

```

1 -- vim: filetype=lua ts=2 sw=2 et:
2 -- (c) 2022, Tim Menzies, timm@ieee.org, opensource.org/licenses/Fair
3 -- Usage of the works is permitted provided that this instrument is retained
4 -- with the works, so that any entity that uses the works is notified of this
5 -- instrument.  DISCLAIMER: THE WORKS ARE WITHOUT WARRANTY.
6
7
8
9
10
11
12
13
14 local help= {}
15 tweak: tries three weak learners for multi-objective optimization
16 (c) 2022, Tim Menzies, timm@ieee.org, opensource.org/licenses/Fair
17
18 learner1: n times, discard half the data furthest from best
19 learner2: classify data according to presence of survivors of learner1
20 learner3: run learner1 on best "best" found by learner2
21
22 USAGE:
23 alias twk="lua tweak.lua "
24 twk [OPTIONS]
25
26 OPTIONS:
27 --boot -b size of bootstrap = 512
28 --cohen -c cohen = 35
29 --conf -C statistical confidence = 0.05
30 --cliffs -l cliff's delta = 0.147
31 --K -K manage low class counts = 1
32 --M -M manage low evidence counts = 2
33 --far -F how far to go for far = 95
34 --p -p coefficient on distance = 2
35 --seed -S seed = 10019
36 --some -s sample size for distances = 512
37 --stop -T how far to go for far = 20
38 --min -m size of min space = .5
39 --best -B best percent = .05
40
41 OPTIONS (other):
42 --dump -d dump stack+exit on error = false
43 --file -f file name = ../etc/data/auto93.csv
44 --help -h show help = false
45 --rnd -r rounding numbers = %.5f
46 --go -g start up action = nothing]] --[[
47
48 ABOUT THE CODE:
49 - Settings generated from "help" string
50 - Settings can be updated from the strings seed in flags
51 - Settings stored in the global "the"
52
53 - Layout code in chunks of size 120 lines (max), broken by line-feed
54 - Chunk1=help; Chunk2=utils; Chunk3=objects; Chunk(last)=demos+start-up
55 - Layout lines 80 chars side (max)
56 - So use 2 spaces for "tab"
57 - Do functions as one-liners (if possible)
58 - In order to define code in any order:
59 - Near the top, define all function and object names as "local"
60 - Otherwise, don't use the "local" keyword (too ugly)
61
62 - Minimize use of map (hard to debug)
63 - Object names are short and UPPER CASE
64 - Private object slots (that should not be printed) start with "_".
65 - Constructors need not return constructed instance.
66 - No inheritance (hard to debug)
67 - For code with many parameters, pass in a dictionary with named fields.
68
69 - Tests in the "go" table at end. Reset settings to defaults after each.
70 - Tests check for error conditions using "ok" not "assert".
71 - Command line "-d go" crashed if test "x" fails, shows stack dump.
72 - Command line "-go x" calls test "go(x)".
73 - Command line "-go all" calls all tests.
74 - Command line "-h" shows help
75 - Command line "-S N" sets random seed (so "$SRANDOM" is "full" random)
76
77 - 2nd last line: look for "rogue" globals (there should be none)
78 - Last line: exit to operating system with number of failures seen in tests
79
80 ABOUT THE CLASSES:
81 - "mean", "mode" are generalized to "mid" (i.e. "mid-point")
82 - "standard deviation", "entropy" are generalized to "div" (i.e. "diversity")
83
84 - BIN holds the class labels seen between "lo" and "hi".
85 - EGS (examples) hold many ROWs, summarized in SYMBolic or NUMeric columns.
86 - COLS is a factory for turning column names into NUMs or SYMs.
87 - Numeric names start with upper case
88 - Goal names ending with "-" or "+" get weights -1,1 for minimize,maximize
89 - Non-numeric class names end with "!"
90 - Columns to be skipped have a name ending with "."
91 - Non-skipped columns are divided into COLS.y and COLS.x (for goal and other)
92 - "mid" and "div" for EGS are computed recursively by "middiv" in NUMs, SYMs
93 - distances between two rows is computed recursively via "dist" in NUMs, SYMs
94
95 - ROW1 before ROW2 (i.e. ROW1<ROW2) if its goals dominate (using [CDOM])
96 - ROWs are clustered by FASTMAP
97 - The distance between two ROWs (i.e. ROW1-ROW2) uses [AHA].
98 - To save space, ROWs are made once but can be passed around different EGS.
99 - ROWs have a "data" pointer where it gets "lo,hi" info needed for distances.
100 - For consistency, "data" is set to the first EGS that holds that row.
101
102 REFERENCES:
103 - [AHA] : Aha : doi.org/10.1007/BF00153759
104 - [CDOM] : Zitzler : doi.org/10.1145/1830483.1830578
105 - [FASTMAP] : Faloutsos : doi.org/10.1145/568271.223812 --]]

```

```

106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
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

```

```

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 == c o l s
325
326 function COLS:new(names, it,num,sym,col)
327 self.names, self.x, self.y, self.all = names, {}, {}, {}
328 for pos,txt in pairs(names) do
329 col = push(self.all, {txt:find"^[A-Z]" and NUM or SYM} (pos,txt))
330 if not txt:find"^[A-Z]" then
331 if txt:find"^[A-Z]" then self.klass = col end
332 push(txt:find"^[A-Z]" and self.y or self.x, col) end end end
333
334 == b i n
335
336 function BIN:new(t)
337 self.pos, self.txt = t.pos, t.txt
338 self.lo, self.hi, self.ys = t.lo, t.hi, t.ys or SYM() end
339
340 function BIN:_tostring()
341 local x,lo,hi,big = self.txt, self.lo, self.hi, math.huge
342 if lo == hi then return fmt("%s=%s",x, lo)
343 elseif hi == big then return fmt("%s>=%s",x, lo)
344 elseif lo == -big then return fmt("%s<=%s", x, hi)
345 else return fmt("%s<=%s<=%s",lo,x,hi) end end
346
347 function BIN:select(t)
348 t = t.cells and t.cells or t
349 local x, lo, hi = t[self.pos], self.lo, self.hi
350 return x=="?" or lo == hi and lo == x or lo <= x and x < hi end
351
352 function BIN:of(x) return self.ys.has[x] or 0 end
353
354 == f o w
355
356 function ROW:new(data,t)
357 self.data,self.cells, self.evaluated,self.rank,self.klass = data,t,false,0, false
358
359 function ROW:_sub(other, cols,d,inc)
360 d, cols = 0, self.data.cols.x
361 for _,col in pairs(cols) do
362 inc = col:dist(self.cells[col.pos], other.cells[col.pos])
363 d = d + inc*the.p end
364 return (d / #cols) ^ (1/the.p) end
365
366 function ROW:_lt(other, s1,s2,e,y,a,b)
367 y = self.data.cols.y
368 s1, s2, e = 0, math.exp(1)
369 for _,col in pairs(y) do
370 a = col:norm(self.cells[col.pos])
371 b = col:norm(other.cells[col.pos])
372 s1 = s1 - e*(col.w * (a - b) / #y)
373 s2 = s2 - e*(col.w * (b - a) / #y) end
374 return s1/#y < s2/#y end
375
376 function ROW:around(rows, rowGap)
377 function rowGap(row) return (row=row, gap=self - row) end
378 return sort(map(rows or self.data.rows, rowGap), lt"gap") end
379
380 function ROW:far(rows) return per(self:around(rows), the.far).row end
381
382 == o r g a n i z a t i o n
383
384 function EGS:new() self.rows,self.cols = {},nil end
385 function EGS:load(f) for t in csv(f) do self:add(t); return self end
386 function EGS:mid(t) return map(t or self.cols.y, function(c) return c:mid()end)end
387 function EGS:div(t) return map(t or self.cols.y, function(c) return c:div()end)end
388
389 function EGS:ranks( any, all,first,now,n)
390 for i,row in pairs(sort(self.rows)) do row.rank = (100*1/#self.rows)/1 end end
391
392 function EGS:evaluated(rows, n)
393 n=0;for _,row in pairs(rows or self.rows) do n=n+(row.evaluated and 1 or 0)end
394 return n end
395
396 function EGS:add(t)
397 if self.cols
398 then t = push(self.rows, t.cells and t or ROW(self,t)).cells
399 for _,col in pairs(self.cols.all) do col:add(t[col.pos]) end
400 else self.cols = COLS(t) end
401 return self end
402
403 function EGS:clone(rows, out)
404 out=EGS():add(self.cols.names)
405 for _,row in pairs(rows or {}) do out:add(row) end
406 return out end
407
408 function EGS:sway(rows,x,stop,rest, rxs,some,y,c,best,mid)
409 rows = rows or self.rows
410 stop = stop or 2*the.best*#rows
411 rest = rest or {}
412 if #rows <= stop then return rows,rest,x end
413 some = many(rows,the.some)
414 x = x or any(some):far(some)
415 y = x:far(some)
416 if y < x then x,y = y,x end
417 c = x - y
418 x.evaluated = true
419 y.evaluated = true
420 rxs = map(rows,function(r) return {r=r, x=((r-x)^2+c^2-(r-y)^2)/(2*c)} end)
421 best = {}
422 for i,rx in pairs(sort(rxs, lt"x")) do
423 push(i<#rows*.5 and best or rest, rx.r) end
424 return self:sway(best,x,stop,rest) end
425
426 function EGS:rbins(rows,B,R,how, v,bins,best,bests)
427 function v(bin, b,r)
428 b = bin:of(true) / (B+0.0001)
429 r = bin:of(false) / (R+0.0001)
430 return b>2/(B+R) end
431 how = how or {}
432 bins = {}
433 for _,col in pairs(self.cols.x) do
434 for _,bin in pairs(col:bins(rows)) do push(bins,bin) end end
435 best = sort(bins,function(a,b) return v(a) > v(b) end)[1]
436 bests= map(rows,function(row) if best:select(row) then return row end end)
437 if #bests < #rows
438 then push(how,best)
439 else return rows,how end
440 end*K

```

```

443 == D E M O S
444
445 function ok(test,msg)
446 print("**", test and "PASS" or "FAIL", msg or "")
447 if not test then
448 GO.fail = GO.fail+1
449 if the.dump then assert(test,msg) end end end
450
451 function GO:new(todo, b4,go)
452 b4={}; for k,v in pairs(the) do b4[k]=v end
453 go={}; for k,_ in pairs(GO) do
454 if k=="new" and type(GO[k])=="function" then go[!#go]=k end end
455 GO.fail = 0
456 for _,x in pairs(todo=="all" and sort(go) or {todo}) do
457 for k,v in pairs(b4) do the[k]=v end
458 math.randomseed(the.seed)
459 if GO[x] then print(k); GO[x]() end end end
460
461 function GO:roque( t)
462 t={}; for _,k in pairs( " G", " VERSION", "arg", "assert", "collectgarbage",
463 "coroutine", "debug", "dofile", "error", "getmetatable", "io", "pairs",
464 "load", "loadfile", "math", "next", "os", "package", "pairs", "pcall",
465 "print", "rawequal", "rawget", "rawlen", "rawset", "require", "select",
466 "setmetatable", "string", "table", "tonumber", "tostring", "type", "utf8",
467 "warn", "xpcall" do t[k]=true end
468 for k,v in pairs(_ENV) do if not t[k] then print("?",k, type(v)) end end end
469
470 function GO:the( t) oo(the) end
471 function GO:eg( n,out)
472 out = true
473 n=0; for row in csv(the.file) do
474 n=n+1; out=out and #row==8
475 if n>1 then out=out and type(row[1])=="number" end end
476 ok(out and n==399); end
477
478 function GO:some( s)
479 s=SOME(); for i=1,10^6 do s:add(R(100)) end
480 oo(s:has()) end
481
482 function GO:num( n,s,t)
483 local function sd(t, n,d,m,m2)
484 n,m,m2=0,0,0;for _,x in pairs(t) do n=n+1; d=x-m; m=m+d/n; m2=m2+d*(x-m) end
485 return {n,d,m,m2}
486 for i=1,5 do; print("**")
487 s=2; for r=1,6 do
488 s=s*4
489 n=NUM(); for i=1,s do push(t, n:add(normal(10,2))) end
490 print(fmt("%7.0f %6.2f %6.2f %6.2f",s,n:mid(),n:div(),sd(t))) end end end
491
492 function GO:rows( eggs)
493 eggs=EGS():load(the.file)
494 map(eggs.cols.x,oo); print("**");
495 map(eggs.cols.y,oo) end
496
497 function GO:dist( eggs, a,b,c,out)
498 eggs = EGS():load(the.file)
499 for i=1,100 do
500 out = true
501 a,b,c = any(eggs.rows), any(eggs.rows), any(eggs.rows)
502 out = out and (b-a) == (a-b) and (a-a)=0 and ((a-b)+(b-c)) >= (a-c) end
503 ok(out,"dist") end
504
505 function GO:sort( eggs,rows,n)
506 eggs = sort(EGS():load(the.file))
507 rows= sort(eggs.rows)
508 n = .05*#rows/1
509 print("all", o(rnds(eggs:mid()))))
510 print("best", o(rnds(eggs:clone(slice(rows, 1, n)):mid()))))
511 print("rest", o(rnds(eggs:clone(slice(rows, n+1)):mid()))))
512 end
513
514 function GO:far( eggs,row2)
515 eggs = EGS():load(the.file)
516 row2=eggs:far(eggs.rows[1])
517 print(row2 - eggs.rows[1])end
518
519 function GO:sway( eggs,best,rest)
520 eggs = EGS():load(the.file)
521 best,rest = eggs:sway()
522 eggs:ranks()
523 for _,row in pairs(eggs.rows) do if row.evaluated then oo(row.cells) end end
524 print("all", o(rnds(eggs:mid()))))
525 print("best", o(rnds(eggs:clone(slice(rows, 1, n)):mid()))))
526 print("rest", o(rnds(eggs:clone(slice(rows, n+1)):mid()))))
527 oo(sort(map(best,function(row) return row.rank end))) end
528
529 function GO:symbols( eggs)
530 eggs = EGS():load(the.file)
531 for i,row in pairs(sort(eggs.rows)) do row.klass = i<#eggs.rows//2 end
532 map(eggs.cols.x[4]:rbins(eggs.rows),oo) end
533
534 function GO:bins( eggs)
535 eggs = EGS():load(the.file)
536 for i,row in pairs(sort(eggs.rows)) do row.klass = i<=.05*#eggs.rows end
537 for col in pairs(eggs.cols.x) do
538 print(fmt("%s",col.txt))
539 map(col:bins(eggs.rows),print) end end
540
541 function GO:shuffle( t)
542 t= {10,20,30,40,50,60,70,80,90}
543 for i=1,20 do oo(shuffle(t)) end end
544
545 function GO:per( t)
546 print(per({10,20,30,40},0.00)) end
547
548

```

```

553 function GO:rbins1() GO:rbins(1) end
554 function GO:rbins( max)
555 max = max or 20
556
557 local randoms1={[]}=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()
558 local randoms2={[]}=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()
559 local sways1={[]}=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()
560 local sways2={[]}=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()
561 local guesses={[]}=NUM(), [.25]=NUM(), [.50]=NUM(), [.75]=NUM()
562 local nguesses=NUM()
563 local depths=NUM()
564 local nsways1=NUM()
565 local nsways2=NUM()
566 local nbests1=NUM()
567 local nbests2=NUM()
568 for i=1,max do
569 print("Un=====")
570 local eggs = EGS():load(the.file)
571 print("What", o(map(eggs.cols.y,function(c) return c.txt end)))
572 print("Mid", o(eggs:mid()))
573 eggs.rows = shuffle(eggs.rows) --- <==== Important!!!!
574 local best1,rest1,top = eggs:sway()
575 print("best1", o(eggs:clone(best1):mid()))
576 nsways1:add(eggs:evaluated())
577 local best2,rest2,top = eggs:sway(best1,top,10)
578 print("best2", o(eggs:clone(best2):mid()))
579 nsways2:add(eggs:evaluated())
580 nbests1:add(#best1)
581 local anys1=many(eggs.rows,1*#best2)
582 local anys2=many(eggs.rows,2*#best2)
583 local classes={}
584 for row in pairs(best2) do push(classes,row).klass
585 s=true
586 for _,row in pairs(slice(shuffle(rest1),1,4*#best2)) do push(classes,row).klass
587 s=false end
588 shuffle(classes)
589 local guess,how = eggs:rbins(shuffle(classes),#best2,4*#best2)
590 print("guesses", o(eggs:clone(guess):mid()))
591 nguesses:add(#guess)
592 depths:add(#how)
593 print("**"); map(how,print)
594 eggs:ranks()
595 local anys1 = sort(map(anys1,function(row) return row.rank end))
596 local anys2 = sort(map(anys2,function(row) return row.rank end))
597 local best1 = sort(map(best1,function(row) return row.rank end))
598 local best2 = sort(map(best2,function(row) return row.rank end))
599 local guess = sort(map(guess,function(row) return row.rank end))
600 for i,num in pairs(randoms1) do num:add(per(anys1,i)) end
601 for i,num in pairs(randoms2) do num:add(per(anys2,i)) end
602 for i,num in pairs(sways1) do num:add(per(best1,i)) end
603 for i,num in pairs(sways2) do num:add(per(best2,i)) end
604 for i,num in pairs(guesses) do num:add(per(guess,i)) end
605 end
606 print("s")
607 for _,kv in pairs({(rand1=randoms1),(rand2=randoms2),(sway1=sways1),( sway2=sways2),(guess=guesses)}) do
608 for k,v in pairs(kv) do
609 local t=map({0, .25, .5, .75},function(p) return fmt("%.5f",v[p]:mid()) end
610 )
611 print("s",the.file,"s",k,table.concat(t)) end end
612
613 print("s")
614 print("s",the.file,"nsways1",nsways1:mid())
615 print("s",the.file,"nsways2",nsways2:mid())
616 print("s",the.file,"nbests1",nbests1:mid())
617 print("s",the.file,"nbests2",nbests2:mid())
618 print("s",the.file,"nguesses",nguesses:mid(),"s",nguesses:div())
619 print("s",the.file,"depths",depths:mid(),"s",depths:div())
620 end
621
622 function GO:ranks( eggs)
623 eggs = EGS():load(the.file)
624 eggs:ranks()
625 for _,row in pairs(eggs.rows) do if row.rank>0 then print(row.rank,o(row.cells))
626 end end end
627
628 == S T A R T
629
630 if pcall(debug.getlocal, 4, 1)
631 then return {the.the,any=any,any=csv,fmt=fmt,many=many,map=map,
632 oo=oo,o=o,obj=obj,per=per,push=push,R=R,
633 rnd=rnd,rnds=rnds,sort=sort,slice=slice,
634 string2thing=string2thing,
635 NUM=NUM, SYM=SYM}
636 else if the.help then print(help) else GO(the.go) end
637 GO:roque()
638 os.exit(fails) end

```