

```

1  -- <img align=left width=150 src=head.png>
2  --
3  -- **[Repo](https://github.com/timm/lu) &M~@& [Issues](https://github.com/timm
4  -- /lua/issues) &M~@& [copy;2022] (LICENSE.md)** Tim Menzies
5  --
6  -- If we choose our AI tools not on their complexity, but
7  -- on their understandability, what would they look like?
8  -- To that end, I've been looking back over
9  -- common themes seen in my
10 -- AI graduate students (30+ students, over 20 years). What I was
11 -- after were the least lines of code that offer the most
12 -- AI functionality-- and which could be mixed and matched in
13 -- novel and interesting ways.
14 --
15 -- The result is this file. My standard "intro to AI" exercise is six
16 -- weeks of homeworks where students rebuild the following code, from
17 -- scratch, in any language they like (except LUA). After that,
18 -- students can review all the assumptions of this code, then read the
19 -- literature looking for other tools that challenge those assumptions.
20 -- That leads to a second a 4-6 week project using these tools as a baseline aga
21 -- inst
22 -- which they can compare other, more complex, approaches.
23 --
24 -- <hr>
25 --
26 -- The need for baselines. XXXX
27 --
28 -- Standard supervised learners assume that all examples have labels.
29 -- When this is not true, then we need tools to incrementally
30 -- (a) summarize what has been seen so far; (b) find and focus
31 -- on the most interesting part of that summary, (c) collect
32 -- more data in that region, then (d) repeat.
33 --
34 -- <a href="div.png"></a>
35 -- To make that search manageable, it is useful to exploit a
36 -- manifold assumption; i.e.
37 -- higher-dimensional data can be approximated in a lower dimensional
38 -- manifold without loss of signal [Ch05,le05].
39 -- Manifolds lead to _continuity_
40 -- effects; i.e. if there are fewer dimensions, then there are more
41 -- similarities between examples.
42 -- Continuity simplifies _clustering_
43 -- (and any subsequent reasoning). More similarities means easier
44 -- clustering. And after clustering, reasoning just means reason about
45 -- a handful of examples (maybe even just one) from each cluster.
46 --
47 -- **ASSIGNMENTS**
48 -- - **Instance selection**: filter the data down to just a few samples per
49 -- cluster, the reason using just those.
50 -- - **Anomaly detection**
51 -- - **Explanation**
52 -- Discretize the numeric ranges (\*) at each level of the recursion,
53 -- then divide the data according what range best selects for one half, or the o
54 -- ther
55 -- at the data at this level of recursion.
56 -- - **Multi-objective optimization**: This code
57 -- can apply Zitzler's multi-objective ranking predicate [Zit04] to prune the
58 -- worst
59 -- half of the data, then recurs on the rest [Ch18]. Assuming a large over-gener
60 -- ation
61 -- of the initial population (to say, 10,000, examples), this can be just as eff
62 -- ective
63 -- as genetic optimization [Ch18], but runs much faster.
64 -- - **Semi-supervised learning**: these applications require only the _2.log(N)
65 -- _ labels at
66 -- of the pair of furthest points seen at each level of recursion.
67 -- - **Privacy**
68 -- - **Planning**
69 -- - **Monitoring**
70 --
71 -- local help = [[
72 --
73 -- 15 == a little lab of lots of LUA learning algorithms.
74 -- (c) 2022, Tim Menzies, BSD 2-clause license.
75 --
76 -- USAGE:
77 -- lua 15.lua [OPTIONS]
78 --
79 -- OPTIONS:
80 -- -cohen F Cohen's delta = .35
81 -- -data N data file = etc/data/auto93.csv
82 -- -Dump stack dump on assert fails = false
83 -- -furthest F far = .9
84 -- -Format S format string = %5.2f
85 -- -keep P max kept items = 512
86 -- -p P distance coefficient = 2
87 -- -seed P set seed = 10019
88 -- -todo S start up action (or 'all') = nothing
89 -- -help show help = false
90 -- -want F recurse until rows^want = .5
91 --
92 -- KEY: N=fileName F=float P=posint S=string
93 --
94 -- NOTES: This code uses Aha's distance measure [Aha91] (that can
95 -- handle numbers and symbols) to recursively divide data based on two
96 -- distant points (these two are found in linear time using the Fastmap
97 -- heuristic [Fa95]).
98 --
99 -- To avoid spurious outliers, this code use the 90% furthest points.
100 --
101 -- To avoid long runtimes, uses a subset of the data to learn where
102 -- to divide data (then all the data gets pushed down first halves).
103 --
104 -- To support explanation, optionally, at each level of recursion,
105 -- this code reports what ranges can best distinguish sibling clusters
106 -- C1,C2. The discretizer is inspired by the ChiMerge algorithm:
107 -- numerics are divided into, say, 16 bins. Then, while we can find
108 -- adjacent bins with the similar distributions in C1,C2, then
109 -- (a) merge then (b) look for other merges.
110 -- ]
111 --
112 -- ## Namespace
113 --
114 -- Cache current globals, use at end to find rogue variables
115 -- local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
116 --
117 -- Defined local names.
118 -- local any, asserts, big, cli, csv, fails, firsts, fmt, goalp, ignorep, klassp
119 -- local lessp, map, main, many, max, merge, min, morep, new, nump, o, oo, per, pop, push
120 -- local r, rows, rnd, rnds, slots, sort, sum, thing, things, unpack
121 --
122 -- Classes have UPPER CASE names.
123 -- local CLUSTER, COLS, EGS, EXPLAIN, NUM, ROWS = {}, {}, {}, {}, {}
124 -- local SKIP, SOME, SPAN, SYM = {}, {}, {}, {}
125 --
126 -- ## Settings
127 -- Parse the help text for flags and defaults (e.g. -keep, 512).
128 -- Check for updates on those details from command line
129 -- (and and there,
130 -- some shortcuts are available;
131 -- e.g. _k N, &rArr; 'keep=N';
132 -- and _booleanFlag, &rArr; 'booleanFlag=not default').
133 -- local the={}
134 -- help:gsub("\n [^\s%+)]*\n" %s{[^\s%+)]", function(key, x)
135 -- for n, flag in ipairs(arg) do
136 -- if flag:sub(1,1)=="-" and key:find("^\s*flag:sub(2)..." then
137 -- x = x=="false" and true or x=="true" and "false" or arg[n+1] end end

```

```

130 if x=="false" then the[key]=false elseif x=="true" then the[key]=true else
131 the[key] = tonumber(x) or x end end )
132
133 -----
134 -- this code reads csv files where the words on line1 define column types.
135 function ignorep(x) return x:find"$" end -- columns to ignore
136 function klassp(x) return x:find"$" end -- symbolic goals to achieve
137 function lessp(x) return x:find"$" end -- number goals to minimize
138 function morep(x) return x:find"$" end -- numeric goals to maximize
139 function nump(x) return x:find"[A-Z]" end -- numeric columns
140 function goalp(x) return morep(x) or lessp(x) or klassp(x) end
141
142 -- strings
143 fmt = string.format
144
145 -- maths
146 big = math.huge
147 max = math.max
148 min = math.min
149 r = math.random
150
151 function rnds(t, f) return map(t, function(x) return rnd(x, f) end) end
152 function rnd(x, f)
153 return fmt(type(x)=="number" and (x~x//1 and f or the.Format) or "%s", x) end
154
155 -- tables
156 pop = table.remove
157 unpack = table.unpack
158 function any(t) return t[#t] end
159 function firsts(a,b) return a[1] < b[1] end
160 function many(t,n, u) u={}; for i=1,n do push(u, any(t)) end; return u end
161 function per(t,p) return t[ (#t*(p or .5))//1 ] end
162 function push(t,x) table.insert(t,x); return x end
163 function sort(t,f) table.sort(t,f); return t end
164
165 -- meta
166 function map(t, f, u) u={}; for k,v in pairs(t) do push(u, f(v)) end; return u end
167 function sum(t, f, n) n=0; for _,v in pairs(t) do n=n+f(v) end; return n end
168 function slots(t, u)
169 u={}
170 for k,v in pairs(t) do k=tostring(k); if k:sub(1,1)=="_" then push(u,k) end end
171 return sort(u) end
172
173 -- print tables, recursively
174 function oo(t) print(o(t)) end
175 function o(t)
176 if type(t)=="table" then return tostring(t) end
177 local key=function(k) return fmt("%s %s", k, o(t[k])) end
178 local u = #t>0 and map(t,o) or map(slots(t), key)
179 return '{ ' .. table.concat(u, " " ) .. " }" end
180
181 -- strings to things
182 function csv(file, x)
183 file = io.input(file)
184 return function()
185 x=io.read(); if x then return things(x) else io.close(file) end end end
186
187 function thing(x)
188 x = x:match"%^%s*(-)%s*$"
189 if x=="true" then return true elseif x=="false" then return false end
190 return tonumber(x) or x end
191
192 function things(x, sep, t)
193 t={}
194 for y in x:gmatch(sep or "([^\s,]+)" do push(t, thing(y)) end
195 return t end

```

```

196 -- misc
197 function distance2Heaven(t,heaven, num,d)
198 for n,txt in pairs(heaven) do
199     num = Num(at,txt)
200     for _,z in pairs(t) do num:add(z.ys[n]) end
201     for _,z in pairs(t) do z.ys[n] = num:distance2Heaven(z.ys[n]) end end
202 d = function(ones) return (sum(ones.ys)/#ones.ys)^.5 end
203 return sort(t, function(a,b) return d(a) < d(b) end) end
204
205 -- CLASSES
206
207 function new(k,t) k.__index=k; k.__tostring=o; return setmetatable(t,k) end
208
209 -- COLS: turns list of column names into NUMs, SYMs, or SKIPs
210 function COLS.new(k,row, i)
211 i = new(k,{all={},x={},y={},names=row})
212 for at,txt in ipairs(row) do push(i.all, i:col(at,txt)) end
213 return i end
214
215 function COLS.add(i,t)
216 for _,col in pairs(i.all) do col:add( t[col.at] ) end
217 return t end
218
219 function COLS.col(i,at,txt, col)
220 if ignorep(txt) then return SKIP:new(at,txt) end
221 col = (nump(txt) and NUM or SYM):new(at,txt)
222 push(goalp(txt) and i.y or i.x, col)
223 if klassp(txt) then i.klass = col end
224 return col end
225
226 -- NUM: summarizes a stream of numbers
227 function NUM.new(k,n,s)
228 return new(k,{n=0,at=n or 0,txt=s or "",has=SOME:new(),ok=false,
229     w=lessp(s or "") and -1 or 1, lo=big, hi=-big}) end
230
231 function NUM.add(i,x)
232 if x == "" then
233     i.n = i.n + 1
234     if i.has:add(x) then i.ok=false end
235     i.lo,i.hi = min(x,i.lo), max(x,i.hi); end end
236
237 function NUM.dist(i,x,y)
238 if x=="?" and y=="?" then return 1
239 elseif x=="?" then y=i:norm(y); x=y<0.5 and 1 or 0
240 elseif y=="?" then x=i:norm(x); y=x<0.5 and 1 or 0
241 else x,y = i:norm(x), i:norm(y) end
242 return math.abs(x-y) end
243
244 function NUM.distance2Heaven(x, w)
245 return ((i.w>0 and 1 or 0) - i:norm(x))^2 end
246
247 function NUM.mid(i) return per(i:sorted(), .5) end
248
249 function NUM.norm(i,x)
250 return math.abs(i.hi-i.lo)<1E-9 and 0 or (x-i.lo)/(i.hi - i.lo) end
251
252 function NUM.sorted(i)
253 if i.ok==false then table.sort(i.has.all); i.ok=true end
254 return i.has.all end
255
256 -- ROWS: manages 'rows', summarized in 'cols' (columns).
257 function ROWS.new(k,init, i)
258 i = new(k,{rows=[],cols=nil})
259 if type(init)=="string" then for t in csv(init) do i:add(t) end end
260 if type(init)=="table" then for t in init do i:add(t) end end
261 return i end
262
263 function ROWS.add(i,t)
264 if i.cols then push(i.rows,i.cols:add(t)) else i.cols=COLS:new(t) end end
265
266 function ROWS.clone(i, j) j = ROWS:new(); j:add(i.cols.names);return j end
267
268 function ROWS.dist(i,row1,row2, d,fun)
269 function fun(col) return col:dist(row1[col.at], row2[col.at])^the.p end
270 return (sum(i.cols.x, fun)/ #i.cols.x)^(1/the.p) end
271
272 function ROWS.furthest(i,row1,rows, fun)
273 function fun(row2) return i:dist(row1,row2), row2 end
274 return unpack(per(sort(map(rows,fun),firsts), the.furthest)) end
275
276 function ROWS.half(i, top)
277 local some, top,c,x,y,tmp,mid,lefts,rights,_
278 some= many(i.rows, the.keep)
279 top = top or i
280 _,x = top:furthest(any(some), some)
281 c,y = top:furthest(x, some)
282 tmp = sort(map(i.rows,function(r) return top:fastmap(r,x,y,c) end),firsts)
283 mid = #i.rows//2
284 lefts, rights = i:clone(), i:clone()
285 for at,row in pairs(tmp) do (at <=mid and lefts or rights):add(row[2]) end
286 return lefts,rights,x,y,c, tmp[mid] end
287
288 function ROWS.mid(i,cols)
289 return map(cols or i.cols.all, function(col) return col:mid() end) end
290
291 function ROWS.fastmap(i, r,x,y,c, a,b)
292 a,b = i:dist(r,x), i:dist(r,y); return ((a^2 + c^2 - b^2)/(2*c), r) end
293
294 -- SKIP: summarizes things we want to ignore (so does nothing)
295 function SKIP.new(k,n,s) return new(k,{n=0,at=at or 0,txt=s or ""}) end
296 function SKIP.add(i,x) return x end
297 function SKIP.mid(i) return "?" end
298
299 -- SOME: keeps a random sample on the arriving data
300 function SOME.new(k,keep) return new(k,{n=0,all={}, keep=keep or the.keep}) end
301 function SOME.add(i,x)
302 i.n = i.n+1
303 if #i.all < i.keep then push(i.all,x) ; return i.all
304 elseif r() < i.keep/i.n then i.all[r(#i.all)]=x; return i.all end end
305
306 -- SYM: summarizes a stream of symbols
307 function SYM.new(k,n,s)
308 return new(k,{n=0,at=n or 0,txt=s or "",has={},most=0}) end
309
310 function SYM.add(i,x,inc)
311 if x == "?" then
312     inc = inc or 1
313     i.n = i.n + inc
314     i.has[x] = inc + (i.has[x] or 0)
315     if i.has[x] > i.most then i.most,i.mode=i.has[x],x end end end
316
317 function SYM.dist(i,x,y) return (x=="?" and y=="?" and 1) or (x==y and 0 or 1) end
318 function SYM.mid(i) return i.mode end
319 function SYM.div(i, p)
320 return sum(i.has,function(k) p=-i.has[k]/i.n;return -p*math.log(p,2) end) end
321
322 function SYM.merge(i,j, k)
323 k = SYM:new(i.at,i.txt)
324 for x,n in pairs(i.has) do k:add(x,n) end
325 for x,n in pairs(j.has) do k:add(x,n) end
326 ei, ej, ejk = i:div(), j:div(), k:div()
327 if i.n==0 or j.n==0 or .99*ek <= (i.n*ei + j.n*ej)/k.n then
328     return k end end
329
330
331 -- CLUSTER
332
333 function CLUSTER:recursively divides data by clustering towards two distant points
334 function CLUSTER.new(k,egs,top)
335 local i,want,left,right
336 i = new(k, {here=egs})
337 top = top or egs
338 want = (#top.rows)^the.want
339 if #egs.rows >= 2*want then
340     left, right, i.x, i.y, i.c, i.mid = egs:half(top)
341     if #left.rows < #egs.rows then
342         i.left = CLUSTER:new(left, top)
343         i.right = CLUSTER:new(right, top) end end
344 return i end
345
346 function CLUSTER.show(i,pre, here)
347 pre = pre or ""
348 here=""
349 if not i.left and not i.right then here= o(i.here:mid(i.here.cols.y)) end
350 print(fmt("%s: %-30s %s", #i.here.rows, pre, here))
351 for _,kid in pairs(i.left, i.right) do
352     if kid then kid:show(pre .. "|.") end end end
353
354 -- EXPLAIN
355
356 function SPAN: keeps a random sample on the arriving data
357 function SPAN.new(k, col, lo, hi, has)
358 return new(k,{col=col,lo=lo,hi=hi or lo,has=has or SYM:new()}) end
359
360 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
361 function SPAN.merge(i,j)
362 local has = i.has:merge(j.has)
363 if now then return SPAN:new(i.col, i.lo, j.hi, has) end end
364
365 function SPAN.select(i,row, x)
366 x = row[i.col.at]
367 return (x=="") or (i.lo==i.hi and x==i.lo) or (i.lo <= x and x < i.hi) end
368
369 function SPAN.score(i) return {i.has.n/i.col.n, i.has:div()} end
370
371 -- EXPLAIN:
372 function EXPLAIN.new(k,egs,top)
373 local i,top,want,left,right,spans,best,yes,no
374 i = new(k, {here = egs})
375 top = top or egs
376 want = (#top.rows)^the.want
377 if #top.rows >= 2*want then
378     left,right = egs:half(top)
379     spans = {}
380     for n,col in pairs(i.cols.x) do
381         for _,s in pairs(col:spans(j.cols.x[n])) do
382             push(spans,{yes=score(i),it=s}) end end
383     best = distance2Heaven(spans,{"", ""})[1]
384     yes,no = egs:clone(), egs:clone()
385     for _,row in pairs(egs.rows) do
386         (best:selects(row) and yes or no):add(row) end -- divide data in two
387     if #yes.rows<#egs.rows then -- make kids if kid size different to parent siz
388         if #yes.rows>=want then i.yes=EXPLAIN:new(yes,top) end
389         if #no.rows >=want then i.no =EXPLAIN:new(no, top) end end end
390 return i end
391
392 function EXPLAIN.show(i,pre)
393 pre = pre or ""
394 if not pre then
395     tmp = i.here:mid(i.here.y)
396     print(fmt("%s: %-30s %s", #i.here.rows, pre, o(i.here:mid(i.here.cols.y))))
397     for _,pair in pairs({true,i.yes},{false,i.no}) do
398         status,kid = unpack(pair)
399         k:shpw(pre .. "|.") end end end
400
401 function SYM.spans(i, j)
402 local xys,all,one,last,xys,x,c n = {},{}
403 for x,n in pairs(i.has) do push(xys, {x,"this",n}) end
404 for x,n in pairs(j.has) do push(xys, {x,"that",n}) end
405 for _,tmp in ipairs(sort(xys,firsts)) do
406     x,c,n = unpack(tmp)
407     if x == last then
408         last = x
409         one = push(all, Span(i,x,x)) end
410     one:add(x,y,n) end
411 return all end
412
413 function NUM.spans(i, j)
414 local xys,all,lo,hi,gap,xys,one,x,c,n = {},{}
415 lo,hi = min(i.lo, j.lo), max(i.hi, j.hi)
416 gap = (hi - lo) / (6/the.cohen)
417 for x,n in pairs(i.has) do push(xys, {x,"this",1}) end
418 for x,n in pairs(j.has) do push(xys, {x,"that",1}) end
419 one = Span:new(i,lo,lo)
420 all = {one}
421 for _,tmp in ipairs(sort(xys,firsts)) do
422     x,c,n = unpack(tmp)
423     if one.hi - one.lo > gap then one = push(all, Span(i, one.hi, x)) end
424     one:add(x,y,n) end
425
426 all[1].j.lo = -big
427 all[#all].hi = big
428 return all end
429
430 function merge(b4, j,n,now,a,b,merged)
431 j,n,now = 0,#b4,{ }
432 while j < #b4 do
433     j = j+1
434     a, b = b4[j], b4[j+1]
435     if b then
436         merged = a:merge(b)
437         if merged then a,j = merged, j+1 end end
438     push(now,a)
439     j = j+1 end
440 return #now == #b4 and b4 or merge(now) end

```

```

446 -- DEMOS
447 --
448 --
449 --
450 fails=0
451 function asserts(test, msg)
452   print(test and "PASS: " or "FAIL: ", msg or "")
453   if not test then
454     fails=fails+1
455     if the.dump then assert(test, msg) end end end
456
457 function EGS.nothing() return true end
458 function EGS.the()   oo(the) end
459 function EGS.rand()  print(r()) end
460 function EGS.some(s, t)
461   s=SOME:new(100)
462   for i=1,100000 do s:add(i) end
463   for j,x in pairs(sort(s.all)) do
464     --if (j % 10)==0 then print("") end
465     --io.write(fmt("%6s", x)) end end
466     fmt("%6s", x) end end
467
468 function EGS.clone( r, s)
469   r = ROWS:new(the.data)
470   s = r:clone()
471   for _,row in pairs(r.rows) do s:add(row) end
472   asserts(r.cols.x[1].lo==s.cols.x[1].lo, "clone.lo")
473   asserts(r.cols.x[1].hi==s.cols.x[1].hi, "clone.hi")
474   end
475
476 function EGS.data( r)
477   r = ROWS:new(the.data)
478   asserts(r.cols.x[1].hi == 8, "data.columns") end
479
480 function EGS.dist( r, rows, n)
481   r = ROWS:new(the.data)
482   rows = r.rows
483   n = NUM:new()
484   for _,row in pairs(rows) do n:add(r:dist(row, rows[1])) end
485   --oo(r.cols.x[2]:sorted()) end
486   o(r.cols.x[2]:sorted()) end
487
488 function EGS.many( t)
489   t={}; for j=1,100 do push(t, j) end
490   --print(oo(many(t, 10))) end
491   o(many(t, 10)) end
492
493 function EGS.far( r, c, row1, row2)
494   r = ROWS:new(the.data)
495   row1 = r.rows[1]
496   c, row2 = r:far(r.rows[1], r.rows) end
497   --print(c, "\n", o(row1), "\n", o(row2)) end
498
499 function EGS.half( r, c, row1, row2)
500   local lefts, rights, x, y, x
501   r = ROWS:new(the.data)
502   r:mid(r.cols.y)
503   lefts, rights, x, y, c = r:half()
504   lefts:mid(lefts.cols.y)
505   rights:mid(rights.cols.y)
506   asserts(true, "half") end
507
508 function EGS.cluster(r)
509   r = ROWS:new(the.data)
510   --CLUSTER:new(r):show() end
511   CLUSTER:new(r) end
512
513 -- start-up
514 if arg[0] == "slua" then
515   if the.help then print(help:gsub("\nNOTES:$", "")) else
516     local b4={}; for k,v in pairs(the) do b4[k]=v end
517     for _,todo in pairs(the.todo=="all" and slots(EGS) or {the.todo}) do
518       for k,v in pairs(b4) do the[k]=v end
519       math.randomseed(the.seed)
520       if type(EGS[todo])=="function" then EGS[todo]() end end
521     end
522     for k,v in pairs(_ENV) do if not b4[k] then print("?", k, type(v)) end end
523     os.exit(fails)
524   else
525     return {CLUSTER=CLUSTER, COLS=COLS, NUM=NUM, ROWS=ROWS,
526            SKIP=SKIP, SOME=SOME, SYM=SYM, the=the, oo=oo, o=o}
527   end
528
529 -- git rid of SOME for rows
530 nss = NUM | SYM | SKIP
531 COLS = all:[nss] +, x:[nss]*, y:[nss]*, klass:col?
532 ROWS = cols:COLS, rows:SOME
533 -- ## References
534 -- [Ah91]:
535 -- Aha, D.W., Kibler, D. & Albert, M.K. Instance-based
536 -- learning algorithms. Mach Learn 6, 37&M-^@M-^S66 (1991).
537 -- https://doi.org/10.1007/BF00153759
538 -- [Boley, 1998]:
539 -- Boley, D., 1998.
540 -- [Principal directions divisive partitioning](https://www-users.cse.umn.edu/~b
541 -- oley/publications/papers/PDDP.pdf)
542 -- Data Mining and Knowledge Discovery, 2(4): 325-344.
543 -- [Ch05]:
544 -- [Semi-Supervised Learning](http://www.molgen.mpg.de/3659531/MITPress--SemiSup
545 -- ervised-Learning)
546 -- (2005) Olivier Chapelle, Bernhard Sch&Auml;kopf, and Alexander Zien (eds).
547 -- MIT Press.
548 -- [Ch18]:
549 -- [Sampling&M-^@M-^] as a Baseline Optimizer for Search-Based Software Engineer
550 -- ing](https://arxiv.org/pdf/1608.07617.pdf),
551 -- Jianfeng Chen; Vivek Nair; Rahul Krishna; Tim Menzies
552 -- IEEE Trans SE, (45)6, 2019
553 -- [Ch22]:
554 -- [Can We Achieve Fairness Using Semi-Supervised Learning?](https://arxiv.org/p
555 -- df/2111.02038.pdf)
556 -- (2022), Joydip Chakraborty, Huy Tu, Suvdeep Majumder, Tim Menzies.
557 -- [Fa195]:
558 -- [Fa195]:
559 -- Christos Faloutsos and King-Ip Lin. 1995. FastMap: a fast algorithm for index
560 -- ing, data-mining and visualization of traditional and multimedia datasets. SIGMO
561 -- D Rec. 24, 2 (May 1995), 163&M-^@M-^S174. DOI:https://doi.org/10.1145/568271.223
562 -- 812
563 -- [Le05]:
564 -- Levina, E., Bickel, P.J.: [Maximum likelihood estimation of intrinsic dimensi
565 -- on](https://www.stat.berkeley.edu/~bickel/mldim.pdf).
566 -- In:
567 -- Advances in neural information processing systems, pp. 777&M-^@M-^S784 (2005)
568 -- [Pl04]:
569 -- Platt, John.
570 -- [FastMap, MetricMap, and Landmark MDS are all Nystrom Algorithms](https://ww
571 -- .microsoft.com/en-us/research/wp-content/uploads/2005/01/nystrom2.pdf)
572 -- AISTATS (2005).
573 -- [Zit04]:
574 -- [Indicator-based selection in multiobjective search](https://link.springer.co
575 -- m/chapter/10.1007/978-3-540-30217-9_84)
576 -- Eckart Zitzler, Simon K&Auml;nzli
577 -- Proc. 8th International Conference on Parallel Problem Solving from Nature (P
578 -- PSN VIII)

```