

binr.py

Page 1/5

```

1 #!/usr/bin/env python3 -B
2 # vim: ts=2:sw=2:sts=2:et
3 """NAME
4     binr.py -- build rules via stochastic incremental XAI
5
6 SYNOPSIS
7     binr.py [OPTIONS] [ACTION]
8
9 DESCRIPTION
10    binr.py performs stochastic incremental Explainable AI (XAI) analysis.
11    It utilizes multi-objective optimization to discretize continuous attributes,
12    cluster rows, and generate rules explaining target variance.
13
14 DATA FORMAT
15    Input is CSV. Header (row 1) defines column roles via naming conventions:
16    [A-Z]*: Numeric (e.g. "Age"), [a-zA-Z]*: Symbolic (e.g. "job*").
17    +*: Maximize (e.g. "Pay+"). -: Minimize (e.g. "Cost-").
18    *X: Ignored (e.g. "id*"). ?: Missing value.
19
20 OPTIONS
21     -h Show help message and exit.
22     -b bins Number of bins for discretization (int, 4).
23     -B budget Max number of samples to consider for each row (int, 30).
24     -C CF Coverage rate for mixing samples (float, 0.8).
25     -F Factor scale between two numbers during mixing (float, 0.3).
26     -e era Rows in an era for incremental processing (int, 10).
27     -p D Distance coefficient for Minkowski distance (int, 2).
28     -r repeats Number of experimental repeats (int, 20).
29     -s seed Random number seed (int, 42).
30     -f file Path to input CSV file (str, ./data/auto93.csv").
31
32 ACTIONS
33     -data [f] Load dataset; print summary of columns and last row.
34     -disty [f] Print distances of rows to "best" goal values.
35     -distx [f] Print distances based on identifier (X) attributes.
36     -disty [f] Print distances based on identifier (Y) attributes.
37     -score [f] Run XAI scoring: guesses next scores via history.
38     -random Test stochastic sampling on generated Eden model.
39     -hcilm Test hill-climbing optimization on generated model.
40     -all Run all defined tests.
41
42 ALGORITHMS
43     Incremental Statistics
44         Means and standard deviations are updated online using Welford's
45         algorithm to ensure numerical stability.
46
47 Distance (Aha)
48     Row differences are calculated using Aha's similarity measure
49     adapted for the Minkowski metric (p=2). Missing values ("*")*
50     are assumed to have maximal distance (1.0).
51
52 Sampling
53     Normal distributions use Irwin–Hall or Marsaglia polar methods.
54     Skewed distributions are modeled using Triangular sampling.
55
56 Optimization (Mixtures)
57     Candidate generation uses strategies from Storn's Differential
58     Evolution (DE) to mix existing samples (extrapolating deltas).
59
60 AUTHOR
61     Tim Menzies (timm@ieee.org)
62
63 COPYRIGHT
64     (c) 2025, MIT License. """

```

binr.py

Page 2/5

```

66 from math import floor,sqrt,cos,log,exp,pi
67 from typing import Any,Iterable
68 import fileinput,random,sys,re
69 rand = random.random
70
71
72 class obj(dict):
73     "Strips with slots accessible via x.slot. And pretty print."
74     def __repr__(i): return "( " + ' '.join(f'{k}:[{v}]' for k in i) + " )"
75     def __setattr__(i, k, v): i[k] = v
76     def __getattribute__(i, k):
77         try: return i[k]
78         except KeyError: raise AttributeError(k)
79
80 the = obj(bins=4, Budget=30, CF=.8, era=10, F=0.3, p=2, repeats=20, seed=42,
81 file="data/autot93.csv")
82
83 # types, upper case
84 QTY = float | int
85 ATOM = QTY | str | bool
86 ROW = list[ATOM]
87 ROWS = list[ROW]
88 NUM,SYM,TRI, COLS = obj, obj,obj,obj,obj
89 COL = list[ATOM] # col TRO
90 COLS = list[list[COL]]
91 DATA = tuple[ROWS, COLS]
92
93 #
94 # Constructors, mixed case
95 def Sym(has_dots=None) -> SYM:
96     "Summarize symbol."
97     return obj(it=Sym, n=0, has=has or {}, bins={})
98
99 def Num(mu=0, sd=1) -> NUM:
100     "Summarize numbers."
101     return obj(it=Num, n=0, mu=mu, sd=sd, m2=0, bins={})
102
103 def Tri(lo=0, mid=0.5, hi=1) -> TRI:
104     "Used to sample from a skewed distribution but sum(adding not defined)."
105     return obj(it=Tri,n=0, lo=lo, mid=mid, hi=hi)
106
107 def Col(at=0, of="") -> COL:
108     "Column in rows of data."
109     it = (Num if of[0].isupper() else Sym)()
110     it.at = at
111     it.of = of
112     it.best = str(of)[-1]!="="
113     return it
114
115 def Cols(names:list[str]) -> COLS:
116     "Factory. Turns column names into columns."
117     cols = [Col(at=i, of=s) for i,s in enumerate(names)]
118     return obj(it=Cols, names=names,
119               all = cols,
120               x = [col for col in cols if str(col.of)[-1] not in "+-X"],
121               y = [col for col in cols if str(col.of)[-1] in "+-"])
122
123 def Data(rows = None) -> DATA:
124     "Summarize rows into columns."
125     return adds(rows, obj(it=Data, n=0, rows=[], cols=None))
126
127 def clone(data:DATA, rows=None) -> DATA:
128     "Mimic the structure of 'data'. Optimally, add some rows."
129     return adds(rows, Data([data.cols.names]))
130
131

```

binr.py

Page 3/5

```

1 def add(i: NUM | SYM | DATA,   # NOTE: TRI not supported (cant decrement lo,hi)
2     item: Any,
3     inc = 1) -> Any: # returns item
4     "Add increment items from columns or data."
5     if item=="": return item
6     i.n += inc
7     if i.iit is Sym: i.has[item] = inc + i.has.get(item,0)
8     elif i.iit is Num:
9         item = float(item)
10    if inc < 0 and i.n < 2:
11        i.n = i.mu + i.sd = i.m2 = 0
12    else:
13        d     = item - i.mu
14        i.mu += inc * d / i.n
15        i.m2 += inc * d * (item - i.mu)
16        i.n  = i.n + 1 if i.n < 2 else sqrt(max(0,i.m2)/(i.n - 1))
17    elif i.iit is Data:
18        if i.cols:
19            row = [add(c, item[c.at], inc) for c in i.cols.all]
20            i.rows.append(row) if inc > 0 else i.rows.remove(row)
21        else: i.cols = Cols(item)
22    return item
23
24
25 def sub(i: item):
26     "Subtract items."
27     return add(i, item, -1)
28
29 def adds(items: Iterable = None, it=None) -> obj: # returns it
30     "Load many items into a TRI/Num/Sym/Data summary."
31     if it is None: it = Num()
32     if str(items)[-4:]==".csv":
33         with open(items, encoding="utf-8") as f:
34             for line in f:
35                 if line: add(it, [s.strip() for s in line.split(",")])
36     else: [add(it, item) for item in (items or [])]
37     return it
38
39
40 def sample(i: TRI | SYM | NUM | list) -> list:
41     "Sample from a TRI/Num/Sym/Data summary."
42     if type(i)==list: return [sample(col) for col in i]
43     if i.iit is Num: return irwinHill3(i.mu, i.sd)
44     if i.iit is Tri:
45         p = (i.mid - i.lo) / (i.hi - i.lo + le-32)
46         u, v = rand(), rand()
47         return (i.lo + (i.hi - i.lo) * (min(u, v) + p * abs(u - v)))
48     if i.iit is Sym:
49         r = rand() * i.n
50         for x, count in i.has.items():
51             r -= count
52             if r <= 0: return x
53         return x # should never get here.
54
55 def mixtures(data: list[COL], np=100) -> Data:
56     "Return n samples nonparametrically: add the delta between two items to a third."
57     any = lambda: random.choice(data.rows)
58     return [mixture(data, any(), any(), any()) for _ in range(np)]
59
60
61 def mixture(data:DATA, a:ROW, b:ROW, c:ROW) -> ROW:
62     "Mutate 'a' by mixing items from 'b,c'."
63     def nump(z): return type(z) in [float,int]
64     d = a[:] keep = random.randrange(len(a))
65     for i, (A,B,C,col) in enumerate(zip(a,b,c,data.cols.all)):
66         if z[i] < keep and rand() < the.CF:
67             d[i] = B if rand() < 0.5 else C
68         if col.it is Num and nump(A) and nump(B) and nump(C):
69             d[i] = wrap(col, A + the.F*(B - C))
70     return d
71
72
73 def wrap(v, v):
74     "Restrain 'v' to the effective min,max range of 'num'."
75     lo,hi = num.mu - 3*num.sd, num.mu + 3*num.sd
76     if v<lo: return hi - ((lo-v) % (hi-lo))
77     if v>hi: return lo + ((v-hi) % (hi-lo))
78     return v
79
80
81

```

## binr.py

Page 4/5

```

208 #def mid(i: COL | DATA) -> ATOM | ROW:
209     "Return the expected value of 'i'."
210     if i.iit is Num: return i.mu
211     if i.iit is Tri: return i.mid
212     if i.iit is Sym: return max(i.has, key=i.has.get)
213     return [mid(col) for col in i.cols.all]
214
215 def shuffle(lst:list) -> list:
216     "Shuffle 'lst in place."
217     random.shuffle(lst); return lst
218
219 def irwinHall3(mu=0,sd=1) -> float:
220     "Fast normal sampling: chartpit.com/share/93Seb44-70Sc-8010-8782-454c0uffa5e"
221     return mu + sd * 2.0 * (rand() + rand() - 1.5)
222
223 def marsagliaPolar(mu=0,sd=1) -> float:
224     "Slightly slower normal sampling."
225     while True:
226         u,v = 2*rand() - 1, 2*rand() - 1
227         s = u*u + v*v
228         if 0 < s < 1: return mu + sd*u*sqrt(-2*log(s)/s)
229
230 def norm(i:NUM, v:QTY) -> float:
231     "Returns 0..1."
232     return 1/(1+exp(-1.7*(v-i.mu)/(i.sd+le-32))) if i.iit is Num and v!="?" else v
233
234 def bin(col:COL, v:ATOM) -> int | ATOM:
235     "Returns 0..bins-1."
236     return floor(the.bins * norm(col, v)) if col.iit is Num and v!="?" else v
237
238 def dist(src:Iterable) -> float:
239     "Minkowski distance."
240     d,n = 0,0
241     for z in src:
242         n += 1
243         d += z ** the.p
244     return (d/n) ** (1/the.p)
245
246 def disty(data:DATA, row:ROW) -> float:
247     "Distance of 'row' to 'best' value in each goal column."
248     return dist(abst(abs(norm(col, row[col.at])) - col.best) for col in data.cols.y)
249
250 def distx(data:DATA, row1:ROW, row2:ROW) -> float:
251     "Distance between 'x' attributes of two rows."
252     return dist(_aha(col, row1[col.at], row2[col.at]) for col in data.cols.x)
253
254 def _aha(col:COL, a:ATOM, b:ATOM) -> float:
255     "If any unknowns, assume max distance."
256     if a==b=="?": return 1
257     if col.iit is Sym: return a != b
258     a,b = norm(col,a), norm(col,b)
259     a = a if a>0.5 else 0 if b>0.5 else 1
260     b = b if b>0.5 else 0 if a>0.5 else 1
261     return abs(a - b)
262
263 #
264 def __scoreGet(model, use, row:ROW) -> ROW:
265     "Compute the score of the bins used by 'row'."
266     n = 0
267     for slot in use:
268         if (v := row[slot.at]) != "?":
269             if bin(model.cols.all[slot.at], v) == slot.of:
270                 n += want(slot)
271     return n
272
273 def scorePut(data:DATA, row:ROW, score:QTY):
274     "Increment the bins used by 'row'."
275     for x in data.cols.x:
276         if (b := bin(x[row.at])) != "?":
277             one = x.bins[b]; X.bins.get(b) or Num()
278             one.at.one.of = x.at, b
279             add(one, score)
280
281 def want(slot): return slot.mu + slot.sd/sqrt(slot.n)
282
283 def top(data):
284     return sorted((slot for x in data.cols.x for slot in x.bins.values()),key=want)
285
286 dump={}
287
288 def score(data:DATA, eps=0.05):
289     "Guess next few scores using scores seen to date."
290     best_score, best_row = le32, None
291     rows = shuffle(data.rows)
292     seen, model = set(), Data([data.cols.names])
293     for i,(seen,r) in enumerate(rows):
294         if len(seen) >= the.Budget: break
295         add(model, r)
296         scorePut(model, r, disty(model, r))
297         seen.add(id(r))
298         if (j:=i) % the.era == 0 and j < len(rows) - 100:
299             use = top(data)
300             use = use[-int(sqrt(len(use)))]
301             for slot in use:
302                 k=(slot.at, slot.of)
303                 dump[k] = 1 + dump.get(k,0)
304                 candidate = min(rows[i+1:i+100], key=lambda r: scoreGet(model,use, r))
305                 seen.add(id(candidate))
306                 if (score := disty(model, candidate)) < best_score - eps:
307                     best_score, best_row = score, candidate
308
309     return best_score
310

```

## binr.py

Page 5/5

```

311 #
312 def o(x):
313     "Pretty print."
314     if type(x) is type(o): return x.__name__ + '()' # type(x) is type(o) : return x.__name__ + '()' # type(x) is float : return str(int(x)) if x == int(x) else f'{x:.2f}' # type(x) is list : return "[" + ','.join(o(y) for y in x) + "]"
315     return str(x)
316
317 #
318 def go_h(_) -> None:
319     print(__doc__)
320
321 def go_the(_) -> None:
322     print(the)
323
324 def go_s(n: str) -> None:
325     the.seed = float(n); random.seed(the.seed)
326
327 def go_sym(_) -> None:
328     print(adds("aaabb",Sym()))
329
330 def go_num(_) -> None:
331     print(adds(irwinHall3(10,2) for _ in range(10**3)))
332
333 def go_data(f = None) -> None:
334     data = Data(f or the.file)
335     print(data.cols.x[-1])
336     print(len(data.rows), data.rows[1])
337
338 def go_disty(f = None):
339     vs, data = Num(), Data(f or the.file)
340     print("[col.of for col in data.cols.all],", "y", sep="\t")
341     Y=lambda row: floor(100*disty(data, row))
342     for r in sorted(data.rows, key=Y)[:20]:
343         print(r,Y(r),sep="\t")
344
345 def go_distx(f = None):
346     xs, data = Num(), Data(f or the.file)
347     print("[col.of for col in data.cols.all],", "x", sep="\t")
348     X=lambda row: floor(100*distx(data, row, data.rows[0]))
349     for r in sorted(data.rows, key=X)[:20]:
350         print(r,X(r),sep="\t")
351
352 def go_inc(f=None):
353     data1 = Data(f or the.file)
354     data2 = clone(data1)
355     for row in data1.rows:
356         add(data2, row)
357         if len(data2.rows)==50: print(o(mid(data2)))
358     print(o(mid(data2)))
359     for row in data1.rows[:-1]:
360         if len(data2.rows)==50: print(o(mid(data2)))
361         sub(data2, row)
362
363 def fx(x) : return 1.61 + 2.1*x[0] - 3.5*(x[1]*x2) + 4*(x[2]**3) - 5*(x[3]**4)
364 def fx(row) : print(obj(best+row, y=f(row)))
365
366 def go_random():
367     eden = [Num(100,10), Num(20,5), Num(10,4), Num(3,2)]
368     fx( min((sample(eden) for _ in range(1000)), key=f))
369
370 def go_hclimb():
371     m = 100,8
372     data = [Num(*X1*,100,10), (*X2*,20,5), (*X3*,10,4), (*X4*,3,2)]
373     eden = [Num(mu,sd) for _,mu,sd in model]
374     data = Data([(s for s,_ in model)] + [sample(eden) for _ in range(m)])
375     for _ in range(r):
376         tmp = clone(data, sorted(data.rows, key=f)[:m//2])
377         fx(tmp.rows[0])
378         data = clone(data, mixtures(tmp,m))
379
380 def go_score(f = None):
381     my = lambda n: floor(100*n)
382     data = Data(f or the.file)
383     print(len(data.rows))
384     for (obj, (mu,sd)) in zip(data.rows, (row[0].mean(mu,sd), row[0].sd(mu,sd)))
385     print(obj(mu,sd))
386     print("sorted(my(score(data)) for _ in range(the.repeats)))")
387     print(sorted((n,k) for k,n in dump.items()))
388
389 def __tests():
390     _tests = {k:fun for k,fun in vars().items() if "go_" in k}
391
392 def go_all():
393     for k,fun in _tests.items():
394         if k != "go_all": print("\n-----+" + k); random.seed(the.seed); fun_()
395
396 #
397 if __name__ == "__main__":
398     for n, s in enumerate(sys.argv):
399         if fn := vars().get(f"go_{s.replace('-', '_')}"):
400             random.seed(the.seed)
401             fn(sys.argv[n+1] if n < len(sys.argv)-1 else None)
402

```