


```

1 #!/usr/bin/env lua
2 -- vim: ts=2 sw=2 et:
3 -- (c) 2022, Tim Menzies
4 -- Usage of the works is permitted provided that this instrument is
5 -- retained with the works, so that any entity that uses the works is
6 -- notified of this instrument.  DISCLAIMER: THE WORKS ARE WITHOUT WARRANTY.
7 -----
8 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
9 local help = {}
10
11 gate: explore the world better, explore the world for good.
12 (c) 2022, Tim Menzies
13
14
15      Ba      Bad <----- planning= (better - bad)
16      56      monitor = (bad - better)
17      -----
18      Be      v
19      4      Better
20      -----
21
22 OPTIONS (inference control):
23 -k      int      Bayes: handle rare classes          = 2
24 -m      int      Bayes: handle rare values           = 1
25 -min     real     min size                           = .5
26 -seed    int      random number seed                 = 10019
27 -keep    int      numbers to keep per column         = 512
28 -wait    int      pre-learning, wait a few examples  = 5
29
30 OTHER:
31 -h          show help                                = false
32 -dump       enable stack dump on failures            = false
33 -file       file with data                          = ../etc/data/auto93.csv
34 -rnd        pretty print control for floats          = %5.3f
35 -todo       start-up action                          = the
36
37 EXAMPLES:
38 lua gate.lua -todo list      : list all actions
39 lua gate.lua -todo all      : run all actions
40 ]]
41 -----
42
43 -- define the local names
44 local the,go,no,fails = {}, {}, {}, 0
45 local abs,updates,cli,coerce,copy,csv,demos,ent,fu,fmt,fmt2,gt,inc,log
46 local lt,map,map2,max,merge,merges,min,new,o,ok,obj,oo,ooo,per,push
47 local r,rnd,rnds,sd,settings,slots,sort,sum
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82

```



```

83 -----
84 -- maths
85 r= math.random
86 abs= math.abs
87 log= math.log
88 min= math.min
89 max= math.max
90 function ent(t, n,e)
91   n=0; for _,v in pairs(t) do n=n+v end
92   e=0; for _,v in pairs(t) do e=e-v/n*log(v/n,2) end; return e end
93
94 function per(t,p) return t[ ((p or .5)*#t) // 1 ] end
95
96 function sd(sorted,f, ninety,ten)
97   if #sorted <= 10 then return 0 end
98   ninety,ten = per(sorted, .90), per(sorted, .10)
99   if f then ninety,ten = f(ninety), f(ten) end
100  return (ninety-ten)/2.564 end -- 2*(1.2 + 0.1*(0.9-0.88493)/(0.9032-0.88493))
101
102 -- lists
103 function inc(f,a,n) f=f or {}; f[a]=(f[a] or 0) + (n or 1) return f end
104 function pushings(x) t[1 + #t] = x; return x end
105 function sort(t,f) table.sort(t,f); return t end
106 function map(t,f, u) u={};for _,v in pairs(t)do u[1+#u]=f(v) end;return u end
107 function map2(t,f, u) u={};for k,v in pairs(t)do u[k] = f(k,v) end;return u end
108
109 function copy(t, u)
110   if type(t) ~= "table" then return t end
111   u={};for k,v in pairs(t) do u[copy(k)]=copy(v) end; return u end
112
113 function slots(t, u,public)
114   function public(k) return tostring(k):sub(1,1) ~= "." end
115   u={};for k,v in pairs(t) do if public(k) then u[1+#u]=k end end
116   return sort(u) end
117
118 -- things to strings
119 fmt= string.format
120 fmt2= function(k,v) return fmt("%s%s",k,v) end
121
122 function ooo(t) print( #t>1 and o(t) or oo(t)) end
123 function o(t,s) return "["..table.concat(map(t,tostring),s or",").."]" end
124 function oo(t,sep, slot)
125   function slot(k) return fmt2(k, t[k]) end
126   return (t.is or"")..o(map(slots(t),slot),sep or" ") end
127
128 function rnds(t,f) return map(t, function(x) return rnd(x,f) end) end
129 function rnd(k,f)
130   return fmt(type(x)=="number" and (x~x//1 and f or the.rnd) or"%s",x) end
131
132 -- strings to things
133 function coerce(x)
134   x = x:match("^%s*(-)%s*$")
135   if x=="true" then return true elseif x=="false" then return false end
136   return math.tointeger(x) or tonumber(x) or x end
137
138 function csv(src, things)
139   function things(s, t)
140     t={}; for y in s:gmatch("[^,]+") do t[1+#t]=coerce(y) end; return t end
141   src = io.input(src)
142   return function(x) x=io.read()
143     if x then return things(x) else io.close(src) end end end
144
145 -- misc
146 function fu(x) return function(t) return t[x] end end
147 function lt(x) return function(t,u) return t[x] < u[x] end end
148 function gt(x) return function(t,u) return t[x] > u[x] end end
149
150 function updates(obj,data)
151   if type(data) == "string"
152   then for row in csv(data) do obj:update(row) end
153   else for _,x in pairs(data or {}) do obj:update(x) end end
154   return obj end
155
156 function merge(i,j, k)
157   k = i + j
158   if k:div()*.95 <= (i.n*i:div() + j.n*j:div())/k.n then return k end end
159
160 function merges(b4, a,b,c,j,n,tmp)
161   j,n,tmp = 1,#b4,{ }
162   while j<=n do
163     a, b = b4[j], b4[j+1]
164     if b then
165       c = merge(a,b)
166       if c then a, j = c, j+1 end end
167     tmp[tmp+1] = a
168     j = j+1 end
169   return #tmp==#b4 and tmp or merges(tmp) end
170
171 -- startup, execution, unit tests
172 function settings(t,help)
173   help:gsub("\n -[!%s+)%%s]+[^\n]*%s([^\n]+)",function(k,x) t[k]=coerce(x) end)
174   return t end
175
176 function cli(the, flag)
177   for k,v in pairs(the) do
178     flag="--".k
179     for n,flag1 in ipairs(arg) do
180       if flag1 == flag then
181         v = v==false and"true" or v==true and"false" or arg[n+1]
182         the[k] = coerce(v) end end end
183   if the.h then os.exit(print(help)) else return the end end
184
185 function ok(test,msg)
186   print("", test and "PASS" or "FAIL", msg or "")
187   if not test then
188     fails= fails+1
189     if the.dump then assert(test,msg) end end end
190
191 function demos(the,go, demo1,defaults)
192   function demo1(txt,f)
193     assert(f, fmt("unknown start-up action: %s",txt))
194     the = copy(defaults)
195     math.randomseed(the.seed or 10019)
196     print(txt)
197     f()
198   end
199   defaults = copy(the)
200   if the.todo=="all"
201   then for _,txt in pairs(slots(go)) do
202     go[txt], demo1(txt) end
203   else demo1(the.todo, go[the.todo]) end end
204
205 -- classes
206 function new(klass,...)
207   local obj = setmetatable({},klass)
208   local res = klass.new(obj,...)
209   if res then obj = setmetatable(res,klass) end
210   return obj end
211
212 function obj(name, t)
213   t={__tostring=oo, is=name or""}; t.__index=t
214   return setmetatable(t, {__call=new}) end

```

```

215 -----
216 local Some,Sym,Num,Bin = obj"Some", obj"Sym", obj"Num", obj"Bin"
217 local Cols,Egs,Nb,Abcd = obj"Cols", obj"Egs", obj"Nb", obj"Abcd"
218 -----
219 function Bin:new(at,name, lo,hi,ys)
220   self.at, self.name = at or 0, name or ""
221   self.lo, self.hi, self.ys = lo, hi or lo, ys or Sym() end
222
223 function Bin:__tostring()
224   local x,lo,hi,big = self.name, self.lo, self.hi, math.huge
225   if lo == hi then return fmt("%s==%s",x, lo)
226   elseif hi == big then return fmt("%s<=%s",x, lo)
227   elseif lo == -big then return fmt("%s<%s",x, hi)
228   else return fmt("%s<=%s<%s",lo,x,hi) end end
229
230 function Bin:select(row)
231   local x, lo, hi = row[self.at], self.lo, self.hi
232   return x=="?" or lo == hi and lo == x or lo <= x and x < hi end
233
234 function Bin:update(x,y)
235   if x<self.lo then self.lo = x end
236   if x>self.hi then self.hi = x end
237   self.ys:update(y) end
238
239 function Bin:div() return self.ys:div() end
240
241 function Bin:__add(other)
242   return Bin(self.at, self.name, self.lo, after.hi, self.ys + other.ys) end
243 -----
244 function Sym:new(at,name)
245   self.at, self.name = at or 0, name or ""
246   self.n, self.has, self.mode, self.most = 0, {}, nil, 0 end
247
248 function Sym:update(x,inc)
249   if x ~= "?" then
250     inc = inc or 1
251     self.n = self.n + inc
252     self.has[x] = inc + (self.has[x] or 0)
253     if self.has[x] > self.most then self.most,self.mode = self.has[x],x end end
254   return x end
255
256 function Sym:mid() return self.mode end
257 function Sym:div() return ent(self.has) end
258
259 function Sym:like(x,prior)
260   return (self.has[x] or 0) + the.m*prior/(self.n + the.m) end
261
262 function Sym:__add(other, out)
263   out=Sym(self.at,self.name)
264   for x,n in pairs(self.has) do out:update(x,n) end
265   for x,n in pairs(other.has) do out:update(x,n) end
266   return out end
267
268 function Sym:bins(other)
269   local out = {}
270   local function known(x) out[x] = out[x] or Bin(self.at, self.name, x,x) end
271   for x,n in pairs(self.has) do known(x); out[x].ys:update("left", n) end
272   for x,n in pairs(other.has) do known(x); out[x].ys:update("right", n) end
273   return map(slots(out), function(k) return out[k] end) end
274 -----
275 function Some:new()
276   self.kept, self.ok, self.n = {}, false, 0 end
277
278 function Some:update(x, a)
279   self.n = 1 + self.n
280   a, self.kept = self, {}
281   if #a < the.keep then self.ok=false; push(a,x)
282   elseif r() < the.keep/self.n then self.ok=false; a[r(#a)]=x end end
283
284 function Some:has()
285   if not self.ok then table.sort(self.kept) end
286   self.ok = true
287   return self.kept end
288 -----
289 function Num:new(at,name)
290   self.at, self.name = at or 0, name or ""
291   self.w = self.name:find"$-" and -1 or 1
292   self.some=Some()
293   self.n,self.mu,self.m2,self.sd,self.lo,self.hi = 0,0,0,0,1E32,-1E32 end
294
295 function Num:update(x,_, a,d)
296   if x ~="?" then
297     self.some:update(x)
298     self.n = self.n + 1
299     self.lo = min(x, self.lo)
300     self.hi = max(x, self.hi)
301     d = x - self.mu
302     self.mu = self.mu + d/self.n
303     self.m2 = self.m2 + d*(x - self.mu)
304     self.sd = (self.m2<0 or self.n<2) and 0 or ((self.m2/(self.n - 1))^0.5) end
305   return x end
306
307 function Num:__add(other, out)
308   out=Num(self.at,self.name)
309   for _,x in pairs(self.some.kept) do out:update(x) end
310   for _,x in pairs(other.some.kept) do out:update(x) end
311   return out end
312
313 function Num:mid() return self.mu end
314 function Num:div() return self.sd end
315
316 function Num:like(x,_)
317   local z, e, pi = 1E-64, math.exp(1), math.pi
318   if x < self.mu - 4*self.sd then return 0 end
319   if x > self.mu + 4*self.sd then return 0 end
320   return e^(-(x - self.mu)^2 / (z + 2*self.sd^2))/(z + (pi*2*self.sd^2)^.5) end
321
322 function Num:norm(x, lo,hi)
323   lo,hi = self.lo, self.hi
324   return x=="?" and x or hi-lo < 1E-9 and 0 or (x - lo)/(hi - lo) end
325
326 function Num:bins(other, tmp,out,now,epsilon,minSize)
327   tmp = {}
328   for _,x in pairs(self.some.kept) do push(tmp, {x=x, y="left"}) end
329   for _,x in pairs(other.some.kept) do push(tmp, {x=x, y="right"}) end
330   tmp = sort(tmp,lt"x") -- ascending on x
331   out = {}
332   now = push(out, Bin(self.at, self.name, tmp[1].x))
333   epsilon = sd(tmp,fu"$") * the.cohen
334   minSize = (#tmp)^the.leaves
335   for j,xy in pairs(tmp) do
336     if j > minSize and j + minSize < #tmp then -- leave enough for other bins
337       if now.ys.n > minSize then -- enough in this bins
338         if xy.x ~ tmp[j+1].x then -- there is a break in the data
339           if now.hi - now.lo > epsilon then -- "now" not trivially small
340             now = push(out, Bin(self.at, self.name, now.hi)) end end end end
341           now:update(xy.x, xy.y) end
342           out[j].lo,hi = -math.huge
343           out[#out].hi = math.huge
344           return merges(out) end

```

```

345 function Cols:new(names, col)
346   self.names, self.all, self.x, self.y, self.klass = names, {}, {}, {}, nil
347   for at,name in pairs(names) do
348     col = push(self.all, (name:find"^[A-Z]" and Num or Sym) (at,name))
349     if not name:find"$" then
350       if name:find"$" then self.klass=col end
351       col.indep = not name:find"[-+]$"
352       push(col.indep and self.x or self.y, col) end end end
353 -----
354 function Egs:new() self.rows, self.cols = {},nil end
355
356 function Egs:clone(data)
357   return updates(Egs():update(self.cols.names), data) end
358
359 function Egs:update(row, add)
360   add = function(col) col:update(row[col.at]) end
361   if self.cols
362     then map(self.cols.all,add); push(self.rows, row)
363     else self.cols = Cols(row) end
364   return self end
365
366 function Egs:mid(cols)
367   return map(cols or self.cols.y, function(col) return col:mid() end) end
368
369 function Egs:div(cols)
370   return map(cols or self.cols.y, function(col) return col:div() end) end
371
372 function Egs:like(row,egs,overall, prior,like,col)
373   prior = (#self.rows + the.k) / (overall + the.k * #egs)
374   like = log(prior)
375   for at,x in pairs(row) do
376     col = self.cols.all[at]
377     if x ~="?" and col.indep then like=like + log(col:like(x,prior)) end end
378   return like end
379
380 function Egs:klass(row) return row[self.cols.klass.at] end
381
382 function Egs:better(row1,row2)
383   local s1, s2, n, e = 0, 0, #self.cols.y, math.exp(1)
384   for _,col in pairs(self.cols.y) do
385     local a = col:norm(row1[col.at])
386     local b = col:norm(row2[col.at])
387     s1 = s1 - e^(col.w * (a - b) / n)
388     s2 = s2 - e^(col.w * (b - a) / n) end
389   return s1 / n < s2 / n end
390
391 function Egs:betters()
392   return sort(self.rows, function(a,b) return self.better(a,b) end) end
393 -----
394 function Nb:new()
395   self.all, self.some, self.log = nil, {}, {} end
396
397 function Nb:update(row)
398   if self.all
399     then if #self.all.rows > the.wait then
400         push(self.log, { want = self.all:klass(row),
401         got = self:classify(row) }) end
402     self:train(row)
403     else self.all = Egs():update(row) end end
404
405 function Nb:train(row, k)
406   k = self.all:klass(row)
407   self.some[k] = self.some[k] or self.all:clone()
408   self.some[k]:update(row)
409   self.all:update(row) end
410
411 function Nb:classify(row, most,klass,tmp,out)
412   most = -math.huge
413   for klass,eg in pairs(self.some) do
414     out = out or klass
415     tmp = eg:like(row, self.some, #self.all.rows)
416     if tmp > most then most,out = tmp, klass end end
417   return out,most end
418 -----
419 function Egs:tree(other,min, kids,score)
420   function gain(col1, col2, all, sum,bins)
421     sum = 0
422     bins = col1:bins(col2)
423     map(bins, function(bin)
424       bin.here = self
425       bin.has = {self:clone(),self:clone()}
426       sum = sum + bin.ys.n/all * bin.ys:div() end)
427     return (bins=bins, gain=sum)
428   end
429   n = #self.rows + #other.rows
430   stop = stop or n^the.min
431   if n < stop
432     then return self
433     else cols = map2(self.col.x, function(at,col)
434       return {w=gain(col, other.col.x[at], n), col=col} end)
435     bins = sort(cols,fu"w")[1].bins
436     for at,eg in pairs(self,other) do
437       for _,row in pairs(eg.rows) do
438         for _,bin in pairs(bins) do
439           sub = bin.has[at]
440           if bin:select(row) then sub:update(row); break end end end end
441           self.kids = map(bins,
442           function(bin) bin.kid = bin.has[1]:tree(bin.has[2]) end) end end
443   -- XXXX not done yet. need to return the ocal kids

```

```

445 -----
446 function Abcd:new(data,rx)
447   self.data, self.rx = data or "", rx or ""
448   self.yes, self.no = 0,0
449   self.known, self.a, self.b, self.c, self.d = {}, {}, {}, {}, {} end
450
451 function Abcd:exists(x, new)
452   new = not self.known[x]
453   inc(self.known,x)
454   if new then
455     self.a[x]=self.yes + self.no; self.b[x]=0; self.c[x]=0; self.d[x]=0 end end
456
457 function Abcd:report( p,out,a,b,c,d,pd,pf,pn,f,acc,g,prec)
458   p = function (z) return math.floor(100*z + 0.5) end
459   out= {}
460   for x,xx in pairs( self.known ) do
461     pd,pf,pn,prec,g,f,acc = 0,0,0,0,0,0
462     a= (self.a[x] or 0); b= (self.b[x] or 0);
463     c= (self.c[x] or 0); d= (self.d[x] or 0);
464     if b+d > 0 then pd = d / (b+d) end
465     if a+c > 0 then pf = c / (a+c) end
466     if a+c > 0 then pn = (b+d) / (a+c) end
467     if c+d > 0 then prec = d / (c+d) end
468     if 1-pf+pd > 0 then g=2*(1-pf) * pd / (1-pf+pd) end
469     if prec+pd > 0 then f=2*prec*pd / (prec + pd) end
470     if self.yes + self.no > 0 then
471       acc= self.yes / (self.yes + self.no) end
472     out[x] = {data=self.data,rx=self.rx,num=self.yes+self.no,
473               a=a,b=b,c=c,d=d,acc=p(acc),
474               prec=p(prec), pd=p(pd), pf=p(pf),f=p(f), g=p(g), class=x} end
475   return out end
476
477 function Abcd:pretty(t, s1,s2,d,s,u)
478   print""
479   s1 = "%10s| %10s| %4s| %4s| %4s| %4s"
480   s2 = "%10s| %3s| %3s| %4s| %3s| %3s|"
481   d,s = "----", (s1 .. s2)
482   print(fmt(s,"db","rx","a","b","c","d","acc","pd","pf","prec","f","g"))
483   print(fmt(s,d,d,d,d,d,d,d,d,d,d,d,d))
484   for x,u in pairs(sort(map(t,function(x) return x end),
485                             function(a,b) return (a.c+a.d> b.c+b.d) end)) do
486     print(fmt(s.." %s", u.data,u.rx,u.a, u.b, u.c, u.d,
487               u.acc, u.pd, u.pf, u.prec, u.f, u.g, u.class)) end end
488
489 function Abcd:adds(gotwants, show)
490   for key,one in pairs(gotwants) do
491     self:exists(one.want)
492     self:exists(one.got)
493     if one.want == one.got then self.yes=self.yes+1 else self.no=self.no+1 end
494     for x,xx in pairs(self.known) do
495       if one.want == x
496       then inc(one.want == one.got and self.d or self.b, x)
497       else inc(one.got == x and self.c or self.a, x) end end end
498   return show and self:pretty(self:report()) or self:report() end

```

```

499 -----
500 function go.list()
501   map(slots(go), function(x) print(fmt("lua gate.lua -todo %s",x)) end) end
502
503 function go.the() ooo(the) end
504
505 function go.sort( t)
506   t={10,9,3}
507   ooo(sort(t)) end
508
509 function go.ent() ok(abs(1.3788 - ent{a=4,b=2,c=1}) < 0.001,"enting") end
510
511 function go.ooo() ooo(cc=1,bb={ff=4,dd=5,bb=6}, aa=3) end
512
513 function go.copy( t,u)
514   t = {a=1,b=2,c={d=3,e=4,f={g=5,h=6}}}
515   u = copy(t)
516   t.c.f.g = 100
517   ok(u.c.f.g ~= t.c.f.g, "deep copy") end
518
519 function go.rnds() ooo(rnds{3.421212, 10.1121, 9.1111, 3.44444}) end
520
521 function go.csv( n)
522   n=0; for row in csv(the.file) do n=n+1 end; ok(n==399,"stuff") end
523
524 function go.some( s)
525   the.keep = 64
526   s = Some(); for i=1,10^6 do s:update(i) end
527   ooo(s:has()) end
528
529 function go.num( n,mu,sd)
530   n, mu, sd = Num(), 10, 1
531   for i=1,10^3 do
532     n:update(mu + sd*math.sqrt(-2*math.log(r()))*math.cos(2*math.pi*r())) end
533   ok(abs(n:mid() - mu) < 0.025, "sd")
534   ok(abs(n:div() - sd) < 0.05, "div") end
535
536 function go.updates( n)
537   print(updates(Num(),{1,2,3,4,5}) + updates(Num(),{11,12,13,14,15}))
538   end
539
540 function go.sym( s,mu,sd)
541   s= Sym()
542   for i=1,100 do
543     for k,n in pairs{a=4,b=2,c=1} do s:update(k,n) end end
544   ooo(s:has) end
545
546 function go.egs(f)
547   for _,col in pairs(updates(Egs(),f or "../etc/data/diabetes.csv").cols.all) do
548     print("\n",col) end end
549
550 function go.clone(f, a,b)
551   a = updates(Egs(),f or "../etc/data/diabetes.csv")
552   b = a:clone(a.rows)
553   print(a.cols.x[1].sd)
554   print(b.cols.x[1].sd)
555   ok(a.cols.x[1].sd == b.cols.x[1].sd, "same y") end
556
557 function go.abcd()
558   local t={}
559   for _ = 1,6 do push(t,{want="yes",got="yes"}) end
560   for _ = 1,2 do push(t,{want="no",got="no"}) end
561   for _ = 1,6 do push(t,{want="maybe",got="maybe"}) end
562   for _ = 1,1 do push(t,{want="maybe", got="no"}) end
563   Abcd():adds(t,true) end
564
565 function go.nb(f, nb)
566   nb = updates(Nb(),f or "../etc/data/diabetes.csv")
567   Abcd():adds(nb.log, true) end
568
569 function go.nbsb()
570   go.nb("../etc/data/soybean.csv") end
571
572 the = settings(the,help)
573
574 if pcall(debug.getlocal, 4, 1)
575 then return {Num=Num, Sym=Sym, Egs=Egs} -- called as sub-module. return classes
576 else the = cli(the) -- update 'the' from command line
577   demos(the,go) -- run some demos
578   for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end
579   os.exit(fails) end

```