

```

1  #!/usr/bin/env python3
2  # vim: ts=2 sw=2 sts=2 et :
3  #
4  #
5  #
6  #
7  #
8  #
9  #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 /sublime.py [OPTIONS]
18 (c)2022 Tim Menzies <timm@ieee.org>, BSD license
19 S.U.B.L.I.M.E. =
20 Sublime's unsupervised bifurcation: let's infer minimal explanations.
21
22 OPTIONS:
23
24 -Max      max numbers to keep      : 512
25 -Some     find 'far' in this many eggs : 512
26 -cautious On any crash, stop+show stack : False
27 -data     data file                  : data/auto93.csv
28 -enough   min leaf size              : .5
29 -help     show help                  : False
30 -far      how far to look in 'Some'   : .9
31 -p        distance coefficient        : 2
32 -seed     random number seed         : 10019
33 -todo     start up task               : nothing
34 -xsmall   Cohen's small effect        : .35
35
36 ## See Also
37
38 [issues](https://github.com/timm/sublime/issues)
39 :: [repo](https://github.com/timm/sublime)
40 :: [view source](https://github.com/timm/sublime/blob/main/docs/pdf)
41
42 ![https://img.shields.io/badge/purpose-se--ai-blueviolet)
43 ![https://img.shields.io/badge/language-python3-orange)
44 ![https://img.shields.io/badge/platform-osx_linux-pink)
45 <a href=https://github.com/timm/sublime/actions/workflows/main.yml><img
46 src=https://github.com/timm/sublime/actions/workflows/main.yml/badge.svg></a>
47 [!DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.5912461.svg)[https://doi.org/10.5281/zenodo.5912461]
48
49 ## Algorithm
50
51 Stochastic clustering to generate tiny models. Uses random projections
52 to divide the space. Then, optionally, explain the clusters by
53 unsupervised iterative dichotomization using ranges that most
54 distinguish sibling clusters.
55
56 ### Example1: just bi-cluster on two distant points
57
58 ...
59 /sublime.py -c -s $RANDOM -t cluster
60
61 .. : 398
62 .. : 199
63 .. : 99
64 .. : 49 Lbs- Acc+ Mpg+
65 .. : 24 : [2255, 15.5, 30]
66 .. : 25 : [2575, 16.4, 30]
67 .. : 50
68 .. : 25 : [2110, 16.4, 30] <== best
69 .. : 25 : [2205, 16, 30]
70 .. : 100
71 .. : 50
72 .. : 25 : [2234, 15.5, 30]
73 .. : 25 : [2278, 16.5, 30]
74 .. : 50
75 .. : 25 : [2220, 15.5, 30]
76 .. : 25 : [2320, 15.8, 30]
77 .. : 199
78 .. : 99
79 .. : 49
80 .. : 24 : [2451, 16.5, 20]
81 .. : 25 : [3021, 15.5, 20]
82 .. : 50
83 .. : 25 : [3425, 17.6, 20]
84 .. : 25 : [3155, 16.7, 20]
85 .. : 100
86 .. : 50
87 .. : 25 : [4141, 13.5, 10]
88 .. : 25 : [4054, 13.2, 20]
89 .. : 50
90 .. : 25 : [4425, 11, 10]
91 .. : 25 : [4129, 13, 10]
92
93
94
95 ### Example2: as above but split on range that most divides data
96
97 ...
98 /sublime.py -c -s $RANDOM -t xplain
99
100 .. : 398 : [2807, 15.5, 20]
101 .. : 167 : [3725, 14.5, 20]
102 .. : 34 : [3609, 13, 20]
103 .. : 133 : [3735, 14.9, 20]
104 .. : 56 : [3336, 17, 20]
105 .. : 22 : [3410, 17.1, 20]
106 .. : 34 : [3233, 17, 20]
107 .. : 77 : [4129, 13.2, 20]
108 .. : 37 : [4274, 13, 10]
109 .. : 40 : [3962, 13.5, 20]
110 .. : 35 : [4054, 13.2, 20]
111 .. : 231 : [2290, 16, 30] <== best

```

```

112 ## License
113
114 Redistribution and use in source and binary forms, with or without
115 modification, are permitted provided that the following conditions are met:
116 1. Redistributions of source code must retain the above copyright notice, this
117 list of conditions and the following disclaimer.
118 2. Redistributions in binary form must reproduce the above copyright notice,
119 this list of conditions and the following disclaimer in the documentation
120 and/or other materials provided with the distribution.
121
122 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
123 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
124 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
125 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
126 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
127 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
128 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
129 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
130 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
131 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
132 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
133
134
135 import traceback, random, copy, math, sys, re
136 from random import random as r
137 from typing import Any

```

```

138 #
139 #
140 #
141 #
142 #
143 #
144 #
145
146 def any(a:list) -> Any:
147     "Return a random item."
148     return a[anywhere(a)]
149
150 def anywhere(a:list) -> int:
151     "Return a random index of list 'a'."
152     return random.randint(0, len(a)-1)
153
154 big = sys.maxsize
155
156 def atom(x):
157     "Return a number or trimmed string."
158     x=x.strip()
159     if x=="True": return True
160     elif x=="False": return False
161     else:
162         try: return int(x)
163         except: return float(x)
164         except: return x.strip()
165
166 def demo(do,all):
167     "Maybe run a demo, if we want it, resetting random seed first."
168     todo = dir(all)
169     if do and do != "all":
170         todo = [x for x in dir(all) if x.startswith(do)]
171     for one in todo:
172         fun = all.__dict__[one, ""]
173         if type(fun)==type(demo):
174             random.seed(the.seed)
175             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
176             try:
177                 fun()
178                 print("PASS:", doc)
179             except Exception as e:
180                 all.fails += 0
181                 if the.cautious: traceback.print_exc(); exit(1)
182                 else: print("FAIL:", doc, e)
183             exit(all.fails)
184
185 def file(f):
186     "Iterator. Returns one row at a time, as cells."
187     with open(f) as fp:
188         for line in fp:
189             line = re.sub(r'(\n|\r|\t)|#.*', '', line)
190             if line:
191                 yield [cell.strip() for cell in line.split(",")]
192
193 def first(a:list) -> Any:
194     "Return first item."
195     return a[0]
196
197 def merge(b4:list) -> list:
198     "While we can find similar adjacent things, merge them."
199     j,n,now = -1, len(b4), []
200     while j < n-1:
201         j += 1
202         a = b4[j]
203         if j < n-2:
204             if merged := a.merge(b4[j+1]):
205                 a = merged
206                 j += 1 # we will continue, after missing one
207             now += [a]
208         # if 'now' is same size as 'b4', look for any other merges.
209     return b4 if len(now)==len(b4) else merge(now)
210
211 class o(object):
212     "Class that can pretty print its slots, with fast inits."
213     def __init__(i, **d): i.__dict__.update(**d)
214     def __repr__(i):
215         pre = i.__class__.__name__ if isinstance(i,o) else ""
216         return pre+'('+'(''.join([f":{k} {v}" for k, v in
217             sorted(i.__dict__.items()) if str(k)[0] != "_"]))+')?'
218
219 def options(doc:str) -> o:
220     """Convert 'doc' to options dictionary using command line args.
221     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
222     those on the command line means 'flip the default value'; (2) args need only
223     mention the first few of a key (e.g. -s is enough to select for -seed)."""
224     d={}
225     for line in doc.splitlines():
226         if line and line.startswith(" -"):
227             key, *_ = line.strip()[1:].split(" ") # get 1st,last word on each line
228             for j,flag in enumerate(sys.argv):
229                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
230                     x="True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
231                     d[key] = atom(x)
232             if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
233     return o(**d)
234
235 def r() -> float:
236     "Return random number 0..1"
237     return random.random()
238
239 def rn(x:float, n=3) -> float:
240     "Round a number to three decimals."
241     return round(x,n)
242
243 def rN(a:list, n=3) -> list:
244     "Round a list of numbers to three decimals."
245     return [rn(x,n=n) for x in a]
246
247 def second(a:list) -> Any:
248     "Return second item."
249     return a[1]
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 class Num(Col):
392     "Summarize numeric columns."
393     def __init__(i,size,**kw):
394         super().__init__(**kw)
395         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, size, False
396
397     def add(i,x: float ,inc=1):
398         "Reservoir sampler. If '._all' is full, sometimes replace an item at random."
399         if x != "":
400             i.n += inc
401             i.lo = min(x,i.lo)
402             i.hi = max(x,i.hi)
403             if len(i._all) < i.max : i.ok=False; i._all += [x]
404             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
405         return x
406
407     def all(i):
408         "Return '._all'.sorted."
409         if not i.ok: i.ok=True; i._all.sort()
410         return i._all
411
412     def dist1(i,x,y):
413         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
414         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
415         else : x,y = i.norm(x), i.norm(y)
416         return abs(x-y)
417
418     def div(i):
419         """Report the diversity of this distribution (using standard deviation).
420         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
421         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
422         return (i.per(.9) - i.per(.1)) / 2.56
423
424     def merge(i,j):
425         "Return two Num's."
426         k = Num(i.max, at=i.at, txt=i.txt)
427         for x in i._all: k.add(x)
428         for x in j._all: k.add(x)
429         return k
430
431     def mid(i):
432         "Return central tendency of this distribution (using median)."
433         return i.per(.5)
434
435     def norm(i,x):
436         "Normalize 'x' to the range 0..1."
437         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
438
439     def per(i,p:float=.5) -> float:
440         "Return the p-th ranked item."
441         a = i.all(); return a[ int(p*len(a)) ]
442
443     def prep(i,x):
444         "Return 'x' as a float."
445         return x if x=="?" else float(x)
446
447     def spans(i,j, bins, out):
448         """Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
449         then count the number of times we the bin on other side.
450         Then merge similar adjacent bins."""
451         lo = min(i.lo, j.lo)
452         hi = max(i.hi, j.hi)
453         gap = (hi-lo) / bins
454         xys = [(x,"this",1) for x in i._all] + [
455             (x,"that",1) for x in j._all]
456         one = Span(i.lo,lo)
457         all = [one]
458         for x,y,n in sorted(xys, key=first):
459             if one.hi - one.lo > gap:
460                 one = Span(i, one.hi,x)
461                 all += [one]
462             one.add(x,y,n)
463         all = merge(all)
464         all[0].lo = -big
465         all[-1].hi = big
466         if len(all) > 1: out += all
467
468 #
469 #
470 #
471 #
472
473 class Explain(o):
474     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
475     def __init__(i,here):
476         i.here, i.span, i.yes, i.no = here, None, None, None
477
478     def show(i,pre=""):
479         if not pre:
480             tmp = i.here.mid(i.here.y)
481             print(f"[{pre:40}]: {len(i.here.rows):5} : {tmp}")
482         if i.yes:
483             s=f"[pre]{i.span.show(True)}"
484             tmp = i.yes.here.mid(i.yes.here.y)
485             print(f"[s:40] : {len(i.yes.here.rows):5} : {tmp}")
486             i.yes.show(pre + "|.")
487         if i.no:
488             s=f"[pre]{i.span.show(False)}"
489             tmp = i.no.here.mid(i.no.here.y)
490             print(f"[s:40] : {len(i.no.here.rows):5} : {tmp}")
491             i.no.show(pre + "|.")
492
493 #
494 #
495 #
496 #
497
498 class Cluster(o):
499     "Tree with 'left','right' samples, broken at median between far points."
500     def __init__(i,here,x=None,y=None,c=None,mid=None):
501         i.here, i.x, i.y, i.c, i.mid, i.left, i.right = here, x, y, c, mid, None, None
502
503     def show(i,pre=""):
504         s = f"[pre:40] : {len(i.here.rows):5}"
505         print(f"[s] " if i.left else f"[s] : {i.here.mid(i.here.y)})")
506         for kid in [i.left, i.right]:
507             if kid: kid.show(pre + "|.")
508
509 #
510 #
511 #
512 #
513
514 class Sample(o):
515     "Load, then manage, a set of examples."
516
517     def __init__(i, the, inits=[]):
518         i.the = the
519         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
520         if str ==type(inits): [i.add(row, True) for row in file(inits)]
521         if list==type(inits): [i.add(row) for row in inits]
522
523
524 def add(i, a, raw=False):
525     pre = lambda a,c: c.prep(a[c.at]) if raw else a[c.at]
526     nump = lambda x : x[0].isupper()
527     skipp = lambda x : x[-1]=="."
528     klassp = lambda x : "[ " in x
529     goalp = lambda x : "+ " in x or "- " in x or klassp(x)
530
531     def col(at,txt):
532         now = Num(i.the.Max,at=at,txt=txt) if nump(txt) else Sym(at=at,txt=txt)
533         where = i.y if goalp(txt) else i.x
534         if not skipp(txt):
535             where += [now]
536             if klassp(txt): i.klass = now
537         return now
538
539     #-----
540     if i.cols: i.rows += [[col.add(pre(a,col)) for col in i.cols]]
541     else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
542
543 def clone(i, inits=[]):
544     out = Sample(i.the)
545     out.add([col.txt for col in i.cols])
546     [out.add(x) for x in inits]
547     return out
548
549 def cluster(i, top=None):
550     """Split the data using random projections. Find the span that most
551     separates the data. Divide data on that span."""
552     here = Cluster(i)
553     top = top or i
554     if len(i.rows) >= 2*(len(top.rows)**i.the.enough):
555         left, right, x, y, c, mid = i.half(top)
556         if len(left.rows) < len(i.rows):
557             here = Cluster(i, x, y, c, mid)
558             here.left = left.cluster(top)
559             here.right = right.cluster(top)
560         return here
561
562 def dist(i,x,y):
563     d = sum( col.dist(x[col.at], y[col.at])**i.the.p for col in i.x )
564     return (d/len(i.x)) ** (1/i.the.p)
565
566 def div(i, cols=None):
567     return [col.div() for col in (cols or i.all)]
568
569 def far(i, x, rows=None):
570     tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
571     return tmp[ int(len(tmp)*i.the.far) ]
572
573 def half(i, top=None):
574     "Using two faraway points 'x,y' break data at median distance."
575     some = i.rows if len(i.rows)<i.the.Some else random.choices(i.rows, k=the.Some
576     e)
577     top = top or i
578     w = any(some)
579     _,x= top.far(w, some)
580     c,y= top.far(x, some)
581     tmp = [r for _,r in sorted([(top.proj(r,x,y,c),r)
582         for r in i.rows],key=first))]
583     mid= len(tmp)//2
584     return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
585
586 def mid(i, cols=None):
587     return [col.mid() for col in (cols or i.all)]
588
589 def proj(i, row, x, y, c):
590     "Find the distance of a 'row' on a line between 'x' and 'y'."
591     a = i.dist(row,x)
592     b = i.dist(row,y)
593     return (a**2 + c**2 - b**2) / (2*c)
594
595 def xplain(i, top=None):
596     """Split the data using random projections. Find the span that most
597     separates the data. Divide data on that span."""
598     here = Explain(i)
599     top = top or i
600     tiny = len(top.rows)**i.the.enough
601     if len(i.rows) >= 2*tiny:
602         left, right, *_ = i.half(top)
603         spans = []
604         [lcol.spans(rcol,6/i.the.xsmall,spans) for lcol,rcol
605             in zip(left.x, right.x)]
606         if len(spans) > 0:
607             here.span = Span.sort(spans)[0]
608             yes, no = i.clone(), i.clone()
609             [(yes if here.span.selects(row) else no).add(row) for row in i.rows]
610             if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
611             if tiny <= len(no.rows) < len(i.rows): here.no = no.xplain(top=top)
612         return here

```

```

610 #
611 #
612 #
613 #
614 #
615 #
616 #
617
618 class Demos:
619     "Possible start-up actions."
620     fails=0
621     def opt():
622         "show the config."
623         print(the)
624
625     def seed():
626         "seed"
627         assert .494 <= r() <= .495
628
629     def num():
630         "check 'Num'."
631         n = Num(512)
632         for _ in range(100): n.add(r())
633         assert .30 <= n.div() <= .31, "in range"
634
635     def sym():
636         "check 'Sym'."
637         s = Sym()
638         for x in "aaaabbc": s.add(x)
639         assert 1.37 <= s.div() <= 1.38, "entropy"
640         assert 'a' == s.mid(), "mode"
641
642     def rows():
643         "count rows in a file."
644         assert 399 == len([row for row in file(the.data)])
645
646     def sample():
647         "sampling."
648         s = Sample(the, the.data)
649         print(the.data, len(s.rows))
650         assert 398 == len(s.rows), "length of rows"
651         assert 249 == s.x[-1].has['l'], "symbol counts"
652
653     def dist():
654         "distance between rows"
655         s = Sample(the, the.data)
656         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
657
658     def far():
659         "distant items"
660         s = Sample(the, the.data)
661         for _ in range(32):
662             a,_ = s.far(any(s.rows))
663             assert a>.5, "large?"
664
665     def clone():
666         "cloning"
667         s = Sample(the, the.data)
668         s1 = s.clone(s.rows)
669         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
670         for k,v in d1.items():
671             assert d2[k] == v, "clone test"
672
673     def half():
674         "divide data in two"
675         s = Sample(the, the.data)
676         s1,s2,*_ = s.half()
677         print(s1.mid(s1.y))
678         print(s2.mid(s2.y))
679
680     def cluster():
681         "divide data in two"
682         s = Sample(the, the.data)
683         s.cluster().show(); print("")
684
685     def xplain():
686         "divide data in two"
687         s = Sample(the, the.data)
688         s.xplain().show(); print("")
689
690 #-----
691 the=options(__doc__)
692 if __name__ == "__main__": demo(the.todo,Demos)
693
694 """
695 Example class
696 """

```