```lua
-- .
local b4={}; for k, _ in pairs(_ENV) do b4[k]=k end
local l,the,help = {},{},[[
gate: guess, assess, try, expand
(c) 2023, Tim Menzies, BSD-2
Learn a little, guess a lot, try the strangest guess, learn a little more, repeat

USAGE:
  lua gate.lua [OPTIONS]

OPTIONS:
  -c --cohen    small effect size               = .35
  -f --file     csv data file name              = ../data/diabetes.csv
  -h --help     show help                       = false
  -k --k        low class frequency kludge      = 1
  -m --m        low attribute frequency kludge  = 2
  -s --seed     random number seed              = 31210
  -t --todo     start up action                 = help]]

-- ------------------------------------------------------------------------------
-- ## Classes
local function isa(x,y) return setmetatable(y,x) end
local function is(s,    t) t={a=s}; t.__index=t; return t end

-- ## Columns
-- ### Numerics

-- Create
local NUM=is"NUM"
function NUM.new(s, n)
  return isa(NUM, {txt=s or "", at=n or 0, n=0, mu=0, m2=0, hi=-1E30, lo=1E30,
                   heaven = (s or ""):find"-$" and 0 or 1}) end

-- Update
function NUM:add(x,     d)
  if x ~="?" then
    self.n  = self.n+1
    d       = x - self.mu
    self.mu = self.mu + d/self.n
    self.m2 = self.m2 + d*(x - self.mu)
    self.lo = math.min(x, self.lo)
    self.hi = math.max(x, self.hi) end end

-- Query
function NUM:mid() return self.mu end

function NUM:div() return self.n < 2 and 0 or (self.m2/(self.n - 1))^.5 end

function NUM:small() return the.cohen*self:div() end

function NUM:norm(x)
  return x=="?" and x or (x - self.lo) / (self.hi - self.lo + 1E-30) end

-- Likelihood
function NUM:like(x,_,      nom,denom)
  local mu, sd = self:mid(), (self:div() + 1E-30)
  nom   = 2.718^(-.5*(x - mu)^2/(sd^2))
  denom = (sd*2.5 + 1E-30)
  return  nom/denom end

-- ### Symbols

-- Create
local SYM=is"SYM"
function SYM.new(s,n)
  return isa(SYM,{txt=s or "", at=n or 0, n=0, has={}, mode=nil, most=0}) end

-- Update
function SYM:add(x)
  if x ~= "?" then
    self.n = self.n + 1
    self.has[x] = 1 + (self.has[x] or 0)
    if self.has[x] > self.most then
      self.most,self.mode = self.has[x], x end end end

-- Query
function SYM:mid() return self.mode end

function SYM:div(    e)
  e=0; for _,v in pairs(self.has) do e=e-v/self.n*math.log(v/self.n,2) end; return e end

function SYM:small() return 0 end

-- Likelihood
function SYM:like(x, prior)
  return ((self.has[x] or 0) + the.m*prior)/(self.n +the.m) end

-- ### Columns
-- A contrainer storing multiple `NUM`s and `SYM`s.

-- Create a set of columns from a set of strings. If uppercase
-- then `NUM`, else `SYM`. `Klass`es end in `!`. Numeric goals to
-- minimize of maximize end in `-`,`+`. Keep all cols in `all`.
-- Also add dependent columns to `y` (anthing ending in `-`,`+`,`!`) and
-- independent columns in `x` (skipping over anyhing ending in `X`).
local COLS=is"COLS"
function COLS.new(row)
  local x,y,all = {},{},{}
  local klass,col
  for at,txt in pairs(row.cells) do
    col = (txt:find"^[A-Z]" and NUM or SYM).new(txt,at)
    all[1+#all] = col
    if not txt:find"X$" then
      if txt:find"!$" then klass=col end
      (txt:find"[!+-]$" and y or x)[at] = col end end
  return isa(COLS,{x=x, y=y, all=all, klass=klass, names=row.cells}) end

-- Update
function COLS:add(row)
  for _,cols in pairs{self.x, self.y} do
    for _,col in pairs(cols) do
      col:add(row.cells[col.at]) end end
  return row end


-- ### ROW

-- Store cells.
local ROW=is"ROW"
function ROW.new(t) return isa(ROW, { cells = t }) end

-- Distance to best values (and _lower_ is _better_).
function ROW:d2h(data, d, n)
  d, n = 0, 0
  for _, col in pairs(data.cols.y) do
      n = n + 1
      d = d + math.abs(col.heaven - col:norm(self.cells[col.at])) ^ 2 end
  return d ^ .5 / n ^ .5 end

-- Return the `data` (from `datas`) that I like the best
function ROW:likes(datas,       n,nHypotheses,most,tmp,out)
  n,nHypotheses = 0,0
  for k,data in pairs(datas) do
    n = n + #data.rows
    nHypotheses = 1 + nHypotheses end
  for k,data in pairs(datas) do
    tmp = self:like(data,n,nHypotheses)
    if most==nil or tmp > most then most,out = tmp,k end end
  return out,most end

-- How much does ROW like `self`. Using logs since these
-- numbers are going to get very small.
function ROW:like(data,n,nHypotheses,       prior,out,v,inc)
  prior = (#data.rows + the.k) / (n + the.k * nHypotheses)
  out   = math.log(prior)
  for _,col in pairs(data.cols.x) do
    v= self.cells[col.at]
    if v ~= "?" then
      inc = col:like(v,prior)
      out = out + math.log(inc) end end
  return math.exp(1)^out end

-- ### Data
-- Store `rows`, summarized in `COL`umns.

-- Create from either a file name or a list of rows
local DATA=is"DATA"
function DATA.new(src,  fun,     self)
  self = isa(DATA,{rows={}, cols=nil})
  if   type(src) == "string"
  then for _,x in l.csv(src)       do self:add(x, fun) end
  else for _,x in pairs(src or {}) do self:add(x, fun) end end
  return self end

-- Update. First time through, assume the row defines the columns.
-- Otherwise, update the columns then store the rows. If `fun` is
-- defined, call it before updating anything.
function DATA:add(t,  fun,row)
  row = t.cells and t or ROW.new(t)
  if   self.cols
  then if fun then fun(self,row) end
       self.rows[1 + #self.rows] = self.cols:add(row)
  else self.cols = COLS.new(row) end end

-- Query
function DATA:mid(cols,   u)
  u = {}; for _, col in pairs(cols or self.cols.all) do u[1 + #u] = col:mid() end
  return ROW.new(u) end

function DATA:div(cols,    u)
  u = {}; for _, col in pairs(cols or self.cols.all) do u[1 + #u] = col:div() end;
  return ROW.new(u) end

function DATA:small(    u)
  u = {}; for _, col in pairs(self.cols.all) do u[1 + #u] = col:small(); end
  return ROW.new(u) end

function DATA:stats(cols,fun,ndivs,    u)
  u = {[".N"] = #self.rows}
  for _,col in pairs(self.cols[cols or "y"]) do
    u[col.txt] = l.rnd(getmetatable(col)[fun or "mid"](col), ndivs) end
  return u end


-- Gate.
function DATA:gate(budget0,budget,some)
  local rows,lite,dark
  local stats,bests = {},{}
  rows = l.shuffle(self.rows)
  lite = l.slice(rows,1,budget0)
  dark = l.slice(rows, budget0+1)
  for i=1,budget do
    local best, rest      = self:bestRest(lite, (#lite)^some)  -- assess
    local todo, selected = self:split(best,rest,lite,dark)
    stats[i] = selected:mid()
    bests[i] = best.rows[1]
    table.insert(lite, table.remove(dark,todo)) end
  return stats,bests end

-- Find the row scoring based on our acquite function.
function DATA:split(best,rest,lite,dark)
  local selected,max,out
  selected = DATA.new{self.cols.names}
  max = 1E30
  out = 1
  for i,row in pairs(dark) do
    local b,r,tmp
    b = row:like(best, #lite, 2)
    r = row:like(rest, #lite, 2)
    if b>r then selected:add(row) end
    tmp = math.abs(b+r) / math.abs(b-r+1E-300)
    --print(b,r,tmp)
    if tmp > max then out,max = i,tmp end end
  return out,selected end

-- Sort on distance to heaven, split off the first `want` items to return
-- a `best` and `rest` data.
function DATA:bestRest(rows, want, best, rest, top)
    table.sort(rows, function(a, b) return a:d2h(self) < b:d2h(self) end)
    best, rest = { self.cols.names }, { self.cols.names }
    for i, row in pairs(rows) do
        if i <= want then best[1 + #best] = row else rest[1 + #rest] = row end
    end
    return DATA.new(best), DATA.new(rest)
end
```

```lua
-- ------------------------------------------------------------------------
-- ## Library Functions


-- ### Linting
function l.rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("E:",k,type(k)) end end end

-- ### Numbers
function l.rnd(n, ndecs)
  if type(n) ~= "number" then return n end
  if math.floor(n) == n  then return n end
  local mult = 10^(ndecs or 2)
  return math.floor(n * mult + 0.5) / mult end

-- ### Lists

-- Sorted keys
function l.keys(t,     u)
  u={}; for k,_ in pairs(t) do u[1+#u]=k end; table.sort(u); return u end

-- Deep copy
function l.copy(t,     u)
  if type(t) ~= "table" then return t end
  u={}; for k,v in pairs(t) do u[l.copy(k)] = l.copy(v) end
  return u end

-- Return a new table, with old items sorted randomly.
function l.shuffle(t,     u,j)
  u={}; for _,x in pairs(t) do u[1+#u]=x; end;
  for i = #u,2,-1 do j=math.random(i); u[i],u[j] = u[j],u[i] end
  return u end

-- Return `t` skipping `go` to `stop` in steps of `inc`.
function l.slice(t, go, stop, inc,    u)
  if go   and go   < 0 then go=#t+go   end
  if stop and stop < 0 then stop=#t+stop end
  u={}
  for j=(go or 1)//1,(stop or #t)//1,(inc or 1)//1 do u[1+#u]=t[j] end
  return u end

-- ### String to Things

-- Coerce string to intm float, nil, true, false, or (it all else fails), a strong.
function l.coerce(s1,     fun)
  function fun(s2)
    if s2=="nil" then return nil else return s2=="true" or (s2~="false" and s2) end end
  return math.tointeger(s1) or tonumber(s1) or fun(s1:match'^%s*(.*%S)') end

-- Parse help string to infer the settings.
function l.settings(s,     t,pat)
  t,pat = {}, "[-][%S]+[^-]+=([%S]+)"
  for k, s1 in s:gmatch(pat) do t[k] = l.coerce(s1) end
  t._help = s
  return t end

-- Return a list of comma seperated values (coerced to things)
function l.cells(s,    t)
  t={}; for s1 in s:gmatch("([^,]+)") do t[1+#t]=l.coerce(s1) end; return t end

-- Return rows of a csv file.
function l.csv(src,    i)
  i,src = 0,src=="-" and io.stdin or io.input(src)
  return function(      s)
    s=io.read()
    if s then i=i+1; return i,l.cells(s) else io.close(src) end end end

-- Update a table of settings using command-line settings.
function l.cli(t)
  for k, v in pairs(t) do
    v = tostring(v)
    for argv,s in pairs(arg) do
      if s=="-"..(k:sub(1,1)) or s=="--"..k then
        v = v=="true" and "false" or v=="false" and "true" or arg[argv + 1]
        t[k] = l.coerce(v) end end end
  if t.help then os.exit(print("\n"..t._help)) end
  return t end

-- ### Things to Strings

-- Emulate sprintf
l.fmt = string.format

-- Print a string of a nested structure.
function l.oo(x) print(l.o(x)); return x end

-- Rerun a string for a nested structure.
function l.o(t,   n,      u)
  if type(t) == "number" then return tostring(l.rnd(t, n)) end
  if type(t) ~= "table"  then return tostring(t) end
  u={}
  for _,k in pairs(l.keys(t)) do
    if tostring(k):sub(1,1) ~= "_" then
      u[1+#u]= #t>0 and l.o(t[k],n) or l.fmt("%s: %s", l.o(k,n), l.o(t[k],n)) end end
  return "{" .. table.concat(u, " ") .. "}" end


```

```lua
-- ------------------------------------------------------------------------
-- ## Examples

-- ### Examples support code

-- Where to store examples
local eg={}

local function run(k,   oops,b4)
  b4 = l.copy(the) -- set up
  math.randomseed(the.seed) -- set up
  oops = eg[k]() ==false
  io.stderr:write(l.fmt("# %s %s\n",oops and "âM-^]M-^L FAIL" or "âM-^\M-^E PASS",k))
  for k,v in pairs(b4) do the[k]=v end -- tear down
  return oops end

-- Run all examples
function eg.all(     bad)
  bad=0
  for _,k in pairs(l.keys(eg)) do
    if k ~= "all" then
      if run(k) then bad=bad+1 end end end
  io.stderr:write(l.fmt("# %s %s fail(s)\n",bad>0 and "âM-^]M-^L FAIL" or "âM-^\M-^E PASS",bad))
  os.exit(bad) end

-- List all example names
function eg.egs()
  for _,k in pairs(l.keys(eg)) do print(l.fmt("lua gate.lua -t %s",k)) end end

-- ### The actual examples
function eg.oo()
  return l.o{a=1,b=2,c=3,d={e=3,f=4}}  == "{a: 1, b: 2, c: 3, d: {e: 3, f: 4}}" end

function eg.the() l.oo(the); return the.help ~= nil and the.seed and the.m and the.k  end

function eg.help() print("\n"..the._help) end

function eg.sym(      s,mode,e)
  s = SYM.new()
  for _, x in pairs{1,1,1,1,2,2,3} do s:add(x) end
  mode, e = s:mid(), s:div()
  print(mode, e)
  return 1.37 < e and e < 1.38 and mode == 1 end

local function norm(mu,sd,    R)
  R=math.random
  return (mu or 0) + (sd or 1) * math.sqrt(-2 * math.log(R()))
                               * math.cos(2 * math.pi * R()) end

function eg.num(      e,mu,sd)
  e = NUM.new()
  for _ = 1,1000 do e:add(norm(10, 2)) end
  mu, sd = e:mid(), e:div()
  print(l.rnd(mu,3), l.rnd(sd,3))
  return 10 < mu and mu < 10.1 and 2 < sd and sd < 2.05 end

function eg.csv(      n)
  n=0
  for i,t in l.csv(the.file) do
    if i%100 == 0 then  n = n + #t; print(i, l.o(t)) end end
  return n == 63 end

function eg.data(     d,n)
  n=0
  d = DATA.new(the.file)
  for i, row in pairs(d.rows) do
    if i % 100 ==0 then n = n + #row.cells; l.oo(row.cells) end end
  l.oo(d.cols.x[1].cells)
  return n == 63 end

local function learn(data,row,  my,kl)
  my.n = my.n + 1
  kl   = row.cells[data.cols.klass.at]
  if my.n > 10 then
    my.tries = my.tries + 1
    my.acc   = my.acc + (kl == row:likes(my.datas) and 1 or 0) end
  my.datas[kl] = my.datas[kl] or DATA.new{data.cols.names}
  my.datas[kl]:add(row) end

function eg.bayes()
  local wme = {acc=0,datas={},tries=0,n=0}
   DATA.new("../data/diabetes.csv", function(data,t) learn(data,t,wme) end)
   print(wme.acc/(wme.tries))
   return wme.acc/(wme.tries) > .72 end

function eg.km()
  print(l.fmt("#%4s\t%s\t%s","acc","k","m"))
  for k=0,3,1 do
    for m=0,3,1 do
      the.k = k
      the.m = m
      local wme = {acc=0,datas={},tries=0,n=0}
      DATA.new("../data/soybean.csv", function(data,t) learn(data,t,wme) end)
      print(l.fmt("%5.2f\t%s\t%s",wme.acc/wme.tries, k,m)) end end end

function eg.stats()
  return  l.o(DATA.new("../data/auto93.csv"):stats())  ==
              "{.N: 398, Acc+: 15.57, Lbs-: 2970.42, Mpg+: 23.84}" end

function eg.sorted(   d)
  d=DATA.new("../data/auto93.csv")
  table.sort(d.rows, function(a,b) return a:d2h(d) < b:d2h(d) end)
  print("",l.o(d.cols.names))
  for i, row in pairs(d.rows) do
    if i < 5  or i> #d.rows - 5 then print(i, l.o(row.cells)) end end end

function eg.gate(stats, bests, d, say,sayd)
  local budget0,budget,some = 4,10,.5
  print(the.seed)
  d = DATA.new("../data/auto93.csv")
  function sayd(row, txt) print(l.o(row.cells), txt, l.rnd(row:d2h(d))) end
  function say( row,txt)  print(l.o(row.cells), txt) end
  print(l.o(d.cols.names),"about","d2h")
  print"#overall" ----------------------------------------
  sayd(d:mid(), "mid")
  say(d:div() , "div")
  say(d:small(),"small=div*"..the.cohen)
  print"#generality" ----------------------------------
  stats,bests = d:gate(budget0, budget, some)
  for i,stat in pairs(stats) do sayd(stat,i+budget0) end
  print"#specifically" ------------------------------------------------
  for i,best in pairs(bests) do sayd(best,i+budget0) end
  print"#optimum" ------------------------------------------------------
  table.sort(d.rows, function(a,b) return a:d2h(d) < b:d2h(d) end)
  sayd(d.rows[1], #d.rows)
  print"#random" -------------------------------------------------------
  local rows=l.shuffle(d.rows)
  rows = l.slice(rows,1,math.log(.05)/math.log(1-the.cohen/6))
  table.sort(rows, function(a,b) return a:d2h(d) < b:d2h(d) end)
  sayd(rows[1]) end

function eg.gate20(    ss,bs,rs,d,stats,bests,rows,stat,best)
  print("#average, #optimistic,#random")
  ss,bs,rs=NUM.new(),NUM.new(),NUM.new()
  for i=1,20 do
    io.write(i," "); io.flush()
    d=DATA.new(the.file)
    d.rows = l.shuffle(d.rows)
    stats,bests = d:gate(4, 16, .5)
    ss:add(stats[#stats]:d2h(d))
    bs:add(bests[#bests]:d2h(d))
    stat,best = stats[#stats]:d2h(d), bests[#bests]:d2h(d)
    rows=l.shuffle(d.rows)
    rows = l.slice(rows,1,math.log(.05)/math.log(1-the.cohen/6))
    table.sort(rows, function(a,b) return a:d2h(d) < b:d2h(d) end)
    rs:add(rows[1]:d2h(d)) end
  print""
  print(l.rnd(ss:mid(),2), l.rnd(bs:mid(),2), l.rnd(rs:mid(),2))
  print(l.rnd(2*ss:div(),2), l.rnd(2*bs:div(),2), l.rnd(2*rs:div(),2))
  end

-- ------------------------------------------------------------------------
-- ## Start-up

the =  l.settings(help)
if not pcall(debug.getlocal,4,1) then run(l.cli(the).todo) end
l.rogues()
return {the=the, COLS=COLS, DATA=DATA, NUM=NUM, ROW=ROW, SYM=SYM}
```