`zymake`: a tool for running experiments

Eric Breck

Outline

- An example
- A shell script?
- A makefile?
- A new language

Introduction

- How do we run experiments (ML, NLP, ...)?
- You may think differently.
- Obviously things “work” now - can they be better?

An Example

- An experiment (Breck, Choi & Cardie, 2007)
- .../A .../O .../O .../A .../O .../O .../B .../O
- Train class-set $=$
  two-way distinction $\{\{A\}\{BO\}\}$, $\{\{B\}\{AO\}\}$,
  three-way distinction $\{\{A\}\{B\}\{O\}\}$,
  compare to baseline $\{\{AB\}\{O\}\}$
- Evaluate prediction (precision/recall) of $\{A\}$, $\{B\}$, $\{AB\}$
- Also 10-fold cross-validation
- Start with shell script

A shell script

```
for fold in 0 .. 9:
  extract-test-data $fold raw-data > $fold.test
  for class in A B A+B:
    extract-2way-training $fold raw-data $class > $fold.$class.train
    train   $fold.$class.train > $fold.$class.model
    predict $fold.$class.model $fold.test > $fold.$class.out
    prep-eval-2way $fold.$class.out > $fold.eval-in
    eval    $class $fold.$class.eval-in > $fold.$class.eval
  extract-3way-training $fold raw-data > $fold.3way.train
  train   $fold.3way.train > $fold.3way.model
  predict $fold.3way.model $fold.test > $fold.3way.out
  for class in A B A+B:
    prep-eval-3way $class $fold.3way.out > $fold.3way.$class.eval-in
    eval   $class $fold.3way.$class.eval-in > $fold.3way.$class.eval
```

Problem: What if something breaks? (it will)

- comment out code that's correct; re-run (brittle)
- script checks at each line whether needed (lots of work)
- other options?
-
- obviously this can happen multiple times.
- add to an existing script (another model, another graph)
  - another script?
  - add to same script?

Problem: Code Duplication

- `eval`, `train`, `predict` occur more than once
- Bad software engineering (propagate changes)
- A different organization?

Avoiding duplication: another shell script

```
for fold in 0 .. 9:
  extract-test-data $fold raw-data > $fold.test
  for class in A B A+B:
    extract-2way-training $fold raw-data $class \
      > $fold.2way.$class.train
  extract-3way-training $fold raw-data > $fold.3way.train
  for class in 2way.A 2way.B 2way.A+B 3way:
    train   $fold.$class.train > $fold.$class.model
    predict $fold.$class.model $fold.test > $fold.$class.out
  for class in A B A+B:
    prep-eval-3way $class $fold.3way.out > $fold.3way.$class.eval-in
    prep-eval-2way $fold.2way.$class.out > $fold.2way.$class.eval-in
    for train in 3way 2way:
      eval $class $fold.$train.$class.eval-in \
        > $fold.$train.$class.eval
```

note: now the list "A B A+B" is duplicated

Problem: non-locality

- each command has to know (and track) irrelevant attributes

```
eval $class $fold.$train.$class.eval-in > $fold.$train.$class.eval
```

- maybe a `makefile`?
  - pick up where we left off
  - code-sharing
  - locality
  - (but: funny variable names, annoying tabs, ...)

A `makefile` (partial)

```
%.train: %.model
  train $< > $@

%.out: %.model %.test
  predict $^ > $@
```

(What if we need args in a different order, or parameters between them?)

A `makefile` (continued)

```
%.test:
   extract-test-data $(fold-of $@) raw-data > $@

%.train:
   extract-2way-training $(fold-of $@) raw-data $(class-of $@) > $@
```

- fold-of? (hard to do)
- file - key-value pairs, not strings
- lists

```
all: $(foreach fold,0 .. 9,$(addsuffix .$(fold),
     $(foreach train,3way 2way,$(addsuffix .$(type),
     $(foreach class,A B A+B, o/example)))))
```

(What if I decide to change the order of the fields?)

- multiple kinds of dependencies

zymake

```
extract-test-data $(fold) raw-data > $().test
extract-2way-training $(fold) raw-data $(class) > $(train="2way").tra
extract-3way-training $(fold) raw-data > $(train="3way").train
train $().train > $().model
predict $().model $().test > $().out
prep-eval-3way $(class) $().out > $(train="3way").eval-in
prep-eval-2way $().out > $(train="2way").eval-in
eval $(class) $().eval-in > $().eval

classes = A B A+B
ways = 2way 3way

: $(fold = *(range 0 9) class = *classes train = *ways).eval
```

We'll break this down piece by piece

One query

```
eval $(class) magical-input-file > $().eval
```

```
: $(class="A").eval
```

this results in executing

```
eval A magical-input-file > o/ex.A.eval
```

- an expression interpolation
- an output file interpolation
- an input file interpolation

Multiple queries

```
eval $(class) magical-input-file > $().eval
```

```
: $(class="A").eval $(class="B").eval
```

this results in executing

```
eval A magical-input-file > o/ex.A.eval
eval B magical-input-file > o/ex.B.eval
```

Lists

```
eval $(class) magical-input-file > $().eval
classes = A B
: $(class=*classes).eval
```

this results in executing

```
eval A magical-input-file > o/ex.A.eval
eval B magical-input-file > o/ex.B.eval
```

More complex matching

```
prep-eval-3way $(class) magical-input > $(train="3way").eval-in
prep-eval-2way magical-input > $(train="2way").eval-in
eval $(class) $().eval-in > $().eval

classes = A B
ways = 2way 3way

: $(class = *classes train = *ways).eval
```

results in (note set of keys varies); see the bug?

```
prep-eval-3way B magical-input > o/ex.B.3way.eval-in
prep-eval-3way A magical-input > o/ex.A.3way.eval-in
prep-eval-2way magical-input > o/ex.2way.eval-in
eval B o/ex.B.3way.eval-in > o/ex.B.3way.eval
eval A o/ex.A.3way.eval-in > o/ex.A.3way.eval
eval B o/ex.2way.eval-in > o/ex.B.2way.eval
eval A o/ex.2way.eval-in > o/ex.A.2way.eval
```

Another example of generated commands

```
predict $().model $().test > $().out

predict o/ex.0.3way.model o/ex.0.test > o/ex.0.3way.out
predict o/ex.1.3way.model o/ex.1.test > o/ex.1.3way.out
predict o/ex.A.0.2way.model o/ex.0.test > o/ex.A.0.2way.out
predict o/ex.A.1.2way.model o/ex.1.test > o/ex.A.1.2way.out
predict o/ex.AB.0.2way.model o/ex.0.test > o/ex.AB.0.2way.out
predict o/ex.AB.1.2way.model o/ex.1.test > o/ex.AB.1.2way.out
predict o/ex.B.0.2way.model o/ex.0.test > o/ex.B.0.2way.out
predict o/ex.B.1.2way.model o/ex.1.test > o/ex.B.1.2way.out
```

Other features and benefits

- Parallel execution (topo-ordered dag)
- Simple interpolation syntax
- Scheme-like extension language
  `(range 0 9)` = `0 1 2 3 4 5 6 7 8 9`
- Automatically create filenames prepended with scriptname, avoiding collisions