


```

1  #!/usr/bin/env lua
2  -- vim : filetype=lua ts=2 sw=2 et :
3  --
4  --
5  
6  --
7  --
8  -- (c)2021 Tim Menzies. Permission is hereby granted, free of charge,
9  -- to any person obtaining a copy of this software and associated
10 -- documentation files (the "Software"), to deal in the Software without
11 -- restriction, including without limitation the rights to use, copy,
12 -- modify, merge, publish, distribute, sublicense, and/or sell copies
13 -- of the Software, and to permit persons to whom the Software is
14 -- furnished to do so, subject to the following conditions:
15 --
16 -- The above copyright notice and this permission notice shall be included in all
17 -- copies or substantial portions of the Software.
18 --
19 -- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
20 -- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
21 -- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
22 -- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
23 -- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
24 -- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
25 -- SOFTWARE
26 local help = [[
27 muse [OPTIONS]
28
29 Tree learner (binary splits on numerics using Gaussian approximation)
30 (c)2021 Tim Menzies <timmm@ieee.org> MIT license.
31
32 OPTIONS:
33 -best X Best examples are in 1..best*size(all) = .2
34 -debug X run one test, show stackdumps on fail = pass
35 -epsilon X ignore differences under epsilon*stdev = .35
36 -Far X How far to look for remove items = .9
37 -file X Where to read data = ../data/auto93.csv
38 -goal X smile, frown, xplor, doubt = smile
39 -h Show help = false
40 -little X size of subset of a list = 1024
41 -more X Use more*#best for rest = 3.5
42 -p X distance calc coefficient = 2
43 -round X Control for rounding numbers = 2
44 -seed X Random number seed; = 10019
45 -Stop X Create subtrees while at least 2*stop eggs = 4
46 -Tiny X Min range size = size(eggs)^tiny = .5
47 -todo X Pass/fail tests to run at start time = pass
48 If "X=all", then run all.
49 If "X=k" then list all.
50 -verbose Show low-level traces. = false
51
52 Data read from "-file" is a csv file whose first row contains column
53 names (and the other row contain data. If a name contains ":",
54 that column will get ignored. Otherwise, names starting with upper
55 case denote numerics (and the other columns are symbolic). Names
56 containing "!" are class columns and names containing "+" or "-"
57 are goals to be maximized or minimized. ]] --[[
58
59 Internally, columns names are read by a COLS object where numeric,
60 symbolic, and ignored columns generate NUM, SYM, and SKIP instances
61 (respectively). After row1, all the other rows are examples ('EG')
62 which are stored in a SAMPLE. As each example is added to a sample,
63 they are summarized in the COLS' objects.
64
65 Note that SAMPLEs can be created from disk data, or at runtimes from
66 lists of examples (see SAMPLE:clone()). --]]
67
68 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
69 local THE = {} -- The THE global stores the global config for this software.
70 -- any line of help text starting with " -" has flag,default as first,last word
71 help:gsub("(\\n[-])([^%s]+)(^\\n)%s{1,3}(\\n)",
72 function(flag,x)
73   for n,word in ipairs(arg) do -- check for any updated to "flag" on command line
74     -- use any command line "word" that matches the start of "flag"
75     if flag:match("^"..word:sub(2).."*") then
76       -- command line "word"s for booleans flip the default value
77       x=(x=="false" and "true") or (x=="true" and "false") or arg[n+1] end end
78     if x=="true" then x=true elseif x=="false" then x=false else x=tonumber(x) or x end
79     THE[flag] = x end)
80
81 THE.seed = THE.seed or 10019
82 if THE.h then return print(help) end

```

```

83 --
84 --
85 --
86 --
87 --
88 --
89 -- meta
90 local function same(x,...) return x end
91 local function upto(x,y) return x < y end
92 local function over(x,y) return not(upto(x,y)) end
93
94 -- sorting
95 local function push(t,x) table.insert(t,x); return x end
96 local function sort(t,f) table.sort(t,f); return t end
97 local function ones(a,b) return a[1] < b[1] end
98
99 -- tables
100 local top,copy,keys,map,sum
101 function copy(t, u) u={};for k,v in pairs(t) do u[k]=v end; return u end
102 function map(t,f, u) u={};for _,v in pairs(t) do u[1+#u]=f(v) end; return u end
103 function sum(t,f, n) n=0 ;for _,v in pairs(t) do n=n+(f or same)(v) end;return n end
104 function top(t,n, u)
105   u={}; for k,v in pairs(t) do if k>n then break end; u[#u+1]=v end; return u end
106
107 function keys(t, u)
108   u={}; for k,_ in pairs(t) do
109     if tostring(k):sub(1,1) ~= "-" then u[1+#u]=k end end;
110     return sort(u) end
111
112 -- printing utils
113 local fmt = string.format
114 local function say(...) if THE.verbose then print(fmt(...)) end end
115 local function btw(...) io.stderr:write(fmt(...).."\n") end
116 local function hue(n,s) return string.format("%27[1m%27[%sm%27[0m",n,s) end
117
118 local o
119 local function out(x) print(o(x)) end
120 function o(t, u,f) -- convert nested tables to a string
121   local function f(k) return fmt(":%s%s", hue(33,k), o(t[k])) end
122   if type(t) ~= "table" then return tostring(t) end
123   u = #t>0 and map(t, o) or map(keys(t), f)
124   return hue(32, (t._is or ""))..{"..table.concat(u, ",").."} end
125
126 -- reading from file
127 local function coerce(x)
128   if x=="true" then return true elseif x=="false" then return false end
129   return tonumber(x) or x end
130
131 local function csv(file, x,line)
132   function line(x, t)
133     t={}; for y in x:gsub("[\t]*",""):gmatch("([^\t]+)") do push(t,coerce(y)) end
134     return t end
135   file = io.input(file)
136   return function( x)
137     x = io.read()
138     if x then return line(x) else io.close(file) end end end
139
140 -- maths
141 local log = math.log
142 local sqrt= math.sqrt
143 local function rnd(x,d, n) n=10^(d or THE.round); return math.floor(x*n+0.5) / n end
144 local function rnds(t,d)
145   return map(t,function(x) return type(x)=="number" and rnd(x,d) or x end) end
146
147 -- random stuff (LUA's built-in randoms give different results on different platfors)
148 local rand
149 local function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
150 function rand(lo,hi)
151   lo, hi = lo or 0, hi or 1
152   THE.seed = (16807 * THE.seed) % 2147483647
153   return lo + (hi-lo) * THE.seed / 2147483647 end
154
155 local function any(t) return t[randi(1,#t)] end
156 local function shuffle(t, j)
157   for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end
158
159 local function some(t,n, u)
160   if n >= #t then return shuffle(copy(t)) end
161   u={}; for i=1,n do push(u,any(t)) end; return u end
162
163 -- objects
164 local function is(x) return getmetatable(x) end
165 local function as(mt,x) return setmetatable(x,mt) end
166 local function of(s, obj)
167   obj = {_is=s, tostring=o}
168   obj.__index = obj
169   return as({__call=function(_,...) return obj.new(...) end},obj) end

```

```

170 -- GOALS
171
172
173
174 local goals={}
175 --function goals.smile(b,r) if b+r>1E-2 and b>r then return b^2/(b+r+1E-31) end end
176 --function goals.frown(b,r) if b+r>1E-2 and r>b then return r^2/(b+r+1E-31) end end
177 function goals.smile(b,r) if b+r>1E-2 then return b^2/(b+r+1E-31) end end
178 function goals.frown(b,r) if b+r>1E-2 then return r^2/(b+r+1E-31) end end
179 function goals.xplor(b,r) if b+r>1E-2 then return 1/(b+r+1E-31) end end
180 function goals.doubt(b,r) if b+r>1E-2 then return (b+r)/(math.abs(b-r)+1E-31) end end
181
182 -- XXXX have to handle breaks in conjuncts
183 function select(cuts, best, rest, lt, merge)
184   local score, parts, merge, fx, show
185   function score(a,b) return a.score >= b.score end
186   function parts(a,b) return a.col.at<b.col.at or a.col.at==b.col.at and a.lo<b.lo end
187   function merge(b4, j,tmp,now,after)
188     j, tmp = 0, {}
189     while j<#b4 do
190       j = j + 1
191       now, after = b4[j], b4[j+1]
192       if after then
193         if now.hi == after.lo then
194           now = {col=now.col, lo=now.lo, hi= after.hi}
195           j = j + 1 end end
196       push(tmp,now) end
197       return #tmp==#b4 and b4 or merge(tmp)
198     end
199   function fx(cuts)
200     function relevant(eg)
201       for _,cut in pairs(cuts) do
202         local x = eg.cells[cut.col.at]
203         if not(x=="?" or cut.lo <= x and x <= cut.hi) then return nil end end
204       return eg end
205     best1 = #map(best,function(eg) return relevant(eg) end) / #best
206     rest1 = #map(rest,function(eg) return relevant(eg) end) / #rest
207     return best1 / (best1 + rest1)
208   end
209   cuts = sort(cuts,score)
210   for j=1,#cuts do
211     rule= merge(sort(top(cuts,j),parts))
212     print(j, fx(egs,rule), table.concat(map(rule,show)," and ")) end end
213
214 -- SYM
215
216
217 local SYM=of"SYM"
218 function SYM.new(inits,at,txt, i)
219   i = as(SYM,{n=0, at=at or 0, txt=txt or "",
220             has={}, mode=nil, most=0})
221   for _,x in pairs(inits or {}) do i:add(x) end
222   return i end
223
224 -- Summarizing
225 function SYM.merge(i,j, k)
226   k = SYM({},i.at, i.txt)
227   for x,n in pairs(i.has) do k:add(x,n) end
228   for x,n in pairs(j.has) do k:add(x,n) end
229   return k end
230
231 function SYM.mid(i) return i.mode end
232 function SYM.spread(i)
233   return sum(i.has, function(n) return -n/i.n*log(n/i.n,2) end) end
234
235 -- update
236 function SYM.add(i,x,n)
237   if x ~= "?" then
238     n = n or 1
239     i.n = n + i.n
240     i.has[x] = (i.has[x] or 0) + n
241     if i.has[x] > i.most then i.mode, i.most = x, i.has[x] end
242     return x end end
243
244 -- querying
245 function SYM.dist(i,x,y) return x==y and 0 or 1 end
246
247 -- discretization
248 function SYM.splits(i,j,cuts, cut,tmp)
249   cuts = cuts or {}
250   xs= keys(i:merge(j).has)
251   if #xs > 1 then
252     for _,x in pairs(xs) do
253       b = i.has[x] or 0
254       r = j.has[x] or 0
255       s = goals[THE.goal]( b/i.n, r/j.n)
256       if s then push(cuts,{score=s,col=i,lo=x,hi=x}) end end end
257     return cuts end

```

```

258 -- SKIP
259
260
261
262
263 -- Columns for values we want to ignore.
264 local SKIP=of"SKIP"
265 function SKIP.new(inits,at,txt)
266   return as(SKIP,{at=at or 0, txt=txt or ""}) end
267
268 function SKIP.mid(i) return "?" end
269 function SKIP.spread(i) return 0 end
270 function SKIP.add(i,x) return x end
271 function SKIP.splits(i,_) return {} end
272
273 -- NUM
274
275
276
277 local NUM=of"NUM"
278 function NUM.new(inits,at,txt, i)
279   i = as(NUM,{n=0, at=at or 0, txt=txt or "",
280             w=(txt or ""):find("-" and -1 or 1,
281               _has={}),
282             mu=0, m2=0, lo=math.huge, hi=-math.huge})
283   for _,x in pairs(inits or {}) do i:add(x) end
284   return i end
285
286 -- summarizing
287 function NUM.mid(i) return i.mu end
288 function NUM.spread(i) return (i.m2/(i.n-1))^0.5 end
289
290 -- updating
291 function NUM.add(i,x, d)
292   if x ~= "?" then
293     push(i._has, x)
294     i.n = i.n + 1
295     d = x - i.mu
296     i.mu = i.mu + d/i.n
297     i.m2 = i.m2 + d*(x-i.mu)
298     i.lo = math.min(x, i.lo)
299     i.hi = math.max(x, i.hi) end
300   return x end
301
302 function NUM.merge(i,j, k)
303   k = NUM({}, i.at, i.txt)
304   for _,v in pairs(i._has) do k:add(v) end
305   for _,v in pairs(j._has) do k:add(v) end
306   return k end
307
308 -- querying
309 function NUM.norm(i,x)
310   return math.abs(i.hi - i.lo) < 1E-9 and 0 or (x-i.lo)/(i.hi-i.lo) end
311
312 function NUM.dist(i,x,y)
313   if x=="?" then y=i:norm(y); x=y>0.5 and 0 or 1
314   elseif y=="?" then x=i:norm(x); y=x>0.5 and 0 or 1
315   else x, y = i:norm(x), i:norm(y) end
316   return math.abs(x-y) end
317
318 -- discretization
319 local spread_merge
320 function NUM.splits(i,j,cuts, xys,tmp,b,r,s)
321   xys, cuts = {},cuts or {}
322   for _,x in pairs(i._has) do push(xys, {x=x, y="best"}) end
323   for _,x in pairs(j._has) do push(xys, {x=x, y="rest"}) end
324   tmp = spread_merge(sort(xys, function(a,b) return a.x < b.x end),
325                     (#xys)^THE.Tiny,
326                     THE.epsilon*(i.n*i:spread() + j.n*j:spread()/(i.n + j.n),
327                     i,
328                     SYM)
329   if #tmp > 1 then
330     for _,cut in pairs(tmp) do
331       b = cut.has.has.best or 0
332       r = cut.has.has.rest or 0
333       s = goals[THE.goal]( b/i.n, r/j.n)
334       if s then cut.score=s; push(cuts,cut) end end end
335   return cuts end
336

```

```

337 --
338 --
339 --
340 --
341 -- Return a list of 'spans' {lo=hi,col=col}.
342 -- Sort the list of pairs 'xys' then split it into 'spans' of cardinality at
343 -- least 'tiny'. Ensure that the max-min of each span is more that 'trivial'.
344 function spread_merge(xys, tiny, trivial,col,yklass)
345   local function mergeable(a,b, new,b4)
346     new = a:merge(b)
347     b4 = (a.n*a:spread() + b.n*b:spread()) / new.n
348     if new:spread()*1.01 <= b4 then return new end
349   end
350   local function merge(b4, j,tmp,simpler,now,after)
351     local j, tmp = 0, {}
352     while j < #b4 do
353       j = j + 1
354       now, after = b4[j], b4[j+1]
355       if after then
356         simpler = mergeable(now.has, after.has)
357         if simpler then
358           now = {col=col, lo=now.lo, hi= after.hi, has=simpler}
359           j = j + 1 end end
360         push(tmp,now) end end
361       return #tmp==#b4 and b4 or merge(tmp)
362     end
363   local function div( spans,span,x,y)
364     span = {col=col,lo=xys[1].x, hi=xys[1].x, has=yklass()}
365     spans = {span}
366     for j,xy in pairs(xys) do
367       x, y = xy.x, xy.y
368       if j < #xys - tiny and -- enough items remaining after split
369         x ~= xys[j+1].x and -- next item is different (so can split here)
370         span.has.n > tiny and -- span has enough items
371         span.hi - span.lo > trivial -- span is not trivially small
372       then span = push(spans, {col=col, lo=span.hi, hi=x, has=yklass()}) -- then new span
373       end
374       span.hi = x
375       span.has:add(y)
376     end
377     spans[1].lo = -math.huge
378     spans[#spans].hi = math.huge
379     return spans
380   end
381   return merge(div()) end

```

```

382 --
383 --
384 --
385 --
386 -- Convert column headers into NUMs and SYMs, etc.
387 local COLS=of"COLS"
388 function COLS.new(names, i, new,what)
389   i = as(COLS, {names=names, xs={}, all={}, ys={}})
390   for n,x in pairs(names) do
391     new = (x:find"." and SKIP or x:match"^[A-Z]" and NUM or SYM) ({},n,x)
392     push(i.all, new)
393     if not x:find"." then
394       if x:find"!" then i.klass = new end
395       what = (x:find"-" or x:find"+") and "ys" or "xs"
396       push(i[what], new) end end
397   return i end
398
399 -- Updates
400 function COLS.add(i,eg)
401   return map(i.all, function(col) col:add(eg[col.at]); return x end) end
402
403 --
404 --
405 --
406 -- One example
407 local EG=of"EG"
408 function EG.new(cells) return as(EG,{cells=cells}) end
409
410 -- Sumamrizing
411 function EG.cols(i,all)
412   return map(all,function(c) return i.cells[c.at] end) end
413
414 -- Queries
415 function EG.dist(i,j,cols, a,b,d,n,inc)
416   d,n = 0,0
417   for _,col in pairs(cols) do
418     a,b = i.cells[col.at], j.cells[col.at]
419     inc = a=="?" and b=="?" and 1 or col:dist(a,b)
420     d = d + inc^THE.p
421     n = n + 1 end
422   return (d/n)^(1/THE.p) end
423
424 -- Sorting
425 function EG.better(i,j,cols, e,n,a,b,s1,s2)
426   n,s1,s2,e = #cols, 0, 0, 2.71828
427   for _,col in pairs(cols) do
428     a = col:norm(i.cells[col.at])
429     b = col:norm(j.cells[col.at])
430     s1 = s1 - e^(col.w * (a-b)/n)
431     s2 = s2 - e^(col.w * (b-a)/n) end
432   return s1/n < s2/n end
433

```

```

434 --
435 -- SAMPLE
436 --
437 --
438 -- SAMPLEs hold many examples
439 local SAMPLE=of"SAMPLE"
440 function SAMPLE.new(inits, i)
441   i = as(SAMPLE, {cols=nil, eggs={}})
442   if type(inits)=="string" then for eg in csv(inits) do i:add(eg) end end
443   if type(inits)=="table" then for eg in pairs(inits) do i:add(eg) end end
444   return i end
445
446 -- Create a new sample with the same structure as this one
447 function SAMPLE.clone(i, inits, tmp)
448   tmp = SAMPLE.new()
449   tmp:add(i.cols.names)
450   for _,eg in pairs(inits or {}) do tmp:add(eg) end
451   return tmp end
452
453 -- Updates
454 function SAMPLE.add(i, eg)
455   eg = eg.cells and eg.cells or eg
456   if i.cols
457   then push(i.egs, EG(eg)); i.cols:add(eg)
458   else i.cols = COLS(eg) end end
459
460 -- Distance queries
461 function SAMPLE.neighbors(i, egl, eggs, cols, dist_eg2)
462   dist_eg2 = function(eg2) return {egl:dist(eg2, cols or i.cols.xs), eg2} end
463   return sort(map(egs or i.egs, dist_eg2), ones) end
464
465 function SAMPLE.distance_farEg(i, egl, eggs, cols, tmp)
466   tmp = i:neighbors(egl, eggs, cols)
467   tmp = tmp[#tmp*THE.Far//1]
468   return tmp[2], tmp[1] end
469
470 -- Unsupervised discretization
471 function SAMPLE.best(i)
472   local rest, div = {}
473   function div(egs, lvl, one, tmp, a, b, c, two, want, low, good)
474     tmp = i:clone(egs)
475     say("%s%s\t%s",
476         string.rep(".", lvl), #egs, o(rnds(tmp:mid(tmp.cols.ys), 1)))
477     if #egs < 2*(#i.egs)^THE.epsilon then
478       return i:clone(egs), i:clone(some(rest, THE.more*#egs)) end
479     one = one or i:distance_farEg(any(egs), eggs, i.cols.xs)
480     two, c = i:distance_farEg(one, eggs, i.cols.xs)
481     for _, eg in pairs(egs) do
482       a = eg:dist(one, i.cols.xs)
483       b = eg:dist(two, i.cols.xs)
484       eg.x = (a^2 + c^2 - b^2)/(2*c) end
485     low = one:better(two, i.cols.ys)
486     good = {}
487     for n, eg in pairs(sort(egs, function(a, b) return a.x < b.x end)) do
488       if n < .5*#egs then push(low and good or rest, eg)
489       else push(low and rest or good, eg) end end
490     return div(good, lvl+1, two) end
491     return div(same(i.egs, THE.little), 0) end
492
493 function SAMPLE.mid(i, cols)
494   return map(cols or i.cols.all, function(col) return col:mid() end) end
495
496 function SAMPLE.spread(i, cols)
497   return map(cols or i.cols.all, function(col) return col:spread() end) end
498
499 function SAMPLE.sorted(i)
500   i.egs= sort(i.egs, function(eg1, eg2) return eg1:better(eg2, i.cols.ys) end)
501   return i.egs end
502

```

```

503 --
504 -- SAMPLE TREE
505 --
506 --
507 function SAMPLE:splits(other, both, place, score)
508   function place(eg, cuts, x)
509     for _, cut in pairs(cuts) do
510       cut.has = cut.has or self:clone()
511       x = eg.cells[cut.at]
512       if x ~= "?" and cut.when(x) then return cut.has:add(eg) end end end
513   function score(cut, m, n)
514     m, n = # (cut.has.egs), #both.egs; print(m, n); return -m/n*log(m/n, 2) end
515   local best, cutsx, cuts, tmp = math.huge
516   for pos, col in pairs(both.cols.xs) do
517     print("eps", col.at, col:spread()*THE.epsilon)
518     cutsx = col:splits(other.cols.xs[pos], col:spread()*THE.epsilon)
519     for _, eg in pairs(both.egs) do place(eg, cutsx) end
520     tmp = sum(cutsx, score)
521     if tmp < best then best, cuts = tmp, cutsx end end
522   return cuts end
523

```

```

524 -----
525 --
526 --   EXAMPLES
527 --
528 --
529 local go={}
530 function go.pass() return true end
531 function go.the( s ) s=o(THE); say("%s",o(s)) end
532 function go.bad( s ) assert(false) end
533
534 function go.sort( u,t)
535   t={}; for i=100,1,-1 do push(t,i) end
536   t=sort(t,function(x,y)
537     if x+y<20 then return x>y else return x<y end end)
538   assert(sum(t,function(x) return x*100 end)==505000)
539   assert(t[1] == 10)
540   assert(t[#t]==100)
541   u=copy(t)
542   t[1] = 99
543   assert(u[1] ~= 99) end
544
545 function go.file( n)
546   for _,t in pairs({{"true",true,"boolean"}, {"false",false,"boolean"},
547     {"42.1",42.1,"number"}, {"32zz","32zz","string"},
548     {"nil","nil","string"}}) do
549     assert(coerce(t[1])==t[2])
550     assert(type(coerce(t[1]))==t[3]) end
551   n = 0
552   for row in csv(THE.file) do
553     n = n + 1
554     assert(#row==8)
555     assert(n==1 or type(row[1])=="number")
556     assert(n==1 or type(row[8])=="number") end end
557
558 function go.rand( t,u)
559   t,u={},{}; for i=1,20 do push(u,push(t,100*rand())) end
560   t= sort(rnds(t,0))
561   assert(t[1]==3 and t[#t]==88)
562   t= sort(some(t,4))
563   assert(#t==4)
564   assert(t[1]==7)
565   assert(79.5 == rnds(shuffle(u))[1])
566 end
567
568 function go.num( cut,min, z,r1,r2,x,y)
569   z = NUM{9,2,5,4,12,7,8,11,9,3,7,4,12,5,4,10,9,6,9,4}
570   assert(7 == z:mid(), 3.06 == rnd(z:spread(),2))
571   x, y = NUM(), NUM()
572   for i=1,20 do x:add(rand(1,5)) end
573   for i=1,20 do y:add(randi(20,30)) end end
574
575 function go.sym( cut,min,w,z)
576   w = SYM{"m","m","m","m","b","b","c"}
577   z = SYM{"a","a","a","a","b","b","c"}
578   assert(1.38 == rnd(z:spread(),2))
579   for _,cut in pairs(w:splits(z)) do say("%s",o(cut)) end
580 end
581
582 function go.sample( s,egs,xs,ys,scopy)
583   s=SAMPLE(THE.file)
584   scopy=s:clone(s.egs)
585   say("%s %s",s.cols.all[1]:spread(), scopy.cols.all[1]:spread())
586   xs,ys= s.cols.xs, s.cols.ys
587   assert(4 == #xs)
588   assert(3 == #ys)
589   egs=s:sorted()
590   say(o(rnds(s:mid(ys),1)))
591   say(o(rnds(map(s:spread(ys),function(x) return .35*x end), 1)));say("")
592   for i=1,10 do say("%s", o(rnds(egs[i]:cols(ys),1))) end; say("")
593   for i=#egs,#egs-10,-1 do say(o(rnds(egs[i]:cols(ys),1))) end
594 end
595
596 function go.dist( s,xs,sorted, show )
597   s=SAMPLE(THE.file)
598   xs= s.cols.xs
599   sorted = s:neighbors(s.egs[1], s.egs,xs)
600   show=function(i) say("%s %s",rnd(sorted[i][1],2),
601     o(sorted[i][2]:cols(xs))) end
602   for i=1,10 do show(i) end; say("")
603   for i=#sorted-10,#sorted do show(i) end end
604
605 function go.far( s,xs,d,eg2)
606   s = SAMPLE(THE.file)
607   xs = s.cols.xs
608   for k,egl in pairs(shuffle(s.egs)) do
609     if k > 10 then break end
610     eg2,d = s:distance_farEg(egl, s.egs, xs)
611     say("%s %s %s",rnd(d), o(egl:cols(xs)), o(eg2:cols(xs))) end end
612
613 function go.best( all,best,rest,cuts)

```

```

614   all = SAMPLE(THE.file)
615   best,rest = all:best()
616   say(o(best.cols.all[1]))
617   say("%s %s", #best.egs, #rest.egs)
618   say("")
619   cuts={}
620   local order=function(a,b) return
621     a.col.at < b.col.at or a.col.at==b.col.at and a.lo < b.lo end
622   for n,coll in pairs(best.cols.xs) do coll:splits(rest.cols.xs[n],cuts) end
623   for _,cut in pairs(sort(cuts,order)) do
624     say(o(at=cut.col.at, lo=cut.lo, hi=cut.hi, score=cut.score, txt=cut.col.txt)) end
625   end
626 --
627 --   START-UP
628 --
629 --
630 local fails,defaults,todos,ok,msg
631 fails, defaults = 0, copy(THE)
632 go[ THE.debug ]()
633
634 todos = THE.todo == "all" and keys(go) or {THE.todo}
635 for _,todo in pairs(todos) do
636   THE = copy(defaults)
637   ok,msg = pcall( go[todo] )
638   if ok then btw("%s %s",hue(32,"-- PASS "),todo)
639     else btw("%s %s %s",hue(31,"-- FAIL "),todo,msg); fails=fails+1 end end
640
641   btw(hue(33,"-- %s error(s)", fails)
642   for k,v in pairs(_ENV) do
643     if not b4[k] then btw(hue(31,"-- rogue? %s %s"),k,type(v)) end end
644   os.exit(fails)

```