Evil Code for Wicked Problems, part 4

A Research Programmer's Guide to World Domination– in Python.
a.k.a. *Lecture Notes, Automated SE,* CS, NC State, Fall'15

by Tim Menzies
#attentionDeficitSquirrel

June 16, 2015

SYNOPSIS: This book is a "how to guide" on model-based reasoning using data mining and search-based tools (with examples taken from software engineering). It is intended for graduate students taking a one semester subject in advanced programming methods as well as researchers developing the next generation of model-based reasoning tools.

Using Python 2.7, the book builds (from the ground up) numerous tiny tools that can tame seemingly complex tasks. The combined toolkit, called RINSE, *represents* models using domain-specific languages. It also supports *inference* across the multiple goals of those models using multi-objective optimization. Further, it can succinctly *summarize* that inference using data miners. Finally, RINSE contains many tools for the *evaluation* of different inference methods.

RINSE is a not some shiny end-user click-and-point GUI package. Rather, it is a starter-kit that demonstrates an novel model-based approach to problem solving where programmers mix and match and extend data miners and multi-objective optimizers.

Contents

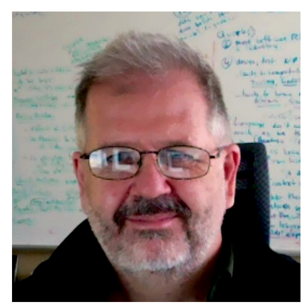
Content Advisory

This book contains strong language, weakly typed (and tapped with glee). This book may contain excessive or gratutious fun– as well as ideas that some readers may (or may not) find disturbing. This book does not necessarily believed or endorse those ideas- but plays with them anyway (and asks you to do the same). This book may include heresies, not suitable for anyone who believes in established wisdom, without adequate experimentation. It is intended for mature audiences only; i.e. those old enough to know there is much left to know. This book may (or may not) contain peanuts or tree nut products. Batteries not included.

Source Code Availability and Copyleft

To download this code, see http://github.com/txt/mase. The software associated with this book is free and unencumbered and released into the public domain.

About the Author



Tim Menzies (Ph.D., UNSW, 1995, http://menzies.us) is a full Professor in CS at North Carolina State University where he teaches software engineering and automated software engineering. His research relates to synergies between human and artificial intelligence, with particular application to data mining for software engineering.

In his career, he has been a lead researcher on projects for NSF, NIJ, DoD, NASA, USDA, as well as joint research work with private companies. He is the author of over 230 referred publications; and is one of the 100 most cited authors in software engineering out of over 80,000 researchers.

Prof. Menzies is an associate editor of IEEE Transactions on Software Engineering, Empirical Software Engineering and the Automated Software Engineering Journal. His community service includes co-founder of the PROMISE project (storing data for repeatable SE experiments); co-program chair for the 2012 conference on Automated SE and the 2015 New Ideas and Emerging Research track at the International Conference on SE; and co-general chair for 2016 International Conference on Software Maintenance and Evolution.

Prof. Menzies can be contacted at `tim.menzies@gmail.com`.

A

(An Introduction)

1 Welcome to the Evil Plan

"The world is a dangerous place to live, not because of the people who are evil, but because of the people who don't do anything about it." - Albert Einstein

The evil plan (by programmers) to take over the world is progressing nicely. Certain parts of that plan were initially somewhat undefined. However, given recent results, this book can now fill in the missing details from part4 of that plan.

But first, a little history. As all programmers know, the initial parts of the plan were completed years ago. Part one was was programmers to adopt a meek and mild persona (possibly even boring and dull).

Part two was, under the guise of that persona, ingratiated ourselves to government and indistrial agenices (education, mining, manufacturing, etc etc). Once there, make our work essential to their day to day opertion. Software is now a prime driven in innovation and all aspects of economic development. Software mediates most aspects of our daily lives such as the stock market models that control the economy; the probablistic models that recommend what books to read; and the pacemakers that govern the beating of our heart.

After that, part three was to make more material available for our inspection and manipluation. To this end, the planet was enclosed a digital network that grants us unprecendented access to petabytes of sensors and effectors. Also, by carefully seeding a few promienet examples of successful programmers (Gates, Jobs, Zuckerburg, thanks guys!), we convinced a lot of people to write lots of little tools, each of which represent or control some thing, somewhere.

Part four was a little tricky but, as shown in this book, it turned out not to be too hard. Having access to many models and much data can be overwhelming– unless some GREAT SECRET can be used to significantly simply all that information. For the longest time, that GREAT SECRET was unknown. However, recent advances have revealed that if we describe something in N dimensions, then there is usually a much smaller set of M dimensions that contain most of the signal. So GREAT SECRET is that is it very easy (and very fast) to find then exploit those few number of M dimensions for solving seemingly complex problems.

With those controllers in hand, we are now free to move to part five; i.e. taking over the world. In fact, the truly evil part of this work is this: *now you know you have the power to change the world.* This also means that (evil laugh) *now you have the guilt if you do not use that power to right the wrongs of the world.* So welcome to a lifetime of discontent (punctuated by the occasionaly, perhaps fleerting, truimphs) as you struggle to solve a very large number of pressing problems facing humanity.

www.shutterstock.com · 94524841

'Nough said. Good luck with that whole world domination thing. One tip: if at first you cannot dominate the whole thing, start out with something smaller. Find some people who have problems, then work with them to make changes that help them. Remember: if you don't try then you won't be able to sleep at night. Ever again (evil laugh).

1.1 Research Programming

Silliness aside, this book is about how to be a *research programmer*. Research programmer's understand the world by:

- Codify out current understanding of "it" into a model.
- Reasoning about the model.

We take this term "research programmer" from Ph.D. Steve Guao's 2012 dissertation.

1.1.1 Challenges with Research Programming

Research programming sounds simple, right? Well, there's a catch (actually, there are several catches).

Firstly, models have to be written and it can be quite a task to create and validate a model of some complex phenomenon.

see also list in sbse14

Secondly, many models related to *wicked problems*; i.e.~problems for which there is no clear best solution. Tittel XXXWorse still, some models relate to _wicked there is final matter of the *goals* that humans want to achieve with those models. When those goals are contradictory (which happens, all too often), then our model-based tools must negotiate complex trade offs between different possibilities.

Thirdly, if wicked problems were not eough, there is also the issue of uncertainty. Many real world models contain large areas of uncertainty, especially if that model relates to something that humans have only been studying for a few decades.

Fourthly, even if you are still not worried about the effectiveness of reserach problem, consider the complexity of real-world phenomonem. Many of these models are so complex that we cannot predict what happens when the parts of that model interact.

Sounds simple, right? Well, there's a catch. Many models related to *wicked problems*; i.e. problems for which there is no clear best solution. Tittel XXXWorse still, some models relate to _wicked there is final matter of the *goals* that humans want to achieve with those models. When those goals are contradictory (which happens, all too often), then our model-based tools must negotiate complex trade offs between different possibilities.

If wicked problems were not eough, there is also the issue of uncertainty. Many real world models contain large areas of uncertainty, especially if that model relates to something that humans have only been studying for a few decades.

And if you are still not worried about the effectiveness of reserach problem, consider the complexity of real-world phenomonem. Many of these models are so complex that we cannot predict what happens when the parts of that model interact.

1.1.2 Parts

- Domain specifc langauges (representation)
- execution (nuktu-objective ootiization)
- evaluation (statistical methods for experimental sciencetists in SE)
- Philophsopy (about what it means to know, and to doubt)

1.1.3 Implications for Software Engineering

Note that research programming changes the nature and focus and role of 21st century software engineering:

- Traditionally, software engineering is about services that meet requirements.
- But with research programming, software engineering is less about service than about search. Research programming's goal is the discovery of interesting features in existing models (or perhaps even the evolution of entirely new kinds of models).

For example, old-fashioned software engineerings might explore small things like strings or "hello world". But with research programmers explore **BIG** things like String Theory or "hello world model of climate change and economic impacts".

The GREAT SECRET

Example

brook's law. DSL in python of CM. data mining.

B

(Before we begin)

2 Lib: Standard Utilities

Standard imports: used everywhere.

2.1 Code Standards

Narrow code (52 chars, max); use i'', notself”, set indent to two characters,

In a repo (or course). Markdown comments (which means we can do tricks like auto-generating this documentation from comments in the file).

Not Python3, but use Python3 headers.

good reseraoiuces for advance people: Norving's infrenqencly asked questions

David Isaacon's Pything tips, tricks, and Hacks.http://www.siafoo.net/article/52

Environemnt that supports matplotlib, scikitlearn. Easy to get there.

Old school: install linux. New school: install virtualbox. Newer school: work online.

To checn if you ahve a suseful envorunment, try the following (isntall pip, matpolotlib, scikitlearn)

Learn Python.

Learn tdd

Attitude to coding. not code byt“set yourself up to et rapid feedback on some issue”

```
import random, pprint, re, datetime, time
from contextlib import contextmanager
import pprint,sys
```

Unit test engine, inspired by Kent Beck.

```
def ok(*lst):
  for one in lst: unittest(one)
  return one

class unittest:
  tries = fails = 0  #  tracks the record so far
  @staticmethod
  def score():
    t = unittest.tries
    f = unittest.fails
    return "# TRIES= %s FAIL= %s %%PASS = %s%%"  % (
      t,f,int(round(t*100/(t+f+0.001))))
  def __init__(i,test):
    unittest.tries += 1
    try:
      test()
    except Exception,e:
      unittest.fails += 1
      i.report(e,test)
  def report(i,e,test):
    print(traceback.format_exc())
    print(unittest.score(),':',test.__name__, e)
```

Simple container class (offers simple initialization).

```
class o:
  def __init__(i,**d)    : i.__dict__.update(**d)
  def __setitem__(i,k,v) : i.__dict__[k] = v
  def __getitem__(i,k)   : return i.__dict__[k]
  def __repr__(i)        : return str(i.items())
  def items(i,x=None)    :
    x = x or i
    if isinstance(x,o):
      return [k,i.items(v) for
              k,v in x.__dict__.values()
              if not k[0] === "_" ]
    else: return x
```

The settings system.

```
the = o()

def setting(f):
  name = f.__name__
  @wraps(f)
  def wrapper(**d):
    tmp = f()
    tmp.update(**d)
    the[name] = tmp
    return tmp
  wrapper()
  return wrapper


@setting
def LIB(): return o(
    seed =  1,
    has  = o(decs = 3,
             skip="_",
             wicked=True),
    show = o(indent=2,
             width=80)
)
#-------------------------------------------------
r    = random.random
any  = random.choice
seed = random.seed
isa  = isinstance

def lt(x,y): return x < y
def gt(x,y): return x > y
def first(lst): return lst[0]
def last(lst): return lst[-1]

def shuffle(lst):
  random.shuffle(lst)
  return lst

def ntiles(lst, tiles=[0.1,0.3,0.5,0.7,0.9],
               norm=False, f=3):
  if norm:
    lo,hi = lst[0], lst[-1]
    lst= g([(x - lo)/(hi-lo+0.0001) for x in lst],f)
  at = lambda x: lst[ int(len(lst)*x) ]
  lst = [ at(tile) for tile in tiles ]

  return lst

def say(*lst):
  sys.stdout.write(', '.join(map(str,lst)))
  sys.stdout.flush()

def g(lst,f=3):
  return map(lambda x: round(x,f),lst)
#-------------------------------------------------
def show(x, indent=None, width=None):
  print(pprint.pformat(has(x),
            indent= indent or the.LIB.show.indent,
            width = width  or the.LIB.show.width))


def cache(f):
  name = f.__name__
  def wrapper(i):
    i._cache = i._cache or {}
    key = (name, i.id)
    if key in i._cache:
      x = i._cache[key]
    else:
      x = f(i) # sigh, gonna have to call it
    i._cache[key] =  x # ensure ache holds 'c'
    return x
  return wrapper

@contextmanager
def duration():
  t1 = time.time()
  yield
  t2 = time.time()
  print("\n" + "-" * 72)
  print("# Runtime: %.3f secs" % (t2-t1))

def use(x,**y): return (x,y)

@contextmanager
def settings(*usings):
  for (using, override) in usings:
    using(**override)
  yield
  for (using,_) in usings:
    using()

@contextmanager
```

```
def study(what,*usings):
  print("\n#" + "-" * 50,
        "\n#", what, "\n#",
        datetime.datetime.now().strftime(
          "%Y-%m-%d %H:%M:%S"))
  for (using, override) in usings:
    using(**override)
  seed(the.LIB.seed)
  show(the)
  with duration():
    yield
  for (using,_) in usings:
    using()
```

3 Pandoc with citeproc-hs

Doe and Roe [2007]

References

John Doe and Jenny Roe. Why water is wet. In Sam Smith, editor, *Third Book*. Oxford University Press, Oxford, 2007.