

```

1 #!/usr/bin/env lua
2 local b4={}; for k,v in pairs(_ENV) do b4[k]=v end;
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

```

a little lite lua learning library

```

21 local options={
22   what = "Small sample multi-objective optimizer.",
23   usage= "(c) 2021 Tim Menzies <tim@ieee.org> unlicense.org",
24   about= [[
25     Sort N examples on multi-goals using a handful of 'hints'; i.e.
26
27     - Evaluate and rank, a few examples (on their y-values);
28     - Sort other examples by x-distance to the ranked ones;
29     - Recurse on the better half (so we sample more and more
30       from the better half, then quarter, then eighth...).
31
32     A regression tree learner then explores the examples (sorted
33     left to right, worst to best). By finding branches that
34     reduce the variance of the index of those examples, this
35     tree reports what attribute ranges select for the better (or
36     worse) examples.  ]],
37
38   how= {{"file",      "-f",      ".", "/data/auto93.csv",    "read data from file"},
39         {"cull",       "-c",       .5,    "cuts per generation"},
40         {"help",       "-h",       false,  "show help"},
41         {"hints",      "-H",       4,     "hints per generation"},
42         {"p",          "-p",       2,     "distance calc exponent"},
43         {"small",      "-s",       .5,    "div list into 'small'"},
44         {"seed",       "-S",       10019, "random number seed"},
45         {"train",      "-t",       .5,    "size of training set"},
46         {"todo",       "-T",       "all",  "run unit test, or 'all'"},
47         {"trivial",    "-v",       .35,   "small delta-trivial'sd"},
48         {"wild",       "-W",       false,  "run tests, no protection"}},
49
50   local the = {}
51   for _,t in pairs(options.how) do -- update defaults from command line
52     the[t[1]] = t[3]
53   end
54   for n,word in ipairs(arg) do if word==t[2] then
55     local new = t[3] and (tonumber(arg[n+1]) or arg[n+1]) or true
56     assert(type(new) == type(the[t[1]]), word.." expects a"..type(the[t[1]]))
57     the[t[1]] = new end end
58
59   local say = function(...) print(string.format(...)) end
60   if the.help then -- print help text
61     say("un%s [OPTIONS]n%s\n%s\nOPTIONS\n", arg[0], options.usage, options.what)
62     for _,t in pairs(options.how) do
63       say("%%s %-9s%-30s%%s", t[2],t[3] and t[1] or"", t[4],t[3] and"" or"" ,t[3] or"") end
64     print("un"..options.about)
65     os.exit() end
66
67   --[[
68   Spans
69   Little languages:
70     - options
71     - data language
72
73   Lesson plan
74   -- w1: sysyems: github. github workplaces. unit tests. doco tools.
75   -- w2: num,sym
76   -- W3: sample
77   -- w4: eval, knn, unfairnessness
78   -- W5:
79   --]]

```

```

80
81
82
83
84 -- Random stuff
85 local Seed,rand,randi
86 Seed = the.seed or 10019
87 -- random integers
88 function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
89 -- Random floats
90 function rand(lo,hi, mult,mod)
91   lo, hi = lo or 0, hi or 1
92   Seed = (16807 * Seed) % 2147483647
93   return lo + (hi-lo) * Seed / 2147483647 end
94
95 -----
96 -- ## Table Stuff
97 local cat,map,lap,top,keys,last,copy,pop,push
98 local sort,firsts,first,second,shuffle,bchop
99 -- Table to string.
100 cat = table.concat
101 -- Return a sorted table.
102 sort = function(t,f) table.sort(t,f); return t end
103 -- Return first,second, last item.
104 first = function(t) return t[1] end
105 second = function(t) return t[2] end
106 last = function(t) return t[#t] end
107 -- Function for sorting pairs of items.
108 firsts = function(a,b) return first(a) < first(b) end
109 -- Add to end, pull from end.
110 pop = table.remove
111 push = function(t,x) table.insert(t,x); return x end
112
113 -- Random order of items in a list (sort in place).
114 function shuffle(t, j)
115   for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end
116
117 -- Collect values, passed through 'f'.
118 function lap(t,f) return map(t,f,1) end
119 -- Collect key,values, passed through 'f'.
120 -- If 'f' returns two values, store as key,value.
121 -- If 'f' returns one values, store at index value.
122 -- If 'f' return nil then add nothing (so 'map' is also 'select').
123 function map(t,f,one, u)
124   u={}; for x,y in pairs(t) do
125     if one then x,y=f(y) else x,y=f(x,y) end
126     if x ~= nil then
127       if y then u[x]=y else u[1+#u]=x end end end
128   return u end
129
130 -- Shallow copy
131 function copy(t, u) u={}; for k,v in pairs(t) do u[k]=v end; return u end
132
133 function top(t,n, u)
134   u={};for k,v in pairs(t) do if k>n then break end; push(u,v) end; return u,end
135
136 -- Return a table's keys (sorted).
137 function keys(t,u)
138   u={}
139   for k, _ in pairs(t) do if tostring(k):sub(1,1)~="_" then push(u,k) end end
140   return sort(u) end
141
142 -- Binary chop (assumes sorted lists)
143 function bchop(t,val,lt,lo,hi, mid)
144   lt = lt or function(x,y) return x < y end
145   while lo <= hi do
146     mid=(lo+hi) // 2
147     if lt(t[mid],val) then lo=mid+1 else hi= mid-1 end end
148   return math.min(lo,#t) end
149
150 -----
151 -- ## Math Stuff
152 local abs,sum,rnd,rnds
153 abs = math.abs
154 -- Round 'x' to 'd' decimal places.
155 function rnd(x,d, n) n=10^(d or 0); return math.floor(x*n+0.5) / n end
156 -- Round list of items to 'd' decimal places.
157 function rnds(t,d) return lap(t, function(x) return rnd(x,d or 2) end) end
158
159 -- Sum items, filtered through 'f'.
160 function sum(t,f)
161   f= f or function(x) return x end
162   out=0; for _,x in pairs(f) do out= out + f(x) end; return out end
163
164 -----
165 -- ## Printing Stuff
166 local out,shout,red,green,yellow,blue,color,fmt
167 fmt = string.format
168 -- Print as red, green, yellow, blue.
169 function color(s,n) return fmt("%27[1m27[%sm%s27[0m",n,s) end
170 function red(s) return color(s,31) end
171 function green(s) return color(s,32) end
172 function yellow(s) return color(s,34) end
173 function blue(s) return color(s,36) end
174
175 -- Printed string from a nested structure.
176 shout = function(x) print(out(x)) end
177 -- Generate string from a nested structures
178 -- (and don't print any contents more than once).
179 function out(t,seen, u,key,value,public)
180   function key(k) return fmt("%s%%s",blue(k),out(t[k],seen)) end
181   function value(v) return out(v,seen) end
182   if type(t) == "function" then return "(...)" end
183   if type(t) ~= "table" then return tostring(t) end
184   seen = seen or {}
185   if seen[t] then return "..." else seen[t] = t end
186   u = #t>0 and lap(t, value) or lap(keys(t), key)
187   return red((t._is or"").."[")..cat(u," " )..red("]") end
188
189 -----
190 -- ## File i/o Stuff
191 -- Return one table per line, split on commsns.
192 local csv
193 function csv(file, line)
194   file = io.input(file)
195   line = io.read()
196   return function( t,tmp)
197     if line then
198       t={}
199       for cell in line:gsub("[\r\n]", ""):gsub("#.*", ""):gmatch("[^\r\n]+") do
200         push(t, tonumber(cell) or cell) end
201       line = io.read()
202       if #t>0 then return t end
203       else io.close(file) end end end
204
205 -----
206 -- ## OO Stuff
207 local has,objj
208 -- Create an instance
209 function has(mt,x) return setmetatable(x,mt) end
210
211 -- Create a class
212 function objj(s, o,new)
213   o = {__is=s, __tostring=out}
214   o.__index = o
215   return setmetatable(o,{__call = function(_,...) return o.new(...) end}) end
216

```

```

217 --
218
219 --
220
221 -- ## Stuff for tracking 'Num'bers.
222 -- 'Num's track a list of number, and can report it sorted.
223 local Num=obj"Num"
224 function Num.new(inits,at, txt, self)
225 self= has(Num,{at=at or 0, txt=txt or "", w=(txt or ""):find"--" and -1 or 1,
226 has={}, n=0, lo=1E32, hi =1E-32, ready=true))
227 for _,one in pairs(inits or {}) do self:add(one) end
228 return self end
229
230 function Num:add(x)
231 if x>self.hi then self.hi = x
232 elseif x<self.lo then self.lo = x end
233 push(self.has,x); self.n=self.n+1; self.ready=false end
234
235 -- Ensure that the returned list of numbers is sorted.
236 function Num:all(x)
237 if not self.ready then table.sort(self.has) end
238 self.ready = true
239 return self.has end
240
241 function Num:dist(a,b)
242 if a=="?" then b=self:norm(b); a = b>.5 and 0 or 1
243 elseif b=="?" then a=self:norm(a); b = a>.5 and 0 or 1
244 else a,b = self:norm(a), self:norm(b) end
245 return abs(a-b) end
246
247 -- Combine two 'num's.
248 function Num:merge(other, new)
249 new = Num.new(self.has)
250 for _,x in pairs(other.has) do new:add(x) end
251 return new end
252
253 -- Return a merged item if that combination
254 -- is simpler than its parts.
255 function Num:mergeable(other, new,b4)
256 new = self:merge(other)
257 b4 = (self.n*self:sd() + other.n*other:sd()) / new.n
258 if b4 >= new:sd() then return new end end
259
260 -- The 'mid' is the 50th percentile.
261 function Num:mid() return self:per(.5) end
262
263 -- Return 'x' normalized 0..1, lo..hi.
264 function Num:norm(x, lo,hi)
265 if x=="?" then return x end
266 lo,hi = self.lo, self.hi
267 return abs(hi - lo) < 1E-32 and 0 or (x - lo)/(hi - lo) end
268
269 -- Return the 'p'-th percentile number.
270 function Num:per(p, t)
271 t = self:all()
272 p = p*#t/#t
273 return #t<2 and t[1] or t[p < 1 and 1 or p>#t and #t or p] end
274
275 -- The 10th to 90th percentile range is 2.56 times the standard deviation.
276 function Num:sd() return (self:per(.9) - self:per(.1))/ 2.56 end
277
278 -- Create one span holding row indexes associated with each number
279 local div -- defined below
280 function Num:spans(egs)
281 local xys,xs = {}, Num()
282 for pos,eg in pairs(egs) do
283 x = eg[self:at]
284 if x == "?" then
285 xs:add(x)
286 push(xys, {x=x,y=pos}) end end
287 return div(xys, -- split xys into spans...
288 #xs*the.small, -- ..where spans are of size sqrt(#xs)..
289 xs:sd()*the.trivial) end -- ..and spans have (last-first)>trivial
290
291 -----
292 -- ## Stuff for tracking 'Sym'bol Counts.
293 -- 'Sym's track symbol counts and the 'mode' (most frequent symbol).
294 local Sym=obj"Sym"
295 function Sym.new(inits,at,txt, self)
296 self= has(Sym,{at=at or 0, txt=txt or "", has={}, n=0, mode=nil, most=0})
297 for _,one in pairs(inits or {}) do self:add(one) end
298 return self end
299
300 function Sym:add(x)
301 self.n = self.n + 1
302 self.has[x] = 1 + (self.has[x] or 0)
303 if self.has[x] > self.most then self.most, self.mode = self.has[x], x end end
304
305 function Sym:dist(a,b) return a==b and 0 or 1 end
306 function Sym:mid() return self.mode end
307
308 -- Create one span holding row indexes associated with each symbol
309 function Sym:spans(egs, xys,x)
310 xys = {}
311 for pos,eg in pairs(egs) do
312 x = eg[self:at]
313 if x == "?" then
314 xys[x] = xys[x] or {}
315 push(xys[x], pos) end end
316 return map(xys, function(x,t) return {lo=x, hi=x, has=Num(t)} end) end
317
318 -----
319 -- ## Stuff for skipping all things sent to a column
320 local Skip=obj"Skip"
321 function Skip.new(_,at,txt) return has(Skip,{at=at or 0, txt=txt or "", n=0}) end
322 function Skip:add(x) self.n = self.n + 1; return x end
323
324 --
325
326 --
327
328 -- Samples store examples. Samples know about
329 -- (a) lo,hi ranges on the numerics
330 -- and (b) what are independent 'x' or dependent 'y' columns.
331 local Sample = obj"Sample"
332 function Sample.new(src,self)
333 self= has(Sample,{names=nil, all={}, ys={}, xs={}, egs={})
334 if src then
335 if type(src)=="string" then for x in csv(src) do self:add(x) end end
336 if type(src)=="table" then for _,x in pairs(src) do self:add(x) end end end
337 return self end
338
339 function Sample:add(eg, ako,what,where)
340 if not self.names
341 then -- create the column headers
342 self.names = eg
343 for at,x in pairs(eg) do
344 ako = x:find"." and Skip or x:match"^[A-Z]" and Num or Sym
345 what = push(self.all, ako({}), at, x))
346 if not x:find"." then
347 where = (x:find"+" or x:find("-")) and self.ys or self.xs
348 push(where, what) end end
349 else -- store another example; update column headers
350 push(self.egs, eg)
351 for at,x in pairs(eg) do if x ~= "?" then self.all[at]:add(x) end end end
352 return self end
353
354 function Sample:better(eg1,eg2, e,n,a,b,s1,s2)
355 n,s1,s2,e = #self.ys, 0, 0, 2.71828
356 for _,num in pairs(self.ys) do
357 a = num:norm(eg1[num.at])
358 b = num:norm(eg2[num.at])
359 s1 = s1 - e^(num.w * (a-b)/n)
360 s2 = s2 - e^(num.w * (b-a)/n) end
361 return s1/n < s2/n end
362
363 function Sample:betters(egs)
364 return sort(egs or self.egs,function(a,b) return self:better(a,b) end) end
365
366 function Sample:clone( inits,out)
367 out = Sample.new():add(self.names)
368 for _,eg in pairs(inits or {}) do out:add(eg) end
369 return out end
370
371 function Sample:dist(eg1,eg2, a,b,d,n,inc)
372 d,n = 0,0
373 for _,col in pairs(self.xs) do
374 a,b = eg1[col.at], eg2[col.at]
375 inc = a=="?" and b=="?" and 1 or col:dist(a,b)
376 d = d + inc*the.p
377 n = n + 1 end
378 return (d/n)^(1/the.p) end
379
380 -- Report mid of the columns
381 function Sample:mid(cols)
382 return lap(cols or self.ys,function(col) return col:mid() end) end
383
384 -- Return spans of the column that most reduces variance
385 function Sample:splitter(cols)
386 function worker(col) return self:splitter1(col) end
387 return first(sort(lap(cols or sample.xs, worker), firsts))[2] end
388
389 -- Return a column's spans, and the expected sd value of those spans.
390 function Sample:splitter1(col, spans,xpect)
391 spans= cols:spacs(self.egs)
392 xpect= sum(spans, function(_,span) return span.n*span:sd()/#self.egs end)
393 return {xpect, spans} end
394
395 -- Split on column with best span, recurse on each split.
396 function Sample:tree(min, node,min,sub,splitter, splitter1)
397 node = {node=self, kids={}}
398 min = min or (#self.egs)^the.small
399 if #self.egs >= 2*min then
400 for _,span in pairs(self:splitter()) do
401 sub = self:clone()
402 for _,at in pairs(span.has) do sub:add(self.egs[at]) end
403 push(node.kids, span)
404 span.has = sub:tree(min) end end
405 return node end
406
407 -- Find which leaf best matches an example 'eg'.:w
408
409 function Sample:where(tree,eg, max,x,default)
410 if #kid.has==0 then return tree end
411 max = 0
412 for _,kid in pairs(tree.node) do
413 if #kid.has > max then default,max = kid,#kid.has end
414 x = eg[kid.at]
415 if x ~= "?" then
416 if x <= kid.hi and x >= kid.lo then
417 return self:where(kid.has,eg) end end end
418 return self:where(default, eg) end
419
420 -----
421 -- Discretization tricks
422 -- Input a list of {(x,y)..} values. Return spans that divide the 'x' values
423 -- to minimize variance on the 'y' values.
424 function div(xys, tiny, dull)
425 function merge(b4) -- merge adjacent spans if combo simpler to he parts
426 local j, tmp = 0, {}
427 while j < #b4 do
428 j = j + 1
429 local now, after = b4[j], b4[j+1]
430 if after then
431 local simpler = now.has:mergeable(after.has)
432 if simpler then
433 now = {lo=now.lo, hi= after.hi, has=simpler}
434 j = j + 1 end end
435 push(tmp,now) end
436 return #tmp==#b4 and b4 or merge(tmp) -- recurse until nothing merged
437 end
438 local spans,span
439 xys = sort(xys, function(a,b) return a.x < b.x end)
440 span = {lo=xys[1].x, hi=xys[1].x, has=Num({})}
441 spans = {span}
442 for j,xy in pairs(xys) do
443 local x, y = xy.x, xy.y
444 if j < #xys - tiny and -- enough items remaining after split
445 x ~= xys[j+1].x and -- next item is different (so can split here)
446 span.has.n > tiny and -- span has enough items
447 span.hi - span.lo > dull -- span is not trivially small
448 then
449 now = push(spans, {lo=x, hi=x, has=Num({)}) -- then new span
450 end
451 span.hi = x
452 span.has:add(y) end
453 return merge(spans) end
454

```

```
454 -- hints
455 --
456 --
```

```
458 -- Sorting on a few y values
459 local hints={}
460 function hints.default(eg) return eg end
461
462 function hints.sort(sample,scorefun, test,train,egs,scored,small)
463   sample = Sample.new(the.file)
464   train,test = {}, {}
465   for i,eg in pairs(shuffle(sample.egs)) do
466     push(i<= the.train*#sample.egs and train or test, eg) end
467   egs = copy(train)
468   small = (#egs)^the.small
469   local i=0
470   scored = {}
471   while #egs >= small do
472     local tmp = {}
473     i = i + 1
474     io.stderr:write(fmt("%s",string.char(96+i)))
475     for j=1,the.hints do
476       egs[j] = (scorefun or hints.default)(egs[j])
477       push(tmp, push(scored, egs[j]))
478     end
479     egs = hints.ranked(scored,egs,sample)
480     for i=1,the.cull*#egs//1 do pop(egs) end
481   end
482   io.stderr:write("\n")
483   train=hints.ranked(scored, train, sample)
484   return #scored, sample:clone(train), sample:clone(test) end
485
486 function hints.ranked(scored,egs,sample,worker, some)
487   function worker(eg) return (hints.rankOfClosest(scored,eg,sample),eg) end
488   scored = sample:betters(scored)
489   return lap(sort(lap(egs, worker),firsts),second) end
490
491 function hints.rankOfClosest(scored,egl,sample, worker,closest)
492   function worker(rank,eg2) return (sample:dist(egl,eg2),rank) end
493   closest = first(sort(map(scored, worker),firsts))
494   return closest[2] end --+ closest[i]/10^8 end
495
```

```
496 -- d (7 17 5 5)
497 --
```

```
498 local eg={}
499 function eg.shuffle( t)
500   t={}
501   for i=1,100 do push(t,i) end
502   assert(#t == #shuffle(t) and t[1] ~= shuffle(t)[1]) end
503
504 function eg.lap()
505   assert(3==lap({1,2},function(x) return x+1 end)[2]) end
506
507 function eg.map()
508   assert(3==map({1,2},function(_,x) return x+1 end)[2]) end
509
510 function eg.tables()
511   assert(20==sort(shuffle({{10,20},{30,40},{40,50}}),firsts)[1][2]) end
512
513 function eg.csv( n,z)
514   n=0
515   for eg in csv(the.file) do n=n+1; z=eg end
516   assert(n==399 and z[#z]==50) end
517
518 function eg.rnds( t)
519   assert(10.2 == first(rnds({10.22,81.22,22.33},1))) end
520
521 function eg.sym( s)
522   s=Sym{"a","a","a","a","b","b","c"}
523   assert("a"==s.mode) end
524
525 function eg.num1( n)
526   n=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
527   assert(.375 == n:norm(25))
528   assert(15.625 == n:sd()) end
529
530 function eg.num2( n1,n2,n3,n4)
531   n1=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
532   n2=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
533   assert(n1:mergeable(n2)~=nil)
534   n3=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
535   n4=Num{100,200,300,400,500,100,200,300,400,500,100,200,300,400,500}
536   assert(n3:mergeable(n4)==nil) end
537
538 function eg.sample( s,tmp,d1,d2,n)
539   s=Sample(the.file)
540   assert(2110 == last(s.egs)[s.all[3].at])
541   local sortl= s:betters(s.egs)
542   local lo, hi = s:clone(), s:clone()
543   for i=1,20 do lo:add(sortl[i]) end
544   for i=#sortl,#sortl-30,-1 do hi:add(sortl[i]) end
545   shout(s:mid())
546   shout(lo:mid())
547   shout(hi:mid())
548   for m,eg in pairs(sortl) do
549     n = bchop(sortl, eg,function(a,b) return s:better(a,b) end)
550     assert(m-n <=2) end end
551
552 function eg.dists( s,tmp,d1,d2,n)
553   s=Sample(the.file)
554   tmp = sort(lap(shuffle(s.egs),
555     function(eg2) return {s:dist(eg2,s.egs[1]), eg2} end),
556     firsts)
557   d1=s:dist(tmp[1][2], tmp[10][2])
558   d2=s:dist(tmp[1][2], tmp[#tmp][2])
559   assert(d1*10<d2) end
560
561 function eg.binsym( s)
562   s=Sample(the.file)
563   print(s.all[6]._is=="Sym")
564   s:splitter1(s.all[6])
565   end
566
567 function eg.hints( s,_,__,evals,sortl,train,test,n)
568   s=Sample(the.file)
569   evals, train,test = hints.sort(s)
570   test.egs = test:betters()
571   for m,eg in pairs(test.egs) do
572     n = bchop(train.egs, eg,function(a,b) return s:better(a,b) end)
573     print(n) end end
574
575 -----
576 -- startup
577 local fails, defaults = 0, copy(the)
578 local function example(k, f,ok,msg)
579   f= eg[k]
580   assert(f,"unknown action ".k)
581   the = copy(defaults)
582   Seed = the.seed
583   if the.wild then return f() end
584   ok,msg = pcall(f)
585   if ok then print(green("PASS"),k)
586   else print(red("FAIL"), k,msg); fails=fails+1 end end
587
588 -- run one or more examples
589 if the.todo=="all" then lap(keys(eg),example) else example(the.todo) end
590 -- print any rogue global variables
591 for k,v in pairs(_ENV) do if not b4[k] then print("?rogue: ",k,type(v)) end end
592 -- exit, return our test failure count.
593 os.exit(fails)
594
595
```

```
596 --[[
597   +---+
598   |   |
599   +---+
600
601 -- seems to be a revers that i need to do .... but dont
602 -- check if shuffle is working
603
604 teaching:
605 - sample is v.useful
606 --]]
```