duo.lua

Contents

For software engineering to knowledge engineering

Ever thought of software engineering as knowledge engineer? Not bracket matching but the creation and stewardship of what humans think is important (as well as methods for how to get it)?

Here by "knowledge" I mean the thing defined by Allen Newell in the classic "Knowedge LEvel" talk (AAAI 1992). To Newell, intelligence was a goal seeking process that (a) identified the goals we want then (b) reflected on what operators were available in order for us to get there then (c) selected the appropriate operators then (d) adapted those operators to the current task. Note that for Newell, how those operators were build was a lover level "symbol-level" issue. These operators could be Python scripts, data mining function, logical axioms, etc. In terms of the knowledge level, the important thing is that we should be able to reflect over them before deciding how to best use them.

In my view this is an engineering process (where engineering means "building a useful project to an acceptable standard given the local constants"). Good knowledge engineers are lazy in that they try to do the most, with least effort, while consuming the fewest resources. Also, I would say that good engineers are stewards, not owners, of their resources and, a good stewart, they try to leave behind the most resources (in the healthiest condition) for those that will come after them.

To do that, good knowledge engineers know their tools, insider and out, since this lets them reconfigure their tools to best suite the current task. $x = b^3$.

`y=f(x)`	no `x`	few `x`	much `x`
no y	model (*)	re	mine
expensive y		explore (+)	
cheap y			mine,otpimize

(*) model

- feature models
- production rules (fuzzing)
- state machines
- bayes nets
- compartmental models
- CNF (or DNF)

(+) explore: - SMO (general term) - TPE (more specific)

`y=f(x)`	no `x` oracle	cheap `x` oracle	expensive `y` oracle
no `y` oracle	stop	cluster & visualize	re(%), ssl (=)
cheap `y` oracle			
expensive `y` oracle			

(%) re = requirements engieering

- show people a few examples, get them to argue it
- do not expect consensus. Instead, collect the (possibly conflicting
- to avoid cognitive overload, only argue the fewest, most informative examples

(=) ssl = semi-supervised learning

- cluster the data
- collect a few (?one) label per cluster
- spread those labels within one cluster
- spread thsoe labels to other clusters

Premable: names in this space

Globals

Trap globals here, so to report rogue globals (at end: see `rogues()`).

```
local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
```

Define our names.

```
local any,asserts,atom,copy,csv,failures,firsts,fmt,go,help
local inc,isa, klass,last,map,new,o,obj,old,push,rand,randi
local rnd,rnds,rogues,seconds,settings,slots,sort,the,xpects
local BAG, COLS, EG, EGS, NB, NUM, RANGE, SYM
```

User Settings

User settings are stored in `the` and derived from the `help` (using `settings()`). These settings can be updated from the command line.

```
local the
help = [[

lua duo.lua [OPTIONS] : data miners using/used by optimizers.
```

```
(c) 2022, Tim Menzies, opensource.org/licenses/MIT
Understands "N" items by peeking at at few (maybe zero) items.

OPTIONS
  -ample  max items in a 'SAMPLE'        = 512
  -bins   max number of bins             = 16
  -Better use 'y' when dividing data     = false
  -Debug  one crash, show stackdump      = false
  -file   file for data                  = ../../data/auto93.csv
  -h      show help                      = false
  -p      coefficient on distance calcs = 2
  -round  print to 'round' decimals      = 2
  -seed   random number seed             = 10019
  -Some   max number items to explore    = 512
  -Tiny   bin size = #t^'Tiny'           = .5
  -todo   start up action ('all'=every) = -]]
```

Library stuff

Misc functions.

OO stuff

Make a new instance by sharing the same metatable.

```
function new(mt,t) return setmetatable(t,mt) end
```

Make a new class using the LUA delegation mechanism. When a field is missing, LUA checks `__index` for any other options. Tables that share that `__index` field all point same methods (i.e. are all members the same class). Similarly, we can share a class name (`_is`); an instance print methods (`o`); and a common instance create protocol (called `klass()` really calls `klass.new(...)`). As a reflection on the power of that delegation mechanism, it is fun to note that this comment is (much) longer than the code itself.

```lua
function klass(s, t)
  t = {_is=s, __tostring=o}
  t.__index = t
  return setmetatable(t,{__call=function(_,...) return t.new(...) end}) end
```

List stuff

```lua
function last(t)       return t[#t] end
function firsts(a,b)   return a[1] < b[1] end -- used for sorting`
function seconds(a,b)  return a[2] < b[2] end -- used for sorting`
function sort(t,f)     table.sort(t,f); return t end
function slots(t,  u)  u={};for k,_ in pairs(t) do u[1+#u]=k end; return u end
function push(t,x)     table.insert(t,x); return x end
function inc(d,k)      d[k]= 1+(d[k] or 0); return k end -- used for counting
```

```lua
function any(t,  n)
  if not n then return t[randi(1,#t)] end
  u={};for j=1,n do push(u, t[randi(1,#t)]) end; return u end
```

```lua
function map(t,f,  u)
  u={};for k,v in pairs(t) do u[#u+1]=f(v) end; return u; end
```

copy implements a deep copy.

```lua
function copy(t,   u)
  if type(t) ~= "table" then return t end
  u={}; for k,v in pairs(t) do u[k]=copy(v) end
  return setmetatable(u, getmetatable(t)) end
```

Display stuff

fmt is for simple prints.

```lua
fmt = string.format
```

o is for printing nested tables.

```lua
function o(t,     u,show)
  function show(k) return fmt(":%s %s", k, t[k]) end
  u= #t>0 and map(t,tostring) or map(sort(slots(t)),show)
  return (t._is or "").."{"..table.concat(u,", ").."}" end
```

rnd returns rounds `x` (and, if non-numeric, it just returns `x`).

```lua
function rnd(x,d,  n)
  n=10^(d or the.round)
  return type(x)~="number" and x or math.floor(x*n+0.5)/n end
```

```
function rnds(t,d) return map(t, function(x) return rnd(x,d) end) end
```

OS Stuff

atom coerces strings to atoms.

```
function atom(x)
  if x=="true" then return true elseif x=="false" then return false end
  return tonumber(x) or x end
```

csv returns comma-seperated rows as a table, with all strings coerced to their right type.

```
function csv(file)
  file = io.input(file)
  return function(    t)
    x=io.read();
    if x then
      t={}; for y in x:gsub("%s+",""):gmatch"([^,]+)" do t[1+#t]=atom(y) end
      return #t>0 and t
    else io.close(file) end end end
```

Settings stuff

For all lines starting with ' -' then grab the first (as a setting) and the last word (as a default value). Look for updates to these settings from the command line, For convenience, this code support partial match on the CLI to the setting name. Also, for flags with boolean code, using that command line flag will flip the default value.

```
function settings(help,       t)
  t = {}
  help:gsub("\n  [-]([^%s]+)[^\n]*%s([^%s]+)", function(flag, x)
    for n,txt in ipairs(arg) do
      if   txt:sub(1,1)=="-" and flag:match("^"..txt:sub(2)..".*")
      then x = x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
    t[flag] = atom(x) end)
  return t end
```

Random stuff

```
function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
function rand(lo,hi)
  the.seed = (16807 * the.seed) % 2147483647
  return (lo or 0) + ((hi or 1) - (lo or 0)) * the.seed / 2147483647 end
```

Math stuff

```
function xpects(t,       sum,n)
  sum,n = 0,0
  for _,one in pairs(t) do n= n + one.n; sum= sum + one.n*one:div() end
  return sum/n end
```

Error stuff

Wraps the “real” assert in code that increments `failures` and only shows a stack dump if `-D` was set of the commend-line.

```
failures=0
function asserts(test,msg)
  msg=msg or ""
  if test then return print("  PASS : "..msg) end
  failures = failures+1
  print("  FAIL : "..msg)
  if the.Debug then assert(test,msg) end end
```

```
function rogues(b4)
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end

-- ------------------------------------------------------------------------------
```

BAGs

```
BAG=klass""
function BAG.new(t) return new(BAG,t or {}) end
```

RANGEs

```
RANGE=klass"RANGE"
```

Create, add, merge

```
function RANGE.new(col,lo,hi,has)
  lo = lo or -math.huge
  return new(RANGE, {n=0,score=nil,col=col, lo=lo, hi=hi or lo, has=has or SYM()}) end
```

```
function RANGE.add(i,x,y)
  i.n = i.n+1
  i.hi = math.max(x, i.hi)
  i.lo = math.min(x, i.lo)
  i.has:add(y) end
```

```
function RANGE.merge(i,j,       k)
 k = RANGE(i.col, i.lo, j.hi, i.has:merged(j.has))
 k.n = i.n + j.n
 if k.has:div()*1.01 <= xpects{i, j} then return k end end
```

Printing stuff

```
function RANGE.__tostring(i)
  if i.lo == i.hi        then return fmt("%s == %s",i.col.txt,i.lo) end
  if i.lo == -math.huge then return fmt("%s < %s",i.col.txt,i.hi) end
  if i.ho ==  math.huge then return fmt("%s >= %s",i.col.txt,i.lo) end
  return fmt("%s <= %s < %s", i.lo, i.col.txt, i.hi) end
```

Queries

```
function RANGE.div(i) return i.has:div() end
```

```
function RANGE.select(i,eg,        x)
  x = eg.has[i.col.at]
  return x=="?" or i.lo <= x and x < i.hi end
```

```
function RANGE.eval(i,goal)
    local best, rest, goals = 0,0,{}
  if not i.score then
    function goals.smile(b,r) return r>b and 0 or b*b/(b+r +1E-31) end
    function goals.frown(b,r) return b<r and 0 or r*r/(b+r +1E-31) end
    function goals.xplor(b,r) return 1/(b+r                +1E-31) end
    function goals.doubt(b,r) return 1/(math.abs(b-r)       +1E-31) end
    for x,n in pairs(i.has) do
      if x==goal then best = best+n/i.n else rest = rest+n/i.n end end
    i.score = best + rest < 0.01 and 0 or goals[the.goal](best,rest) end
  return i.score end
```

```
EG=klass"EGS"
function EG.new(t) return new(EG, {has=t}) end
```

```
function EG.better(eg1,eg2,egs)
  local s1,s2,e,n,a,b = 0,0,10,#egs.cols.y
  for _,col in pairs(egs.cols.y) do
    a  = col:norm(eg1.has[col.at])
    b  = col:norm(eg2.has[col.at])
    s1 = s1 - e^(col.w * (a-b)/n)
    s2 = s2 - e^(col.w * (b-a)/n) end
  return s1/n < s2/n end
```

```
function EG.cols(i,cols) return map(cols,function(x) return i.has[x.at] end) end
```

```
function EG.dist(i,j,egs,     a,b,d,n)
  d,n = 0, #egs.cols.x + 1E-31
  for _,col in pairs(egs.cols.x) do
    a,b = i.has[col.at], j.has[col.at]
    d   = d + col:dist(a,b) ^ the.p end
  return (d/n) ^ (1/the.p) end
```

```
EGS=klass"EGS"
function EGS.new(i) return new(EGS, {rows={}, cols=nil}) end

function EGS.add(i,eg)
  eg = eg.has and eg.has or eg -- If eg has data buried inside, expose it.
  if i.cols then push(i.rows, EG(i.cols:add(eg))) else i.cols=COLS(eg) end end

function EGS.from(t, i) i=EGS();for _,eg in pairs(t) do i:add(eg) end;return i end
function EGS.file(f, i) i=EGS();for   eg in csv(f)   do i:add(eg) end;return i end

function EGS.clone(i,inits,    j)
  j = EGS()
  j:add(map(i.cols.all, function(col) return col.txt end))
  for _,x in pairs(inits or {}) do  j:add(x) end
  return j end

function EGS.cluster(i, rows)
  local far,zero,one,two,ones,twos,both,a,b,c
  function far(eg1,   fun,tmp)
    fun = function(eg2) return {eg2, eg1:dist(eg2,i)} end
    tmp = sort(map(tmp, fun), seconds)
    return table.unpack(tmp[#tmp*your.far//1])
  end -----------------------------------------
  rows  = #rows > the.Some and any(rows, the.Some) or rows
  zero  = any(rows)
  one   = i:far(zero)
  two,c = i:far(one)
  ones,twos,both = {},{},{}
  for _,eg in pairs(rows) do
    a = eg:dist(one, i)
    b = eg:dist(two, i)
    push(both, {(a^2 + c^2 - b^2) / (2*c),eg}) end
  for n,pair in pairs(sort(both, firsts)) do
    (n <= #both//2 and ones or twos):add(pair[2]) end
  if the.Better and two:better(one,i) then ones,twos=twos,ones end
  return ones, twos end

function EGS.mid(i,cols)
  return map(cols or i.cols.all, function(col) return col:mid() end) end

function EGS.superRanges(i,top)
  local one, two = top:cluster(i.rows)
  local best, out, col2, tmp, ranges = math.huge
  for n,col1 in pairs(one.cols.x) do
    col2   = two.cols.x[n]
    ranges = col1:superRanges( col1:ranges(col2))
    if #ranges > 1 then
      tmp = xpects(ranges)
      if tmp < best then best, out = tmp, ranges end end end
  return out, lefts, firsts end

COLS=klass"COLS"
function COLS.new(eg,     i,now,where)
  i = new(COLS,{all={}, x={}, y={}})
  for at,s in pairs(eg) do    -- First row. Create the right columns
    now   = push(i.all, (s:find"^[A-Z]" and NUM or SYM)(at,s))
    if not s:find":" then
      where = (s:find"-" or s:find"+") and i.y or i.x
      push(where, now) end end
  return i end
```

```
function COLS.add(i,eg)
  assert(#eg == #i.all,"expected a different number of cells")
  return map(i.all, function(col) return col:add(eg[col.at]) end) end
```

SYM: summarize stream of symbols
`lua SYM=klass"SYM" function SYM.new(at,txt) return new(SYM,{at=at or 0, txt=txt or "", n=0, has={},mode=nil,most=0}) end`
`lua function SYM.add(i,x,count) if x=="?" then count = count or 1 i.has[x] = count + (i.has[x] or 0) if i.has[x] > i.most then i.most,i.mode = i.has[x],x end end return x end`
`lua function SYM.merge(i,j,     k) k= SYM(i.at, i.txt) for x,count in pairs(i.has) do k:add(x,count) end for x,count in pairs(j.has) do k:add(x,count) end return k end`
“‘lua
‘‘‘
dist stuff
`lua function SYM.dist(i,x,y) return x=="?" and y=="?" and 1 or x==y and 0 or 1 end`
stats stuff
`lua function SYM.mid(i) return i.mode end function SYM.div(i,   e) e=0; for _,n in pairs(i.has) do e=e-n/i.n*math.log(n/i.n,2) end; return e end`
discretization stuff
`lua function SYM.superRanges(i,ranges) return ranges end function SYM.ranges(i,j,   t,out) t,out = {},{} for x,n in pairs(i.has) do t[x]= t[x] or SYM(); t[x]:add("best",n) end for x,n in pairs(j.has) do t[x]= t[x] or SYM(); t[x]:add("rest",n) end for x,stats in pairs(t) do push(out, RANGE(i,x,x,stats)) end return out end`

Columns

NUM: summarize streams of numbers

```
NUM=klass"NUM"
```

Create, add, merge

```
function NUM.new(at,txt)
  return new(NUM,{at=at or 0, txt=txt or "", n=0, has={}, ready=false,
                 w=(txt or ""):find"-" and -1 or 1}) end

function NUM.add(i,x,     pos)
  if x ~="?" then
    i.n= i.n + 1
    if     #i.has < the.ample  then pos = 1       + #i.has
    elseif rand() < #i.has/i.n then pos = rand() * #i.has end
    if pos then i.ready=false; i.has[pos//1]= x end end
  return x end
```

```
function NUM.merge(i,j,         k)
  k = NUM(i.at, i.txt)
  for _,x in pairs(i.has) do k:add(x) end
  for _,x in pairs(j.has) do k:add(x) end
  return k end
```

Distance stuff

```
function NUM.norm(i,x,    a)
  a=i:all(); return  (a[#a]-a[1]) < 1E-9 and 0 or (x-a[1])/(a[#a] - a[1]) end
function NUM.dist(i,x,y)
  if     x=="?" and y=="?" then return 1
  elseif x=="?"            then y= i:norm(y); x=y>.5 and 0 or 1
  elseif y=="?"            then x= i:norm(x); y=x>.5 and 0 or 1
  else   x,y = i:norm(x), i:norm(y) end
  return math.abs(x-y) end
```

Queries

```
function NUM.lo(i)  return i:all()[1]  end
function NUM.hi(i)  return last(i:all()) end
function NUM.mid(i) return i:per(.5) end
function NUM.div(i) return (i:per(.9) - i:per(.1))/2.56 end
function NUM.per(i,p,   a) a=i:all(); return a[math.min(#a,1+(p*#a//1))]; end
function NUM.all(i)
  if not i.ready then table.sort(i.has); i.ready=true end; return i.has end
```

Discretization

Until no new merges are found, try combining adjacent ranges.

```
function NUM.superRanges(i,b4)
  local j,tmp,one,two,both = 0, {}
  while j < #b4 do
    j = j + 1
    one, two = b4[j], b4[j+1]
    if two then
      both = one:merge(two)
      if both then  -- both is as simple as the original one,two
        now=both
        j=j+1 end end -- skip over merged range
    push(tmp,now) end
  return #tmp==#b4 and b4 or i:superRanges(tmp) end
```

Divide `i,j` numbers into `the.bins` ranges.

```
function NUM.ranges(i,j, yklass)
  local out,lo,hi,gap = {}
  lo  = math.min(i:lo(), j:lo())
  hi  = math.max(i:hi(), j:hi())
  gap = (hi-lo)/the.bins
  for b=1,the.bins do
    here   = lo + (b-1)*gap
    out[b] = RANGE(i, here, here+gap, (yklass or SYM)()) end
  for _,x in pairs(i._has.all) do out[(x-lo)//gap]:add(x,"best") end
  for _,x in pairs(j._has.all) do out[(x-lo)//gap]:add(x,"rest") end
  out[1].lo    = -math.huge
  out[#out].hi =  math.huge
  return out end
```

```
NB=klass"NB"
function NB.new() return new(NB, {k=1,m=2,names=BAG(),n, hs=0,h={}, f={}}) end
```

```
function NB.read(i, file)
  for row in csv(file) do if row then i:add(n,row) end end end
```

```
function NB.add(i, n,row,          k,klass)
  if n==0 then i.names=row else
    k=#row
    if n > 5 then print(row[k], i:classify(row)) end
    klass=row[k]
    if not i.h[klass] then i.hs=i.hs+1; i.h[klass]=0 end
    inc(i.h,row[k])
    i.n=i.n+1
    for col,x in pairs(row) do
      if col~=k and x~="?" then
        inc(i.f, {col,x,klass}) end end end  end
```

```
function NB.classify(i,row,      best)
  best=-1
  for klass,nh in pairs(i.h) do
    local prior = (nh+i.k)/(i.n + i.k*i.hs)
    local tmp   = prior
    for col,x in pairs(row) do
      if col ~= #row and x~="?" then
        tmp = tmp * ((i.f[{col,x,klass}] or 0) +i.m*prior)/(nh+i.m) end end
    if tmp > best then best,out=tmp,klass end end
  return klass end
```

```
go={}
function go.copy(   a,b)
  a={1,2,3,{40,50}}; b=copy(a); b[4][1]=400
  asserts(a[4][1]~=b[4][1],"deep copy") end
```

```
function go.range(  r)
  r=RANGE(NUM(10,"fred"),31)
  r:add(23,32)
  assert(tostring(r) == "23 <= fred < 31", "print ok") end
```

```
function go.num(    m,n)
  m=NUM();
  for j=1,20 do m:add(j) end; n=copy(m)
  for j=1,20 do n:add(j) end
  asserts(6.25 == rnd(n:div()),"sd ok") end
```

```
function go.egs(     egs)
  egs = EGS.file(the.file)
  asserts(egs.cols.y[1]:hi()==5140,"most seen") end
```

```
function go.clone(      egs1,egs2,s1,s2)
  egs1 = EGS.file(the.file)
  egs2 = egs1:clone(egs1.rows)
  s1   = o(egs1.cols.y)
  s2   = o(egs2.cols.y)
  assert(o(rnds(egs1:mid()))==o(rnds(egs2:mid())),"cloning") end
```

```
function go.order(   egs,t)
  egs = EGS.file(the.file)
  t = sort(egs.rows, function(a,b) return a:better(b,egs) end)
  for j=1,5      do print("<",o(t[j]:cols(egs.cols.y))) end; print("")
  for j=#t-5,#t do print(">",o(t[j]:cols(egs.cols.y))) end end
```

start up stuff

```
the = settings(help)
old = copy(the)
if the.h then
  print(help)
else
  failures = 0
  for _,it in pairs(the.todo=="all" and slots(go) or {the.todo}) do
    if go[it] then print(it); go[it](); the = old end end -- do, then reset
  rogues(b4) end
```

“‘lua os.exit(failures)