

```

1 #!/usr/bin/env python3
2 # vim: ts=2 sw=2 sts=2 et :
3
4 #
5 #      dew
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 /sublime.py [OPTIONS]
31 (c)2022 Tim Menzies <timmm@ieee.org> unlicense.org.
32 Sublime's unsupervised bifurcation:
33 let's infer minimal explanations.
34
35 OPTIONS:
36
37 -Max      max numbers to keep      : 512
38 -Some     find 'far' in this many eggs : 512
39 -cautious On any crash, stop+show stack : False
40 -data     data file                 : ./data/aut093.csv
41 -enough   min leaf size             : .5
42 -help     show help                 : False
43 -far      how far to look in 'Some'   : .9
44 -p        distance coefficient       : 2
45 -seed     random number seed        : 10019
46 -todo     start up task              : nothing
47 -xsmall   Cohen's small effect      : .35
48
49 ## See Also
50
51 [issues](https://github.com/timmm/sublime/issues)
52 âM-^@e [repo](https://github.com/timmm/sublime)
53 âM-^@e [view src](https://raw.githubusercontent.com/timmm/sublime/main/docs/sublime.pdf)
54
55 ## Algorithm
56
57 Stochastic clustering to generate tiny models. Uses random projections
58 to divide the space. Then, optionally, explain the clusters by
59 unsupervised iterative dichotomization using ranges that most
60 distinguish sibling clusters.
61
62 e.g.1: just bi-cluster on two distant points
63
64 ...
65 /sublime.py -c -s $RANDOM -t cluster
66
67 .. : 398
68 .. : 199
69 .. : 99
70 .. : 49 Lbs- Acc+ Mpg+
71 .. : 24 : [2255, 15.5, 30]
72 .. : 25 : [2575, 16.4, 30]
73 .. : 50
74 .. : 25 : [2110, 16.4, 30] <== best
75 .. : 25 : [2205, 16, 30]
76 .. : 100
77 .. : 50
78 .. : 25 : [2234, 15.5, 30]
79 .. : 25 : [2278, 16.5, 30]
80 .. : 50
81 .. : 25 : [2220, 15.5, 30]
82 .. : 25 : [2320, 15.8, 30]
83 .. : 199
84 .. : 99
85 .. : 49
86 .. : 24 : [2451, 16.5, 20]
87 .. : 25 : [3021, 15.5, 20]
88 .. : 50
89 .. : 25 : [3425, 17.6, 20]
90 .. : 25 : [3155, 16.7, 20]
91 .. : 100
92 .. : 50
93 .. : 25 : [4141, 13.5, 10]
94 .. : 25 : [4054, 13.2, 20]
95 .. : 50
96 .. : 25 : [4425, 11, 10]
97 .. : 25 : [4129, 13, 10]
98
99 e.g. #2, as above but split on range that mist divides data
100
101 ...
102 /sublime.py -c -s $RANDOM -t xplain
103
104 .. : 398 Lbs- Acc+ Mgg+
105 .. : [2807, 15.5, 20]
106 198 <= Lbs < 454 : 167 : [3725, 14.5, 20]
107 .. Modl < 72 : 34 : [3609, 13, 20]
108 .. Modl >= 72 : 133 : [3735, 14.9, 20]
109 .. Cylr < 8 : 56 : [3336, 17, 20]
110 .. 77 <= Modl < 82 : 22 : [3410, 17.1, 20]
111 .. Modl < 77 or Modl >= 82 : 34 : [3233, 17, 20]
112 .. Cylr >= 8 : 77 : [4129, 13.2, 20]
113 .. Modl < 75 : 37 : [4274, 13, 10]
114 .. Modl >= 75 : 40 : [3962, 13.5, 20]
115 .. Lbs >= 302 : 35 : [4054, 13.2, 20]
116 Lbs < 198 or Lbs >= 454 : 231 : [2290, 16, 30] <== best
117

```

```

118 ## License
119
120 This is free and unencumbered software released into the public
121 domain.
122
123 Anyone is free to copy, modify, publish, use, compile, sell, or
124 distribute this software, either in source code form or as a compiled
125 binary, for any purpose, commercial or non-commercial, and by any
126 means.
127
128 In jurisdictions that recognize copyright laws, the author or authors
129 of this software dedicate any and all copyright interest in the
130 software to the public domain. We make this dedication for the
131 benefit of the public at large and to the detriment of our heirs
132 and successors. We intend this dedication to be an overt act of
133 relinquishment in perpetuity of all present and future rights to
134 this software under copyright law.
135
136 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
137 EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
138 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
139 IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
140 OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
141 ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
142 OR OTHER DEALINGS IN THE SOFTWARE.
143
144 For more information, please refer to <http://unlicense.org/>
145
146
147 import traceback, random, math, sys, re
148 from random import random as r
149 from typing import Any

```

```

150 #
151 #
152 #
153 #
154 #
155 #
156 #
157
158 def any(a:list) -> Any:
159     "Return a random item."
160     return a[anywhere(a)]
161
162 def anywhere(a:list) -> int:
163     "Return a random index of list 'a'."
164     return random.randint(0, len(a)-1)
165
166 big = sys.maxsize
167
168 def atom(x):
169     "Return a number or trimmed string."
170     x=x.strip()
171     if x=="True": return True
172     elif x=="False": return False
173     else:
174         try: return int(x)
175         except:
176             try: return float(x)
177             except: return x.strip()
178
179 def demo(do,all):
180     "Maybe run a demo, if we want it, resetting random seed first."
181     todo = dir(all)
182     if do and do != "all":
183         todo = [x for x in dir(all) if x.startswith(do)]
184     for one in todo:
185         fun = all.__dict__.get(one,"")
186         if type(fun)==type(demo):
187             random.seed(the.seed)
188             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
189             try:
190                 fun()
191                 print("PASS:", doc)
192             except Exception as e:
193                 all.fails += 0
194                 if the.cautious: traceback.print_exc(); exit(1)
195                 else: print("FAIL:", doc, e)
196             exit(all.fails)
197
198 def file(f):
199     "Iterator. Returns one row at a time, as cells."
200     with open(f) as fp:
201         for line in fp:
202             line = re.sub(r'([\\n\\r\\v\\t])#.*', '', line)
203             if line:
204                 yield [atom(cell.strip()) for cell in line.split(",")]
205
206 def first(a:list) -> Any:
207     "Return first item."
208     return a[0]
209
210 def merge(b4:list) -> list:
211     "While we can find similar adjacent things, merge them."
212     j,n,now = -1,len(b4),[]
213     while j < n-1:
214         j += 1
215         a = b4[j]
216         if j < n-2:
217             if merged := a.merge(b4[j+1]):
218                 a = merged
219                 j += 1 # we will continue, after missing one
220             now += [a]
221         # if 'now' is same size as 'b4', look for any other merges.
222     return b4 if len(now)==len(b4) else merge(now)
223
224 class o(object):
225     "Class that can pretty print its slots, with fast inits."
226     def __init__(i, **d): i.__dict__.update(**d)
227     def __repr__(i):
228         pre = i.__class__.__name__ if isinstance(i,o) else ""
229         return pre+str(
230             {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})
231
232 def options(doc:str) -> o:
233     """Convert 'doc' to options dictionary using command line args.
234     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
235     those on the command line means 'flip the default value'; (2) args need only
236     mention the first few of a key (e.g. -s is enough to select for -seed)."""
237     d={}
238     for line in doc.splitlines():
239         if line and line.startswith(" -"):
240             key, _, x = line.strip()[1:].split("#") # get 1st,last word on each line
241             for j,flag in enumerate(sys.argv):
242                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
243                     x= "True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
244             d[key] = atom(x)
245     if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
246     return o(**d)
247
248 def r() -> float:
249     "Return random number 0..1"
250     return random.random()
251
252 def rn(x:float, n=3) -> float:
253     "Round a number to three decimals."
254     return round(x,n)
255
256 def rN(a:list, n=3) -> list:
257     "Round a list of numbers to three decimals."
258     return [rn(x,n=n) for x in a]
259
260 def second(a:list) -> Any:
261     "Return second item."
262     return a[1]

```

```

263 #
264 #
265 #
266 #
267 #
268 #
269 #
270
271 #
272 #
273 #
274 #
275 #
276
277 class Span(o):
278     """Given two 'Sample's and some 'x' range 'lo..hi'.
279     a 'Span' holds often that range appears in each 'Sample'."""
280     def __init__(i,col, lo, hi, ys=None):
281         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
282
283     def add(i, x:float, y:Any, inc=1) -> None:
284         "y' is a label identifying one 'Sample' or another."
285         i.lo = min(x, i.lo)
286         i.hi = max(x, i.hi)
287         i.ys.add(y,inc)
288
289     def merge(i, j): # -> Span|None
290         "If the merged span is simpler, return that merge."
291         a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
292         if (i.ys.n==0 or j.ys.n==0 or
293             c.div()*.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
294             return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
295
296     def selects(i,row:list) -> bool:
297         "True if the range accepts the row."
298         x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi
299
300     def show(i, positive=True) -> None:
301         "Show the range."
302         txt = i.col.txt
303         if positive:
304             if i.lo == i.hi: return f"[{txt}] == {i.lo}"
305             elif i.lo == -big: return f"[{txt}] < {i.hi}"
306             elif i.hi == big: return f"[{txt}] >= {i.lo}"
307             else: return f"[{i.lo}] <= {txt] < {i.hi}"
308         else:
309             if i.lo == i.hi: return f"[{txt}] != {i.lo}"
310             elif i.lo == -big: return f"[{txt}] >= {i.hi}"
311             elif i.hi == big: return f"[{txt}] < {i.lo}"
312             else: return f"[{txt}] < {i.lo} or {txt] >= {i.hi}"
313
314     def support(i) -> float:
315         "Returns 0..1."
316         return i.ys.n / i.col.n
317
318     @staticmethod
319     def sort(spans : list) -> list:
320         "Good spans have large support and low diversity."
321         divs, supports = Num(), Num()
322         sn = lambda s: supports.norm( s.support())
323         dn = lambda s: divs.norm( s.ys.div())
324         f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
325         for s in spans:
326             divs.add( s.ys.div())
327             supports.add(s.support())
328         return sorted(spans, key=f)
329
330 #
331 #
332 #
333
334 class Col(o):
335     "Summarize columns."
336     def __init__(i,at=0,txt=""):
337         i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "-" in txt else 1)
338
339     def dist(i,x:Any, y:Any) -> float:
340         return 1 if x=="?" and y=="?" else i.dist1(x,y)
341
342 #
343 #
344 #
345
346 #
347
348 class Sym(Col):
349     "Summarize symbolic columns."
350     def __init__(i,**kw):
351         super().__init__(**kw)
352         i.has, i.mode, i.mode = {}, None, 0
353
354     def add(i, x:str, inc:int=1) -> str:
355         "Update symbol counts in 'has', updating 'mode' as we go."
356         if x != " ":
357             i.n += inc
358             tmp = i.has[x] = inc + i.has.get(x,0)
359             if tmp > i.mode: i.mode, i.mode = tmp, x
360             return x
361
362     def dist(i,x:str, y:str) -> float:
363         "Distance between two symbols."
364         return 0 if x==y else 1
365
366     def div(i):
367         "Return diversity of this distribution (using entropy)."
368         p = lambda x: x / (1E-31 + i.n)
369         return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
370
371     def merge(i, j):
372         "Merge two 'Sym's."
373         k = Sym(at=i.at, txt=i.txt)
374         for x,n in i.has.items(): k.add(x,n)
375         for x,n in j.has.items(): k.add(x,n)
376         return k
377
378     def mid(i):
379         "Return central tendency of this distribution (using mode)."
380         return i.mode
381
382     def spans(i, j, out):
383         """For each symbol in 'i' and 'j', count the
384         number of times we see it on either side."""
385         xys = [(x,"this",n) for x,n in i.has.items()] + [
386             (x,"that",n) for x,n in j.has.items()]
387         one, last = None, None
388         all = []
389         for x,y,n in sorted(xys, key=first):
390             if x != last:
391                 last = x
392                 one = Span(i, x, x)
393                 all += [one]
394             one.add(x,y,n)
395         if len(all) > 1 : out += all

```

```

395 #
396 #
397 #
398 #
399 class Num(Col):
400     "Summarize numeric columns."
401     def __init__(i,**kw):
402         super().__init__(**kw)
403         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
404
405     def add(i,x: float ,inc=1):
406         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
407         if x != "":
408             i.n += inc
409             i.lo = min(x,i.lo)
410             i.hi = max(x,i.hi)
411             if len(i._all) < i.max : i.ok=False; i._all += [x]
412             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
413         return x
414
415     def all(i):
416         "Return '_all', sorted."
417         if not i.ok: i.ok=True; i._all.sort()
418         return i._all
419
420     def dist1(i,x,y):
421         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
422         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
423         else : x,y = i.norm(x), i.norm(y)
424         return abs(x-y)
425
426     def div(i):
427         """Report the diversity of this distribution (using standard deviation).
428         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
429         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
430         return (i.per(.9) - i.per(.1)) / 2.56
431
432     def merge(i,j):
433         "Return two 'Num's."
434         k = Num(at=i.at, txt=i.txt)
435         for x in i._all: k.add(x)
436         for x in j._all: k.add(x)
437         return k
438
439     def mid(i):
440         "Return central tendency of this distribution (using median)."
441         return i.per(.5)
442
443     def norm(i,x):
444         "Normalize 'x' to the range 0..1."
445         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
446
447     def per(i,p:float=.5) -> float:
448         "Return the p-th ranked item."
449         a = i.all(); return a[ int(p*len(a)) ]
450
451     def spans(i,j, out):
452         """Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
453         then count the number of times we hit the bin on other side.
454         Then merge similar adjacent bins."""
455         lo = min(i.lo, j.lo)
456         hi = max(i.hi, j.hi)
457         gap = (hi-lo) / (6/the.xsmall)
458         xys = [(x,"this",1) for x in i._all] + [(x,"that",1) for x in j._all]
459         one = Span(i.lo,lo)
460         all = [one]
461         for x,y,n in sorted(xys, key=first):
462             if one.hi - one.lo > gap:
463                 one = Span(i, one.hi,x)
464                 all += [one]
465                 one.add(x,y,n)
466             all = merge(all)
467             all[0].lo = -big
468             all[-1].hi = big
469             if len(all) > 1: out += all
470
471 #
472 #
473 #
474 #
475 #
476
477 class Explain(o):
478     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
479     def __init__(i,here):
480         i.here, i.span, i.yes, i.no = here, None, None, None
481
482     def show(i,pre=""):
483         if not pre:
484             tmp = i.here.mid(i.here.y)
485             print(f"{pre[:40]}: {len(i.here.rows):5} : {tmp}")
486         if i.yes:
487             s=f"{pre}{i.span.show(True)}"
488             tmp = i.yes.here.mid(i.yes.here.y)
489             print(f"{s[:40]}: {len(i.yes.here.rows):5} : {tmp}")
490             i.yes.show(pre + "|. ")
491         if i.no:
492             s=f"{pre}{i.i.span.show(False)}"
493             tmp = i.no.here.mid(i.no.here.y)
494             print(f"{s[:40]}: {len(i.no.here.rows):5} : {tmp}")
495             i.no.show(pre + "|. ")
496
497 #
498 #
499 #
500 #
501 #
502 class Cluster(o):
503     "Tree with 'left','right' samples, broken at median between far points."
504     def __init__(i,here,x=None,y=None,c=None,mid=None):
505         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
506
507     def show(i,pre=""):
508         s = f"{pre[:40]}: {len(i.here.rows):5}"
509         print(f"{s} if i.left else f'{s} : {i.here.mid(i.here.y)}")
510         for kid in [i.left,i.right]:
511             if kid: kid.show(pre + "|. ")

```

```

512 #
513 #
514 #
515 #
516 #
517
518 class Sample(o):
519     "Load, then manage, a set of examples."
520     def __init__(i,init=[]):
521         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
522         if str == type(inits): [i.add(row) for row in file(inits)]
523         if list == type(inits): [i.add(row) for row in inits]
524
525     def add(i,a):
526         def col(at,txt):
527             what = Num if txt[0].isupper() else Sym
528             now = what(at=at, txt=txt)
529             where = i.y if "==" in txt or "-" in txt or "!" in txt else i.x
530             if txt[-1] != " ":
531                 where += [now]
532                 if "==" in txt: i.klass = now
533             return now
534         #-----
535         if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
536         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
537
538     def clone(i,init=[]):
539         out = Sample()
540         out.add([col.txt for col in i.cols])
541         [out.add(x) for x in inits]
542         return out
543
544     def cluster(i,top=None):
545         """Split the data using random projections. Find the span that most
546         separates the data. Divide data on that span."""
547         here = Cluster(i)
548         top = top or i
549         if len(i.rows) >= 2*(len(top.rows)**the.enough):
550             left,right,x,y,c,mid = i.half(top)
551             if len(left.rows) < len(i.rows):
552                 here = Cluster(i,x,y,c,mid)
553             here.left = left.cluster(top)
554             here.right = right.cluster(top)
555         return here
556
557     def dist(i,x,y):
558         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
559         return (d/len(i.x)) ** (1/the.p)
560
561     def div(i,cols=None):
562         return [col.div() for col in (cols or i.all)]
563
564     def far(i, x, rows=None):
565         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
566         return tmp[ int(len(tmp)*the.far) ]
567
568     def half(i, top=None):
569         "Using two faraway points 'x,y' break data at median distance."
570         some= i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
571         top = top or i
572         w = any(some)
573         _,x= top.far(w, some)
574         c,y= top.far(x, some)
575         tmp = [r for _,r in sorted([(top.proj(r,x,y,c),r)
576                                     for r in i.rows],key=first))]
577         mid= len(tmp)//2
578         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
579
580     def mid(i,cols=None):
581         return [col.mid() for col in (cols or i.all)]
582
583     def proj(i,row,x,y,c):
584         "Find the distance of a 'row' on a line between 'x' and 'y'."
585         a = i.dist(row,x)
586         b = i.dist(row,y)
587         return (a**2 + c**2 - b**2) / (2*c)
588
589     def xplain(i,top=None):
590         """Split the data using random projections. Find the span that most
591         separates the data. Divide data on that span."""
592         here = Explain(i)
593         top = top or i
594         tiny = len(top.rows)**the.enough
595         if len(i.rows) >= 2*tiny:
596             left, right, *_ = i.half(top)
597             spans = []
598             [icol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
599             if len(spans) > 0:
600                 here.span = Span.sort(spans)[0]
601                 yes, no = i.clone(), i.clone()
602                 [yes if here.span.selects(row) else no].add(row) for row in i.rows
603                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
604                 if tiny <= len(no.rows ) < len(i.rows): here.no = no.xplain(top=top)
605             return here
606
607

```

```

608 #
609 #
610 #
611 #
612 #
613 #
614 #
615
616 class Demos:
617     "Possible start-up actions."
618     fails=0
619     def opt():
620         "show the config"
621         [print(f"{k}>10}={v}") for k,v in the.__dict__.items()]
622
623     def seed():
624         "seed"
625         assert .494 <= r() <= .495
626
627     def num():
628         "check 'Num'."
629         n = Num()
630         for _ in range(100): n.add(r())
631         assert .30 <= n.div() <= .31, "in range"
632
633     def sym():
634         "check 'Sym'."
635         s = Sym()
636         for x in "aaaabbc": s.add(x)
637         assert 1.37 <= s.div() <= 1.38, "entropy"
638         assert 'a' == s.mid(), "mode"
639
640     def rows():
641         "count rows in a file."
642         assert 399 == len([row for row in file(the.data)])
643
644     def sample():
645         "sampling"
646         s = Sample(the.data)
647         assert 398 == len(s.rows), "length of rows"
648         assert 249 == s.x[-1].has[1], "symbol counts"
649
650     def dist():
651         "distance between rows"
652         s = Sample(the.data)
653         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
654
655     def far():
656         "distant items"
657         s = Sample(the.data)
658         for _ in range(32):
659             a,_ = s.far(any(s.rows))
660             assert a>.5, "large?"
661
662     def clone():
663         "cloning"
664         s = Sample(the.data)
665         s1 = s.clone(s.rows)
666         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
667         for k,v in d1.items():
668             assert d2[k] == v, "clone test"
669
670     def half():
671         "divide data in two"
672         s = Sample(the.data); s1,s2,*_ = s.half()
673         print(s1.mid(s1.y))
674         print(s2.mid(s2.y))
675
676     def cluster():
677         "divide data in two"
678         s = Sample(the.data)
679         s.cluster().show(); print("")
680
681     def xplain():
682         "divide data in two"
683         s = Sample(the.data)
684         s.xplain().show(); print("")
685
686 #-----
687 the=options(__doc__)
688 if __name__ == "__main__": demo(the.todo,Demos)
689
690 """
691 all config local to Sample
692 Example class
693 """

```