

```

1 -----
2 ---
3 ---
4 ---
5 ---
6 ---
7 ---
8 ---
9 ---
10 -----
11 -- Recursively divide data based on two
12 -- distant points (found in linear time using the Fastmap
13 -- heuristic [Fa95]). Then find and print the attribute range
14 -- that best distinguishes these halves. Recurse on each half.
15 -- (which is sort of like PDDP [Bo98] but faster; and we
16 -- offers a human-readable description for each division).
17 -- To find those ranges, this code uses a variant of the ChiMerge
18 -- discretizer (but we select on entropy and size,
19 -- not the Chi statistic)
20 -- To avoid spurious outliers, this code separates using '-furthest=.9';
21 -- i.e. the 90% furthest points.
22 -- To avoid long runtimes, this code only searches at most '-keep=512'
23 -- randomly selected examples to find those furthest points.
24 -- To support multi-objective optimization, this code reads csv files
25 -- whose headers may contain markers for "minimize this" or "maximize
26 -- that" (see the 'lessp, morep' functions).
27 -- To support explanation, optionally, at each level of recursion,
28 -- this code reports what ranges can best distinguish sibling clusters
29 -- C1,C2. The discretizer is inspired by the ChiMerge algorithm:
30 -- numerics are divided into, say, 16 bins. Then, while we can find
31 -- adjacent bins with the similar distributions in C1,C2, then
32 -- (a) merge then (b) look for other merges.
33 local help = {}
34
35 l5 == a little LUA learning library
36 (c) 2022, Tim Menzies, BSD 2-clause license.
37
38 USAGE:
39     lua l5.lua [OPTIONS]
40
41 OPTIONS:
42 -cohen      -c  F  Cohen's delta          = .35
43 -data       -d  N  data file              = ../etc/data/auto93.csv
44 -Dump       -D          stack dump on assert fails = false
45 -furthest   -f  F  far                    = .9
46 -Format     -F  S  format string          = %5.2f
47 -keep       -k  P  max kept items         = 512
48 -p          -p          distance coefficient    = 2
49 -seed       -s  F  set seed              = 10019
50 -todo       -t  S  start up action (or 'all') = nothing
51 -help       -h          show help          = false
52 -want       -w  F  recurse until rows>want   = .5
53
54 KEY: N=fileName F=float P=posint S=string
55 []
56
57 -- ## Definitions
58
59 -- Cache current names (used at end to find rogue variables)
60 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
61
62 -- Define locals.
63 local any, asserts, big, cli, distance2Heaven
64 local fails, firsts, fmt, goalp, ignorep, klassp
65 local lessp, map, main, many, max, merge, min, morep, new, nump, o, oo, per, pop, push
66 local r, rows, rnd, rnds, slots, sort, sum, thing, things, file2things, unpack
67
68 -- Define classes
69 local CLUSTER, COLS, EGS, EXPLAIN, NUM, ROWS = {}, {}, {}, {}, {}
70 local SKIP, SOME, SPAN, SYM = {}, {}, {}, {}
71
72 -- Define parameter settings.
73 -- Update parameter defaults from command line. Allow for some shorthand:
74 -- e.g. _k N_ &Arr; 'keep=N';
75 -- and _booleanFlag_ &Arr; 'booleanFlag=not default').
76 local the={}
77 help:gsub("\n ([-])([^\s+])([%s]+)(-)([^\s+])([^\n]*)s([^\s+])",
78 function(long, key, short, x)
79     for n, flag in ipairs(arg) do
80         if flag==short or flag==long then
81             x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
82         if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
83             the[key] = tonumber(x) or x end end )
84
85 -- ### Define headers for row1 of csv files
86
87 -- Columns to ignore
88 function ignorep(x) return x:find"%" end
89 -- Symbolic class columns.
90 function klassp(x) return not nump(x) and x:find"!$" end
91 -- Goal columns to minimize
92 function lessp(x) return nump(x) and x:find"$" end
93 -- Goal columns to maximize
94 function morep(x) return nump(x) and x:find"+$" end
95 -- Numeric columns
96 function nump(x) return x:find"^[A-Z]" end
97 -- Dependent columns
98 function goalp(x) return morep(x) or lessp(x) or klassp(x) end

```

```

99 ---
100 ---
101 ---
102 ---
103 ---
104 ---
105 -- ## Misc Utils
106
107 -- Strings
108 fmt = string.format
109
110 -- Maths
111 big = math.huge
112 max = math.max
113 min = math.min
114 r = math.random
115
116 function rnds(t,f) return map(t, function(x) return rnd(x,f) end) end
117 function rnd(x,f)
118     return fmt(type(x)=="number" and (x~x//1 and f or the.Format) or "%s",x) end
119
120 -- Tables
121 pop = table.remove
122 unpack = table.unpack
123 function any(t) return t[#t] end
124 function firsts(a,b) return a[1] < b[1] end
125 function many(t,n, u) u={}; for i=1,n do push(u,any(t)) end; return u end
126 function per(t,p) return t[ (#t*(p or .5))//1 ] end
127 function push(t,x) table.insert(t,x); return x end
128 function sort(t,f) table.sort(t,f); return t end
129
130 -- Meta
131 function map(t,f, u) u={}; for k,v in pairs(t) do push(u,f(v)) end; return u end
132 function sum(t,f, n) n=0; for _,v in pairs(t) do n=n+f(v) end; return n end
133 function slots(t, u)
134     u={}
135     for k,v in pairs(t) do k=tostring(k); if k:sub(1,1)~="_" then push(u,k) end end
136     return sort(u) end
137
138 -- Print tables, recursively
139 function oo(t) print(o(t)) end
140 function o(t)
141     if type(t)~="table" then return tostring(t) end
142     local key=function(k) return fmt("%s %s",k,o(t[k])) end
143     local u = #t>0 and map(t,o) or map(slots(t),key)
144     return ' '..table.concat(u, " " ).."}" end
145
146 -- Coerce strings to things
147 function thing(x)
148     x = x:match"^(%s*)(-)%s*$"
149     if x=="true" then return true elseif x=="false" then return false end
150     return tonumber(x) or x end
151
152 function things(x,sep, t)
153     t={}; for y in x:gmatch(sep or "([^\s]+)") do push(t,thing(y)) end
154     return t end
155
156 function file2things(file, x)
157     file = io.input(file)
158     return function()
159         x=io.read(); if x then return things(x) else io.close(file) end end end
160
161 -- ### Misc stuff
162
163 -- Multi-objectives. Normalized, scored via distance to heaven.
164 function distance2Heaven(t,heaven, num,d)
165     for n,txt in pairs(heaven) do
166         num = Num(at,txt)
167         for _,z in pairs(t) do num:add(z.ys[n]) end
168         for _,z in pairs(t) do z.ys[n] = num:distance2heaven(z.ys[n]) end end
169         d = function(ony) return (sum(ony.ys)/#ony.ys)^.5 end
170         return sort(t, function(a,b) return d(a) < d(b) end) end
171
172 -- While merges found: merge similar adjacent ranges j and j+1 then jump to j+2.
173 function merge(b4, j,n,now,a,b,merged)
174     print(#b4)
175     j,n,now = 0, #b4, {}
176     while j < #b4 do
177         j = j+1
178         a, b = b4[j], b4[j+1]
179         if b then
180             merged = a:merge(b)
181             if merged then a,j = merged, j+1 end end
182             push(now,a)
183             j = j+1 end
184         return #now == #b4 and b4 or merge(now) end
185
186 -- Objects
187 function new(k,t) k.__index=k; k.__tostring=o; return setmetatable(t,k) end

```

```

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 ---
430 ---
431 ---
432 ---
433 ---
434 ---
435 ---
436 ---
437 ---
438 ---
439 ---
440 ---
441 ---
442 ---
443 ---
444 ---
445 ---

```

CLASSES

COLS

```

--- ## COLS
--- Factory: Turns list of column names into NUMs, SYMs, or SKIPs
function COLS.new(k, row, i, createl)
  createl = function(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, inits, i)
  i = new(k, {rows={}, cols=nil})
  if type(inits)=="table" then for t in inits do i:add(t) end end
  if type(inits)=="string" then for t in file2things(inits) 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[3]) 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, row, left, right, c, a, b, x, y)
  a, b = i:dist(row, left), i:dist(row, right);
  x = (a^2 + c^2 - b^2)/(2*c)
  x = max(0, min(x, 1))
  return {x, (x^2-a^2)^.5, row} 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, p)
  return SYM.div(i, p)
  return -sum(i.has, function(n) p=n/i.n; return p*math.log(p, 2) end) end

function SYM.merge(i, j, k, ei, ej, ek)
  k = SYM:new(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
  ei, ej, ek = i:div(), j:div(), k:div()
  if i.n==0 or j.n==0 or 1.01*ek <= (i.n*ei + j.n*ej)/k.n then
    return k end end

function CLUSTER
  CLUSTER: recursively divides data by clustering towards two distant points
  function CLUSTER.new(k, eggs, top)
    local i, want, left, right
    i = new(k, {here=eggs})
    top = top or eggs
    want = (#top.rows)*the.want
    if #eggs.rows >= 2*want then
      left, right, i.x, i.y, i.c, i.mid = eggs:half(top)
      if #left.rows < #eggs.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("%s: %~30s", #i.here.rows, pre, here))
  for _, kid in pairs(i.left, i.right) do
    if kid then kid:show(pre .. "|.") end end end

function 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 or 1) 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, eggs, top)
  local i, top, want, left, right, spans, best, yes, no
  i = new(k, {here = eggs})
  top = top or eggs
  want = (#top.rows)*the.want
  if #top.rows >= 2*want then
    left, right = eggs: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(), it=s}) end end
    best = distance2heaven(spans, {"+", "-"}[1])
    yes, no = eggs:clone(), eggs:clone()
    for row in pairs(eggs.rows) do
      (best:selects(row) and yes or no):add(row) end -- divide data in two
    if yes.rows<#eggs.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("%s: %~30s", #i.here.rows, pre, o(i.here:mid(i.here.cols.y))))
    for _, pair in pairs({true, i.yes}, {false, i.no}) do
      status, kid = unpack(pair)
      k:shpw(pre .. "|.") end end end

function SPANS
  function SYM.spans(i, j)
    local xys, all, one, last, x, y, 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, y, n = unpack(tmp)
      if x == last then
        last = x
        one = push(all, SPAN:new(i, x, x)) end
      one:add(x, y, n) end
    return all end

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

```

```

446 ----
447 ----
448 ----
449 ----
450 ----
451 ----
452 fails=0
453 function asserts(test, msg)
454   print(test and "PASS: " or "FAIL: ",msg or "")
455   if not test then
456     fails=fails+1
457     if the.dump then assert(test,msg) end end end
458
459 function EGS.nothing() return true end
460 function EGS.the()      oo(the) end
461 function EGS.rand()     print(r()) end
462 function EGS.some(s,t)
463   s=SOME:new(100)
464   for i=1,100000 do s:add(i) end
465   asserts(100==#s.all,"length")
466   for j,x in pairs(sort(s.all)) do
467     --if (j % 10)==0 then print("") end
468     --io.write(fmt("%6s",x)) end end
469     fmt("%6s",x) end end
470
471 function EGS.sum( s)
472   print(sum({1,2,3,4,5},function(x) return x*1 end)) end
473
474 function EGS.div( s)
475   s = SYM:new()
476   for _,x in pairs{"a", "a", "a", "a", "b", "b", "c"} do print(x); s:add(x) end
477   print(s:div()) end
478
479 function EGS.clone( r,s)
480   r = ROWS:new(the.data)
481   s = r:clone()
482   for _,row in pairs(r.rows) do s:add(row) end
483   asserts(r.cols.x[1].lo==s.cols.x[1].lo,"clone.lo")
484   asserts(r.cols.x[1].hi==s.cols.x[1].hi,"clone.hi")
485   end
486
487 function EGS.data( r)
488   r = ROWS:new(the.data)
489   asserts(r.cols.x[1].hi == 8, "data.columns") end
490
491 function EGS.dist( r,rows,n)
492   r = ROWS:new(the.data)
493   rows = r.rows
494   n = NUM:new()
495   for _,row in pairs(rows) do n:add(r:dist(row, rows[1])) end
496   oo(rnds(n:sorted()))
497   --oo(r.cols.x[2]:sorted())
498   o(r.cols.x[2]:sorted()) end
499
500 function EGS.many( t)
501   t={};; for j=1,1000 do push(t,j) end
502   --print(oo(many(t, 10))) end
503   oo(many(t, 10)) end
504
505 function EGS.far( r,c,row1,row2)
506   r = ROWS:new(the.data)
507   row1 = r.rows[1]
508   c,row2 = r:far(r.rows[1], r.rows) end
509   --print(c,"\n",o(row1),"\n", o(row2)) end
510
511 function EGS.half( r,c,row1,row2)
512   local lefts,rights,x,y,x
513   r = ROWS:new(the.data)
514   r:mid(r.cols.y)
515   lefts,rights,x,y,c = r:half()
516   lefts:mid(lefts.cols.y)
517   rights:mid(rights.cols.y)
518   asserts(199==#lefts.rows,"left rows")
519   asserts(199==#rights.rows,"right rows")
520   asserts(true,"half") end
521
522 function EGS.cluster(r)
523   r = ROWS:new(the.data)
524   --CLUSTER:new(r):show() end
525   CLUSTER:new(r):show() end
526
527 function EGS.numspan( r,c,row1,row2)
528   local lefts,rights,x,y,x
529   r = ROWS:new(the.data)
530   r:mid(r.cols.y)
531   lefts,rights,x,y,c = r:half()
532   for n,col in pairs(lefts.cols.x) do
533     print("-----")
534     for x,span in pairs(lefts.cols.x[n]:spans(rights.cols.x[n]) or {}) do
535       print(span.col.txt, span.lo,span.hi) end end
536   end
537
538 -- start-up
539 if arg[0] == "S.lua" then
540   if the.help then print(help) else
541     local b4={};; for k,v in pairs(the) do b4[k]=v end
542     for _,todo in pairs(the.todo=="all" and slots(EGS) or {the.todo}) do
543       for k,v in pairs(b4) do the[k]=v end
544       math.randomseed(the.seed)
545       if type(EGS[todo])=="function" then EGS[todo]() end end
546     end
547     for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
548     os.exit(fails)
549   else
550     return {CLUSTER=CLUSTER, COLS=COLS, NUM=NUM, ROWS=ROWS,
551            SKIP=SKIP, SOME=SOME, SYM=SYM,the=the,oo=oo,o=o}
552   end
553
554 -- git rid of SOME for rows
555 -- nss = NUM | SYM | SKIP
556 -- COLS = all:[nss]+, x:[nss]+, y:[nss]+, klass:col?
557 -- ROWS = cols:COLS, rows:SOME
558
559 -- [Ah91]: Aha, D.W., Kibler, D. & Albert, M.K. Instance-based learning algori
560 thms. Mach Learn 6, 37&M-^@M-^S66 (1991). https://doi.org/10.1007/BF00153759
561 --

```