

```
1: # README.md
2:
3:
4: On a 'BSD-compatible' system, to convert and print the text file
5: myfile.txt to printer ps99 without a banner page, you would enter
6: at the Unix prompt:
7:
8:   enscript -Pps99 -h myfile.txt
9:
10: To make a PostScript file of the text file myfile.txt and save it
11: as myfile.ps, you would enter:
12:
13:   enscript -pmyfile.ps myfile.txt
14:
15: To redirect the output of enscript as input for another command,
16: append another dash to the -p option, for example:
17:
18:   enscript -2rG -p- myfile.txt | psduplex | lpr -Pps99 -h
19:
20: ## asd asd as dsa dasdas
21:
22: This will convert myfile.txt into a two-column, rotated PostScript
23: file with ai_large header, send it through the _psduplex filter to
24: print on both sides of the paper, and finally print the result
25: without a banner page to the printer named ps99.
26:
27: ### asdasdasdasdas
28:
29: For more information about enscript, read its man page by entering
30: the following:
31:
32:   man enscript
33:
34:   adas | asdas | a asdas
35:   -----|-----|-----
36:   asdas | 34 | 323
37:   sadasd | asdasds | asdas
38:
39: At Indiana University, enscript is installed on Steel and Libra.
40:
41: - asda dasd asdas asd as das das das as ada d as d as as sa d
42:   as as as d asas asdadasas
43: - asdadas
44:   asdas
45:
```

```

1: #!/usr/bin/env python3.9
2: # vim: ts=2 sw=2 sts=2 et :
3: # autopep8 --exclude 'E20,E401,E226,E301,E302,E41'
4: """
5:     \ Contrast set learning
6:     \ (c) Tim Menzies, 2021, unlicense.org
7:     \ Cluster, then reports, just the
8:     \ v deltas between similar clusters.
9: """
10: import re, sys, math, copy, argparse, random, datetime, itertools
11:
12: def config(): return dict(
13:     BINS=( float, .5, 'bins are of size n**BINS'),
14:     COLS=( str, 'x', 'columns to use for inference'),
15:     DATA=( str, './data/auto2.csv', 'where to read data'),
16:     FAR=( float, .9, 'where to look for far things'),
17:     GOAL=( str, 'best', 'learning goals: best|rest|other'),
18:     IOTA=( float, .3, 'small = sd**iota'),
19:     K=( int, 2, 'bayes low class frequency hack'),
20:     M=( int, 1, 'bayes low range frequency hack'),
21:     P=( int, 2, 'distance calculation exponent'),
22:     SAMPLES=(int, 20, '#samples to find far things?'),
23:     SEED=( int, 10013, 'seed for random numbers'),
24:     TOP=( int, 10, 'focus on this many'),
25:     UNSAFE=( bool, False, 'run example, no protection'),
26:     VERBOSE=(bool, False, 'set verbose'),
27:     XAMPLE=( str, "", "'-x ls' lists all, '-x all' runs all"))
28:
29: class o(object):
30:     def __init__(i, **k): i.__dict__.update(**k)
31:     def __setitem__(i, k, v): i.__dict__[k] = v
32:     def __getitem__(i, k): return i.__dict__[k]
33:     def __repr__(i): return i.__class__.__name__ + str(
34:         {k: v for k, v in i.__dict__.items() if k[0] != "_"})
35:
36: # Columns
37: class Col(o):
38:     "Store columns in 'Col', 'Skip', 'Sym', 'Num'."
39:     def __init__(i, at=0, txt="", inits=[]):
40:         i.n, i.at, i.txt = 0, at, txt
41:         i.w = -1 if "-" in txt else 1
42:         [i.add(x) for x in inits]
43:
44:     def add(i, x, n=1):
45:         if x != "?":
46:             i.n += n
47:             x = i.add1(x, n)
48:         return x
49:
50:     def merge(i, j):
51:         k = i.__class__(at=i.at, txt=i.txt)
52:         [k.add(x, n) for has in (i.has, j.has) for x, n in has.items()]
53:         return k
54:
55:     def merged(i, j):
56:         k = i.merge(j)
57:         vi, vj, vk = i.var(), j.var(), k.var()
58:         if vk < (i.n*vi + j.n*vj) / (i.n + j.n):
59:             # if vi + vj < 0.01 or vk * .95 < (i.n*vi + j.n*vj) / (i.n + j.n):
60:             return k
61:
62: #
63: class Skip(Col):
64:     def add1(i, x, n=1): return x
65: #
66: class Sym(Col):
67:     def __init__(i, **kw):
68:         i.has, i.mode, i.most = {}, None, 0
69:         super().__init__(**kw)
70:
71:     def add1(i, x, n=1):
72:         new = inc(i.has, x, n)
73:         if new > i.most:
74:             i.most, i.mode = new, x
75:         return x
76:
77:     def bins(i, j, _):
78:         for k in (i.has | j.has):
79:             yield True, i.has.get(k, 0), i.at, (k, k)
80:             yield False, j.has.get(k, 0), j.at, (k, k)
81:
82:     def dist(i, x, y): return 0 if x == y else 1
83:     def mid(i): return i.mode
84:
85:     def var(i):
86:         return sum(-v/i.n * math.log(v/i.n) for v in i.has.values())
87:
88: #
89: class Num(Col):
90:     def __init__(i, **kw):
91:         i._all, i.ok = [], False
92:         super().__init__(**kw)
93:
94:     def add1(i, x, n):

```

```

95:         x, i.ok = float(x), False
96:     for _ in range(n):
97:         i._all += [x]
98:     return x
99:
100: def all(i):
101:     if not i.ok:
102:         i.ok = True
103:         i._all = sorted(i._all)
104:     return i._all
105:
106: def bins(i, j, the):
107:     xy = [(z, True) for z in i._all] + [(z, False) for z in j._all]
108:     iota = the.IOTA * (i.n*i.sd() + j.n*j.sd()) / (i.n + j.n)
109:     for ((lo, hi), sym) in bins(xy, iota=iota, size=len(xy)**the.BINS):
110:         yield True, sym.has.get(True, 0), i.at, (lo, hi)
111:     yield False, sym.has.get(False, 0), j.at, (lo, hi)
112:
113: def dist(i, x, y):
114:     if x == "?":
115:         y = i.norm(y)
116:         x = 1 if y < 0.5 else 0
117:     elif y == "?":
118:         x = i.norm(x)
119:         y = 1 if x < 0.5 else 0
120:     else:
121:         x, y = i.norm(x), i.norm(y)
122:     return abs(x-y)
123:
124: def mid(i): return per(i.all(), p=.5)
125:
126: def norm(i, x):
127:     if x == "?":
128:         return x
129:     a = i.all()
130:     return max(0, min(1, (x-first(a))/(last(a)-first(a)+1E-32)))
131:
132: def sd(i) : return (per(i.all(), .9) - per(i.all(), .1))/2.56
133: def span(i) : return (first(i.all()), last(i.all()))
134: def var(i) : return i.sd()
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 display(i, s, rule):
192:     def n(x): return int(x) if x == int(x) else x
193:     return f"{s:>.5f}: " + ' ' and '.join(
194:         (i.cols.names[k])+" " = "+(
195:             str(n(lo)) if lo == hi else f"{n(lo)}..{n(hi)}")
196:         for k, (lo, hi) in sorted(rule.items()))
197:
198: def header(i, a):
199:     i.cols.names = a
200:     for at, x in enumerate(a):
201:         new = Skip if i.skipp(x) else (Num if i.nump(x) else Sym)
202:         new = new(at=at, txt=x)
203:         i.cols.all += [new]
204:         if not i.skipp(x):
205:             i.cols["y" if i.y(x) else "x"] += [new]
206:         if i.klassp(x):
207:             i.cols.klass = new
208:
209: def klassp(i, x): return "!" in x
210: def nump(i, x): return x[0].isupper()
211: def skipp(i, x): return "?" in x
212: def yp(i, x): return "-" in x or "+" in x or i.klassp(x)
213: def ys(i): return [col.mid() for col in i.cols.y]
214: def ysd(i): return [col.sd() for col in i.cols.y]
215: # _____
216: def stratify(src):
217:     all, klass = None, {}
218:     for n, row in enumerate(src):
219:         if all:
220:             kl = row[all.cols.klass.at]
221:             here = klass[kl] = klass.get(kl, None) or all.clone()
222:             here.add(row)
223:             all.add(row)
224:         else:
225:             all = Rows([row])
226:     return o(all=all, klass=klass)
227:
228: # Discretizations
229: # Use 'bins' to divide numeric data into ranges.
230: def bins(xy, iota=0, size=30):
231:     def merge(b4):
232:         j, tmp, n = 0, [], len(b4)
233:         while j < n:
234:             ((lo, _), ay) = a = b4[j]
235:             if j < n - 1:
236:                 ((_, hi), by) = b4[j + 1]
237:                 if cy := ay.merged(by):
238:                     a = ((lo, hi), cy)
239:                     j += 1
240:             tmp += [a]
241:             j += 1
242:     return merge(tmp) if len(tmp) < len(b4) else b4
243:
244: def divide(xy):
245:     bin = o(x=Num(), y=Sym())
246:     bins = [bin]
247:     for i, (x, y) in enumerate(xy):
248:         if bin.x.n >= size and x != b4:
249:             if i < len(xy)-size and bin.x.wide(iota):
250:                 bin = o(x=Num(), y=Sym())
251:                 bins += [bin]
252:             bin.x.add(x)
253:             bin.y.add(y)
254:             b4 = x
255:     return bins
256:
257: xy = sorted(xy, key=first)
258: return merge([(bin.x.span(), bin.y) for bin in divide(xy)])
259:
260: # Learn class deltas
261: def contrasts(here, there, the):
262:     goal = ('best' : lambda b, r: b**2/(b+r),
263:            'rest' : lambda b, r: r**2/(b+r),
264:            'other' : lambda b, r: 1/(b+r) ][the.GOAL]
265:
266: def like(d, kl):
267:     out = prior = (hs[kl] + the.K) / (n + the.K*2)
268:     for at, span in d.items():
269:         f = has.get((kl, at, span), 0)
270:         out *= (f + the.M*prior) / (hs[kl] + the.M)
271:     return out
272:
273: def top(a): return sorted(a, reverse=True, key=first)[:the.TOP]
274:
275: def val(d):
276:     for k, v in list(d.items()):
277:         if v == None:
278:             del d[k]
279:     s = goal(like(d, True), like(d, False)) if d else 0
280:     return s, d
281:
282: has = {(klass, at, (lo, hi)): f

```

```

283:         for col1, col2 in zip(here.cols.x, there.cols.x)
284:         for klass, f, at, (lo, hi) in col1.bins(col2, the))
285:     n = len(here.rows) + len(there.rows)
286:     hs = {True: len(here.rows), False: len(there.rows)}
287:     all = set([(at, span) for (_, at, span) in has])
288:     for x, ys in sorted(all):
289:         print(f"\n{here.cols.names[x]}")
290:         [print(f"\t{y}") for y in ys if y]
291:     solos = [val({at: span}) for (_, at, span) in all]
292:     ranges = {}
293:     for _, d in top(solos):
294:         for k in d:
295:             if k not in ranges:
296:                 ranges[k] = set([None])
297:                 ranges[k].add(d[k])
298:             print("\nrules:")
299:             for s, rule in top([val(d) for d in dict_product(ranges)]):
300:                 if s and rule:
301:                     print("\t"+here.display(s, rule))
302:
303: # Report
304: class Report(o):
305:     def __init__(i):
306:         i.rules = []
307:     def span(i, x):
308:         lo, hi = x
309:         return str(lo) if lo == hi else f"{lo}..{hi}"
310:     def rule(i, d): pass
311:     def the(i, d):
312:         print("\noptions:")
313:         for k, v in d.items():
314:             print(f" {k:>15s} = {v}")
315:     def report(i, the):
316:         print(datetime.datetime.now())
317:         i.the(the)
318:
319: # Misc utils
320: # string stuff
321: def color(end="\n", **kw):
322:     s, a, z = "", "\u001b[", ";1m"
323:     c = dict(black=30, red=31, green=32, yellow=33,
324:             purple=34, pink=35, blue=36, white=37)
325:     for col, txt in kw.items():
326:         s = s+a + str(c[col]) + z+txt+"033[0m"
327:     print(s, end=end)
328:
329: def mline(m): m += [""*len(str(x)) for x in m[-1]]
330:
331: def mlines(matrix):
332:     s = [str(e) for e in row] for row in matrix]
333:     lens = [max(map(len, col)) for col in zip(*s)]
334:     fmt = ' | '.join('{:>0}')*len(lens)
335:     for row in [fmt.format(*row) for row in s]:
336:         print(row)
337:
338: def norepeats(m):
339:     b4, new = None, []
340:     for xs in m:
341:         if b4:
342:             tmp = [" " if old == x else x for old, x in zip(b4, xs)]
343:         else:
344:             tmp = xs
345:         b4 = xs
346:         new += [tmp]
347:     return new
348:
349: # maths stuff
350: def r3(a): return [round(x, 3) for x in a]
351:
352: # dictionary stuff
353: def has(d, k): return d.get(k, 0)
354: def inc(d, k, n=1): tmp = d[k] = n + d.get(k, 0); return tmp
355:
356: def dict_product(d):
357:     keys = d.keys()
358:     for p in itertools.product(*d.values()):
359:         yield dict(zip(keys, p))
360:
361: # list stuff
362: def first(a): return a[0]
363: def last(a): return a[-1] # $Label(comment)$
364: def per(a, p=.5): return a[int(p*len(a))]
365:
366: # file stuff
367: def csv(src=None, sep=",", zap=r'([\n\r ]|#.*)':
368:     if lines := (not src and sys.stdin or
369:                  type(src) == list and src or
370:                  "\n" in src and src.splitlines() or
371:                  None):
372:         for line in lines:
373:             if line := re.sub(zap, "", line):
374:                 yield line.split(sep)
375:     else:
376:         with open(src) as lines:

```

```

377:     for line in lines:
378:         if line := re.sub(zap, "", line):
379:             yield line.split(sep)
380:
381: # command-line stuff
382: def cli(use, txt, config):
383:     fmt = argparse.RawTextHelpFormatter
384:     parse = argparse.ArgumentParser
385:     used, p = {}, parse(prog=use, description=txt,
386:                        formatter_class=fmt)
387:     for k, (_, b4, h) in sorted(config.items()):
388:         k0 = k[0]
389:         used[k0] = c = k0 if k0 in used else k0.lower()
390:         if b4 == False:
391:             p.add_argument("-"+c, dest=k, default=False, help=h,
392:                           action="store_true")
393:         else:
394:             p.add_argument("-"+c, dest=k, default=b4,
395:                           help=h + " [" + str(b4) + "]",
396:                           type=type(b4), metavar=k)
397:     return o(**p.parse_args().__dict__)
398:
399: # Unit tests
400: class Eg:
401:     def ls(the):
402:         "list all examples."
403:         print("\nexamples:")
404:         for k, f in vars(Eg).items():
405:             if k[0] != " ":
406:                 print(f" {k:<13} {f.__doc__}")
407:
408:     def _fail(the):
409:         "testing failure"
410:         assert False, "failing"
411:
412:     def csv(the, file="./data/vote.csv"):
413:         "simple load of data into a table"
414:         for r in csv(file):
415:             print(r)
416:
417:     def data(the, file="./data/vote.csv"):
418:         "simple load of data into a table"
419:         r = Rows(csv(file))
420:         assert 435 == len(r.rows)
421:         assert 195 == r.cols.all[1].has['y']
422:
423:     def nclasses(the, file="./data/diabetes.csv", kl="positive"):
424:         "ead data with nclasses"
425:         rs = stratify(csv(file))
426:         assert 2 == len(rs.klass)
427:         assert 268 == len(rs.klass[kl].rows)
428:         assert 768 == len(rs.all.rows)
429:         assert 3.90625 == rs.klass[kl].cols.all[0].sd()
430:
431:     def bins(the, file="./data/diabetes.csv",
432:             k1="positive", k2="negative"):
433:         "discretize some data"
434:         rs = stratify(csv(file))
435:         bins1(rs.klass[k1], rs.klass[k2], the)
436:
437:     def bestrest(the, file="./data/auto93.csv"):
438:         "discretize some multi-goal data"
439:         r = Rows(csv(file))
440:         goods, bads = r.best(the)
441:         bins1(goods, bads, the)
442:
443:     def con1(the, file="./data/auto93.csv"):
444:         "discretize some multi-goal data"
445:         r = Rows(csv(file))
446:         goods, bads = r.best(the)
447:         contrasts(goods, bads, the)
448:
449:     def con2(the):
450:         "discretize china"
451:         Eg.con1(the, "./data/china.csv")
452:
453:     def con3(the, file="./data/diabetes.csv",
454:             k1="positive", k2="negative"):
455:         "contrast sets from diabetes"
456:         rs = stratify(csv(file))
457:         contrasts(rs.klass[k1], rs.klass[k2], the)
458:
459:     def bins1(goods, bads, the):
460:         for good, bad in zip(goods.cols.x, bads.cols.x):
461:             bins = sorted(good.bins(bad, the))
462:             if len(bins) > 1:
463:                 print(f"\n{good.txt}")
464:                 for bin in bins:
465:                     print(f" {bin}")
466:
467: # Main program
468: def main(the):
469:     def run(fun, fails, the):
470:         s = f" {fun.__name__:<12}"

```

```

471:         the = copy.deepcopy(the)
472:         the.report = Report()
473:         random.seed(the.SEED)
474:         if the.UNSAFE:
475:             print("unsafe mode:")
476:             fun(the)
477:             sys.exit()
478:         try:
479:             fun(the)
480:             color(green=(chr(10003) + s), white=fun.__doc__)
481:         except Exception as err:
482:             fails = fails + 1
483:             color(red=(chr(10007) + s), white=str(err))
484:         return fails
485:     #
486:     fails = 0
487:     if the.XAMPLE == "all":
488:         for k, f in vars(Eg).items():
489:             if k[0] != " " and k != "ls":
490:                 fails = run(f, fails, the)
491:     else:
492:         if the.XAMPLE and the.XAMPLE in vars(Eg):
493:             f = vars(Eg)[the.XAMPLE]
494:             if the.XAMPLE == "ls":
495:                 f(the)
496:             else:
497:                 fails = run(f, fails, the)
498:     sys.exit(fails)
499:
500: if __name__ == "__main__":
501:     main(cli("./keys", __doc__, config()))

```