

```

1 # &nbsp; 
2 # 'bingo.py' reads a CSV file ('-f') then
3 # (1) bins the rows into '-B' Bins along '-d' random projections; then
4 # (2) actively learns by scoring '-a' random bins, then for '-b' iterations,
5 # extrapolates from 2 examples to label best 'y'-guess; then
6 # (3) '-c' top bin items are labeled for evaluation.
7 #
8 # Success here means that trees learned from (from 'a+b' labels) finds stuff
9 # as good as anything else (after seeing very few labels).
10 #
11 ### Coding conventions:
12 # - 'the' is for CLI config,
13 # - 'row' is a list,
14 # - prefix '.' means 'private',
15 # - 'col' or 'c' means 'Num' or 'Sym',
16 # - 'i' means 'self',
17 # - 'd,a,n,s' = dict, array, num, str,
18 # - Structs (no classes), since polymorphic code can be shown together,
19 # - '.' in structs denotes type
20 # - Uppercase functions are constructors (e.g. 'Sym'), and their matching
21 # lowercase names are variables from that constructor (e.g. 'sym'),
22 # - 'eg_xxx' are CLI demos (e.g. '-xxx'),
23 # - In CSV input files, uppercase names on row1 denotes numeric; '+/-'
24 # - show 'y'-goals (others 'x').
25 """
26 bingo.py: stochastic landscape analysis for multi objective reasoning
27 (c) 2025 Tim Menzies, <timmm@ieee.org>. MIT license
28
29 Options, with their (defaults):
30
31 -B Bins number of bins (10)
32 -a rows labelled at random during cold start (4)
33 -b rows labelled while reflecting on labels seen so far (30)
34 -c rows labels while testing the supposed best bin (5)
35 -d dims number of dimensions (4)
36 -f file csv file for data (./moot/optimize/misc/aut93.csv)
37 -G Got directory to cache downloaded data files (~tmp/moot)
38 -g get github repo storing example data files (timmm/moot)
39 -k k Bayes hack for rare classes (1)
40 -m m Bayes hack for rare frequencies (2)
41 -p p minkowski coefficient (2)
42 -r rseed random number seed (1234567891)
43 -z zero ignore bins with zero items; 0=auto choose (0)
44 -h show help
45
46 """
47 import urllib.request, random, math, sys, re, os
48
49 sys.dont_write_bytecode = True
50 pick=random.choice
51 picks=random.choices
52 BIG=1E32
53
54 ### Command-line -----
55
56 # Reset slots from CLI flags, matching on first letter of slot.
57 # e.g. '-f file1' sets 'd["file"]' to 'file1'. If current value is a boolean then
58 # flags reverse old value. e.g. '-v 'negates' (e.g.) 'd["verbose"]=False'.
59 def cli(d):
60     for k, v in d.items():
61         for c, arg in enumerate(sys.argv):
62             if arg == "-" + k[0]:
63                 d[k] = coerce("False" if str(v) == "True" else (
64                     "True" if str(v) == "False" else (
65                         sys.argv[c + 1] if c < len(sys.argv) - 1 else str(v)))
66
67 # String to thing
68 def coerce(x):
69     for what in (int, float):
70         try: return what(x)
71     except: pass
72     x = x.strip()
73     y = x.lower()
74     return (y == "true") if y in ("true", "false") else x
75
76 def eg_h():
77     "print help text"
78     print(__doc__, "\nExamples:")
79     for s, fun in globals().items():
80         if s.startswith("eg_"):
81             print(f" {re.sub('eg_', '--', s)}>6) {fun.__doc__}")
82
83 def eg_all():
84     "run all examples"
85     for s, fun in globals().items():
86         if s.startswith("eg_"):
87             if s != "eg_all":
88                 print(f"\n# {s} { '-'*40 }\n# {fun.__doc__}\n")
89                 random.seed(the.rseed)
90                 fun()
91
92 ### Settings -----
93 # Structs with named fields + pretty print.
94 class o:
95     __init__ = lambda i, **d: i.__dict__.update(**d)
96     __repr__ = lambda i: \
97         (f.__name__ if (f:=i.__dict__.get("it")) else "") + cat(i.__dict__)
98
99 # Parse the '__doc__' string to generate 'the' config variable.
100 the= o(**[m[1]: coerce(m[2])
101             for m in re.finditer(r"=w+{s+(w+)[^()\\s*([*])s*})", __doc__])
102
103
104 def eg_the() -> None:
105     "Print the configuration."
106     print(the)
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

155 ### Read -----
156 def csv(s):
157     with open(webdata(s) or s, 'r', newline='') as f:
158         for line in f:
159             yield [coerce(s) for s in line.strip().split(',')]
160
161 def webdata(fn):
162     if fn.startswith(the.get):
163         cdir = os.path.expanduser(the.Got)
164         os.makedirs(cdir, exist_ok=True)
165         lfn = fn[len(the.get)+1:]
166         lpath = os.path.join(cdir, lfn)
167         if not os.path.exists(lpath):
168             rurl = f"https://github.com/{the.get}/tree/master/{fn}"
169             urllib.request.urlretrieve(rurl, lpath)
170         return lpath
171
172 def eg_csv():
173     "Print csv data."
174     m = 0
175     for n, row in enumerate(csv(the.file)):
176         if n>0: assert( int is type(row[0]) )
177         m += len(row)
178         if n%50==0: print(n, row)
179     assert(m==398)
180
181 def eg_cols():
182     "Print csv data."
183     cols = (lbs, acc, mpg) = Cols( next(csv(the.file)) ).y
184     assert mpg.heaven==1 and lbs.heaven==0 and acc.at==6
185     [print(cat(col)) for col in cols]
186
187

```

```

187 ### Update -----
188 # 'sub' is just 'add'ing -1.
189 def sub(i, v, purge=False): # -> v
190     return add(i, inc=-1, purge=purge)
191
192 # If 'v' is unknown, then ignore. Else, update.
193 def add(i, v, inc=1, purge=False): # -> v
194     def _sym(sym, s): # update symbol counts
195         sym.has[s] = inc + sym.has.get(s, 0)
196
197     def _data(data, row): # keep the new row, update the cols summaries.
198         if inc < 0:
199             if purge: data.rows.remove(v)
200             [sub(col, row[col.at], col, inc) for col in data.cols.all]
201         else:
202             data.rows += [[add(col, row[col.at], inc) for col in data.cols.all]]
203
204     def _num(num, n): # update lo, hi, mean and _m2 (used in sd calculation)
205         num.lo = min(n, num.lo)
206         num.hi = max(n, num.hi)
207         if inc < 0 and num.n < 2:
208             num._m2 = num.mu = num.n = 0
209         else:
210             d = n - num.mu
211             num.mu += inc * (d / num.n)
212             num._m2 += inc * (d * (n - num.mu))
213
214     if v != "?":
215         i.n += inc
216         (_num if i.it is Num else (_sym if i.it is Sym else _data))(i, v)
217     return v
218
219 def eg_num():
220     g=lambda: random.gauss(10,2)
221     num = Num(g) for _ in range(256))
222     assert 10 < mid(num) < 10.05 and 2 < div(num) < 2.1
223
224 def eg_sym():
225     sym = Sym("aaaabbc")
226     assert mid(sym) == "a" and 1,37 < div(sym) < 1.38
227
228 def eg_data():
229     model = Data(csv(the.file)).cols.x[2]
230     assert 3.69 < div(model) < 3.7
231     assert model.lo == 70 and model.hi == 82
232
233 # def add sub
234
235

```

```

235 ### Reports -----
236 def mids(data): return [mid(col) for col in data.cols.all]
237
238 def mid(col):
239     return col.mu if col.it is Num else max(col.has, key=col.has.get)
240
241 def div(col):
242     def _num(num):
243         return (max(num._m2,0)/(num.n - 1))*0.5
244
245     def _sym(sym):
246         return -sum(v/sym.n * math.log(v/sym.n, 2) for v in sym.has.values() if v>0)
247
248     return (_num if col.it is Num else _sym)(col)
249
250 ### Bayes -----
251 def like(data, row, nall=2, nh=100):
252     prior = (data.n + the.k) / (nall + the.k*nh)
253     tmp = [pdf(c,row[c.at], prior, nall, nh)
254            for c in data.cols.x if row[c.at] != "??"]
255     return sum(math.log(n) for n in tmp + [prior] if n>0)
256
257 def pdf(col,v, prior=0, nall=2, nh=100):
258     def _sym(sym,s):
259         return (sym.has.get(s,0) + the.m*prior) / (n + the.m + 1/BIG)
260
261     def _num(num,n):
262         sd = num.div() or 1 / BIG
263         var = 2 * sd * sd
264         z = (n - num.mu) ** 2 / var
265         return min(1, max(0, math.exp(-z) / (2 * math.pi * var) ** 0.5))
266
267     return (_num if col.it is Num else _sym)(col,v)
268
269
270 ### Distance -----
271 def norm(i,v):
272     return v if (v=="?" or i.it is not Num) else (v - i.lo)/(i.hi - i.lo + 1/BIG)
273
274 def dist(col,v,w):
275     def _sym(sym,s1,s2):
276         return s1 != s2
277
278     def _num(num,n1,n2):
279         n1,n2 = norm(num,n1), norm(num,n2)
280         n1 = n1 if n1 != "?" else (0 if n2 > 0.5 else 1)
281         n2 = n2 if n2 != "?" else (0 if n1 > 0.5 else 1)
282         return abs(n1 - n2)
283
284     return 1 if v=="?" and w=="?" else (_num if col.it is Num else _sym)(col,v,w)
285
286 def minkowski(a):
287     total, n = 0, 1 / BIG
288     for x in a:
289         n += 1
290         total += x**the.P
291     return (total / n)**(1 / the.P)
292
293 def ydist(data, row):
294     return minkowski(abs(norm(c,row[c.at]) - c.heaven) for c in data.cols.y)
295
296 def xdist(data, row1, row2):
297     return minkowski(dist(c,row1[c.at], row2[c.at]) for c in data.cols.x)
298
299

```

```

298 ### Clustering -----
299 def project(data, row, a, b): # -> 0,1,2 .. the.bins-1
300     D = lambda row1,row2: xdist(data,row1,row2)
301     c = D(a,b)
302     if c==0: return 0
303     return (D(row, a)**2 + c**2 - D(row, b)**2) / (2 * c * c)
304
305 def bucket(data,row,a,b):
306     return min(int( project(data,row,a,b) * the.bins), the.bins - 1)
307
308 def extrapolate(data,row,a,b):
309     ya, yb = ydist(data,a), ydist(data,b)
310     return ya + project(data,row,a,b) * (yb - ya)
311
312 def poles(data): # -> List[Row]
313     r0, *some = picks(data.rows, k=the.some + 1)
314     out = [max(some, key=lambda r1: xdist(data,r1, r0))]
315     for _ in range(the.dims):
316         out += [max(some, key=lambda r2: sum(xdist(data,r1,r2) for r1 in out))]
317     return out
318
319 def lsh(data, poles): # -> Dict[Tuple, List[Row]]
320     buckets = {}
321     for row in data.rows:
322         k = tuple(bucket(row, a, b) for a, b in zip(poles, poles[1:]))
323         buckets[k] = buckets.get(k) or clone(data)
324         add(buckets[k], row)
325     return buckets
326
327 def neighbors(c, hi):
328     def go(i, p):
329         if i == len(c):
330             t = tuple(p)
331             if t != c and all(0 <= x < hi for x in t):
332                 yield t
333         else:
334             for d in [-1, 0, 1]:
335                 yield from go(i+1, p + [c[i] + d])
336     yield from go(0, [])
337
338

```

```

339 ### Tree -----
340 ops = {'<=': lambda x,y: x <= y,
341        '==': lambda x,y: x == y,
342        '>': lambda x,y: x > y}
343
344 def selects(row, op, at, y): x=row[op]; return x=="?" or ops[op](x,y)
345
346 def cuts(col,rows,Y,Klass):
347     def _sym(sym):
348         n,d = 0,{}
349         for row in rows:
350             if (x := row[col.at]) != "?":
351                 n = n + 1
352                 d[x] = d.get(x) or Klass()
353                 add(d[x], Y(row))
354         return o(div = sum(c.n/n * div(c) for c in d.values()),
355                 hws = [{"==",col.at,k} for k,v in d.items()])
356
357     def _num(num):
358         out, b4, lhs, rhs = None, None, Klass(), Klass()
359         xys = [{"r[num.at], add(rhs, Y(r))} for r in rows if r[num.at] != "?"]
360         xpect = div(rhs)
361         for x, y in sorted(xys, key=lambda xy: x[0]):
362             if x != b4:
363                 if the.leaf <= lhs.n <= len(xys) - the.leaf:
364                     tmp = (lhs.n * div(lhs) + rhs.n * div(rhs)) / len(xys)
365                     if tmp < xpect:
366                         xpect, out = tmp, [{"<=", num.at, b4}, {">", num.at, b4}]
367                     add(lhs, sub(rhs,y))
368                     b4 = x
369             if out:
370                 return o(div=xpect, hws=out)
371
372     return (_sym if col.it is Sym else _num)(col)
373
374 def tree(data1, rows=None, Klass=Num, how=None):
375     Y = lambda row: ydist(data1,row)
376     rows = rows or data1.rows
377     data2.kids = []
378     data2.how = how
379     data2 = clone(data1, rows)
380     data2.ys = Num(Y(row) for row in rows)
381     if len(rows) >= the.leaf:
382         cuts = [x for c in data1.cols.x if (x := cuts(c,rows,Y,Klass=Klass))]
383         if cuts:
384             for how in sorted(cuts, key=lambda cut: cut.div)[0].hws:
385                 rows1 = [row for row in rows if selects(row, *how)]
386                 if the.leaf <= len(rows1) < len(rows):
387                     data2.kids += [tree(data1, rows1, Klass=Klass, how=how)]
388     return data2
389
389 def nodes(data1, lvl=0, key=None):
390     yield lvl, data1
391     for data2 in (sorted(data1.kids, key=key) if key else data1.kids):
392         yield from nodes(data2, lvl + 1, key=key)
393
394 def leaf(data1,row):
395     for data2 in data1.kids or []:
396         if selects(row, *data2.decision):
397             return leaf(data2, row)
398     return data1
399
400 def show(data, key=lambda z:z.ys.mu):
401     stats = data.ys
402     win = lambda x: 100-int(100*(x-stats.lo)/(stats.mu - stats.lo))
403     print(f"{'d2h':>4} {'win':>4} {'n':>4} ")
404     print(f"{'-----':>4} {'-----':>4} {'-----':>4} ")
405     for lvl, node in nodes(data, key=key):
406         leafp = len(node.kids)==0
407         post = " " if leafp else ""
408         xplain = ""
409         if lvl > 0:
410             op,at,y = node.decision
411             xplain = f"[data.cols.all[at].txt] {op} {y}"
412         print(f"[node.ys.mu:4.2f] [win(node.ys.mu):4] {node.n:4} {(lvl-1)*' '}{xplain}" + post)
413
414

```

```

414 ### Utils -----
415 def cat(v):
416     it = type(v)
417     inf = float('inf')
418     if it is list: return " + ".join(map(cat, v)) + "]"
419     if it is float: return str(int(v)) if -inf < v < inf and v == int(v) else f"[v
:3g]"
420     if it is dict: return cat([f":{k} {cat(w)}" for k, w in v.items()])
421     if it in [type(abs), type(cat)]: return v.__name__
422     return str(v)
423
424

```

```

424 ### Start-up -----
425 if __name__ == "__main__":
426     cli(the.__dict__)
427     for n, s in enumerate(sys.argv):
428         if fun := globals().get("eg" + s.replace("-", "_")):
429             random.seed(the.rseed)
430             fun()

```