

## keysl

```

5: #!/usr/bin/env python3.9
6: # vim: ts=2 sw=2 sts=2 et :
7: """
8:  _-|_ (c) Tim Menzies, 2021, unlicense.org
9:  / _|_ The delta between things is
10:  \ _-|_ simpler than the things.
11:  v
12: """
13: import re, sys, math, copy, argparse, random, itertools
14:
15: def config(): return dict(
16:     BINS=(float, .5, 'bins are of size n**BINS'),
17:     COLS=(str, 'x', 'columns to use for inference'),
18:     DATA=(str, './data/auto2.csv', 'where to read data'),
19:     FAR=(float, .9, 'where to look for far things'),
20:     GOAL=(str, 'best', 'learning goals: best|rest|other'),
21:     IOTA=(float, .3, 'small = sd**iota'),
22:     K=(int, 2, 'bayes low class frequency hack'),
23:     M=(int, 1, 'bayes low range frequency hack'),
24:     P=(int, 2, 'distance calculation exponent'),
25:     SAMPLES=(int, 20, '#samples to find far things?'),
26:     SEED=(int, 10013, 'seed for random numbers'),
27:     VERBOSE=(bool, False, 'set verbose'),
28:     TOP=(int, 20, 'focus on this many'),
29:     WILD=(int, False, 'run example, no protection'),
30:     XAMPLE=(str, "", 'egs: '-x ls' lists all, '-x all' runs all'))
31:
32: class o(object):
33:     def __init__(i, **k): i.__dict__.update(**k)
34:     def __getitem__(i, k, v): i.__dict__[k] = v
35:     def __getitem__(i, k): return i.__dict__[k]
36:     def __repr__(i): return i.__class__.__name__ + str(
37:         {k: v for k, v in i.__dict__.items() if k[0] != "_"})
38:
39: # Columns
40: class Col(o):
41:     "Store columns in 'Col', 'Skip', 'Sym', 'Num'."
42:     def __init__(i, at=0, txt="", inits=[]):
43:         i.n, i.at, i.txt = 0, at, txt
44:         i.w = -1 if "-" in txt else 1
45:         [i.add(x) for x in inits]
46:
47:     def add(i, x, n=1):
48:         if x != "?":
49:             i.n += n
50:             x = i.add1(x, n)
51:         return x
52: #
53: class Skip(Col):
54:     def add1(i, x, n=1): return x
55: #
56: class Sym(Col):
57:     def __init__(i, **kw):
58:         i.has, i.mode, i.most = {}, None, 0
59:         super().__init__(**kw)
60:
61:     def add1(i, x, n=1):
62:         new = inc(i.has, x, n)
63:         if new > i.most:
64:             i.most, i.mode = new, x
65:         return x
66:
67:     def bins(i, j, _):

```

```

68:         for k in (i.has | j.has):
69:             yield i.at, (k, k)
70:
71:     def dist(i, x, y): return 0 if x == y else 1
72:
73:     def ent(i):
74:         return sum(-v/i.n * math.log(v/i.n) for v in i.has.values())
75:
76:     def merge(i, j):
77:         k = Sym(at=i.at, txt=i.txt)
78:         [k.add(x, n) for has in (i.has, j.has) for x, n in has.items()]
79:         return k
80:
81:     def merged(i, j):
82:         k = i.merge(j)
83:         e1, n1, e2, n2, e, n = i.ent(), i.n, j.ent(), j.n, k.ent(), k.n
84:         tmp = n1/n*e1 + n2/n*e2
85:         # print(e1, e2, e, tmp)
86:         if e1 + e2 < 0.01 or e * .95 < tmp:
87:             return k
88:
89:     def mid(i): return i.mode
90: #
91: class Num(Col):
92:     def __init__(i, **kw):
93:         i._all, i.ok = [], False
94:         super().__init__(**kw)
95:
96:     def add1(i, x, n):
97:         x, i.ok = float(x), False
98:         for _ in range(n):
99:             i._all += [x]
100:         return x
101:
102:     def all(i):
103:         if not i.ok:
104:             i.ok = True
105:             i._all = sorted(i._all)
106:         return i._all
107:
108:     def bins(i, j, the):
109:         xy = [(z, True) for z in i._all] + [(z, False) for z in j._all]
110:         iota = the.IOTA * (i.n*i.sd() + j.n*j.sd()) / (i.n + j.n)
111:         for ((lo, hi), _) in bins(xy, iota=iota, size=len(xy))*the.BINS:
112:             yield i.at, (lo, hi)
113:
114:     def dist(i, x, y):
115:         if x == "?":
116:             y = i.norm(y)
117:             x = 1 if y < 0.5 else 0
118:         elif y == "?":
119:             x = i.norm(x)
120:             y = 1 if x < 0.5 else 0
121:         else:
122:             x, y = i.norm(x), y.norm(y)
123:         return abs(x-y)
124:
125:     def mid(i): return per(i.all(), p=.5)
126:
127:     def norm(i, x):
128:         if x == "?":
129:             return x
130:         a = i.all()

```

```

131:         return max(0, min(1, (x-first(a))/(last(a)-first(a)+1E-32)))
132:
133:     def sd(i) : return (per(i.all(), .9) - per(i.all(), .1))/2.56
134:     def span(i) : return (first(i.all()), last(i.all()))
135:     def wide(i, n=0): return last(i.all()) - first(i.all()) >= n
136:
137: # Row and Rows
138: class Row(o):
139:     def __init__(i, lst, rows=None): i.rows, i.cells = rows, lst
140:
141:     def __lt__(i, j):
142:         goals = i.rows.cols.y
143:         s1, s2, n = 0, 0, len(goals)
144:         for col in goals:
145:             a = col.norm(i.cells[col.at])
146:             b = col.norm(j.cells[col.at])
147:             s1 -= math.e**(col.w * (a - b) / n)
148:             s2 -= math.e**(col.w * (b - a) / n)
149:         return s1 / n < s2 / n
150:
151:     def dist(i, j, the):
152:         d = n = 1E-32
153:         for col in i.rows.cols[the.COLS]:
154:             n += 1
155:             x, y = i.cells[at], j.cells[at]
156:             d += 1 if x == "?" and y == "?" else col.dist(x, y) ** the.P
157:         return (d/n) ** (1/the.P)
158:
159:     def far(i, rows, the):
160:         tmp = [(dist(i, j), j) for _ in range(the.SAMPLE)]
161:         return per(sorted(tmp, key=first), the.FAR)
162:
163:     def ys(i): return [i.cells[col.at] for col in i.rows.cols.y]
164: #
165: class Rows(o):
166:     def __init__(i, inits=[]):
167:         i.rows = []
168:         i.cols = o(all=[], names=[], x=[], y=[], klass=None)
169:         [i.add(x) for x in inits]
170:
171:     def add(i, a): i.data(a) if i.cols.names else i.header(a)
172:
173:     def best(i, the):
174:         i.rows.sort()
175:         ds = [the.IOTA*y.sd() for y in i.cols.y]
176:         best, rest = i.clone(), i.clone()
177:         for n, row in enumerate(i.rows):
178:             bestp = False
179:             for n1, n2, d in zip(i.rows[0].ys(), row.ys(), ds):
180:                 bestp |= abs(n1-n2) <= d
181:             (best if bestp else rest).add(row)
182:         return best, rest
183:
184:     def clone(i, inits=[]): return Rows([i.cols.names] + inits)
185:
186:     def data(i, a):
187:         a = a.cells if type(a) == Row else a
188:         i.rows += [Row([col.add(a[col.at]) for col in i.cols.all],
189:             rows=i)]
190:
191:     def header(i, a):
192:         i.cols.names = a
193:         for at, x in enumerate(a):

```

## keys1

```

194: new = Skip if i.skipp(x) else (Num if i.nump(x) else Sym)
195: new = new(at=at, txt=x)
196: i.cols.all += [new]
197: if not i.skipp(x):
198:     i.cols["y" if i.y(x) else "x"] += [new]
199:     if i.klassp(x):
200:         i.cols.klass = new
201:
202: def klassp(i, x): return "!" in x
203: def nump(i, x): return x[0].isupper()
204: def skipp(i, x): return "?" in x
205: def ys(i): return [col.mid() for col in i.cols.y]
206: def ysd(i): return [col.sd() for col in i.cols.y]
207: def yp(i, x): return "-" in x or "+" in x or i.klassp(x)
208: #
209: def stratify(src):
210:     all, klass = None, {}
211:     for n, row in enumerate(src):
212:         if all:
213:             kl = row[all.cols.klass.at]
214:             here = klass[kl] = klass.get(kl, None) or all.clone()
215:             here.add(row)
216:             all.add(row)
217:         else:
218:             all = Rows([row])
219:     return o(all=all, klass=klass)
220:
221: # Discretizations
222: # Use 'bins' to divide numeric data into ranges.
223: def bins(xy, iota=0, size=30):
224:     def merge(b4):
225:         j, tmp, n = 0, [], len(b4)
226:         while j < n:
227:             ((lo, _), ay) = a = b4[j]
228:             if j < n - 1:
229:                 ((_, hi), by) = b4[j + 1]
230:                 if cy := ay.merged(by):
231:                     a = ((lo, hi), cy)
232:                     j += 1
233:             tmp += [a]
234:             j += 1
235:     return merge(tmp) if len(tmp) < len(b4) else b4
236:
237: def divide(xy):
238:     bin = o(x=Num(), y=Sym())
239:     bins = [bin]
240:     for i, (x, y) in enumerate(xy):
241:         if bin.x.n >= size and x != b4:
242:             if i < len(xy)-size and bin.x.wide(iota):
243:                 bin = o(x=Num(), y=Sym())
244:                 bins += [bin]
245:             bin.x.add(x)
246:             bin.y.add(y)
247:             b4 = x
248:     return bins
249:
250: xy = sorted(xy, key=first)
251: return merge([(bin.x.span(), bin.y) for bin in divide(xy)])
252:
253: # Learn class deltas
254: def contrasts(here, there, the):
255:     goal = {'best': lambda b, r: b**2/b+r,
256:             'rest': lambda b, r: r**2/(b+r),

```

```

257:             'other': lambda b, r: 1/(b+r) ][the.GOAL]
258:
259: def like(d, kl):
260:     out = prior = (hs[kl] + the.K) / (n + the.K*2)
261:     for at, span in d.items():
262:         f = has.get((kl, (at, span)), 0)
263:         out *= (f + the.M*prior) / (hs[kl] + M)
264:     return out
265:
266: def val(d): return goal(like(d, True), like(d, False)), d
267: def top(a): return sorted(a, reversed=True, key=first)[:the.TOP]
268:
269: has = {(kl, (at, (lo, hi))): f
270:         for col1, col2 in zip(here.cols.x, there.cols.x)
271:         for at, (lo, hi) in col1.bins(col2, the)}
272: n = len(here.rows, there.rows)
273: hs = {True: len(here.rows), False: len(there.rows)}
274: solos = [val(dict(at=x)) for at, x in set([z for _, z in has])]
275: ranges = {}
276: for _, d in top(solos):
277:     for k in d:
278:         ranges[k] = ranges.get(k, set()).add(d[k])
279: for rule in top([val(d) for d in dict_product(ranges)]):
280:     print(rule)
281:
282: # Misc utils
283: # string stuff
284: def color(end="\n", **kw):
285:     s, a, z = "", "\u001b[", ";1m"
286:     c = dict(black=30, red=31, green=32, yellow=33,
287:             purple=34, pink=35, blue=36, white=37)
288:     for col, txt in kw.items():
289:         s = s+a + str(c[col]) + z+txt+"\033[0m"
290:     print(s, end=end)
291:
292: def mline(m): m += ["-"*len(str(x)) for x in m[-1]]
293:
294: def printm(matrix):
295:     s = [str(e) for e in row] for row in matrix
296:     lens = [max(map(len, col)) for col in zip(*s)]
297:     fmt = ' | '.join('{{: > {} }}'.format(x) for x in lens)
298:     for row in [fmt.format(*row) for row in s]:
299:         print(row)
300:
301: # maths stuff
302: def r3(a): return [round(x, 3) for x in a]
303:
304: # dictionary stuff
305: def has(d, k): return d.get(k, 0)
306: def inc(d, k, n=1): tmp = d[k] = n + d.get(k, 0); return tmp
307:
308: def dict_product(d):
309:     keys = d.keys()
310:     for p in itertools.product(*d.values()):
311:         yield dict(zip(keys, p))
312:
313: # list stuff
314: def first(a): return a[0]
315: def last(a): return a[-1] # $Vlabel{comment}$
316: def per(a, p=.5): return a[int(p*len(a))]
317:
318: # file stuff
319: def csv(f=None, sep=","):

```

```

320: def prep(s): return re.sub(r'([\n\t\r ]#.*)', '', s)
321: if f:
322:     with open(f) as fp:
323:         for s in fp:
324:             if s := prep(s):
325:                 yield s.split(sep)
326:
327: else:
328:     for s in sys.stdin:
329:         if s := prep(s):
330:             yield s.split(sep)
331:
332: # command-line stuff
333: def cli(use, txt, config):
334:     used, p = {}, argparse.ArgumentParser(prog=use, description=txt,
335:                                           formatter_class=argparse.RawTextHelpForma
336:                                           ter)
337:     for k, (_, b4, h) in config.items():
338:         k0 = k[0]
339:         used[k0] = c = k0 if k0 in used else k0.lower()
340:         if b4 == False:
341:             p.add_argument("-"+c, dest=k, default=False,
342:                             help=h,
343:                             action="store_true")
344:         else:
345:             p.add_argument("-"+c, dest=k, default=b4,
346:                             help=h + " [" + str(b4) + "]",
347:                             type=type(b4), metavar=k)
348:     return o(**p.parse_args().__dict__)
349:
350: # Unit tests
351: class Eg:
352:     def ls(the):
353:         "list all examples."
354:         print("\nexamples:")
355:         for k, f in vars(Eg).items():
356:             if k[0] != "_":
357:                 print(f" {k[:13]} {f.__doc__}")
358:
359: def fail(the):
360:     "testing failure"
361:     assert False, "failing"
362:
363: def data(the, file="./data/vote.csv"):
364:     "simple load of data into a table"
365:     r = Rows(csv(file))
366:     assert 435 == len(r.rows)
367:     assert 195 == r.cols.all[1].has['y']
368:
369: def nclasses(the, file="./data/diabetes.csv", kl="positive"):
370:     "read data with nclasses"
371:     rs = stratify(csv(file))
372:     assert 2 == len(rs.klass)
373:     assert 268 == len(rs.klass[kl].rows)
374:     assert 768 == len(rs.all.rows)
375:     assert 3.90625 == rs.klass[kl].cols.all[0].sd()
376:
377: def bins(the, file="./data/diabetes.csv",
378:          k1="positive", k2="negative"):
379:     "discretize some data"
380:     rs = stratify(csv(file))
381:     bins1(rs.klass[k1], rs.klass[k2], the)
382:
383: def bestrest(the, file="./data/auto93.csv"):

```

```
382: "discretize some multi-goal data"
383: r = Rows(csv(file))
384: goods, bads = r.best(the)
385: bins1(goods, bads, the)
386:
387: def bins1(goods, bads, the):
388:     for good, bad in zip(goods.cols.x, bads.cols.x):
389:         bins = list(good.bins(bad, the))
390:         if len(bins) > 1:
391:             print(f"\n{good.txt}")
392:             for bin in bins:
393:                 print("\t", bin)
394:
395: # Main program
396: def main(the):
397:     def run(fun, fails, the):
398:         s = f" {fun.__name__:<12}"
399:         if the.WILD:
400:             print("raw")
401:             fun(copy.deepcopy(the))
402:             sys.exit()
403:         try:
404:             fun(copy.deepcopy(the))
405:             random.seed(the.SEED)
406:             color(green=(chr(10003) + s), white=fun.__doc__)
407:         except Exception as err:
408:             fails = fails + 1
409:             color(red=(chr(10007) + s), white=str(err))
410:         return fails
411:     #
412:     fails = 0
413:     if the.XAMPLE == "all":
414:         for k, f in vars(Eg).items():
415:             if k[0] != "_" and k != "Is":
416:                 fails = run(f, fails, the)
417:     else:
418:         if the.XAMPLE and the.XAMPLE in vars(Eg):
419:             f = vars(Eg)[the.XAMPLE]
420:             if the.XAMPLE == "Is":
421:                 f(the)
422:             else:
423:                 fails = run(f, fails, the)
424:     sys.exit(fails)
425:
426:
427: if __name__ == "__main__":
428:     main( cli( ".keys", __doc__, config()) )
```