How to Code Less

(101 tips and tricks for smarter, smaller, coding)

Tim Menzies (timm@ieee.org)

2023-06-17

Contents

Chapter 1

Less, but Better?

One of the leading figures in design in the 20th century was Dieter Rams. His impact on the field cannot be over-stated. For example, if you look at any 21st century Apple computing product, you can see the lines and stylings he created in the 1950s:

Rams' designs were guided by several principles including:

- Good design makes a product understandable
- Good design is as little design as possible
- Back to simplicity. Back to purity. Less, but better.

Here, I apply "less, but better" to software engineering and knowledge engineering. At the time of this writing, there is something of a gold rush going on where everyone is racing to embrace very large and complex models. I fear that in all that rush, unless we pay more attention to the basics, we are going to forget a lot of hard-won lessons about how to structure code and how to explore new problems. Sure, sometimes new problems will need to more intricate and memory hungry and energy hungry methods. But always? I think not.

So let's review the basics, and see how they can be used to build understandable tools that run very fast. Then you will know enough when to use those tools, or when to reach for something else that is much more complicated.

These basic are divided in two:

- *software engineering* (SE): all the things I want intro-to-SE graduate students to know about cdding. These tips subdivide into:
  - team tips
  - system tips
  - scripting tips (here, I use Python for the scripting since many people have some understanding of that language).
- *knowledge engineering* (KE): all the things I want SE grad students to know about explainable AI and analytics.

Everything here will be example based. This book will present a small program (under a 1000 lines) that illustrates many of the tips and tricks I want to show.

By the way, that code is interesing in its own right.
`Tiny.py` is explainable multi-objective semi-supervised learner. If you do not know those terms, just relax. All they mean is that my AI tools generate tiny models you can read and understand, and that those tools describe how to reach mulitple goals, and that this is all done with the least amount of labels on the data. If that sounds complicated, it really isn't (see below).

Chapter 2

Less is More

The more you code, they more you have to insepct, test, monitor, port, package, document and maintain. The less you code, the easier it is to change and update and fix things.

Sometimes, the way to code less is to reject functionality if it means adding much more code for very little relative gain. For example, many programs have configuration variables that change what the code does, and those variables can be set from the command-line. Some libraries let you generating command-line interfaces direct from the code. One of the nicest is Vladimir Keleshev's `docopt` tool which builds the interface by parsing the docstring at top of file. `docopt` is under 500 lines of code and it works as well as other tools that are thousads to tens of thousands lines long.

My own alternative to `docopt` is 20 lines long. I present it here as a little case study is how to code less. Like `docopt`, my `settings` tool parses the command line from the top-of-file docstring:

```
"""
SYNOPSIS:
  less: look around just a little, guess where to search.

USAGE:
  ./less.py [OPTIONS] [-g ACTIONS]

OPTIONS:

  -b  --bins    max number of bins    = 16
  -c  --cohen   size significant separation = .35
  -f  --file    data csv file         = "../data/auto93.csv"
  -g  --go      start up action       = "nothing"
  -h  --help    show help             = False
  -k  --keep    how many nums to keep = 512
  -l  --lazy    lazy mode             = False
  -m  --min     min size              = .5
  -r  --rest    ratio best:rest       = 3
  -s  --seed    random number seed    = 1234567891
  -t  --top     explore top  ranges   = 8
  -w  --want    goal                  = "mitigate"
"""
```

asdas

```
import random,math,sys,ast,re
from termcolor import colored
from functools import cmp_to_key
from ast import literal_eval as thing

class BAG(dict): __getattr__ = dict.get
the = BAG(**{m[1]:thing(m[2])
             for m in re.finditer(r"\n\s*-\w+\s*--(\w+)[^=]*=\s*(\S+)",__doc__)})

random.seed(the.seed)    # set random number seed
R = random.random        # short cut to random number generator
isa = isinstance         # short cut for checking types
big = 1E30               # large number

egs={}                                  # place to store examples
def eg(f): egs[f.__name__]= f; return f # define one example
def run1():                             # run one example
  a=sys.argv; return a[1:] and a[1] in egs and egs[a[1]]()

@eg
def thed(): print(the)

class base(object):
  def __repr__(i):
    return i.__class__.__name__+str({k:v for k,v in i.__dict__.items() if k[0] !=
           "_"})

class ROW(base):
  def __init__(i, cells=[]): i.cells=cells

class COL(base):
  def __init__(i, at="",txt=""): i.at,i.txt = at,txt
  def add(i,x):
    if x != "?":
      i.n += 1
      i.add1(x)

def rnd(x,decimals=None):
  return round(x,decimals) if decimals else x

def per(a,p=.5):
  return a[int(max(0,min(len(a)-1,p*len(a))))]

@eg
def rnded(): assert 3.14 == rnd(math.pi,2)

@eg
def pered(): assert 33 == per([i for i in range(100)], .33)

class NUM(COL):
  def __init__(i, **d):
    COL.__init__(i,**d)
    i.w = -1 if len(i.txt) > 0 and i.txt[-1] == "-" else 1
    i._has,i.ready = [],False
    i.lo, i.hi = big, -big
  def has(i):
    if not i.ready:
      i.ready=True
      i._has.sort()
      i.lo,i.hi = i._has[0], i._has[-1]
    return i._has
  def norm(i,x):
    return x if x=="?" else  (x-i.lo)/(x.hi - x.lo + 1/big)
  def mid(i,decimals=None):
    return rnd( per(i.has(),.5), decimals)
  def div(i,decimals=None):
    return rnd( (per(i.has(),.9) - per(i.has(),.1))/2.56, decimals)
  def add1(i,x):
    a = i._has
    if   len(a) < the.keep  : i.ready=False; a += [x]
    elif R() < the.keep/i.n : i.ready=False; a[ int(len(a)*R()) ] = x
  def sub1(i,x): raise(DeprecationWarning("sub not defined for NUMs"))

@eg
def numed():
  n = NUM()
  for i in range(1000):  n.add(i)
  print(n.mid(), n.div())

class SYM(base):
  def __init__(i,**d):
    COL.__init__(i,**d)
    i.counts,i.mode, i.most = {},None,0
  def mid(i,**_): return i.mode
  def div(i, decimals=None):
    a = i.counts
    return rnd( - sum(a[k]/i.n * math.log(a[k]/i.n,2) for k in a if a[k] > 0),
                decimals)
  def add1(i,x):
    now = i.counts[x] = 1 + i.counts.get(x,0)
    if now > i.most: i.most, i.mode = now, x
  def sub(i,x):
    i.n -= 1
    i.counts[x] -= 1

def stats(cols, fun="mid", decimals=2):
  fun = lambda i,d:i.mid(d) if fun=="mid" else lambda i:i.div(d)
  return BAG(N=cols[1].n, **{col.txt:fun(col,decimals) for col in cols})

class COLS(base):
  def __init__(i,names):
    i.x,i.y, i.names = names,[],[]
    i.all = [(NUM(n,s) if s[0].isupper() else SYM(n,s)) for n,s in
             enumerate(names)]
    for col in i.all:
      z = col.txt[-1]
      if z != "X":
        if z=="!": i.klass= col
        (i.y if z in "-+!" else i.y).append(col)
  def add(i,row):
    for cols in [i.x, i.y]:
      for col in cols: col.add(row.cells[col.at])
    return row

def csv(file):
  def coerce(x):
    try: return ast.literal_eval(x)
    except: return x
  with open(file) as fp:
    for line in fp:
      line = re.sub(r'([\n\t\r"\' ]|#.*)', '', line)
      if line:
        yield [coerce(s.strip()) for s in line.split(",")]

class DATA(base):
  def __init__(i,src=[]):
    i.cols, i.rows = None,[]
    [i.add(row) for row in src]
  def add(i,row):
    row = ROW(row) if isa(row,list) else row
    if i.cols:
      i.rows += [i.cols.add(row)]
    else:
      i.cols = COLS(row.cells)
  def clone(i,rows=[]):
    return DATA([i.cols.names] + rows)
```

```
  def sort(i,rows=[]):
    return sorted(rows or i.rows, key=cmp_to_key(lambda r1,r2: i.better(r1,r2)))
  def better(i,row1,row2):
    s1, s2, n = 0, 0, len(i.cols.y)
    for col in i.cols.y:
      a, b = col.norm(row1.cells[col.at]), col.norm(row2.cells[col.at])
      s1  -= math.exp(col.w * (a - b) / n)
      s2  -= math.exp(col.w * (b - a) / n)
    return s1 / n < s2 / n


if __name__ == "__main__":
  if sys.argv[1:]: run1()
```