```python
#!/usr/bin/env python3
"""
nb.py: naive bayes classifier
(c) 2026, Tim Menzies, MIT license.

USAGE
  python3 nb.py [OPTIONS] [FILE]

DESCRIPTION
   Incremental naive bayes. Training and testing are interleaved: after
   burn−in, each row is classified then added to the training set.

OPTIONS
   −h          Show help.
   −k k=1      Bayes low frequency hack for symbolic attributes.
   −m m=2      Bayes low frequency hack for class priors.
   −w wait=5   Start classifying after seeing "some" rows

EXAMPLES
   −−the       Print config settings.
   −−sym       Test symbolic column.
   −−num       Test numeric column.
   −−csv F     Print rows from CSV file.
   −−nb F      Run naive bayes on CSV file.

INPUT FORMAT
   Comma−separated values. First row defines column names. Uppercase
   names (Age, Weight) are numeric; lowercase (name, color) are symbolic.
   Suffixes: "!" class label, "X" ignore.
   Missing values: "?".

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
CODING STANDARD

  Type Hints (single letter)
   i:instance(Obj)   t:target(dict)   s:string    n:number
   r:row(list)       c:col(Obj)       v:value     f:file/filename
   d:delta/data      k:class/key      b4:before(prior)

  Class System
   Obj(dict):        Base class, provides dot notation access (d.x).
   CamelCase(args):  Factory functions (Sym, Num, Data) returning Obj.
   camelcase:        "data" (or e.g. "data1") is created by Data()

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
API

  # Constructors
  Sym(n = 0, s = "")   −− Create symbolic column at position n, name s.
  Num(n = 0, s = "")   −− Create numeric column at position n, name s.
  Data(s = "", items = []) −− Create dataset from list of rows/items.
  clone(d, rows = [])  −− Create new Data with same structure as d.
  Cols(row)            −− Generate column headers from a list of names.

  # Classifier
  nb(items)            −− Run incremental Naive Bayes on item iterator.

  # Methods (Functional)
  add(i, v)            −− Update counts (Sym) or Welford stats (Num).
                     −− If i is Data, add row and update cols.
  like(i, v, prior)   −− Calculate likelihood of v given column i.
  likes(i, r, nall, nh)−− Calculate log−likelihood of row r given Data i.

  # Utilities
  cast(s)            −− Parse string to int, float, or strip whitespace.
  csv(file)            −− Iterator yielding rows from CSV file.
  o(t)               −− Pretty print object/dict t.
"""


import re, sys, math
from math import sqrt, exp, log
BIG = 1E32
the={}

# --- functions -----------------------------------------------------
def csv(f):
  with open(f) as file:
    for s in file: yield [cast(x) for x in s.split(",")]

def cast(s):
  try: return int(s)
  except ValueError:
    try: return float(s)
    except ValueError: return s.strip()

def o(t):
  match t:
   case dict(): return "{"+" ".join(f":{k} {o(t[k])}" for k in t)+"}"
   case float(): return f"{int(t)}" if int(t) == t else f"{t:.2f}"
   case list()|tuple(): return str([o(x) for x in t])
   case _: return str(t)

class Obj(dict):
  __getattr__,__setattr__,__repr__=dict.__getitem__,dict.__setitem__,o

# --- objects ------------------------------------------------------
def Sym(n=0, s=""): return Obj(at=n, txt=s, n=0, has={})
def Num(n=0, s=""): return Obj(at=n, txt=s, n=0, mu=0, m2=0)
def Col(n=0, s=""): return (Num if s[0].isupper() else Sym)(n,s)

def Data(s="", items=[]):
  d = Obj(txt=s, rows=[], cols=None)
  [add(d, r) for r in items]
  return d

def Cols(row):
  all = [Col(n,s) for n,s in enumerate(row)]
  return Obj(names=row, all=all,
             x=[c for c in all if not re.search(r"[!X]$", c.txt)],
             y=[c for c in all if re.search(r"!$", c.txt)])

def add(i, v):
  if "rows" in i: # Data
    if not i.cols: i.cols = Cols(v)
    else: i.rows.append([add(c, v[c.at]) for c in i.cols.all])
  elif v != "?":
    i.n += 1
    if "mu" in i: d = v - i.mu; i.mu += d/i.n; i.m2 += d*(v - i.mu)
    else: i.has[v] = 1 + i.has.get(v, 0) # Sym
  return v

# --- bayes ---------------------------------------------------------
def like(i, v, prior=0):
  if "mu" in i: # Num
    sd = 0 if i.n < 2 else (i.m2/(i.n - 1))**.5
    var = sd**2 + 1/BIG
    return (1/sqrt(2*math.pi*var)) * exp(-((v - i.mu)**2)/(2*var))
  else:    # Sym
    n = i.has.get(v, 0) + the.k*prior
    return max(1/BIG, n/(i.n + the.k + 1/BIG))

def likes(i, r, nall, nh):
  b4 = (len(i.rows) + the.m)/(nall + the.m*nh)
  return log(b4) + sum(log(like(c, r[c.at], b4))
                       for c in i.cols.x if r[c.at] != "?")

def nb(rows):
  all, klasses, nh, out = None, {}, 0, Sym()
  for n, row in enumerate(rows):
    if n==0: all = Data("all", [row])
    else:
      k = row[all.cols.y[0].at]
      if k not in klasses: nh +=1; klasses[k]=Data(k,[all.cols.names])
      if (n - 1) > the.wait:
        fn = lambda cat:likes(klasses[cat],row,n-1,nh)
        add(out, (max(klasses, key=fn), k)) #(predicted, actual)
      add(klasses[k], row)
  return out

# --- main ----------------------------------------------------------
def eg_h(_):    print(__doc__)
def eg__the(_): print(o(the))
def eg__sym(_): print(add(add(add(Sym(),"a"),"a"),"b"))
def eg__num(_): print([add(Num(), x) for x in [10,20,30,40]][-1])
def eg__csv(f): [print(r) for r in csv(f)]
def eg__nb(f):  [print(n,*x) for x,n in nb(csv(f)).has.items()]

the=Obj(**{k:cast(v) for k,v in re.findall(r"(\S+)=(\S+)",__doc__)})
if __name__ == "__main__":
  for j,s in enumerate(sys.argv):
    if f := vars().get(f"eg{s.replace('−', '_')}"):
      f(sys.argv[j+1] if j+1 < len(sys.argv) else None)
```

```lua
#!/usr/bin/env lua
local help = [[
lib.lua: general utilities
(c) 2026, Tim Menzies, MIT license.

USAGE
   lua lib.lua [OPTIONS]

EXAMPLES
    -h          Show help.
    --order     Test sorted key iteration.
    --iter      Test iterator over tables and functions.
    --csv F     Print rows from CSV file.

--------------------------------------------------------------------
CODING STANDARD

  Type Hints (single letter)
    i:instance t:table u:output_table r:row n:number
    s:string v:value k:key f:function d:delta j:index items:iterator

  Multiple Same-Type Params
    Base + suffix: nall, nh, n1, n2

  Class System
    UPPERCASE:metatable (SYM,NUM)  CamelCase:constructor (Sym,Num)
    lowercase:instance (data,col)

  Collision Avoidance
    file (not f, since f is function)

  Function Signatures
    Params before extra spaces; locals after:
      function sum(t,f,    n)   -- t,f:params; n:local

--------------------------------------------------------------------
API
  ITERATION
    lib.iter(items)           -- iterator over tables or functions
    lib.order(t)              -- sorted key iteration

  FUNCTIONAL
    n = lib.sum(t,f)          -- sum f(v) over t
    u = lib.kap(t,f)          -- map t by f(k,v)
    u = lib.sel(t,f)          -- filter t by f(v)
    k = lib.most(t,f)         -- key where f(k,v) is max

  CONVERSION
    v = lib.cast(s)           -- string to int/float/trimmed string
    t = lib.casts(s)          -- comma-separated string to table

  IO
    lib.csv(file)             -- iterator over CSV rows
    s = lib.o(t)              -- pretty print
]]

local int = math.tointeger
local fmt = string.format
local BIG = 1E32

-- iteration -----------------------------------------------------
local function iter(t,    more,state,key)
  if type(t)=="function" then return t end
  more,state,key = pairs(t)
  return function(v) key,v = more(state,key); return v end end

local function order(t,     u,j)
  if #t>0 then return ipairs(t) end
  u,j = {},0
  for k in pairs(t) do u[#u+1]=k end; table.sort(u)
  return function()j=j+1; if u[j] then return u[j],t[u[j]] end end end

-- meta ----------------------------------------------------------
local function isa(mt,t) mt.__index=mt; return setmetatable(t,mt) end

-- conversion ----------------------------------------------------
local function cast(s)
  return int(s) or tonumber(s) or s:match"^%s*(.-)%s*$" end

local function casts(s,    t)
  t={}; for x in s:gmatch"[^,]+" do t[1+#t]=cast(x) end; return t end

-- functional ----------------------------------------------------
local function sum(t,f,   n)
  n=0; for _,v in pairs(t) do n=n+f(v) end
  return n end

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

local function sel(t,f,  u)
  u={}; for _,v in pairs(t) do if f(v) then u[1+#u]=v end end
  return u end

local function most(t,f,   n,out,tmp)
  n = -BIG; for k,v in pairs(t) do
    tmp = f(k,v); if tmp and tmp > n then n,out = tmp,k end end
  return out end

-- io ------------------------------------------------------------
local function csv(file,    src)
  src = assert(io.open(file))
  return function(s)
    s=src:read()
    if s then return casts(s) else src:close() end end end

local function o(t,     u,mt)
  if math.type(t)=="float" then return fmt("%.2f",t) end
  if type(t)~="table" then return tostring(t) end
  mt=getmetatable(t); u={}
  for k,v in order(t) do
    u[1+#u]=#t>0 and o(v) or fmt(":%s %s",k,o(v)) end
  return (mt and mt._is or "").."{"..table.concat(u," ").."}" end

-- demos ---------------------------------------------------------
local eg={}

eg["-h"]= function(_) print("\n"..help) end

eg["--order"]= function(_)
  for k,v in order({z=1,a=2,m=3}) do print(k,v) end end

eg["--iter"]= function(_,    t,f)
  t={}; for x in iter({2,4,8}) do t[1+#t]=x end; print(o(t))
  t={}; f=function(n,   j) j=0;
          return function() j=j+1
                            if j<=n then return j*10 end end end
  for x in iter(f(8)) do t[1+#t]=x end; print(o(t)) end

eg["--csv"]= function(f) for row in csv(f) do print(o(row)) end end

-- main ----------------------------------------------------------
if arg[0] and arg[0]:find"lib" then
  for j,s in pairs(arg) do if eg[s] then eg[s](arg[j+1]) end end end

return {BIG=BIG, iter=iter, order=order, sum=sum, kap=kap, sel=sel,
        most=most, cast=cast, casts=casts, csv=csv, isa=isa, o=o,
        run=run, eg=eg}
```

```lua
#!/usr/bin/env lua
local help = [[
nb.lua: naive bayes classifier
(c) 2026, Tim Menzies, MIT license.

USAGE
   lua nb.lua [OPTIONS] [FILE]

DESCRIPTION
    Incremental bayes. Training and testing are interleaved: after
    burn-in, each row is classified then added to the training set.

OPTIONS
    -h          Show help.
    -k k=1      Bayes low frequency hack for symbolic attributes.
    -m m=2      Bayes low frequency hack for class priors.
    -w wait=5   Start classifying after seeing "some" rows.

EXAMPLES
    --the       Print config settings.
    --sym       Test symbolic column.
    --num       Test numeric column.
    --col       Test column creation.
    --cols      Test column set creation.
    --data F    Load data, print first y column.
    --like      Test likelihood calculations.
    --likes F   Test row likelihood.
    --nb F      Run naive bayes on CSV file.

INPUT FORMAT
    Comma-separated values. First row defines column names. Uppercase
    names (Age, Weight) are numeric; lowercase (name) are symbolic.
    Suffixes: "!" class label, "X" ignore. Missing values: "?".

----------------------------------------------------------------------
CODING STANDARD

  Type Hints (single letter)
    i:instance t:table u:output_table r:row n:number p:probability
    s:string v:value k:key f:function d:delta j:index items:iterator

  Multiple Same-Type Params
    Base + suffix: nall, nh, n1, n2

  Class System
    UPPERCASE:metatable (SYM,NUM)  CamelCase:constructor (Sym,Num)
    lowercase:instance (data,col)

  Collision Avoidance
    file (not f, since f is function)

  Function Signatures
    Params before extra spaces; locals after:
      function sum(t,f,    n)   -- t,f:params; n:local
]]

local l = require"lib"
local o,isa,iter,csv,sel,BIG = l.o,l.isa,l.iter,l.csv,l.sel,l.BIG
local sqrt,exp,log,max = math.sqrt,math.exp,math.log,math.max
local the = {}

-- tyoes ---------------------------------------------------------------
local DATA,COLS,SYM = {_is="DATA"}, {_is="COLS"}, {_is="SYM"}
local NUM = {_is="NUM"}
local Data,Cols,Sym,Num,Col

local function Sym(n,s)
  return isa(SYM,{at=n or 0, txt=s or "", n=0, has={}}) end

local function Num(n,s)
  return isa(NUM,{at=n or 0, txt=s or "", n=0, mu=0, m2=0, sd=0}) end

function Col(n,s) return (s:find"^[A-Z]" and Num or Sym)(n,s) end

local function adds(items,t)
  t=t or Num();for v in iter(items or {})do t:add(v) end; return t end

function Data(s,items)
  return adds(items or {},isa(DATA,{txt=s or "",rows={},cols=nil}))end

function Cols(row,    all)
  all = l.kap(row, Col)
  return isa(COLS, {names=row, all=all,
    x = sel(all, function(c) return not c.txt:find"[!X]$" end),
    y = sel(all, function(c) return c.txt:find"!$" end)}) end

function clone(data,rows)
  return adds(rows, Data(data.txt, {data.cols.names})) end

-- add ------------------------------------------------------------------
function DATA.add(i,row)
  if not i.cols then i.cols=Cols(row) else
    i.rows[1+#i.rows] = row
    for _,col in pairs(i.cols.all) do col:add(row[col.at]) end end end

function SYM.add(i,v)
  if v~="?" then i.n=i.n+1; i.has[v]=1+(i.has[v] or 0) end end

function NUM.add(i,v,    d)
  if v~="?" then
    i.n=i.n+1; d=v-i.mu; i.mu=i.mu+d/i.n; i.m2=i.m2+d*(v-i.mu)
    i.sd = i.n<2 and 0 or sqrt(i.m2/(i.n-1)) end end

-- bayes ----------------------------------------------------------------
function SYM.like(i,v,prior,    n)
  n = (i.has[v] or 0) + the.k*(prior or 0)
  return max(1/BIG, n/(i.n + the.k + 1/BIG)) end

function NUM.like(i,v,    z,var)
  z=1/BIG; var=i.sd^2 + z
  return (1/sqrt(2*math.pi*var)) * exp(-((v - i.mu)^2)/(2*var)) end

function DATA.likes(i,row,nall,nh,    b4)
  b4 = (#i.rows + the.m)/(nall + the.m*nh)
  return log(b4) + l.sum(i.cols.x, function(c)
    return row[c.at]~="?" and log(c:like(row[c.at],b4)) or 0 end) end

local function nb(items,   all,klasses,n,nk,klass,train,seen,classify)
  klasses, n, nk = {}, 0, 0
  function klass(row) return row[all.cols.y[1].at] end
  function train(row) klasses[klass(row)]:add(row) end
  function seen(k)
    if not klasses[k] then
      nk=nk+1; klasses[k]=clone(all); klasses[k].txt=k end end
  function classify(row)
    return l.most(klasses,function(_,d)return d:likes(row,n,nk)end)end

  for row in iter(items) do
    if not all then all=Data("all",{row}) else
      seen(klass(row))
      if n > the.wait then print(classify(row), klass(row)) end
      n=n+1; train(row) end end end

-- demos ----------------------------------------------------------------
local eg={}

eg["-h"]    = function(_) print("\n"..help) end
eg["--the"]= function(_) print(o(the)) end
eg["--sym"]= function(_) print(o(adds({"a","a","a","b","c"},Sym())))end
eg["--num"]= function(_) print(o(adds({10,20,30,40}))) end
eg["--col"]= function(_) print(o(Col(1,"Age")), o(Col(2,"name"))) end

eg["--cols"]= function(_)
   print(o(Cols({"Name","Age","Weight-","Class!"}).y)) end

eg["--data"]= function(f) print(o(Data("",csv(f)).cols.y[1])) end

eg["--like"]= function(_,    num,sym)
  num=adds({10,20,30,40,50}); sym=adds({"a","a","a","b","c"},Sym())
  print(num:like(30), sym:like("a",0.5)) end

eg["--likes"]= function(f,    data)
  data=Data("",csv(f));print(data:likes(data.rows[1],#data.rows,2))end

eg["--nb"]= function(f) nb(csv(f)) end

-- main -----------------------------------------------------------------
for k,v in help:gmatch("(%S+)=(%S+)") do the[k]=l.cast(v) end
if arg[0] and arg[0]:find"nb" then
  for j,s in pairs(arg) do if eg[s] then eg[s](arg[j+1]) end end end

return {the=the, SYM=SYM, NUM=NUM, DATA=DATA, COLS=COLS,
        Sym=Sym, Num=Num, Data=Data, Cols=Cols, Col=Col, adds=adds,
        clone=clone, nb=nb, eg=eg}
```