binr.py Page 1/4

```python
#!/usr/bin/env python3 -B
# vim: ts=2:sw=2:sts=2:et
"""
binr.py : build rules via stochastic incremental XAI
(c) 2025, Tim Menzies, timm@ieee.org, mit-license.org

Options:

  -h          Show help.
  -b bins=4     Number of bins for discretization (int).
  -B Budget=30  Max rows to eval (int).
  -C CF=0.8     crossover rate
  -F F=0.3      scale factor between two nums.
  -e era=10     Number of rows in an era (int)
  -p p=2        Distance coefficient
  -r repeats=20 Number of experimental repeats (int).
  -s seed=42    Random number seed (int).
  -f file=../data/auto93.csv File to load (str).
"""
from math import floor,sqrt,cos,log,exp,pi
from typing import Any,Iterable
import fileinput,random,sys,re
rand = random.random

class obj(dict):
  "Structs with slots accessiable via x.slot. And pretty print."
  def __repr__(i): return "{" + ' '.join(f":{k} {o(i[k])}" for k in i) + "}"
  def __setattr__(i, k, v): i[k] = v
  def __getattr__(i, k):
    try: return i[k]
    except KeyError: raise AttributeError(k)

the = obj(bins=4, Budget=30, CF=.8, era=10, F=0.3, p=2, repeats=20, seed=42,
          file="../data/auto93.csv")


# types, upper case
QTY  = float | int
ATOM = QTY | str | bool
ROW  = list[ATOM]
ROWS = list[ROW]
NUM,SYM,TRI, COLS = obj, obj,obj,obj
COL  = NUM | SYM
COLS = list[list[COL]]
DATA = tuple[ROWS, COLS]

# ------------------------------------------------------------------------------
# Constructors, mixed case
def Sym(has=None) -> SYM:
  "Summarize symbol."
  return obj(it=Sym, n=0, has=has or {}, bins={})

def Num(mu=0,sd=1) -> NUM:
  "Summarize numbers."
  return obj(it=Num, n=0, mu=mu, sd=sd, m2=0, bins={})

def Tri(lo=0,mid=0.5,hi=1) -> TRI:  # TRI used for generation (no updates)
  return obj(it=Tri,n=0, lo=lo, mid=mid, hi=hi)

def Col(at=0, of=" ") -> COL:
  "Column in rows of data."
  it = (Num if of[0].isupper() else Sym)()
  it.at = at
  it.of = of
  it.best = str(of)[-1]!="-"
  return it

def Cols(names:list[str]) -> COLS:
  "Factory. Turns column names into columns."
  cols = [Col(at=i, of=s) for i,s in enumerate(names)]
  return obj(it=Cols, names=names,
             all = cols,
             x   = [col for col in cols if str(col.of)[-1] not in "+-X"],
             y   = [col for col in cols if str(col.of)[-1] in "+-"])

def Data(rows = None) -> DATA:
  "Summarize rows into columns."
  return adds(rows, obj(it=Data, n=0, rows=[], cols=None))

def clone(data, rows=None) -> DATA:
  "Mimic the structure of `data`. Optinally, add some roes."
  return adds(rows, Data([data.cols.names]))


```

binr.py Page 2/4

```python
# ------------------------------------------------------------------------------
def add(i: NUM | SYM | DATA,   # NOTE: TRI not supported (cant decrement lo,hi)
        item: Any,
        inc = 1) -> Any: # returns item
  "Add or subtract items from columns or data."
  if item=="?": return item
  i.n += inc
  if   i.it is Sym:
    i.has[item] = inc + i.has.get(item,0)
  elif i.it is Num:
    item = float(item)
    if inc < 0 and i.n < 2:
      i.n = i.mu = i.sd = i.m2 = 0
    else:
      d     = item - i.mu
      i.mu += inc * d / i.n
      i.m2 += inc * d * (item - i.mu)
      i.sd  = 0 if i.n < 2 else sqrt(max(0,i.m2)/(i.n - 1))
  elif i.it is Data:
    if i.cols:
      row = [add(c, item[c.at], inc) for c in i.cols.all]
      i.rows.append(row) if inc > 0 else i.rows.remove(row)
    else: i.cols = Cols(item)
  return item

def sub(i,item):
  "Subtract items."
  return add(i,item,-1)

def adds(items:Iterable = None, it=None ) -> obj: # returns it
  "Load many items into `it` (defeault is `Num()`)."
  it = it or Num()
  if str(items)[-4:]==".csv":
    with open(items, encoding="utf-8") as f:
      for line in f:
        if line: add(it, [s.strip() for s in line.split(",")])
  else: [add(it, item) for item in (items or [])]
  return it

# ------------------------------------------------------------------------------
def sample(i: TRI | SYM | NUM | list) -> list:
  "Sample a value from a TRI / Num / Sym / Data summary."
  if type(i)==list : return [sample(col) for col in i]
  if i.it is Num   : return irwinHall3(i.mu, i.sd)
  if i.it is Tri:
    denom = (i.hi - i.lo) if (i.hi - i.lo) != 0 else 1e-32
    p = (i.mid - i.lo) / denom
    u, v = rand(), rand()
    return i.lo + (i.hi - i.lo) * (min(u, v) + p * abs(u - v))
  if i.it is Sym:
    r = rand() * i.n
    for x, count in i.has.items():
      r -= count
      if r <= 0: return x
    return x

def mixes(data: list[COL], np=100) -> Data:
  "Return a new data containing `n` samples from data."
  any = lambda: random.choice(data.rows)
  return [mix(data, any(), any(), any()) for _ in range(np)]

def mix(data:DATA, a:ROW, b:ROW, c:ROW) -> ROW:
  "Mutate `a` by mixing items from `b,c`."
  def nump(z): return type(z) in [float,int]
  d = a[:]
  keep = random.randrange(len(a))
  for j,(A,B,C,col) in enumerate(zip(a,b,c,data.cols.all)):
    if j != keep and rand() < the.CF:
      d[j] = B if rand() < 0.5 else C
      if col.it is Num and nump(A) and nump(B) and nump(C):
        d[j] = wrap(col, A + the.F*(B - C))
  return d

def wrap(num,v):
  "Restrain `v` to the effective min,max range of `num`."
  lo,hi = num.mu - 3*num.sd, num.mu + 3*num.sd
  if v<lo: return hi - ((lo-v) % (hi-lo))
  if v>hi: return lo + ((v-hi) % (hi-lo))
  return v


```

binr.py Page 3/4

```python
# ------------------------------------------------------------------------------
def mid(i: COL | DATA) -> ATOM | ROW:
  "Return the expected value of `i`."
  if i.it is Num: return i.mu
  if i.it is Tri: return i.mid
  if i.it is Sym: return max(i.has, key=i.has.get)
  return [mid(col) for col in i.cols.all]

def shuffle(lst:list) -> list:
  "Shuffle `lst` in place."
  random.shuffle(lst); return lst

def irwinHall3(mu=0,sd=1) -> float:
  "Fast normal sampling: chatgpt.com/share/6935eb44-705c-8010-8782-454c0aff8a5e"
  return mu + sd * 2.0 * (rand() + rand() + rand() - 1.5)

def marsagliaPolar(mu=0,sd=1) -> float:
  "Slightly slower normal sampling."
  while 1:
    u,v = 2*rand()-1, 2*rand()-1
    s = u*u + v*v
    if 0 < s < 1: return mu + sd*u*sqrt(-2*log(s)/s)

def norm(num:Num, v:QTY) -> float:
  "Returns 0..1."
  return 1/(1+exp(-1.702 * (v- num.mu)/(num.sd + 1e-32))) if v != "?" else v

def bin(col:COL, v:ATOM) -> int | ATOM:
  "Returns 0..bins-1."
  return floor( the.bins * norm(col,v) ) if v!="?" and col.it is Num else v

def dist(src:Iterable) -> float:
  "Mankoski distance."
  d,n = 0,0
  for d1 in src:
    n += 1
    d += d1 ** the.p
  return (d/n) ** (1/the.p)

def disty(data:DATA, row:ROW) -> float:
  "Distance of `row` to `best` values in each goal column."
  return dist(abs(norm(col, row[col.at]) - col.best) for col in data.cols.y)

def distx(data:DATA, row1:ROW, row2:ROW) -> float:
  "Distance between `x` attributes of two rows."
  return dist(_aha(col, row1[col.at], row2[col.at]) for col in data.cols.x)

def _aha(col:COL, a:ATOM, b:ATOM) -> float:
  "If any unknowns, assume max distance."
  if a==b=="?": return 1
  if col.it is Sym : return a != b
  a,b = norm(col,a), norm(col,b)
  a = a if a != "?" else (0 if b>0.5 else 1)
  b = b if b != "?" else (0 if a>0.5 else 1)
  return abs(a - b)

# ------------------------------------------------------------------------------
def scoreGet(use, row:ROW) -> ROW:
  "Sum the score of the bins used by `row`."
  n = 0
  for num in use:
    if (v := row[num.at]) != "?":
      print(v, num, bin(num,v))
      if bin(num, v) == num.of:
        n += want(num)
      print(22)
  return n

def scorePut(data:DATA, row:ROW, score:QTY):
  "Increment the bins used by `row`."
  for x in data.cols.x:
    if (b := bin(x, row[x.at])) != "?":
      one = x.bins[b] = x.bins.get(b) or Num()
      one.at, one.of = x.at, b
      add(one, score)

def want(num): return num.mu + num.sd/sqrt(num.n)

def top(data):
  return sorted((num for x in data.cols.x for num in x.bins.values()),key=want)

def score(data:DATA, eps=0.05):
  "Guess next few scores using scores seen to date."
  best_score, best_row = 1e32, None
  rows = shuffle(data.rows)
  seen, model = set(), Data([data.cols.names])
  for j, row in enumerate(rows):
    if len(seen) >= the.Budget: break
    add(model, row)
    scorePut(model, row, disty(model, row))
    seen.add(id(row))
    if (j+1) % the.era == 0 and j < len(rows) - 100:
      use = top(model)[:5]
      candidate = min(rows[j+1:j+20], key=lambda r: scoreGet(use, r))
      seen.add(id(candidate))
      if (score := disty(model, candidate)) < best_score - eps:
        best_score, best_row = score, candidate
  return best_score


```

```python
# ---------------------------------------------------------------------------
def o(x):
  "Pretty print."
  if type(x) is type(o) : return x.__name__ + '()'
  if type(x) is float : return str(int(x)) if x == int(x) else f"{x:.2f}"
  if type(x) is list : return "["+('.'.join(o(y) for y in x))+"]"
  return str(x)

# ---------------------------------------------------------------------------
def go_h(_) -> None:
  print(__doc__)

def go__the(_) -> None:
  print(the)

def go_s(n: str) -> None:
  the.seed = float(n); random.seed(the.seed)

def go__sym(_) -> None:
  print(adds("aaaabbc",Sym()))

def go__num(_) -> None:
  print(adds(irwinHall3(10,2) for _ in range(10**3)))

def go__data(f = None) -> None:
  data = Data(f or the.file)
  print(data.cols.x[-1])
  print(len(data.rows),data.rows[1])

def go__disty(f = None):
  ys, data = Num(), Data(f or the.file)
  print(*[col.of for col in data.cols.all],"y",sep="\t")
  Y=lambda row: floor(100*disty(data,row))
  for r in sorted(data.rows,key=Y)[::20]:
    print(*r,Y(r),sep="\t")

def go__distx(f = None):
  xs, data = Num(), Data(f or the.file)
  print(*[col.of for col in data.cols.all],"x",sep="\t")
  X=lambda row1: floor(100*distx(data,row1, data.rows[0]))
  for r in sorted(data.rows,key=X)[::20]:
    print(*r,X(r),sep="\t")

def go__inc(f=None):
  data1 = Data(f or the.file)
  data2 = clone(data1)
  for row in data1.rows:
    add(data2,row)
    if len(data2.rows)==50: print(o(mid(data2)))
  print(o(mid(data2)))
  for row in data1.rows[::-1]:
    if len(data2.rows)==50: print(o(mid(data2)))
    sub(data2,row)

def f(x)    : return 1.61 + 2.1*x[0] - 3.5*(x[1]*2) + 4*(x[2]**3) - 5*(x[3]**4)
def fx(row) : print(obj(best=row, y=f(row)))

def go__random(_):
  eden = [Num(100,1), Num(20,5), Num(10,4), Num(3,2)]
  fx( min((sample(eden) for _ in range(1000)), key=f))

def go__hclimb(_):
  m,r    = 100,9
  model = [("X1",100,1), ("X2",20,5), ("X3",10,4), ("X4",3,2)]
  eden  = [Num(mu,sd) for _,mu,sd in model]
  data  = Data([[s for s,_,_ in model]] + [sample(eden) for _ in range(m)])
  for _ in range(r):
    tmp = clone(data, sorted(data.rows, key=f)[:m//2])
    fx(tmp.rows[0])
    data = clone(data, mixes(tmp,m))

def go__score(f= None):
  my   = lambda n: floor(100*n)
  data = Data(f or the.file)
  print(len(data.rows))
  ys   = adds(my(disty(data,row)) for row in data.rows)
  print(obj(mu=ys.mu,sd=ys.sd))
  print(*sorted(my(score(data)) for _ in range(the.repeats)))

_tests= {k:fun for k,fun in vars().items() if "go__" in k}

def go__all(_):
  for k,fun in _tests.items(): print("\n-----"+k); fun(_)

# ---------------------------------------------------------------------------
if __name__ == "__main__":
  for n, s in enumerate(sys.argv):
    if fn := vars().get(f"go{s.replace('-','_')}"):
      random.seed(the.seed)
      fn(sys.argv[n+1] if n < len(sys.argv)-1 else None)
```