

# Functions

```
1 -- - Recursively divide data based on two
2 -- distant points (found in linear time using the Fastmap
3 -- heuristic [Fa95]). Then find and print the attribute range
4 -- that best distinguishes these halves. Recurse on each half.
5 -- (which is sort of like PDDP [Bo98] but faster; and we
6 -- offers a human-readable description for each division).
7 -- - To find those ranges, this code uses a variant of the ChiMerge
8 -- discretizer (but we select on entropy and size,
9 -- not the Chi statistic)
10 -- - To avoid spurious outliers, this code separates using '-furthest=.9';
11 -- i.e. the 90% furthest points.
12 -- - To avoid long runtimes, this code only searches at most '-keep=512'
13 -- randomly selected examples to find those furthest points.
14 -- - To support multi-objective optimization, this code reads csv files
15 -- whose headers may contain markers for "minimize this" or "maximize
16 -- that" (see the 'lessp, morep' functions).
17 -- - To support explanation, optionally, at each level of recursion,
18 -- this code reports what ranges can best distinguish sibling clusters
19 -- C1,C2. The discretizer is inspired by the ChiMerge algorithm:
20 -- numerics are divided into, say, 16 bins. Then, while we can find
21 -- adjacent bins with the similar distributions in C1,C2, then
22 -- (a) merge them (b) look for other merges.
23 local help = {}
24
25 15 == a little lab of lots of LUA learning algorithms.
26 (c) 2022, Tim Menzies, BSD 2-clause license.
27
28 USAGE:
29 lua 15.lua [OPTIONS]
30
31 OPTIONS:
32 -cohen -c F Cohen's delta = .35
33 -data -d N data file = etc/data/auto93.csv
34 -Dump -D stack dump on assert fails = false
35 -furthest -f F far = .9
36 -Format -F S format string = %5.2f
37 -keep -k P max kept items = 512
38 -p -p P distance coefficient = 2
39 -seed -s P set seed = 10019
40 -todo -t S start up action (or 'all') = nothing
41 -help -h show help = false
42 -want -w F recurse until rows>want = .5
43
44 KEY: N=fileName F=float P=posint S=string
45
46 ]]
47
48 -- ## Definitions
49
50 -- ### Cache current names (used at end to find rogue variables)
51 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
52
53 -- ### Define locals.
54 local any, asserts, big, cli, csv, fails, firsts, fmt, goalp, ignorep, klassp
55 local lessp, map, main, many, max, merge, min, morep, new, nump, o, oo, per, pop, push
56 local r, rows, rnd, rnds, slots, sort, sum, thing, things, unpack
57
58 -- ### Define classes
59 local CLUSTER, COLS, EGS, EXPLAIN, NUM, ROWS = {}, {}, {}, {}, {}
60 local SKIP, SOME, SPAN, SYM = {}, {}, {}, {}
61
62 -- ### Define parameter settings.
63 -- Update parameter defaults from command line. Allow for some shorthand:
64 -- e.g. _k N _rArr; 'keep=N';
65 -- and _booleanFlag_ _rArr; 'booleanFlag=not default').
66 local thes={}
67 help:gsub("\n [^-][^%s+](%s)+([^-][^%s+])\n"*(^%s+)", function(key, flag1, x)
68 for n, flag2 in ipairs(arg) do
69 if flag1==flag2 or "-"..key=="flag2" then
70 x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
71 if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
72 the[key] = tonumber(x) or x end end )
73
74 -- ### Define headers for row1 of csv files
75
76 -- Columns to ignore
77 function ignorep(x) return x:find"%" end
78 -- Symbolic classes
79 function klassp(x) return not nump(x) and x:find"%" end
80 -- Goals to minimize
81 function lessp(x) return nump(x) and x:find"-" end
82 -- Goals to maximize
83 function morep(x) return nump(x) and x:find"+" end
84 -- Numeric columns
85 function nump(x) return x:find"%[A-Z]" end
86 -- Dependent attributes
87 function goalp(x) return morep(x) or lessp(x) or klassp(x) end
88
89 ---
90 ---
91 ---
92 ---
93
94 -- ## Misc Utils
95
96 -- ### Strings
97 fmt = string.format
98
99 -- ### Maths
100 big = math.huge
101 max = math.max
102 min = math.min
103 r = math.random
104
105 function rnds(t,f) return map(t, function(x) return rnd(x,f) end) end
106 function rnd(x,f)
107 return fmt(type(x)=="number" and (x~x//1 and f or the.Format) or "%s",x) end
108
109 -- ### Tables
110 pop = table.remove
111 unpack = table.unpack
112 function any(t) return t[#t] < b[1] end
113 function firsts(a,b) u={}; for i=1,n do push(u,any(t)) end; return u end
114 function many(t,n, u) u={}; for i=1,n do push(u,any(t)) end; return u end
115 function per(t,p) return t[ (#t*(p or .5))//1 ] end
116 function push(t,x) table.insert(t,x); return x end
117 function sort(t,f) table.sort(t,f); return t end
118
119 -- ### Meta
120 function map(t,f, u) u={};for k,v in pairs(t) do push(u,f(v)) end; return u end
121 function sum(t,f, n) n=0; for _,v in pairs(t) do n=n+f(v) end; return n end
122 function slots(t, u)
123 u={}
124 for k,v in pairs(t) do k=tostring(k);if k:sub(1,1)~="_" then push(u,k) end end
125 return sort(u) end
126
127 -- ### Print tables, recursively
128 function oo(t) print(o(t)) end
129 function o(t)
130 if type(t)~="table" then return tostring(t) end
131 local key=function(k) return fmt("%.5s",k,o(t[k])) end
132 local u = #t>0 and map(t,o) or map(slots(t),key)
133 return ' ' ..table.concat(u, " ") .."}" end
134
135 -- ### Coerce strings to things
136 function csv(file, x)
137 file = io.input(file)
138 return function()
139 x=io.read(); if x then return things(x) else io.close(file) end end end
140
141 function thing(x)
142 x = x:match"^(%s*)(-)%s*$"
143 if x=="true" then return true elseif x=="false" then return false end
144 return tonumber(x) or x end
145
146 function things(x,sep, t)
147 t={}
148 for y in x:gmatch(sep or"([^\,]+)") do push(t,thing(y)) end
149 return t end
150
151 -- ### Misc stuff
152
153 -- Multi-objectives. Normalized, scored via distance to heaven.
154 function distance2Heaven(t,heaven, num,d)
155 for n,txt in pairs(heaven) do
156 num = Num(at,txt)
157 for _,z in pairs(t) do num:add(z.ys[n]) end
158 for _,z in pairs(t) do z.ys[n] = num:distance2heaven(z.ys[n]) end end
159 d = function(one) return (sum(one.ys)/#one.ys)^.5 end
160 return sort(t, function(a,b) return d(a) < d(b) end) end
161
162 -- While we can find similar adjacent ranges, then merge them.
163 function merge(b4, j,n,now,a,b,merged)
164 j,n,now = 0, #b4, {}
165 while j < #b4 do
166 j = j+1
167 a, b = b4[j], b4[j+1]
168 if b then
169 merged = a:merge(b)
170 if merged then a, j = merged, j+1 end end
171 push(now,a)
172 j = j+1 end
173 return #now == #b4 and b4 or merge(now) end
174
175 -- Objects
176 function new(k,t) k.__index=k; k.__tostring=o; return setmetatable(t,k) end
```

```

177 ---
178 ---
179 ---
180 ---
181 ---
182 ---
183 ---
184 ---
185 ---
186 ---
187 ---
188 ---
189 ---
190 ---
191 ---
192 ---
193 ---
194 ---
195 ---
196 ---
197 ---
198 ---
199 ---
200 ---
201 ---
202 ---
203 ---
204 ---
205 ---
206 ---
207 ---
208 ---
209 ---
210 ---
211 ---
212 ---
213 ---
214 ---
215 ---
216 ---
217 ---
218 ---
219 ---
220 ---
221 ---
222 ---
223 ---
224 ---
225 ---
226 ---
227 ---
228 ---
229 ---
230 ---
231 ---
232 ---
233 ---
234 ---
235 ---
236 ---
237 ---
238 ---
239 ---
240 ---
241 ---
242 ---
243 ---
244 ---
245 ---
246 ---
247 ---
248 ---
249 ---
250 ---
251 ---
252 ---
253 ---
254 ---
255 ---
256 ---
257 ---
258 ---
259 ---
260 ---
261 ---
262 ---
263 ---
264 ---
265 ---
266 ---
267 ---
268 ---
269 ---
270 ---
271 ---
272 ---
273 ---
274 ---
275 ---
276 ---
277 ---
278 ---
279 ---
280 ---
281 ---
282 ---
283 ---
284 ---
285 ---
286 ---
287 ---
288 ---
289 ---
290 ---
291 ---
292 ---
293 ---
294 ---
295 ---
296 ---
297 ---
298 ---
299 ---
300 ---
301 ---
302 ---
303 ---
304 ---
305 ---
306 ---
307 ---
308 ---
309 ---
310 ---
311 ---
312 ---
313 ---
314 ---
315 ---
316 ---
317 ---
318 ---
319 ---
320 ---
321 ---
322 ---
323 ---
324 ---
325 ---
326 ---
327 ---
328 ---
329 ---
330 ---
331 ---
332 ---
333 ---
334 ---
335 ---
336 ---
337 ---
338 ---
339 ---
340 ---
341 ---
342 ---
343 ---
344 ---
345 ---
346 ---
347 ---
348 ---
349 ---
350 ---
351 ---
352 ---
353 ---
354 ---
355 ---
356 ---
357 ---
358 ---
359 ---
360 ---
361 ---
362 ---
363 ---
364 ---
365 ---
366 ---
367 ---
368 ---
369 ---
370 ---
371 ---
372 ---
373 ---
374 ---
375 ---
376 ---
377 ---
378 ---
379 ---
380 ---
381 ---
382 ---
383 ---
384 ---
385 ---
386 ---
387 ---
388 ---
389 ---
390 ---
391 ---
392 ---
393 ---
394 ---
395 ---
396 ---
397 ---
398 ---
399 ---
400 ---
401 ---
402 ---
403 ---
404 ---
405 ---
406 ---
407 ---
408 ---
409 ---
410 ---
411 ---
412 ---
413 ---
414 ---
415 ---
416 ---
417 ---
418 ---
419 ---
420 ---
421 ---
422 ---
423 ---
424 ---
425 ---
426 ---
427 ---
428 ---
429 ---

```

# CLASSES

## COLS

```

-- ## COLS
-- Factory: Turns list of column names into NUMs, SYMs, or SKIPs
function COLS.new(k,row, i,create1)
  createl = function(i,at,txt, col)
    if ignorep(txt) then return SKIP:new(at,txt) end
    col = (nump(txt) and NUM or SYM):new(at,txt)
    push(goalp(txt) and i.y or i.x, col)
    if klassp(txt) then i.klass = col end
    return col
  end
  i = new(k, {all={},x={},y={},names=row})
  for at,txt in ipairs(row) do push(i.all, createl(at,txt)) end
  return i end

function COLS.add(i,t)
  for _,col in pairs(i.all) do col:add( t[col.at] ) end
  return t end

```

## NUM

```

-- NUM: summarizes a stream of numbers
function NUM.new(k,n,s)
  return new(k, {n=0,at=n or 0,txt=s or "",has=SOME:new(),ok=false,
    w=lessp(s or "") and -1 or 1, lo=big, hi=-big}) end

function NUM.add(i,x)
  if x ~= "" then
    i.n = i.n + 1
    if i.has:add(x) then i.ok=false end
    i.lo,i.hi = min(x,i.lo), max(x,i.hi); end end

function NUM.dist(i,x,y)
  if x=="?" and y=="?" then return 1
  elseif x=="?" then y=i:norm(y); x=y<0.5 and 1 or 0
  elseif y=="?" then x=i:norm(x); y=x<0.5 and 1 or 0
  else x,y = i:norm(x), i:norm(y) end
  return math.abs(x-y) end

function NUM.distance2heaven(x, w)
  return ((i.w>0 and 1 or 0) - i:norm(x))^2 end

function NUM.mid(i) return per(i:sorted(), .5) end

function NUM.norm(i,x)
  return math.abs(i.hi-i.lo)<1E-9 and 0 or (x-i.lo)/(i.hi - i.lo) end

function NUM.sorted(i)
  if i.ok==false then table.sort(i.has.all); i.ok=true end
  return i.has.all end

```

## ROWS

```

-- ROWS: manages 'rows', summarized in 'cols' (columns).
function ROWS.new(k,init, i)
  i = new(k, {rows={},cols=nil})
  if type(init)=="string" then for t in csv(init) do i:add(t) end end
  if type(init)=="table" then for t in init do i:add(t) end end
  return i end

function ROWS.add(i,t)
  if i.cols then push(i.rows,i.cols:add(t)) else i.cols=COLS:new(t) end end

function ROWS.clone(i, j) j= ROWS:new(); j:add(i.cols.names);return j end

function ROWS.dist(i,row1,row2, d,fun)
  function fun(col) return col:dist(row1[col.at], row2[col.at])^the.p end
  return (sum(i.cols.x, fun)/ #i.cols.x)^(1/the.p) end

function ROWS.furthest(i,row1,rows, fun)
  function fun(row2) return (i:dist(row1,row2), row2) end
  return unpack(per(sort(map(rows,fun),firsts), the.furthest)) end

function ROWS.half(i, top)
  local some, top,c,x,y,tmp,mid,lefts,rights,_
  some= many(i.rows, the.keep)
  top = top or i
  _,x = top:furthest(any(some), some)
  c,y = top:furthest(x, some)
  tmp = sort(map(i.rows,function(r) return top:fastmap(r,x,y,c) end),firsts)
  mid = #i.rows//2
  lefts, rights = i:clone(), i:clone()
  for at,row in pairs(tmp) do (at < mid and lefts or rights):add(row[2]) end
  return lefts,rights,x,y,c, tmp[mid] end

function ROWS.mid(i,cols)
  return map(cols or i.cols.all, function(col) return col:mid() end) end

function ROWS.fastmap(i, r,x,y,c, a,b)
  a,b = idist(r,x), idist(r,y); return {(a^2 + c^2 - b^2)/(2*c), r} end

```

## SKIP

```

-- SKIP: summarizes things we want to ignore (so does nothing)
function SKIP.new(k,n,s) return new(k, {n=0,at=at or 0,txt=s or ""}) end
function SKIP.add(i,x) return x end
function SKIP.mid(i) return "?" end

```

## SOME

```

-- SOME: keeps a random sample on the arriving data
function SOME.new(k,keep) return new(k, {n=0,all={}, keep=keep or the.keep}) end
function SOME.add(i,x)
  i.n = i.n + 1
  if #i.all < i.keep then push(i.all,x) ; return i.all
  elseif r() < i.keep/i.n then i.all[r(#i.all)]=x; return i.all end end

```

## SYM

```

-- SYM: summarizes a stream of symbols
function SYM.new(k,n,s)
  return new(k, {n=0,at=n or 0,txt=s or "",has={},most=0}) end

function SYM.add(i,x,inc)
  if x == "?" then
    inc = inc or 1
    i.n = i.n + inc
    i.has[x] = inc + (i.has[x] or 0)
    if i.has[x] > i.most then i.most,i.mode=i.has[x],x end end end

function SYM.dist(i,x,y) return (x=="?" and y=="?" and 1) or (x==y and 0 or 1) end
function SYM.mid(i) return i.mode end

```

## CLUSTER

```

-- CLUSTER: recursively divides data by clustering towards two distant points
function CLUSTER.new(k,egs,top)
  local i,want,left,right
  i = new(k, {here=egs})
  top = top or egs
  want = (#top.rows)^the.want
  if #egs.rows >= 2*want then
    left, right, i.x, i.y, i.c, i.mid = egs:half(top)
    if #left.rows < #egs.rows then
      i.left = CLUSTER:new(left, top)
      i.right = CLUSTER:new(right, top) end end
  return i end

function CLUSTER.show(i,pre, here)
  pre = pre or ""
  here=""
  if not i.left and not i.right then here= o(i.here:mid(i.here.cols.y)) end
  print(fmt("%6s: %-30s%", #i.here.rows, pre, here))
  for _,kid in pairs(i.left, i.right) do
    if kid then kid:show(pre .. "|.") end end end

```

## SPAN

```

-- SPAN: keeps a random sample on the arriving data
function SPAN.new(k, col, lo, hi, has)
  return new(k, {col=col,lo=lo,hi=hi or lo,has=has or SYM:new()}) end

function SPAN.add(i,x,y,n) i.lo,i.hi=min(x,i.lo),max(x,i.hi); i.has:add(y,n) end
function SPAN.merge(i,j)
  local has = i.has:merge(j.has)
  if now then return SPAN:new(i.col, i.lo, j.hi, has) end end

function SPAN.select(i,row, x)
  x = row[i.col.at]
  return (x=="?" or (i.lo==i.hi and x==i.lo) or (i.lo <= x and x < i.hi) end

function SPAN.score(i) return (i.has.n/i.col.n, i.has:div()) end

```

## EXPLAIN

```

--- ## EXPLAIN:
function EXPLAIN.new(k,egs,top)
  local i,top,want,left,right,spans,best,yes,no
  i = new(k, {here = egs})
  top = top or egs
  want = (#top.rows)^the.want
  if #top.rows >= 2*want then
    left,right = egs:half(top)
    spans = {}
    for n,col in pairs(i.cols.x) do
      for _,s in pairs(col:spans(j.cols.x[n])) do
        push(spans,{ys=s:score(),its=j}) end end
    best = distance2heaven(spans,{"+", "-"}[1])
    yes,no = egs:clone(), egs:clone()
    for _,row in pairs(egs.rows) do
      (best:selects(row) and yes or no):add(row) end -- divide data in two
    if #yes.rows<#egs.rows then -- make kids if kid size different to parent siz
      if #yes.rows==want then i.yes=EXPLAIN:new(yes,top) end
      if #no.rows >=want then i.no=EXPLAIN:new(no, top) end end end
    return i end

function EXPLAIN.show(i,pre)
  pre = pre or ""
  if not pre then
    tmp = i.here:mid(i.here.y)
    print(fmt("%6s: %-30s%", #i.here.rows, pre, o(i.here:mid(i.here.cols.y))))
    status,kid = unpack(pair)
    k:shpw(pre .. "|.") end end end

```

## SPANS

```

function SYM.spans(i, j)
  local xys,all,one,last,xys,x,c,n = {},{}
  for x,n in pairs(i.has) do push(xys, {x,"this",n}) end
  for x,n in pairs(j.has) do push(xys, {x,"that",n}) end
  for _,tmp in ipairs(sort(xys,firsts)) do
    x,c,n = unpack(tmp)
    if x ~= last then
      last = x
      one = push(all, Span(i,x,x)) end
    one:add(x,y,n) end
  return all end

function NUM.spans(i, j)
  local xys,all,lo,hi,gap,xys,one,x,c,n = {},{}
  lo,hi = min(i.lo, j.lo), max(i.hi,j.hi)
  gap = (hi - lo) / (6/the.cohen)
  for x,n in pairs(i.has) do push(xys, {x,"this",1}) end
  for x,n in pairs(j.has) do push(xys, {x,"that",1}) end
  one = Span:new(i,lo,lo)
  all = {one}
  for _,tmp in ipairs(sort(xys,first)) do
    x,c,n = unpack(tmp)
    if one.hi - one.lo > gap then one = push(all, Span(i, one.hi, x)) end
    one:add(x,y) end
  all = merge(all)
  all[1] .lo = -big
  all[#all].hi = big
  return all end

```

```

430 ---
431 ---
432 ---
433 ---
434 ---
435 ---
436 fails=0
437 function asserts(test, msg)
438   print(test and "PASS: " or "FAIL: ",msg or "")
439   if not test then
440     fails=fails+1
441     if the.dump then assert(test,msg) end end end
442
443 function EGS.nothing() return true end
444 function EGS.the()      oo(the) end
445 function EGS.rand()     print(r()) end
446 function EGS.some(s,t)
447   s=SOME:new(100)
448   for i=1,100000 do s:add(i) end
449   for j,x in pairs(sort(s.all)) do
450     --if (j % 10)==0 then print("") end
451     --io.write(fmt("%6s",x)) end end
452     fmt("%6s",x) end end
453
454 function EGS.clone( r,s)
455   r = ROWS:new(the.data)
456   s = r:clone()
457   for _,row in pairs(r.rows) do s:add(row) end
458   asserts(r.cols.x[1].lo==s.cols.x[1].lo, "clone.lo")
459   asserts(r.cols.x[1].hi==s.cols.x[1].hi, "clone.hi")
460   end
461
462 function EGS.data( r)
463   r = ROWS:new(the.data)
464   asserts(r.cols.x[1].hi == 8, "data.columns") end
465
466 function EGS.dist( r,rows,n)
467   r = ROWS:new(the.data)
468   rows = r:rows
469   n = NUM:new()
470   for _,row in pairs(rows) do n:add(r:dist(row, rows[1])) end
471   --oo(r.cols.x[2]:sorted()) end
472   o(r.cols.x[2]:sorted()) end
473
474 function EGS.many( t)
475   t={} for j=1,100 do push(t,j) end
476   --print(oo(many(t, 10))) end
477   o(many(t, 10)) end
478
479 function EGS.far( r,c,row1,row2)
480   r = ROWS:new(the.data)
481   row1 = r:rows[1]
482   c,row2 = r:far(r:rows[1], r:rows) end
483   --print(c, "\n", o(row1), "\n", o(row2)) end
484
485 function EGS.half( r,c,row1,row2)
486   local lefts, rights,x,y,x
487   r = ROWS:new(the.data)
488   r:mid(r.cols.y)
489   lefts, rights,x,y,c = r:half()
490   lefts:mid(lefts.cols.y)
491   rights:mid(rights.cols.y)
492   asserts(true, "half") end
493
494 function EGS.cluster(r)
495   r = ROWS:new(the.data)
496   --CLUSTER:new(r):show() end
497   CLUSTER:new(r) end
498
499 -- start-up
500 if arg[0] == "slua" then
501   if the.help then print(help:gsub("\nNOTES:$", "")) else
502     local b4={} for k,v in pairs(the) do b4[k]=v end
503     for _,todo in pairs(the.todo=="all" and slots(EGS) or (the.todo)) do
504       for k,v in pairs(b4) do the[k]=v end
505       math.randomseed(the.seed)
506       if type(EGS[todo])=="function" then EGS[todo]() end end
507     end
508     for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
509     os.exit(fails)
510   else
511     return {CLUSTER=CLUSTER, COLS=COLS, NUM=NUM, ROWS=ROWS,
512            SKIP=SKIP, SOME=SOME, SYM=SYM,the=the,oo=oo,o=o}
513   end
514   -- git rid of SOME for rows
515   -- nss = NUM | SYM | SKIP
516   -- COLS = all:[nss]+, x:[nss]*, y:[nss]*, klass;col?
517   -- ROWS = cols:COLS, rows:SOME
518   --
519   -- [Ah91]: Aha, D.W., Kibler, D. & Albert, M.K. Instance-based learning algoritms. Mach Learn 6, 37&M-^@M-^S66 (1991). https://doi.org/10.1007/BF00153759
520   --

```