

```

1  -- vim : ft=lua et sw=2 ts=2 :
2  local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end --used later (to find rogues)
3  local help = {}
4
5  sl == S.U.B.L.I.M.E. == Sublime's unsupervised
6  bifurcation: let's infer minimal explanations.
7  (c) 2022, Tim Menzies, BSD 2-clause license.
8
9  USAGE:
10     lua sl.lua [OPTIONS]
11
12  OPTIONS:
13     -Dump          stack dump on assert fails = false
14     -data N        data file = etc/data/auto93.csv
15     -enough F      recurse until rows'enough = .5
16     -far F         far = .9
17     -keep P        max kept items = 512
18     -p P          distance coefficient = 2
19     -seed P        set seed = 10019
20     -todo S        start up action (or 'all') = nothing
21     -help          show help = false
22
23  KEY: N=fileName F=float P=posint S=string
24  ]]
25  local any, asserts, big, cli, fails, firsts, fmt, goalp, ignorep, klassp
26  local lessp, map, main, many, max, min, morep, new, nump, o, oo, per, pop, push
27  local r, rows, slots, sort, sum, thing, things, unpack
28  local CLUSTER, COLS, EGS, NUM, ROWS, SKIP, SOME, SYM = {}, {}, {}, {}, {}, {}, {}
29
30  local thes={}
31  help:gsub("\n [-]([%s]+)[^n]*%s([%s]+)", function(key, x)
32      for n, flag in ipairs(arg) do
33          if flag:sub(1,1)=="-" and key:find("^"..flag:sub(2)..".*") then
34              x = x=="false" and true or arg[n+1] end end
35      if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
36          the[key] = tonumber(x) or x end end )
37
38  for k,v in pairs(thes) do
39      print(k,v,type(v)) end
40  -----
41  -- strings
42  fmt = string.format
43
44  -- maths
45  big = math.huge
46  max = math.max
47  min = math.min
48  r = math.random
49
50  -- column headers
51  function goalp(x) return morep(x) or lessp(x) or klassp(x) end
52  function ignorep(x) return x:find"$" end
53  function klassp(x) return x:find"$" end
54  function lessp(x) return x:find"$" end
55  function morep(x) return x:find"+$" end
56  function nump(x) return x:find"%[A-Z]" end
57
58  -- tables
59  pop = table.remove
60  unpack = table.unpack
61  function any(t) return t[#t] end
62  function firsts(a,b) return a[1] < b[1] end
63  function many(t,n, u) u={}; for i=1,n do push(u,any(t)) end; return u end
64  function per(t,p) return t[ (#t*(p or .5))/1 ] end
65  function push(t,x) table.insert(t,x); return x end
66  function sort(t,f) table.sort(t,f); return t end
67
68  -- meta
69  function map(t,f, u) u={};for k,v in pairs(t) do push(u,f(v)) end; return u end
70  function sum(t,f, n) n=0; for _,v in pairs(t) do n=n+f(v) end; return n end
71  function slots(t, u)
72      u={}
73      for k,v in pairs(t) do k=tostring(k); if k:sub(1,1)=="-" then push(u,k) end end
74      return sort(u) end
75
76  -- print tables, recursively
77  function oo(t) print(o(t)) end
78  function o(t)
79      if type(t)=="table" then return tostring(t) end
80      local key=function(k) return fmt("%s%s",k,o(t[k])) end
81      local u = #t>0 and map(t,o) or map(slots(t),key)
82      return ' {'..table.concat(u, " ").."}' end
83
84  -- strings to things
85  function rows(file, x)
86      file = io.input(file)
87      return function()
88          x=io.read(); if x then return things(x) else io.close(file) end end end
89
90  function thing(x)
91      x = x:match"%s*(-)%s*$"
92      if x=="true" then return true elseif x=="false" then return false end
93      return tonumber(x) or x end
94
95  function things(x,sep, t)
96      t={}
97      for y in x:gmatch(sep or"([^\s]+)") do push(t,thing(y)) end
98      return t end

```

```

99  --
100  -- CLASSES
101  --
102  function new(k,t) k.__index=k; k.__tostring=o; return setmetatable(t,k) end
103
104  -- COLS: turns list of column names into NUMs, SYMs, or SKIPs
105  function COLS.new(k,row, i)
106      i = new(k, {all={}, x={}, y={}, names=row})
107      for at,txt in ipairs(row) do push(i.all, i:col(at,txt)) end
108      return i end
109
110  function COLS.add(i,t)
111      for _,col in pairs(i.all) do col:add( t[col.at] ) end
112      return t end
113
114  function COLS.col(i,at,txt, col)
115      if ignorep(txt) then return SKIP:new(at,txt) end
116      col = (nump(txt) and NUM or SYM):new(at,txt)
117      push(goalp(txt) and i.y or i.x, col)
118      if klassp(txt) then i.klass = col end
119      return col end
120
121  -- NUM: summarizes a stream of numbers
122  function NUM.new(k,n,s)
123      return new(k, {n=0, at=n or 0, txt=s or "", has=SOME:new(), ok=false,
124          w=lessp(s or "") and -1 or 1, lo=big, hi=-big}) end
125
126  function NUM.add(i,x)
127      if x == "" then
128          i.n = i.n + 1
129          if i.has:add(x) then i.ok=false end
130          i.lo, i.hi = min(x, i.lo), max(x, i.hi); end end
131
132  function NUM.dist(i,x,y)
133      if x=="?" and y=="?" then return 1
134      elseif x=="?" then y=i:norm(y); x=y<0.5 and 1 or 0
135      elseif y=="?" then x=i:norm(x); y=x<0.5 and 1 or 0
136      else x,y = i:norm(x), i:norm(y) end
137      return math.abs(x-y) end
138
139  function NUM.mid(i) return per(i:sorted(), .5) end
140
141  function NUM.norm(i,x)
142      return math.abs(i.hi-i.lo)<1E-9 and 0 or (x-i.lo)/(i.hi - i.lo) end
143
144  function NUM.sorted(i)
145      if i.ok==false then table.sort(i.has.all); i.ok=true end
146      return i.has.all end
147
148  -- ROWS: manages 'rows', summarized in 'cols' (columns).
149  function ROWS.new(k,init, i)
150      i = new(k, {rows=SOME:new(), cols=nil})
151      if type(init)=="string" then for row in rows(init) do i:add(row) end end
152      if type(init)=="table" then for row in init do i:add(row) end end
153      return i end
154
155  function ROWS.add(i,row)
156      if i.cols then i.rows:add( i.cols:add(row) )
157      else i.cols = COLS:new(row) end end
158
159  function ROWS.clone(i, j) j= ROWS:new(); j:add(i.cols.names);return j end
160
161  function ROWS.dist(i,row1,row2, d,fun)
162      function fun(col) return col:dist(row1[col.at], row2[col.at])^the.p end
163      return (sum(i.cols.x, fun) / #i.cols.x)^(1/the.p) end
164
165  function ROWS.far(i,row1,rows, fun)
166      function fun(row2) return i:dist(row1,row2), row2 end
167      return unpack(per(sort(map(rows,fun),firsts), the.far)) end
168
169  function ROWS.half(i, top)
170      local some, top,c,x,y,tmp,mid,lefts,rights,_
171      some= many(i.rows.all, the.keep)
172      top = top or i
173      _x = top:far(any(some), some)
174      c,y = top:far(x, some)
175      tmp = sort(map(i.rows.all,function(r) return top:project(r,x,y,c) end),firsts)
176      mid = #i.rows.all//2
177      lefts, rights = i:clone(), i:clone()
178      for at,row in pairs(tmp) do (at <=mid and lefts or rights):add(row[2]) end
179      return lefts,rights,x,y,c, tmp[mid] end
180
181  function ROWS.mid(i,cols)
182      return map(cols or i.cols.all, function(col) return col:mid() end) end
183
184  function ROWS.project(i, r,x,y,c, a,b)
185      a,b = i:dist(r,x), i:dist(r,y); return ((a^2 + c^2 - b^2)/(2*c), r) end
186
187  -- SKIP: summarizes things we want to ignore (so does nothing)
188  function SKIP.new(k,n,s) return new(k, {n=0, at=at or 0, txt=s or ""}) end
189  function SKIP.add(i,x) return x end
190  function SKIP.mid(i) return "?" end
191
192  -- SOME: keeps a random sample on the arriving data
193  function SOME.new(k,keep) return new(k, {n=0, all={}, keep=keep or the.keep}) end
194  function SOME.add(i,x)
195      i.n = i.n+1
196      if #i.all < i.keep then push(i.all,x) ; return i.all
197      elseif r() < i.keep/i.n then i.all[r(#i.all)]=x; return i.all end end
198
199  -- SYM: summarizes a stream of symbols
200  function SYM.new(k,n,s)
201      return new(k, {n=0, at=n or 0, txt=s or "", has={}, most=0}) end
202
203  function SYM.dist(i,x,y) return (x=="?" and y=="?" and 1) or (x==y and 0 or 1) end
204  function SYM.mid(i) return i.mode end
205  function SYM.div(i, fun)
206      function fun(k, p) p = -i.has[k]/i.n; return -p*math.log(p,2) end
207      return sum(i.has, fun) end
208
209  function SYM.add(i,x,inc)
210      if x == "?" then
211          inc = inc or 1
212          i.n = i.n + inc
213          i.has[x] = inc + (i.has[x] or 0)
214          if i.has[x] > i.most then i.most=i.has[x],x end end end
215
216  function SYM.merge(i,j, k)
217      k = SYM:new(i.at,i.txt)
218      for x,n in pairs(i.has) do k:add(x,n) end
219      for x,n in pairs(j.has) do k:add(x,n) end
220      ei, ej, ek = i:div(), j:div(), k:div()
221      if i.n==0 or j.n==0 or .99*ek <= (i.n*ei + j.n*ej)/k.n then
222          return k end end
223
224  225

```

```

225 -- CLUSTER: recursively divides data by clustering towards two distant points
226 function CLUSTER.new(k,sample,top)
227   local i,enough,left,right
228   top = top or sample
229   i = new(k, {here=sample})
230   enough = top.rows.n^the.enough
231   if sample.rows.n >= 2*enough then
232     left, right, i.x, i.y, i.c, i.mid = sample:half(top)
233     if left.rows.n < sample.rows.n then
234       i.left = CLUSTER:new(left, top)
235       i.right = CLUSTER:new(right, top) end end
236   return i end
237
238 function CLUSTER.show(i,pre, here)
239   pre = pre or ""
240   here=""
241   if not i.left and not i.right then here= o(i.here:mid(i.here.cols.y)) end
242   print(fmt("%6s:~30s~%",i.here.rows.n, pre, here))
243   for _,kid in pairs{i.left, i.right} do
244     if kid then kid:show(pre .. "~.") end end end

```

```

245 --
246 -- DEMOS
247 --
248 --
249 fails=0
250 function asserts(test, msg)
251   print(test and "PASS: "or "FAIL: ",msg or "")
252   if not test then
253     fails=fails+1
254     if the.dump then assert(test,msg) end end end
255
256 function EGS.nothing() return true end
257 function EGS.the() oo(the) end
258 function EGS.rand() print(r()) end
259 function EGS.some(s,t)
260   s=SOME:new(100)
261   for i=1,100000 do s:add(i) end
262   for j,x in pairs(sort(s.all)) do
263     if (j % 10)==0 then print("") end
264     io.write(fmt("%6s",x)) end end
265
266 function EGS.clone( r,s)
267   r = ROWS:new(the.data)
268   s = r:clone()
269   for _,row in pairs(r.rows.all) do s:add(row) end
270   asserts(r.cols.x[1].lo==s.cols.x[1].lo,"clone.lo")
271   asserts(r.cols.x[1].hi==s.cols.x[1].hi,"clone.hi")
272   end
273
274 function EGS.data( r)
275   r = ROWS:new(the.data)
276   asserts(r.cols.x[1].hi == 8, "data.columns") end
277
278 function EGS.dist( r,rows,n)
279   r = ROWS:new(the.data)
280   rows = r.rows.all
281   n = NUM:new()
282   for _,row in pairs(rows) do n:add(r:dist(row, rows[1])) end
283   oo(r.cols.x[2]:sorted()) end
284
285 function EGS.many( t)
286   t={}; for j=1,100 do push(t,j) end
287   print(oo(many(t, 10))) end
288
289 function EGS.far( r,c,row1,row2)
290   r = ROWS:new(the.data)
291   row1 = r.rows.all[1]
292   c,row2 = r:far(r.rows.all[1], r.rows.all)
293   print(c,"~n",o(row1),"~n", o(row2)) end
294
295 function EGS.half( r,c,row1,row2)
296   local lefts,rights,x,y,x
297   r = ROWS:new(the.data)
298   oo(r:mid(r.cols.y))
299   lefts,rights,x,y,c = r:half()
300   oo(lefts:mid(lefts.cols.y))
301   oo(rights:mid(rights.cols.y))
302   end
303
304 function EGS.cluster(r)
305   r = ROWS:new(the.data)
306   CLUSTER:new(r):show() end
307
308 -- start-up
309 if arg[0] == "sl.lua" then
310   oo(the)
311   if the.help then print(help) else
312     local b4={}; for k,v in pairs(the) do b4[k]=v end
313     for _,todo in pairs(the.todo=="all" and slots(EGS) or {the.todo}) do
314       for k,v in pairs(b4) do the[k]=v end
315       math.randomseed(the.seed)
316       if type(EGS[todo])=="function" then EGS[todo]() end end
317   end
318   for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
319   OS.exit(fails)
320 else
321   return {CLUSTER=CLUSTER, COLS=COLS, NUM=NUM, ROWS=ROWS,
322     SKIP=SKIP, SOME=SOME, SYM=SYM,the=the,oo=oo,o=o}
323 end
324 -- git rid of SOME for rows
325 -- nss = NUM | SYM | SKIP
326 -- COLS = all:[nss]+, x:[nss]*, y:[nss]*, klass:col?
327 -- ROWS = cols:COLS, rows:SOME

```