

```

1
2
3
4
5
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
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

```



```

(defpackage :tiny (:use :cl))
(in-package :tiny)
(mapc #'load '("lib/macros" "lib/math" "lib/strings" "lib/lists"
               "lib/sets" "lib/structs" "lib/egs" ))

(defvar my (settings "
TINY: semi-supervised multi-objective explanation facility.
(c) 2022 Tim Menzies, BSD-2 clause license

USAGE: lisp eg.lisp [OPTIONS] [ARG]"
  '( (file "lib" "help file" " " "lib/data/auto93.lisp")
    (help "h" "show help" " " " ")
    (keep "k" "items to keep" " " 256)
    (k "k" "nb low attributes classes" 1)
    (m "m" "nb low frequency classes" 2)
    (p "p" "distance coefficient" 2)
    (seed "s" "random number seed" 10019)
    (example "e" "example to run" "ls")))

(mapc #'load '("col/sample" "row/row" "col/sym" "col/num" "col/cols" "row/rows"))

[Symbolic representation of a complex geometric pattern]

; Hold one record.
(defstruct+ row cells ; cells
  _cols ; pointer to someone who can say what are (e.g.) lo,hi
  _evald ; have we used the y values

(defun make-row (cols l) (%make-row :cells l :_cols cols))

(defun method better ((row1 row) (row2 row))
  (let* ((s1 0) (s2 0) (d 0) (n 0)
        (cols (? row1 _cols y))
        (n (length cols)))
    (setf (? row1 _evald) t)
    (setf (? row2 _evald) t)
    (dolist (col cols (< (/ s1 n) (/ s2 n)))
      (with-slots (at w) col
        (let ((x (norm col (elt (? row1 cells) at)))
              (y (norm col (elt (? row2 cells) at))))
          (decf s1 (exp (* w (/ (- x y) n))))
          (decf s2 (exp (* w (/ (- y x) n))))))))))

(defun method around ((row1 row) rows)
  (labels ((two (row2) (cons (dist (? row1 _cols) row1 row2) row2)))
    (sort (mapcar 'two rows) 'car))

[Symbolic representation of a complex geometric pattern]

; Place to hold rows, and their summaries.
(defstruct+ rows rows ; all the rows
  cols ; summaries of all the columns

(defun make-rows (&optional src (i (%make-rows)))
  (labels ((top.row.is.special (x) (if (? i cols)
    (push (add i x) (? i rows))
    (setf (? i cols) (make-cols x))))
    (if (stringp src)
      (with-lines src (lambda (line) (top.row.is.special (cells line))))
      (mapcar #'top.row.is.special src)
      i))

(defun method clone ((i rows) &optional src)
  (make-rows (cons (? i cols names) src)))

(defun method add ((i rows) (lst cons)) (add i (make-row i lst))
(defun method add ((i rows) (row1 row))
  (dolist (cols '(, (? i cols x) , (? i cols y)) row1)
    (dolist (col cols)
      (add col (elt (? row1 cells) (? col at)))))

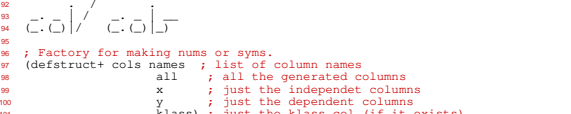
(defun method dist ((self rows) (row1 row) (row2 row))
  (let ((d 0) (n 0) (p (i my p)))
    (dolist (col (? self cols x))
      (incf n)
      (incf d (expt (dist col (elt (? row1 cells) (? col at))
                      (elt (? row2 cells) (? col at))
                      p)))
    (expt (/ d n) (/ 1 p)))

```

```

92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
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

```



```

; Factory for making nums or syms.
(defstruct+ cols names ; list of column names
  all ; all the generated columns
  x ; just the independent columns
  y ; just the dependent columns
  klass ; just the klass col (if it exists)

(defun make-cols (lst)
  (let ((all x y) (at -1))
    (dolist (str lst (%make-cols
      (names lst :x x :y y :klass kl :all (reverse all)))
      (let* ((what (if (upper-case-p (char str 0)) #'make-num #'make-sym))
            (col (funcall what str (incf at))))
        (push col all)
        (unless (eq #\~ (charn str))
          (if (member (charn str) '(\! \# \- \+)) (push col y) (push col x))
          (if (eq #\! (charn str)) (setf kl col)))))))

[Symbolic representation of a complex geometric pattern]

; Summarize numeric columns.
(defstruct+ num (txt "") ; column name
  (at 0) ; column position
  (n 0) ; #items seen
  (w 1) ; (1/-1) = (maximize, minimize)
  (lo most-positive-fixnum) ; least seen
  (hi most-negative-fixnum) ; most seen
  (_kept (make-sample))) ; items seen

(defun make-num (&optional (s "") (n 0))
  (%make-num :txt s :at n :w (if (eq #\~ (charn s)) -1 1)))

(defun method add ((i num) (lst cons)) (dolist (x lst i) (add i x))
(defun method add ((i num) x)
  (unless (eq x #\?)
    (with-slots (lo hi) i
      (incf (? i n))
      (add (? i _kept) x)
      (setf lo (min x (? i lo))
            hi (max x (? i hi)))))

(defun method norm ((i num) x)
  (with-slots (lo hi) i
    (cond ((eq x #\?) x)
          ((< (- hi lo) 1E-9) 0)
          (t (/ (- x lo) (- hi lo)))))

(defun method dist ((i num) x y)
  (cond ((and (eq #\? x) (eq #\? y))
    (return-from dist 1))
    ((eq #\? x) (setf y (norm i y) x (if (< y .5) 1 0)))
    ((eq #\? y) (setf x (norm i x) y (if (< x .5) 1 0)))
    (t (setf x (norm i x) y (norm i y)))
    (abs (- x y)))

(defun method div ((i num)) (div (? i _kept))
(defun method mid ((i num)) (mid (? i _kept))

(defun method discretize ((i num) x &optional (bins (? my bins)))
  (with-slots (lo hi) i
    (let ((b (/ (- hi lo) bins)))
      (if (= hi lo) 1 (* b (floor (+ .5 (/ x b)))))))

[Symbolic representation of a complex geometric pattern]

; Keep up to "max" numbers (after which, replace any old with new).
(defstruct+ sample
  _kept ; where to keep
  (make-array 2 :fill-pointer 0 :adjustable t))
(n 0)
max ; how many to keep
ok ; nil if items added and list not resorted yet

(defun make-sample (&optional (max (! my keep)))
  (%make-sample :max max))

(defun method add ((i sample) (x number))
  (incf (? i n))
  (let ((size (length (? i _kept))))
    (cond ((< size (? i max))
      (setf (? i ok) nil)
      (vector-push-extend x (? i _kept)))
      ((< (randf) (/ (? i n) (? i max)))
      (setf (? i ok) nil)
      (setf (elt (? i _kept) (randi size)) x))))

(defun method per ((i sample) p)
  (let* ((all (sorted i))
        (n (1- (length all))))
    (elt all (max 0 (min n (floor (* p n))))))

(defun method mid ((i sample)) (per i .5)
(defun method div ((i sample)) (/ (- (per i .9) (per i .1)) 2.58))

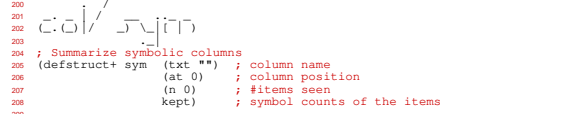
(defun method sorted ((i sample))
  (unless (? i ok)
    (sort (? i _kept) #'<)
    (setf (? i ok) t))
  (? i _kept))

```

```

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

```



```

; Summarize symbolic columns
(defstruct+ sym (txt "") ; column name
  (at 0) ; column position
  (n 0) ; #items seen
  kept ; symbol counts of the items

(defun make-sym (&optional n) (%make-sym :txt s :at n))

(defun method add ((i sym) (lst cons)) (dolist (x lst i) (add i x))
(defun method add ((i sym) x)
  (unless (eq x #\?)
    (incf (? i n))
    (incf (getf x (? i kept)))))

(defun method adds ((i sym) x inc)
  (incf (? i n) inc)
  (incf (getf x (? i kept)) inc))

(defun method div ((i sym))
  (labels ((fun (p) (* -1 (* p (log p 2)))))
    (loop for _ . n in (? i kept) sum (fun (/ n (? i n)))))

(defun method mid ((i sym))
  (loop for (key . n) in (? i kept) maximizing n return key))

(defun method dist ((i sym) x y)
  (cond ((and (eq #\? x) (eq #\? y)) 1)
        ((equal x y) 0)
        (t 1)))

```

```

233 lib/ init:cos
234
235 ; Simple alist access
236 (defmacro ! (l x) `(cdr (assoc 'x,l)))
237
238 ; ? obj x y z == (slot-value (slot-value (slot-value obj 'x) 'y) 'z)
239 (defmacro g (s x &rest xs)
240   (if (null xs) `(slot-value ,s 'x) `(? (slot-value ,s 'x) ,@xs)))
241
242 ; Endure !st has a slot for 'x'. If missing, initialize it with 'init'.
243 (defmacro gsta (x !st &optional (init 0))
244   `(cdr (or (assoc ,x !st :test #'equal)
245             (car (setf !st (cons (cons ,x ,init) !st))))))
246 lib/ init:hs
247
248 ; round
249 (defun rnd (number &optional (digits 3))
250   (let* ((div (expt 10 digits))
251         (tmp (/ (round (* number div)) div)))
252     (if (zerop digits) (floor tmp) (float tmp))))
253
254 ; Random number control (since reseeding in LISP is... strange).
255 (defvar *seed* 10013)
256
257 (defun randf (&optional (n 1.0))
258   (setf *seed* (mod (+ 16807.0d0 *seed* 2147483647.0d0)
259                     (* n (- 1.0d0 (/ *seed* 2147483647.0d0)))))
260
261 (defun randi (&optional (n 1)) (floor (* n (/ (randf 1000000000.0) 1000000000))))
262 lib/ str:inqs
263
264 ; Last thing from a string
265 (defun phasn (x)
266   (and (stringp x)
267        (> (length x) 0)
268        (char x (1- (length x)))))
269
270 ; Kill leading trailing whitespace.
271 (defun trim (x) (string-trim '(#\Space #\Tab #\Newline) x))
272
273 ; Turn 'x' into a number or string or "?"
274 (defun thing (x &aux (y (trim x)))
275   (cond ((string= y "?") #?)
276         ((string= y "i") t)
277         ((string= y "nil") nil)
278         (t (let ((z (read-from-string y nil nil)))
279              (if (numberp z) z y)))))
280
281 ; Divide 'str' on 'char', filtering all items through 'filter'.
282 (defun splits (str &key (char #\,) (filter #'identity))
283   (loop for start = 0 then (1+ finish)
284         for finish = (position char str :start start)
285         collecting (funcall filter (subseq str start finish))
286         until (null finish)))
287
288 ; String to lines or cells of things
289 (defun lines (string) (splits string :char #\Newline))
290 (defun cells (string &key (char #\,) (splits string :char char :filter #'thing))
291
292 ; Call 'fun' for each line in 'file'.
293 (defun with-lines (file fun)
294   (with-open-file (s file)
295     (loop (funcall fun (or (read-line s nil) (return))))))
296 lib/ str:inqs
297
298 ; sort predicates
299 (defun lt (x) (lambda (a b) (< (slot-value a x) (slot-value b x))))
300 (defun gt (x) (lambda (a b) (> (slot-value a x) (slot-value b x))))
301
302 (defun car< (x) (lambda (a b) (< (car a) (car b))))
303 (defun car> (x) (lambda (a b) (> (car a) (car b))))
304 lib/ str:inqs
305
306 ; Update 'default' from command line. Boolean flags just flip defaults.
307 (defun cli (key,flag,help,default)
308   (destructuring-bind (key flag help default) key,flag,help,default
309     (let* ((args #+clisp ext:*args*
310            #+sbcl sb-ext:*posix-argv*)
311           (it (member flag args :test 'equalp)))
312       (cons key (cond ((not it) default)
313                  ((equal default t) nil)
314                  ((equal default nil) t)
315                  (t (thing (second it)))))))
316
317 ; Update settings. If 'help' is set, print help.
318 (defun settings (header options)
319   (let ((tmp (mapcar #'cli options)))
320     (when (! tmp help)
321       (format t "~&-[-a-~%-~%OPTIONS:~%" (lines header))
322       (dolist (one options)
323         (destructuring-bind (_ flag help default) one
324           (format t " ~a -a ~a ~a ~a" flag help default))))
325     tmp))

```

```

336 lib/ str:inqs
337
338 ; Creates &x for constructor, enables pretty print, hides slots with "_" prefix.
339 (defmacro defstruct+ (x &body body)
340   (let* ((slots (mapcar (lambda (x) (if (consp x) (car x) x) body))
341         (show (remove-if (lambda (x) (eq #_ (char (symbol-name x) 0))) slots)))
342     `(:progn
343       (defstruct (,x (:constructor ,(intern (format nil "%MAKE--a" x)))) ,@body)
344       (defmethod print-object ((self ,x) str)
345         (labels ((fun (y) (format nil "~(-a)-a" y (slot-value self y))))
346           (format str "-a" (cons ',x (mapcar #'fun ',show)))))))
347
348 lib/ str:inqs
349
350 ; Define one demos.
351 (defvar *eggs* nil)
352 (defmacro eg (what arg doc &rest src)
353   `(push (list ',what ',doc (lambda (arg ,@src)) *eggs*)))
354
355 ; Run 'one' (or 'all') the demos. Reset globals between each run.
356 ; Return to the operating systems the failure count (so fails=0 means "success").
357 (defun demos (settings all &optional one)
358   (let ((fails 0)
359         (resets (copy-list settings)))
360     (dolist (trio all)
361       (destructuring-bind (what doc fun) trio
362         (setf what (format nil "~(-a)-" what))
363         (when (member what (list 'all one) :test 'equalp)
364           (loop for (key . value) in resets do
365             (setf (cdr (assoc key settings)) value))
366           (setf *seed* (or (cdr (assoc 'seed settings)) 10019))
367           (unless (eq t (funcall fun))
368             (incf fails)
369             (format t "~&FAIL[-a]-a ~%" what doc))))))
370     #+clisp (ext:exitit fails)
371     #+sbcl (sb-ext:exit :code fails)))

```

```

375 lib/ str:inqs
376
377 ; Test suite
378 (load "tiny")
379 (in-package :tiny)
380
381 (eg my () "show options" (pprint my) t)
382
383 (eg sym () "sym"
384   (let ((s (add (make-sym) 'a a a b b c)))
385     (and (= 1.379 (rnd (div s))) (eq 'c (mid s)))))
386
387 (eg sample () "sample"
388   (setf (! my keep) 64)
389   (let ((s (make-sample)))
390     (dotimes (i 100) (add s (1- i)))
391     (and (= 32.170544 (div s)) (= 56 (mid s)))))
392
393 (eg num () "num nums"
394   (setf (! my keep) 64)
395   (let ((n (make-num)))
396     (dotimes (i 100) (add n (1- i)))
397     (and (= 98 (? n hi)) (= 32.170544 (div n)) (= 56 (mid n)))))
398
399 (eg cols () "cols"
400   (print (make-cols '("aa" "bb" "Height" "Weight-" "Age-"))))
401
402 (eg lines () "lines"
403   (with-lines "../data/auto93.csv"
404     (lambda (x) (print (cells x)))))
405
406 (eg rows () "rows"
407   (let ((rows (make-rows "../data/auto93.csv")))
408     (print (? (? rows cols) y)))
409   t)
410
411 (eg dist () "dist"
412   (let (all
413         (r (make-rows "../data/auto93.csv")))
414     (dolist (two (cdr (? r rows)))
415       (push (dist r (car (? r rows) two) all))
416       (format t "~(-,3f)-" (sort all #'<))
417       t))
418   (demos my *eggs* (! my example)))

```