPLE:spread(cols)
372   return map(cols or self.cols.all,function(_,col) return col:spread() end) end
373
374 function SAMPLE:sorted()
375   self.egs= sort(self.egs, function(eg1,eg2) return eg1:better(eg2,self.cols.ys) end)
376   return self.egs end
377

```

```

378 --
379 -- SAMPLE TREE
380 --
381 --
382 -- need to sort first
383
384 -- how to score
385 function SAMPLE:splits(other,both, cuts,unplaced,place,score)
386   function guess(todos,cuts)
387     for _,todo in pairs(todos) do
388       local f=function(_,cut)
389         return {Row(cut.has:mid()):dist(todo, both.cols.xs),cut} end
390       sort(map(cuts,f),firsts)[1][2].has:add(todo) end
391     return cuts end
392   function divide(cuts, todos,placed)
393     todos = {}
394     for _,eg in pairs(both.egs) do
395       placed = false
396       for _,cut in pairs(cuts) do
397         if cut.what(eg.cells[cut.at])
398         then cut.has = cut.has or self:clone()
399             cut.has:add(eg)
400             placed = true
401             break end end
402         if not placed then push(todos, eg) end end
403     return guess(todos,cuts) end
404   function score(cut, m,n)
405     m,n = #cut.has.egs,both.egs; return -m/n*log(m/n,2) end
406   local best, cutsx, tmp = math.huge
407   for pos,col in pairs(both.cols.xs) do
408     cutsx = col:splits(other.cols.xs[pos])
409     tmp = sum(divide(cutsx),score)
410     if tmp < best then best,cuts = tmp,cutsx end end
411   return cuts end
412
413 function SAMPLE:tree(top)
414   top = top or self
415   one,two = self:twain(self.egs, top.cols.xs)
416   for _,cut in pairs(one:splits(two,self)) do
417     if cut.stats.n > (#top.egs)^THE.Tiny then
418       cut.sub= cut.has:tree(top) end end end
419
420 function SAMPLE:show(tree)
421   local vals=function(a,b) return a.val < b.val end
422   local function show1(tree,pre)
423     if #tree.kids==0 then io.write(fmt("==> %s[%s]",tree.mode, tree.n)) end
424     for _,kid in pairs(sort(tree.kids,vals)) do
425       io.write("\n"..fmt("%s%s",pre, showDiv(i, kid.at, kid.val)))
426       show1(kid.sub, pre.."|..") end
427   end -----
428   show1(tree,""); print("") end
429

```

```

430 -----
431 --
432 --  E X A M P L E S
433 --
434 --
435 local go={}
436 function go.ls()
437   print("\nlua"..arg[0].." -todo ACTION\n\nACTIONS:")
438   for _,k in pairs(keys(go)) do print("-todo",k) end end
439
440 function go.pass() return true end
441 function go.the() shout(TH) end
442 function go.bad( s) assert(false) end
443
444 function go.sort( u,t)
445   t={}; for i=100,1,-1 do push(t,i) end
446   t=sort(t,function(x,y)
447     if x+y<20 then return x>y else return x<y end end)
448   assert(sum(t,function(x) return x*100 end)==505000)
449   assert(t[1] == 10)
450   assert(t[#t]==100)
451   u=copy(t)
452   t[1] = 99
453   assert(u[1] ~= 99) end
454
455 function go.out( s)
456   assert({"age 21 :milestones {1 2 3 4} :name tim"}==out(
457     {name='tim', age=21, milestones={1,2,3,4}}))end
458
459 function go.file( n)
460   for _,t in pairs({"true",true,"boolean"}, {"false",false,"boolean"},
461     {"42.1",42.1,"number"}, {"32zz","32zz","string"},
462     {"nil","nil","string"}} do
463     assert(coerce(t[1])==t[2])
464     assert(type(coerce(t[1]))==t[3]) end
465   n =0
466   for row in csv(TH) do
467     n = n + 1
468     assert(#row==8)
469     assert(n==1 or type(row[1])=="number")
470     assert(n==1 or type(row[8])=="number") end end
471
472 function go.rand( t,u)
473   t,u={},{}; for i=1,20 do push(u,push(t,100*rand())) end
474   t= sort(rnds(t,0))
475   assert(t[1]==3 and t[#t]==88)
476   t= sort(some(t,4))
477   assert(#t==4)
478   assert(t[1]==7)
479   assert(79.5 == rnds(shuffle(u))[1])
480 end
481
482 function go.num( cut,min)
483   local z = NUM{9,2,5,4,12,7,8,11,9,3,7,4,12,5,4,10,9,6,9,4}
484   assert(7 == z:mid(), 3.06 == rnd(z:spread(),2))
485   local r1,r2 = roots(2.5, 5, 1.1, .9)
486   assert(rnd(r2,2)==3.8)
487   local x, y = NUM(), NUM()
488   for i=1,20 do x:add(rand(1,5)) end
489   for i=1,20 do y:add(randi(20,30)) end
490   for _,cut in pairs(x:splits(y)) do shout(cut) end end
491
492 function go.sym( cut,min)
493   local w = SYM{"m","m","m","m","b","b","b","c"}
494   local z = SYM{"a","a","a","a","b","b","b","c"}
495   assert(1.38 == rnd(z:spread(),2))
496   for _,cut in pairs(w:splits(z)) do shout(cut) end end
497
498 function go.sample( s,egs,xs,ys)
499   s=SAMPLE(TH,file)
500   xs,ys= s.cols.xs, s.cols.ys
501   assert(4 == #xs)
502   assert(3 == #ys)
503   egs=s:sorted()
504   shout(rnds(s:mid(ys),1));
505   shout(rnds(map(s:spread(ys),function(_,x) return .35*x end), 1)); print("")
506   for i=1,10 do shout(rnds(egs[i]:mid(ys),1)) end; print("")
507   for i=#egs,#egs-10,-1 do shout(rnds(egs[i]:mid(ys),1)) end end
508
509 function go.kordered( s,n)
510   s = ordered(slurp())
511   n = #s.egs
512   shout(s.heads)
513   for i=1,15 do shout(s.egs[i].cells) end
514   print("#")
515   for i=n,n-15,-1 do shout(s.egs[i].cells) end end
516
517 function go.ksymcuts( s,xpect,cuts)
518   s=ordered(slurp())
519   print(out(s.xs),out(s.ys))

```

```

520   xpect,cuts = symcuts(7,s.egs, "origin")
521   for _,cut in pairs(cuts) do print(xpect, out(cut)) end end
522
523 function go.knumcuts( s,xpect,cuts)
524   s=ordered(slurp())
525   xpect,cuts = numcuts(s,2,s.egs,"Displcement")
526   if xpect then
527     for _,cut in pairs(cuts) do print(xpect, out(cut)) end end end
528
529 function go.katcuts( s,cuts,at,ynum)
530   s=ordered(slurp())
531   ynum=NUM(a); map(s.egs,function(_,eg) add(ynum, eg.klass) end)
532   at,cuts = at_cuts(s,egs,sd(ynum)*THE.epsilon, (#s.egs)^THE.Tiny)
533   for _,cut in pairs(cuts) do print(at, out(cut)) end end
534
535 --
536 --  S T A R T - U P
537 --
538 local fails, defaults = 0, copy(TH)
539 go[ TH.debug ]()
540 local todos = TH.todo == "all" and keys(go) or {TH.todo}
541 for _,todo in pairs(todos) do
542   TH = copy(defaults)
543   local ok,msg = pcall( go[todo] )
544   if ok then io.write(hue(32,"PASS")..todo.."")
545   else io.write(hue(31,"FAIL")..todo.." "..msg.."")
546     fails=fails+1 end end
547
548 for k,v in pairs(_ENV) do if not b4[k] then print("?:",k,type(v)) end end
549 os.exit(fails)

```