zymake: a lightweight, computational workflow system for NLP and machine learning

Eric Breck
Cornell University
20 June 2008

How do I run all this stuff?

- Many programs
  - Tokenizer, tagger, stemmer, parser, learning algorithm, evaluation, gnuplot, ...
- Many options
  - Varying parameters, cross-validation, different algorithms, ...

Desiderata for an experimental system

- Reproducibility
- Simplicity
- Life-cycle
- Combinatorial experiments

An example experiment

- Goal: Identify *direct* and *indirect* opinion expressions
- Predict both at once (3way), or each separately (2way)?
- Evaluate on *direct* and *indirect*.
- 10-fold cross-validation

A shell script

```
for fold in `seq 0 9`; do
  train-3way $fold data $fold.3way.model
  predict $fold $fold.3way.model data >$fold.3way.out
  for class in direct indirect; do
    eval $class $fold.3way.out>$fold.3way.$class.eval
    train-2way $class $fold data $fold.$class.model
    predict $fold $fold.$class.model data >
      $fold.$class.out
    eval $class $fold.$class.out > $fold.$class.eval
  done
done
```

Problems

- Re-running the script
  - Programs break, later processing, ...
- Problematic filenames
  - `$fold.$class.eval ; $fold.3way.$class.eval`
- Modularization

A makefile

```
%.model:
  train-3way data $@

%.out: %.model
  predict $^ data > $@

%.eval: %.out
  eval $^ > $@
```

A makefile

```
%.model:
  train-3way data $@

%.out: %.model
  predict $^ data > $@

%.eval: %.out
  eval $^ > $@
```

But wait...

A problem with parameters

```
train-3way $fold data $fold.3way.model
predict $fold $fold.3way.model data >$fold.3way.out
eval $class $fold.3way.out > $fold.3way.$class.eval
```

```
%.model:
  train-3way $fold data $@

%.out: %.model
  predict $fold $^ data > $@

%.eval: %.out
  eval $class $^ > $@
```

Problems

- Filenames are opaque strings
- Combinatorial target
  - `all: 0.2way.direct.eval ... 9.3way.indirect.eval`
- Other dependencies

zymake

- Re-run like makefiles
- Key-value filenames
- Combinatorial files
- Simple syntax

Funny name...

zymake-ifying the shell script

```
for fold in `seq 0 9`; do
  train-3way $fold data $fold.3way.model
  predict $fold $fold.3way.model data >$fold.3way.out
  for class in direct indirect; do
    eval $class $fold.3way.out>$fold.3way.$class.eval
    train-2way $class $fold data $fold.$class.model
    predict $fold $fold.$class.model data >
      $fold.$class.out
    eval $class $fold.$class.out > $fold.$class.eval
  done
done
```

zymake-ifying the shell script

Ignore cross-validation for a moment

```
train-3way data 3way.model
predict 3way.model data > 3way.out
for class in direct indirect; do
  eval $class 3way.out > 3way.$class.eval
  train-2way $class data $class.model
  predict $class.model data > $class.out
  eval $class $class.out > $class.eval
done
```

zymake-ifying the shell script

Ignore cross-validation for a moment

```
train-3way data 3way.model
predict 3way.model data > 3way.out
for class in direct indirect; do
  eval $class 3way.out > 3way.$class.eval
  train-2way $class data $class.model
  predict $class.model data > $class.out
  eval $class $class.out > $class.eval
done
```

The zymake file

```
train-2way data $(> way="2way").model

train-3way $(class) data $(> way="3way").model

predict $().model data > $(>).out

eval $(class) $().out > $(>).eval
```

zymake-ifying the shell script

```
# Rule #1: how to create a 3way .model file
train-3way data $(> way="3way").model

# Rule #1: how to create a 2way .model file
train-2way $(class) data $(> way="2way").model

# Rule #3: how to create a .out file
predict $().model data > $(>).out

# Rule #4: how to create a .eval file
eval $(class) $().out > $(>).eval
```

zymake-ifying the shell script

```
# Rule #1: how to create a 3way .model file
train-3way data $(> way="3way").model

# Rule #1: how to create a 2way .model file
train-2way $(class) data $(> way="2way").model

# Rule #3: how to create a .out file
predict $().model data > $(>).out

# Rule #4: how to create a .eval file
eval $(class) $().out > $(>).eval
```

Variables

Input files

Output files

zymake-ifying the shell script

```
# Rule #1: how to create a 3way .model file
train-3way data $(> way="3way").model

# Rule #1: how to create a 2way .model file
train-2way $(class) data $(> way="2way").model

# Rule #3: how to create a .out file
predict $().model data > $(>).out

# Rule #4: how to create a .eval file
eval $(class) $().out > $(>).eval

: $(way="2way" class="direct").eval
  $(way="3way" class="direct").eval
  $(way="2way" class="indirect").eval
  $(way="3way" class="indirect").eval
```

zymake-ifying the shell script

```
# Rule #1: how to create a 3way .model file
train-3way data $(> way="3way").model

# Rule #1: how to create a 2way .model file
train-2way $(class) data $(> way="2way").model

# Rule #3: how to create a .out file
predict $().model data > $(>).out

# Rule #4: how to create a .eval file
eval $(class) $().out > $(>).eval

ways = 2way 3way
class = direct indirect

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

zymake-ifying the shell script

```
# Rule #1: how to create a 3way .model file
train-3way $(fold) data $(> way="3way").model

# Rule #1: how to create a 2way .model file
train-2way $(fold) $(class) data $(> way="2way").model

# Rule #3: how to create a .out file
predict $(fold) $().model data > $(>).out

# Rule #4: how to create a .eval file
eval $(class) $().out > $(>).eval

ways = 2way 3way
class = direct indirect
folds = 0 1 2 3 4 5 6 7 8 9

: $(way=*ways class=*classes fold=*folds).eval
```

Benefits of zymake

- Reproducibility
- Simplicity
- Experimental life-cycle
- Combinatorial experiments

Related work

- make replacements: ant, SCons, maven, ...
- Scientific workflow systems, e.g. Pegasus/ Wings
- NLP frameworks: GATE, UIMA

Parallel execution

- Straightforward - execute DAG elements in parallel
- Remote execution
  - Current: simple ssh-based system
  - Potentially could interface to various cluster systems (e.g. Condor's DAGman)

Future extensions

- Varying the DAG at run-time
- Optional control over generated filenames
- Version control

Conclusion

- NLP and ML experiments require running complex interdependent processes
- zymake offers a superior alternative to common approaches

Thank you!

- Questions?

- Suggestions?

- Try it!
  - http://www.cs.cornell.edu/~ebreck/zymake