

```

1  #!/usr/bin/env lua
2  local b4={}; for k,v in pairs(_ENV) do b4[k]=v end;
3
4  --
5  --
6  -- a little lite
7  --
8  --
9  --
10 --
11 --
12 --
13 --
14 --
15 --
16 --
17 --
18 --
19 --
20 --
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   for n,word in ipairs(arg) do if word==t[2] then
54     local new = t[3] and (tonumber(arg[n+1]) or arg[n+1]) or true
55     assert(type(new) == type(the[t[1]]), word.." expects a "..type(the[t[1]]))
56     the[t[1]] = new end end
57
58 local say = function(...) print(string.format(...)) end
59 if the.help then -- print help text
60   say("un%s [OPTIONS]n%s\n%s\nOPTIONS:n", arg[0], options.usage, options.what)
61   for _,t in pairs(options.how) do
62     say("%%s %-9s%%s%%s", t[2],t[3] and t[1] or"", t[4],t[3] and"" or"" ,t[3] or"" )end
63   print("un"..options.about)
64   os.exit() end
65
66 --[[
67 Spans
68 Little languages:
69   - options
70   - data language
71
72 Lesson plan
73 -- w1: sysyems: github. github workplaces. unit tests. doco tools.
74 -- w2: num,sym
75 -- W3: sample
76 -- w4: eval, knn, unfairnessness
77 -- W5:
78 --]]
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   lo,hi = lo or 1, hi or #t
146   while lo <= hi do
147     mid=(lo+hi) // 2
148     if lt(t[mid],val) then lo=mid+1 else hi= mid-1 end end
149   return math.min(lo,#t) end
150
151 -----
152 -- ## Maths Stuff
153 local abs,sum,rnd,rnds
154 abs = math.abs
155 -- Round 'x' to 'd' decimal places.
156 function rnd(x,d, n) n=10^(d or 0); return math.floor(x*n+0.5) / n end
157 -- Round list of items to 'd' decimal places.
158 function rnds(t,d) return lap(t, function(x) return rnd(x,d or 2) end) end
159
160 -- Sum items, filtered through 'f'.
161 function sum(t,f)
162   f= f or function(x) return x end
163   out=0; for _,x in pairs(f) do out= out + f(x) end; return out end
164
165 -----
166 -- ## Printing Stuff
167 local out,shout,red,green,yellow,blue,color,fmt
168 fmt = string.format
169 -- Print as red, green, yellow, blue.
170 function color(s,n) return fmt("%27[1m27[%sm%s27[0m",n,s) end
171 function red(s) return color(s,31) end
172 function green(s) return color(s,32) end
173 function yellow(s) return color(s,34) end
174 function blue(s) return color(s,36) end
175
176 -- Printed string from a nested structure.
177 shout = function(x) print(out(x)) end
178 -- Generate string from a nested structures
179 -- (and don't print any contents more than once).
180 function out(t,seen, u,key,value,public)
181   function key(k) return fmt("%s%%s",blue(k),out(t[k],seen)) end
182   function value(v) return out(v,seen) end
183   if type(t) == "function" then return "(...)" end
184   if type(t) ~= "table" then return tostring(t) end
185   seen = seen or {}
186   if seen[t] then return "..." else seen[t] = t end
187   u = #t>0 and lap(t, value) or lap(keys(t), key)
188   return red((t._is or "").."[" ..cat(u," " )..red("]") end
189
190 -----
191 -- ## File i/o Stuff
192 -- Return one table per line, split on commsns.
193 local csv
194 function csv(file, line)
195   file = io.input(file)
196   line = io.read()
197   return function( t,tmp)
198     if line then
199       t={}
200       for cell in line:gsub("[\r\n]","",):gsub("#","",):gmatch("([^\r\n]+)") do
201         push(t, tonumber(cell) or cell) end
202       line = io.read()
203       if #t>0 then return t end
204       else io.close(file) end end end
205
206 -----
207 -- ## OO Stuff
208 local has,objj
209 -- Create an instance
210 function has(mt,x) return setmetatable(x,mt) end
211
212 -- Create a class
213 function objj(s, o,new)
214   o = {_is=s, __tostring=out}
215   o._index = o
216   return setmetatable(o,{__call = function(_,...) return o.new(...) end}) end
217

```

```

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.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 --
320 --
321 --
322 -- Samples store examples. Samples know about
323 -- (a) lo,hi ranges on the numerics
324 -- and (b) what are independent 'x' or dependent 'y' columns.
325 local Sample = obj"Sample"
326 function Sample.new(src,self)
327 self= has(Sample,{names=nil, all={}, ys={}, xs={}, egs={}})
328 if src then
329 if type(src)=="string" then for x in csv(src) do self:add(x) end end
330 if type(src)=="table" then for _,x in pairs(src) do self:add(x) end end end
331 return self end
332
333 function Sample:add(eg, name,datum)
334 function name(at,new, weight, where, what)
335 if new:find"." then return end
336 what = (new:match("[A-Z]",x) and Num or Sym())({},at,new)
337 where = (new:find(".*") or new:find("-")) and self.ys or self.xs
338 push(self.all, what)
339 push(where, what)
340 end
341 if not self.names
342 then self.names = eg
343 map(eg, function(at,x) name(at,x) end)
344 else push(self.egs, eg)
345 lap(self.all, function(col) if new=="?" then col:add(eg[col.at]) end end) end
346 return self end
347
348 function Sample:better(eg1,eg2, e,n,a,b,s1,s2)
349 n,s1,s2,e = #self.ys, 0, 0, 2.71828
350 for _,num in pairs(self.ys) do
351 a = num:norm(eg1[num.at])
352 b = num:norm(eg2[num.at])
353 s1 = s1 - e^(num.w * (a-b)/n)
354 s2 = s2 - e^(num.w * (b-a)/n) end
355 return s1/n < s2/n end
356
357 function Sample:betters(egs)
358 return sort(egs or self.egs,function(a,b) return self:better(a,b) end) end
359
360 function Sample:clone( inits,out)
361 out = Sample.new():add(self.names)
362 for _,eg in pairs(inits or {}) do out:add(eg) end
363 return out end
364
365 function Sample:dist(eg1,eg2, a,b,d,n,inc)
366 d,n = 0,0
367 for _,col in pairs(self.xs) do
368 a,b = eg1[col.at], eg2[col.at]
369 inc = a=="?" and b=="?" and 1 or col:dist(a,b)
370 d = d + inc*the.p
371 n = n + 1 end
372 return (d/n)^(1/the.p) end
373
374 -- Report mid of the columns
375 function Sample:mid(cols)
376 return lap(cols or self.ys,function(col) return col:mid() end) end
377
378 -- Return spans of the column that most reduces variance
379 function Sample:splitter(cols)
380 function worker(col) return self:splitter1(col) end
381 return first(sort(lap(cols or sample.xs, worker), firsts))[2] end
382
383 -- Return a column's spans, and the expected sd value of those spans.
384 function Sample:splitter1(col, spans,xpect)
385 spans= cols:spacs(self.egs)
386 xpect= sum(spans, function(_,span) return span.n*span:sd()/#self.egs end)
387 return {xpect, spans} end
388
389 -- Split on column with best span, recurse on each split.
390 function Sample:tree(min, node,min,sub,splitter, splitter1)
391 node = {node=self, kids={}}
392 min = min or (#self.egs)^the.small
393 if #self.egs >= 2*min then
394 for _,span in pairs(self:splitter()) do
395 sub = self:clone()
396 for _,at in pairs(span.has) do sub:add(self.egs[at]) end
397 push(node.kids, span)
398 span.has = sub:tree(min) end end
399 return node end
400
401 -- Find which leaf best matches an example 'eg':.w
402
403 function Sample:where(tree,eg, max,x,default)
404 if #kid.has==0 then return tree end
405 max = 0
406 for _,kid in pairs(tree.node) do
407 if #kid.has > max then default,max = kid,#kid.has end
408 x = eg[kid.at]
409 if x ~= "?" then
410 if x <= kid.hi and x >= kid.lo then
411 return self:where(kid.has,eg) end end end
412 return self:where(default, eg) end
413
414 -----
415 -- Discretization tricks
416 -- Input a list of {(x,y)..} values. Return spans that divide the 'x' values
417 -- to minimize variance on the 'y' values.
418 function div(xys, tiny, dull)
419 function merge(b4) -- merge adjacent spans if combo simpler to he parts
420 local j, tmp = 0, {}
421 while j < #b4 do
422 j = j + 1
423 local now, after = b4[j], b4[j+1]
424 if after then
425 local simpler = now.has:mergeable(after.has)
426 if simpler then
427 now = {lo=now.lo, hi= after.hi, has=simpler}
428 j = j + 1 end end
429 push(tmp,now) end
430 return #tmp==#b4 and b4 or merge(tmp) -- recurse until nothing merged
431 end
432 local spans,span
433 xys = sort(xys, function(a,b) return a.x < b.x end)
434 span = {lo=xys[1].x, hi=xys[1].x, has=Num({})}
435 spans = {span}
436 for j,xy in pairs(xys) do
437 local x,y = xy.x, xy.y
438 if j < #xys - tiny and -- enough items remaining after split
439 x ~= xys[j+1].x and -- next item is different (so can split here)
440 span.has.n > tiny and -- span has enough items
441 span.hi - span.lo > dull -- span is not trivially small
442 then
443 now = push(spans, {lo=x, hi=x, has=Num({)}) -- then new span
444 end
445 span.hi = x
446 span.has:add(y) end
447 return merge(spans) end

```

```
448 -- | i n t i n g
449 --
450 --
```

```
452 -- Sorting on a few y values
453 local hints={}
454 function hints.default(eg) return eg end
455
456 function hints.sort(sample,scorefun, test,train,egs,scored,small)
457   sample = Sample.new(the.file)
458   train,test = {}, {}
459   for i,eg in pairs(shuffle(sample.egs)) do
460     push(i<= the.train*#sample.egs and train or test, eg) end
461   egs = copy(train)
462   small = (#egs)^the.small
463   local i=0
464   scored = {}
465   while #egs >= small do
466     local tmp = {}
467     i = i + 1
468     io.stderr:write(fmt("%s",string.char(96+i)))
469     for j=1,the.hints do
470       egs[j] = (scorefun or hints.default)(egs[j])
471       push(tmp, push(scored, egs[j]))
472     end
473     egs = hints.ranked(scored,egs,sample)
474     for i=1,the.cull*#egs//1 do pop(egs) end
475   end
476   io.stderr:write("\n")
477   train=hints.ranked(scored, train, sample)
478   return #scored, sample:clone(train), sample:clone(test) end
479
480 function hints.ranked(scored,egs,sample,worker, some)
481   function worker(eg) return (hints.rankOfClosest(scored,eg,sample),eg) end
482   scored = sample:betters(scored)
483   return lap(sort(lap(egs, worker),firsts),second) end
484
485 function hints.rankOfClosest(scored,egl,sample, worker,closest)
486   function worker(rank,eg2) return (sample:dist(egl,eg2),rank) end
487   closest = first(sort(map(scored, worker),firsts))
488   return closest[2] end --+ closest[i]/10^8 end
489
```

```
490 -- | r n d s
491 --
492 --
```

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

```
590 --[[
591   +---+
592   |   |
593   +---+
594
595 -- seems to be a revers that i need to do .... but dont
596 -- check if shuffle is working
597
598 teaching:
599 - sample is v.useful
600 --]]
```