

## xai4.py

Page 1/3

```

1 #!/usr/bin/env python3 -B
2 """xai.py: explainable multi-objective optimization
3 (c) 2025 Tim Menzies, MIT license"""
4 import ast,sys,random
5 from math import sqrt,exp,floor
6 from types import SimpleNamespace as obj
7
8 ATOM = str | int | float
9 ROW = list[ATOM]
10 ROWS = list[ROW]
11 NUM, SYM, DATA = obj,obj,obj
12 COL = NUM | SYM
13 THING = COL | DATA
14
15 BIG=1e32
16 the=obj(bins=7, budget=30, seed=1)
17
18 ### Constructors -----
19 def Sym(): return obj(it=SYM, n=0, has="")
20 def Num(): return obj(it=NUM, n=0, mu=0, m2=0)
21
22 def Col(at=0, txt=""):
23     col = (Num if txt[0].isupper() else Sym)()
24     col.at, col.txt, col.best = at, txt, 0 if txt[-1]=="-" else 1
25     return col
26
27 def Cols(names): # (list[str]) -> Cols
28     cols = [Col(a,s) for n,s in enumerate(names)]
29     return obj(it=cols, names=names, all=cols,
30               x=[col for col in cols if col.txt[-1] not in "+-X"],
31               y=[col for col in cols if col.txt[-1] in "+-X"])
32
33 def Data(rows=None):
34     return adds(rows, obj(it=DATA, rows=[], n=0, cols=None, _centroid=None))
35
36 def clone(data, rows=None): return adds(rows, Data([data.cols.names]))
37
38 ### Functions -----
39 def adds(src, l=None): # (src:Iterable, ?l) -> l
40     l = 1 or Num() # [add(i,v) for v in src]; return l
41
42 def sub(i, v): return add(i, v, inc=False)
43
44 def add(i, v, inc=True):
45     if v!="":
46         if Data is l.it and not i.cols: i.cols = Cols(v) # initializing, not adding
47         else:
48             i.n += 1 # adding
49             if Sym is l.it: i.has[v] = inc + i.has.get(v,0)
50             elif Num is l.it: d = v-i.mu; i.mu += inc*d/i.n; i.m2 += inc*d*(v-i.mu)
51             else:
52                 l._centroid = None # old centroid now out of date
53                 [add(col,v[col.at],inc) for col in i.cols.all] # recursive add to cols
54                 (i.rows.append if inc else i.rows.remove)(v) # handle row storage
55             return v # convention: always return the thing being added
56
57 def norm(num,n):
58     z = (n - num.mu) / sd(num); return 1 / (1 + exp(-1.7 * max(-3, min(3, z))))
59
60 def sd(num):
61     return 1/BIG + (0 if num.n < 2 else sqrt(max(0,num.m2)/(num.n - 1)))
62
63 def mid(col): return col.mu if Num is col.it else max(col.has, key=col.has.get)
64
65 def mids(data):
66     data._centroid = data._centroid or [mid(col) for col in data.cols.all]
67     return data._centroid
68
69 def disty(data, row):
70     ys = data.cols.y
71     return sqrt(sum(abs(norm(y,row[y.at]) - y.best)**2 for y in ys) / len(ys))
72
73 def distx(data, row1,row2):
74     xs = data.cols.x
75     return sqrt(sum(_aha(x, row1[x.at], row2[x.at])**2 for x in xs) / len(xs))
76
77 def _aha(col,u,v):
78     if u==v=="": return 1
79     if Sym is col.it: return u != v
80     u,v = norm(col,u), norm(col,v)
81     u = u if u != "" else (0 if v>0.5 else 1)
82     v = v if v != "" else (0 if u>0.5 else 1)
83     return abs(u - v)
84
85 def peaking(data,rows): # best if rows arrived shuffled
86     d = clone(data, row=[the.warm]) # all rows labelled by this function
87     a,z = clone(data),clone(data) # best, rest labelled rows
88     x = lambda r: distx(d,r,mids(a)) - distx(d,r,mids(z)) # <0 if closest to best
89     y = lambda r: disty(d,r) # distance of goals to "heaven" (best values)
90     d.rows.sort(key=y)
91     adds(d.rows[:the.warm/2], a)
92     adds(d.rows[the.warm/2:], z)
93     for r in rows[the.warm:]:
94         if d.n >= the.budget: break
95         elif x(r) < 0:
96             add(d, add(a,r))
97         if a.n > sqrt(d.n): # too many best things
98             a.rows.sort(key=y)
99             add(z, sub(a, a.rows[-1])) # demote worse row in best to rest
100     d.rows.sort(key=x)
101     return obj(model=x, labelled=d)
102
103 ### Cutting -----
104 def score(num): return num.mu + sd(num) / (sqrt(num.n) + 1/BIG)
105
106 def cut(data, rows):
107     all_bins = (b for col in data.cols.x for b in cuts(col, rows, data))
108     return min(all_bins, key=lambda b: score(b.y), default=None)
109
110 def cuts(col, rows, data):
111     d, xys = [], [(r[col.at], disty(data, r)) for r in rows if r[col.at]!=""]
112     for x, y in sorted(xys):
113         k = x if Sym is col.it else floor(the.bins * norm(col, x))
114         if k not in d: d[k] = obj(at=col.at, txt=col.txt, xlo=x, xhi=x, y=Num())
115         add(d[k].y, y)
116         d[k].xhi = x
117     return _complete(col, sorted(d.values(), key=lambda b: b.xlo))
118
119 def _complete(col, lst):
120     if Num is col.it:
121         for n, b in enumerate(lst):
122             b.xlo = lst[n-1].xhi if n > 0 else -BIG
123             b.xhi = lst[n+1].xlo if n < len(lst)-1 else BIG
124         return lst
125
126 ### Main -----
127 def select(rule, row):
128     if (x:=row[rule.at]) == "" or rule.xlo == rule.xhi == x: return True
129     return rule.xlo < x < rule.xhi
130

```

## xai4.py

Page 2/3

```

131 def xai(data):
132     print(o(the))
133     print(*data.cols.names)
134     def go(rows, lvl=0, prefix=""):
135         ys = Num(); rows.sort(key=lambda row: add(ys, disty(data, row)))
136         print(f"[{o(rows)[len(rows)/2]}]: {o(mu=ys.mu, n=ys.n, sd=sd(ys)):25s} {prefix}")
137         if rule := cut(data, rows):
138             now = [row for row in rows if select(rule, row)]
139             if 4 < len(now) < len(rows):
140                 go(now, lvl + 1, f"[{...} *{lvl}|{rule.txt}| {o(rule.xlo)}..{o(rule.xhi)}]")
141         go(data.rows, 0)
142
143 def six(data):
144     seen = clone(data)
145     unique=set()
146     def go(rows, lvl=0, prefix=""):
147         ys = Num(); rows.sort(key=lambda row: add(ys, disty(data, row)))
148         some = shuffle(rows)[:the.budget]
149         for row in some:
150             add(seen, row)
151             unique.add(tuple(row))
152         if rule := cut(seen, some):
153             now = [row for row in rows if select(rule, row)]
154             if 4 < len(now) < len(rows):
155                 return go(now, lvl + 1, f"[{...} *{lvl}|{rule.txt}| {o(rule.xlo)}..{o(rule.xhi)}]")
156             return int(100*ys.mu)
157     return go(data.rows, 0)
158

```

## xai4.py

Page 3/3

```

159 ## Lib -----
160 def o(v=None, dec=2,**d):
161     isa = isinstance
162     if d: v=d
163     if isa(v, (int, float)): return f"{round(v,dec):}"
164     if isa(v, list): return f"[{','.join(o(k,dec) for k in v)}]"
165     if isa(v, tuple): return f"({','.join(o(k,dec) for k in v)})"
166     if callable(v): return v._name
167     if hasattr(v, "_dict_"): v = vars(v)
168     if isa(v,(dict,tuple)): return f"{{'{}'.'.join(f'{k}|{o(v[k],dec)}' for k in v) *}}}"
169     return str(v)
170
171 def coerce(s):
172     try: return int(s)
173     except Exception as _:
174         try: return float(s)
175         except Exception as _:
176             s=s.strip()
177             return ("true":True, "false":False).get(s,s)
178
179 def csv(fileName):
180     with open(fileName,encoding="utf-8") as f:
181         for i in f:
182             if (l:=l.split("%")[0]).strip():
183                 yield (coerce(x) for x in l.split(","))
184
185 def shuffle(lst): return random.shuffle(lst); return lst
186
187 # -----
188 def go_h(_=None):
189     "show help"
190     print(_doc_+"\n\nOptions:\n")
191     for k,f in globals().items():
192         if k.startswith("go_") and f._doc_:
193             left, right = f._doc_.split("\n")
194             left = k[2:].replace("_","-") + " " + left.strip()
195             d = f._defaults_
196             default = f"(default {d[0]})" if d else ""
197             print(f"  {left:15} {right.strip()}{default}")
198
199 def go_s(s=1):
200     "INT: set random SEED"
201     the.seed = coerce(s); random.seed(the.seed)
202
203 def go_b(b=5):
204     "INT: set number of BINS used on discretization"
205     the.bins = coerce(s)
206
207 def go_n(n=40):
208     "INT: set BUDGET for rows labelled each round"
209     the.budget = coerce(s)
210
211 def go_all(file="data.csv"):
212     "FILE: run all actions that use a FILE"
213     for k,f in globals().items():
214         if k.startswith("go_") and k != "go_all":
215             print(f"u{f.__name__}"); f._doc_ = f"FILE: {f.__doc__}"; f._name_ = f"u{f.__name__}"; f._defaults_ = {}
216
217 def go_csv(file="data.csv"):
218     "FILE: test csv loading"
219     for n,row in enumerate(csv(file)):
220         if n % 40 == 0: print(n, row)
221
222 def go_data(file="data.csv"):
223     "FILE: test adding columns from file"
224     data = Data(csv(file))
225     print(*data.cols.names)
226     for col in data.cols.x: print(o(col))
227
228 def go_clone(file="data.csv"):
229     "FILE: test echoing structure of a table to a new table"
230     data1 = Data(csv(file))
231     data2 = clone(data1,data1.rows)
232     assert data1.cols.x[1].mu == data2.cols.x[1].mu
233
234 def go_disty(file="data.csv"):
235     "FILE: can we sort rows by their distance to heaven?"
236     data=Data(csv(file))
237     print(*data.cols.names)
238     for row in sorted(data.rows, key=lambda r: disty(data,r))[:40]:
239         print(*row)
240
241 def go_xai(file="data.csv"):
242     "FILE: can we succinctly list main effects in a table?"
243     print("u{xai}"+file)
244     xai(Data(csv(file)))
245
246 def go_six(file="data.csv"):
247     "FILE: robt xai, but in each loop, just read BUDGET rows"
248     xai(Data(csv(file))); print("")
249     go_s(the.seed)
250     for b in [5,10,20,30]:
251         go_b(the.budget)
252         print(b,sorted(six(Data(csv(file)))) for _ in range(20)))
253
254 if __name__ == "__main__":
255     for n, s in enumerate(sys.argv):
256         if (x:=vars()).get(f"u{replace('_', '-')}") is None:
257             fn(sys.argv[n+1]) if n < len(sys.argv) - 1 else fn(None)
258

```