

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 at-
12    tributes, 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-z]*: Symbolic (e.g. "Job").
17    "+": Maximize (e.g. "Pay*").    "-": Minimize (e.g. "Cost-*").
18    "X": Ignored (e.g. "idx*").    "?" : 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 rows to evaluate during scoring (int, 30).
24    -C CF       Crossover rate for mixing samples (float, 0.8).
25    -F F        Scale factor between two numbers during mixing (float, 0.3).
26    -c era      Rows in an era for incremental processing (int, 10).
27    -p p        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    --dist [f] Print distance of rows to "best" goal values.
35    --distx [f] Print distances based on independent (X) attributes.
36    --inc [f] Test incremental loading (Welford's) by adding/subbing rows.
37    --score [f] Run XAI scoring; guesses next scores via history.
38    --random    Test stochastic sampling on generated Eden model.
39    --hclimb   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.***
65
66

```

binr.py

Page 2/5

```

67 from math import floor, sqrt, cos, log, exp, pi
68 from typing import Any, Iterable
69 import fileinput, random, sys, re
70 rand = random.random
71
72 class obj(dict):
73     """Structs with slots accessible via x.slot. And pretty print."
74     def __repr__(i): return "[" + ", ".join(f"{k} {o[i[k]]}" for k in i) + "]"
75     def __setattr__(i, k, v): i[k] = v
76     def __getattr__(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/auto93.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
89 COL = NUM | SYM # not TRI
90 COLS = list[list[COL]]
91 DATA = tuple[ROWS, COLS]
92
93 -----
94 # Constructors, mixed case
95 def Sym(has:dist=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 (sub)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 "+-X"])
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'. Optionally, add some rows."
129     return adds(rows, Data([data.cols.names]))
130
131

```

binr.py

Page 3/5

```

132 #
133 def add(i: NUM | SYM | DATA, # NOTE: TRI not supported (cant decrement lo,hi)
134        item: Any,
135        inc = 1) -> Any: # returns item
136     """Add or subtract items from columns or data."
137     if item=="?": return item
138     i.n += inc
139     if i.it is Sym: i.has[item] = inc + i.has.get(item, 0)
140     elif i.it is Num:
141         item = float(item)
142         if inc < 0 and i.n < 2:
143             i.n = i.mu = i.sd = i.m2 = 0
144         else:
145             d = item - i.mu
146             i.mu += inc * d / i.n
147             i.m2 += inc * d * d / (item - i.mu)
148             i.sd = 0 if i.n < 2 else sqrt(max(0, i.m2) / (i.n - 1))
149     elif i.it is Data:
150         row = [add(c, item[c.at], inc) for c in i.cols.all]
151         i.rows.append(row) if inc > 0 else i.rows.remove(row)
152         else: i.cols = Cols(item)
153     return item
154
155 def sub(i, item):
156     """Subtract items."
157     return add(i, item, -1)
158
159 def adds(items:Iterable = None, it=None) -> obj: # returns it
160     """Load many items into 'it' (default is 'Num()')."
161     it = it or Num()
162     if str(items)[-4:]=="*.csv":
163         with open(items, encoding="utf-8") as f:
164             for line in f:
165                 if line: add(it, [s.strip() for s in line.split(",")])
166     else: [add(it, item) for item in (items or [])]
167     return it
168
169 -----
170 def sample(i: TRI | SYM | NUM | list) -> list:
171     """Sample a value from a TRI / Num / Sym / Data summary."
172     if type(i)==list: return [sample(col) for col in i]
173     if i.it is Num: return irwinHall3(i.mu, i.sd)
174     if i.it is Tri:
175         p = (i.mid - i.lo) / (i.hi - i.lo + 1e-32)
176         u, v = rand(), rand()
177         return i.lo + (i.hi - i.lo) * (min(u, v) + p * abs(u - v))
178     if i.it is Sym:
179         r = rand() * i.n
180         for x, count in i.has.items():
181             if r <= 0: return x
182             return x # should never get here.
183
184 def mixtures(data: list[COL], np=100) -> Data:
185     """Return 'n' samples nonparametrically: add the delta between two items to a third."
186     any = lambda: random.choice(data.rows)
187     return [mixture(data, any(), any(), any()) for _ in range(np)]
188
189 def mixture(data:DATA, a:ROW, b:ROW, c:ROW) -> ROW:
190     """Mutate 'a' by mixing items from 'b,c'."
191     def nump(z): return type(z) in [float, int]
192     d = a[z]
193     keep = random.randrange(len(a))
194     for j, (A,B,C,col) in enumerate(zip(a,b,c,data.cols.all)):
195         if j != keep and rand() < the.CF:
196             d[j] = B if rand() < 0.5 else C
197         if col.it is Num and nump(A) and nump(B) and nump(C):
198             d[j] = wrap(col, A + the.F*(B - C))
199     return d
200
201 def wrap(num,v):
202     """Restrain 'v' to the effective min,max range of 'num'."
203     lo,hi = num.mu - 3*num.sd, num.mu + 3*num.sd
204     if v<lo: return hi - ((lo-v) % (hi-lo))
205     if v>hi: return lo + ((v-hi) % (hi-lo))
206     return v
207
208

```

```

208 #
209 def mid(i: COL | DATA) -> ATOM | ROW:
210     "Return the expected value of 'i'."
211     if i.it is Num: return i.mu
212     if i.it is Tri: return i.mid
213     if i.it is Sym: return max(i.has, key=i.has.get)
214     return [mid(col) for col in i.cols.all]
215
216 def shuffle(lst:list) -> list:
217     "Shuffle 'lst' in place."
218     random.shuffle(lst); return lst
219
220 def irwinHall3(mu=0,sd=1) -> float:
221     "Fast normal sampling: chatgpt.com/share/6935eb44-705c-8010-8782-454c0af8a5c"
222     return mu + sd * 2.0 * (rand() + rand() + rand() - 1.5)
223
224 def marsagliaPolar(mu=0,sd=1) -> float:
225     "Slightly slower normal sampling."
226     while 1:
227         u,v = 2*rand()-1, 2*rand()-1
228         s = u*u + v*v
229         if 0 < s < 1: return mu + sd*u*sqrt(-2*log(s)/s)
230
231 def norm(it:NUM, v:QTV) -> float:
232     "Returns 0.1."
233     return 1/(1+exp(-1.7*(v-i.mu)/(i.sd+1e-32))) if i.it is Num and v!="?" else v
234
235 def bin(col:COL, v:ATOM) -> int | ATOM:
236     "Returns 0.bins-1."
237     return floor(the.bins * norm(col,v)) if col.it is Num and v!="?" else v
238
239 def dist(src:Iterable) -> float:
240     "Mankoski distance."
241     d,n = 0,0
242     for z in src:
243         n += 1
244         d += z ** the.p
245     return (d/n) ** (1/the.p)
246
247 def disty(data:DATA, row:ROW) -> float:
248     "Distance of 'row' to 'best' values in each goal column."
249     return dist(abs(norm(col, row[col.at]) - col.best) for col in data.cols.y)
250
251 def distx(data:DATA, row1:ROW, row2:ROW) -> float:
252     "Distance between 'x' attributes of two rows."
253     return dist(_aha(col, row1[col.at], row2[col.at]) for col in data.cols.x)
254
255 def _aha(col:COL, a:ATOM, b:ATOM) -> float:
256     "If any unknowns, assume max distance"
257     if a==b=="?": return 1
258     if col.it is Sym: return a != b
259     a,b = norm(col,a), norm(col,b)
260     a = a if a != "?" else (0 if b>0.5 else 1)
261     b = b if b != "?" else (0 if a>0.5 else 1)
262     return abs(a - b)
263
264 #
265 def scoreGet(model, use, row:ROW) -> ROW:
266     "Sum the score of the bins used by 'row'."
267     n = 0
268     for slot in use:
269         if (v := row[slot.at]) != "?:
270             if bin(model.cols.all[slot.at], v) == slot.of:
271                 n += want(slot)
272     return n
273
274 def scorePut(data:DATA, row:ROW, score:QTV):
275     "Increment the bins used by 'row'."
276     for x in data.cols.x:
277         if (b := bin(x, row[x.at])) != "?:
278             one = x.bins[b] = x.bins.get(b) or Num()
279             one.at, one.of = x.at, b
280             add(one, score)
281
282 def want(slot): return slot.mu + slot.sd/sqrt(slot.n)
283
284 def top(data):
285     return sorted((slot for x in data.cols.x for slot in x.bins.values()),key=want)
286
287 dump={}
288
289 def score(data:DATA, eps=0.05):
290     "Guess next few scores using scores seen to date."
291     best_score, best_row = 1e32, None
292     rows = shuffle(data.rows)
293     seen, model = set(), Data([data.cols.names])
294     for j, row in enumerate(rows):
295         if len(seen) >= the.Budget: break
296         add(model, row)
297         scorePut(model, row, disty(model, row))
298         seen.add(id(row))
299         if (j+1) % the.era == 0 and j < len(rows) - 100:
300             use = top(model)
301             use = use[-int(sqrt(len(use))):]
302             for slot in use:
303                 k=(slot.at, slot.of)
304                 dump[k] = 1 + dump.get(k,0)
305             candidate = min(rows[j+1:j+100], key=lambda r: scoreGet(model,use, r))
306             seen.add(id(candidate))
307             if (score := disty(model, candidate)) < best_score - eps:
308                 best_score, best_row = score, candidate
309     return best_score
310
311

```

```

311 #
312 def o(k):
313     "Pretty print."
314     if type(x) is type(o): return x.__name__ + '()'
315     if type(x) is float: return str(int(x)) if x == int(x) else f"{x:.2f}"
316     if type(x) is list: return "["+' '.join(o(y) for y in x)+"]"
317     return str(x)
318
319 #
320 def go_h(_) -> None:
321     print(_doc_)
322
323 def go_the(_) -> None:
324     print(the)
325
326 def go_s(n: str) -> None:
327     the.seed = float(n); random.seed(the.seed)
328
329 def go_sym(_) -> None:
330     print(adds("aaabbc",Sym()))
331
332 def go_num(_) -> None:
333     print(adds(irwinHall3(10,2) for _ in range(10**3)))
334
335 def go_data(f = None) -> None:
336     data = Data(f or the.file)
337     print(data.cols.x[-1])
338     print(len(data.rows),data.rows[1])
339
340 def go_disty(f = None):
341     ys, data = Num(), Data(f or the.file)
342     print("["col.of for col in data.cols.all], "y",sep="")
343     Y=lambda row: floor(100*distx(data,row))
344     for r in sorted(data.rows,key=Y)[:20]:
345         print("r,Y(r),sep=")
346
347 def go_distx(f = None):
348     xs, data = Num(), Data(f or the.file)
349     print("["col.of for col in data.cols.all], "x",sep="")
350     X=lambda row1: floor(100*distx(data,row1, data.rows[0]))
351     for r in sorted(data.rows,key=X)[:20]:
352         print("r,X(r),sep=")
353
354 def go_inc(f=None):
355     datal = Data(f or the.file)
356     data2 = clone(datal)
357     for row in datal.rows:
358         add(data2,row)
359     if len(data2.rows)==50: print(o(mid(data2)))
360     print(o(mid(data2)))
361     for row in datal.rows[1:-1]:
362         if len(data2.rows)==50: print(o(mid(data2)))
363         sub(data2,row)
364
365 def f(x) : return 1.61 + 2.1*x[0] - 3.5*(x[1]**2) + 4*(x[2]**3) - 5*(x[3]**4)
366 def fx(row) : print(obj(best=row, y=f(row)))
367
368 def go_random():
369     eden = [Num(100,10), Num(20,5), Num(10,4), Num(3,2)]
370     fx(min((sample(eden) for _ in range(1000)), key=f))
371
372 def go_hclimb():
373     m,r = 100,9
374     model = [{"X1",100,10}, {"X2",20,5}, {"X3",10,4}, {"X4",3,2}]
375     eden = [Num(mu,sd) for _,mu,sd in model]
376     data = Data([s for s,_,_ in model]) + [sample(eden) for _ in range(m)]
377     for _ in range(n):
378         tmp = clone(data, sorted(data.rows, key=f)[:m//2])
379         fx(tmp.rows[0])
380         data = clone(data, mixtures(tmp,m))
381
382 def go_score(f= None):
383     my = lambda n: floor(100*n)
384     data = Data(f or the.file)
385     print(len(data.rows))
386     ys = adds(my(disty(data,row)) for row in data.rows)
387     print(obj(mu=ys,mu,sd=ys.sd))
388     print("sorted(my(score(data)) for _ in range(the.repeats)))
389     print(sorted((n,k) for k,n in dump.items()))
390
391 _tests= {k:fun for k,fun in vars().items() if "go_" in k}
392
393 def go_all():
394     for k,fun in _tests.items():
395         if k != "go_all": print("n----- "+k); random.seed(the.seed); fun(_)
396
397 #
398 if __name__ == "__main__":
399     for n, s in enumerate(sys.argv):
400         if fn := vars().get(f"go[{s.replace('-', '_')}]"):
401             random.seed(the.seed)
402             fn(sys.argv[n+1] if n < len(sys.argv)-1 else None)

```