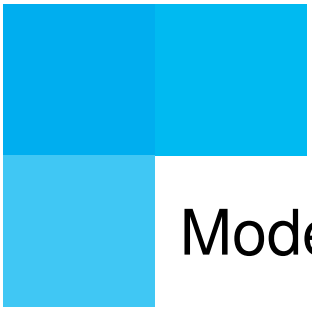
The Ohio State University



Modelblocks

Repository documentation

Last updated: February 23, 2017

Contents

Chapter 1

Overview

Modelblocks is a collection of recipes related to cognitive modeling of sentence processing.

It is implemented as a set of directories containing Makefiles, consisting of:

- a **config** directory (‘`config`’) which contains a set of ‘`user-*-directory.txt`’ files
  (these files contain locations of external code or data not part of Modelblocks);
- a set of **resource** directories, prefixed with ‘`resource-`’
  (recipes and code in these directories are designed to be reusable); and
- a set of **project** directories (those not prefixed with ‘`resource-`’, other than ‘`config`’)
  (recipes and code in these directories are designed to perform a single experiment).

Each resource or project directory contains a ‘`Makefile`’ file, containing its relevant recipes.

Resource and project directories also contain scripts, code, and curated data used in their recipes.

Makefiles in project directories usually begin with a default item, which replicates the main result of the project, followed by a colon, e.g.:

```
#### default item: the main product of this directory...
genmodel/wsj02to21.gcg15.linetrees:
```

This allows the main result of the project to be replicated by moving to that directory and typing:

```
make
```

Project Makefiles then contain a preamble, consisting of some useful make commands:

```
#### preamble...
.SUFFIXES:
.SECONDEXPANSION:
```

followed by a list of defined subdirectories which included files will make use of:

```
#### directories used by included items...
BIN := bin
GENMODEL := genmodel
```

Project Makefiles can then inherit recipes from resource directories:

```
#### included make items...
include $(dir $(CURDIR))resource-general/Makefile
```

Resource directories cannot include other resource directories because of limitations in the GNU make `include` command.

Please note that because Modelblocks primarily uses GNU Make to automate the code in experiments, dependencies to targets are created before the targets themselves. Thus, even though the recipes below are presented in API format with inputs and outputs, **you do not need to make the inputs before making an output target**. It should be sufficient to just type

```
make <stem>.<extension>
```

from the relevant directory to produce an output — make will handle the rest. For that reason, generally the most important information in the following entries is the "Stem template" field, which explains how to write and delimit parameters to targets so that they can be successfully made.

Generally, targets with extensions like 'widgetgadgets' will be formatted as plain-text matrices with rows for 'widgets' (delimited by newlines) and columns for 'gadgets' (delimited by spaces).

E.g. 'tokdecs' files are formatted as matrices with a row for each 'tok(en)' (delimited by newlines) and a column for each 'dec(ision)' (delimited by spaces).

Chapter 2

Repository Policy

Modelblocks has a local OSU git repository, which `ling.osu.edu` account holders can access with:

`git clone ssh://<username>@ling.osu.edu/home/compling/modelblocks-repository`

Here are a few guidelines about repository etiquette within this project:

1. VERY IMPORTANT: Do not push any code or data that is available elsewhere, especially if the license is not open-source! If you do, someone will have to purge the comp ling copy of the repo and ask all users to delete their local repos!

   This should instead be handled using '`config/user-*-directory.txt`' pointer files.

2. Only add project or resource directories to the repo that are likely to be useful to other people.

   Course projects, etc., can be maintained in private repos within your modelblocks directory.

3. Only create resource directories for recipes that are shared by more than one project.

Chapter 3

Terms

Modelblocks assumes the following definitions of terms:

- 'line': a sentence, title, or caption, which is syntactically independent of its neighbors.

  (e.g. '`April in Poetry`')
- 'token' ('tok'): an (arbitrary) elementary unit of syntactic annotation (may not be a word).

  (e.g. '`they`', '`do`', '`n't`', '`?`')
- 'item': a word as delimited in a source corpus.[1]

  (e.g. '`they`', '`don't?`')
- 'decision' ('dec'): an elementary unit of (processing) activity, as defined in annotated corpora.

  (e.g. a decision to fork at token 3)
- 'parameter' ('par'): the predictors and result of a decision according to a model.

  (e.g. '`F 2 VP : 1`')
- 'event' ('ev'): an elementary unit of stimulus in human subject data (i.e. regression targets).

  (e.g. exposure of subject 3 to region 22 of the stimulus)

[1] In some cases, parser input may need be tokenized more finely than the experimental stimuli are (e.g. splitting out punctuation and contractions). In these cases, the 'token'/'item' distinction becomes important. Tokens are always nested inside items.

Chapter 4

RESOURCE-LINETREES

RESOURCE-LINETREES contains methods for generating and manipulating syntactic annotations and evaluating parser accuracy.

4.1 %.linetoks

Extracts word sequences from an input %.linetrees file (i.e. deletes structure from a phrase-structure tree).

File format:

Specification:

$$\langle row \rangle \rightarrow \Big(\langle token \rangle \Big(\text{`` '' } \langle token \rangle \Big)^{*} \Big)^{?}$$

Example:

```
The cat did n't sleep .
```

4.1.1 Recipe: %.linetoks

Extracts word sequences from input trees.

Stem template:
⟨*corpus-specification*⟩ '.linetoks'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.1.2 Recipe: %.morph.linetoks

Uses Morfessor to segment input %.linetoks.

Stem template:
⟨*corpus-specification*⟩ '.morph.linetoks'

Input(s):

⟨*corpus-specification*⟩ '`.linetoks`'

4.1.3 Recipe: `%.rev.linetoks`

Reverses input `%.linetoks`.

Stem template: ⟨*corpus-specification*⟩ '`.rev.linetoks`'

Input(s):

⟨*corpus-specification*⟩ '`.linetoks`'

4.2 `%.linetrees`

Defines trees for each 'line' (sentence, title, caption, etc.) in a corpus using a generalization of the Penn Treebank standard.

File format:

Specification:

⟨*row*⟩ → ⟨*tree*⟩

⟨*tree*⟩ → '`(`' ⟨*node-label*⟩ (' ' ⟨*tree*⟩)$^{+}$ '`)`'

⟨*tree*⟩ → ⟨*node-label*⟩

Example:

```
(S (S (NP The cat) (VP did n't sleep)) (. .))
```

4.2.1 Recipe: `%.linetrees`

Generates a specification-conformant linetrees file from another file containing syntactic annotations.

Option 1: Converting external annotations

Stem template:
⟨*corpus-specification*⟩ '`.linetrees`'

Input(s):

External annotation (handlers provided by associated `RESOURCE-*` directory).

Option 2: Converting from `%.tokdecs`

Stem template:
⟨*corpus-specification*⟩ '`.linetrees`'

Input(s):

⟨*corpus-specification*⟩ '`.tokdecs`'

Option 3: Parsing source text using a probability model

Stem template:
⟨*test-set-specification*⟩ '.' ⟨*dash-delimited-trained-model-name*⟩ '-parsed.linetrees'

Input(s):

⟨*test-set-specification*⟩ '.linetoks'

⟨*dot-delimited-trained-model-name*⟩ '.parweights'

4.2.2 Recipe: %.first.linetrees

Extracts the first ⟨*number*⟩ of lines from the input.

Stem template:
⟨*corpus-specification*⟩ '.' ⟨*number*⟩ 'first.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.3 Recipe: %.last.linetrees

Extracts the last ⟨*number*⟩ of lines from the input.

Stem template:
⟨*corpus-specification*⟩ '.' ⟨*number*⟩ 'last.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.4 Recipe: %.onward.linetrees

Extracts lines ⟨*number*⟩ and onward from the input.

Stem template:
⟨*corpus-specification*⟩ '.' ⟨*number*⟩ 'onward.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.5 Recipe: %.maxwords.linetrees

Extracts only lines from the input containing at most ⟨*number*⟩ words.

Stem template:
⟨*corpus-specification*⟩ '.' ⟨*number*⟩ 'maxwords.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.6 Recipe: %.nolabel.linetrees

Removes syntactic labels from the input.

Stem template:
⟨*corpus-specification*⟩ '.nolabel.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.7 Recipe: %.nopunc.linetrees

Removes punctuation from the input.

Stem template:
⟨*corpus-specification*⟩ '.nopunc.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.2.8 Recipe: %.fromdeps.linetrees

Converts dependency representations into constituency tree representations using the [**?**] algorith (produces the flattest constituency trees permitted by the dependency graph).

Stem template:
⟨*corpus-specification*⟩ '.fromdeps.linetrees'

Input(s):

⟨*corpus-specification*⟩ '.tokdeps'

4.3 %.lineitems

Extracts word sequences from a corpus of psycholinguistic stimuli (where tokenizations generally differ from those used for parsing), preserving original tokenization.

File format:

Specification:

$$\langle row \rangle \rightarrow \Big(\langle item \rangle \Big(\text{`\ '} \langle item \rangle \Big)^{*} \Big)^{?}$$

Example:

```
The cat didn't sleep.
```

4.3.1 Recipe: %.lineitems

Stem template:
⟨*corpus-specification*⟩ '.lineitems'

Input(s):

External corpus

4.4 %.conll

Generates CoNLL-style dependencies from input %.linetrees by assigning the most frequently co-occurring child as the head.

File format: *See CoNLL documentation*

4.4.1 Recipe: %.conll

Stem template:
⟨*corpus-specification*⟩ '.conll'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.5 %.tokdeps

Generates Stanford-style dependencies from input %.linetrees by assigning the most frequently co-occurring child as the head.

File format: *See Stanford Dependencies documentation*

4.5.1 Recipe: %.tokdeps

Stem template:
⟨*corpus-specification*⟩ '.tokdeps'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

4.6 %.syneval

Generates human-readable report of differences between two %.linetrees inputs (typically a gold and a hypothesized sequence of trees).

File format: *Human-readable*

4.6.1 Recipe: %.syneval

Stem template:
⟨*dash-delimited-test-set-name*⟩ ⟨*dot-delimited-gold-edits*⟩ '.' ⟨*processing-specifications*⟩ '.syneval'

Input(s):

RESOURCE-GENERAL/srcmodel/new.prm

⟨*dot-delimited-test-set-name*⟩ ⟨*dot-delimited-gold-edits*⟩ '.linetrees' (control/baseline linetrees)

⟨*dot-delimited-test-set-name*⟩ '.' ⟨*processing-specifications*⟩ '.linetrees' (test linetrees)

4.7 %.bootstrapsignif

Generates human-readable report of statistical significance of difference between two %.syneval files (typically a control and a test sequence of eval scores) obtained via bootstrap resampling.

File format: *Human-readable*

4.7.1 Recipe: %.bootstrapsignif

Stem template:
⟨*common-onset*⟩ '..' ⟨*distinct-ctrl*⟩ '..' ⟨*distinct-test*⟩ '..' ⟨*common-coda*⟩ '.bootstrapsignif'

Input(s):

RESOURCE-GENERAL/srcmodel/new.prm

⟨*common-onset*⟩ '.' ⟨*distinct-ctrl*⟩ '.' ⟨*common-coda*⟩ '.syneval' ⟨*common-coda*⟩ (control/baseline syneval)

⟨*common-onset*⟩ '.' ⟨*distinct-test*⟩ '.' ⟨*common-coda*⟩ '.syneval' ⟨*common-coda*⟩ (test syneval)

4.8 %(.d).constiteval.txt

Alternative to %.syneval with additional measures relevant to unsupervised PoS tagging, parsing, and word segmentation (e.g. tagging and consituent labeling V-Measures, segmentation accuracy, etc.). Can be used when gold and test tokenizations differ arbitrarily. Generates human-readable report of differences between two %.linetrees inputs (typically a gold and a hypothesized sequence of trees).

File format: *Human-readable*

4.8.1 Recipe: %(.d.)constiteval.txt

Without the .d. option, generates summary metrics for entire input corpus. With it, generates additional per-sentence metrics (more informative but consumes much more disk space).

Stem template:
⟨*dash-delimited-test-set-name*⟩ ⟨*dot-delimited-gold-edits*⟩ '.' ⟨*processing-specifications*⟩ '.syneval'

Input(s):

⟨*distinct-ctrl*⟩ '.' ⟨*common-coda*⟩ ⟨.⟩ '.syneval' ⟨*common-coda*⟩ (control/baseline linetrees)

⟨*distinct-test*⟩ '.'
dtypecommon – *coda* ⟨.⟩ *dcode.syneval* ⟨*common-coda*⟩ (test linetrees)

4.8.2 Recipe: %.constiteval.txt

Stem template:
⟨*distinct-ctrl*⟩ '..' ⟨*distinct-test*⟩ '..' ⟨*common-coda*⟩ '.constiteval.txt'

Input(s):

⟨*distinct-ctrl*⟩ '.' ⟨*common-coda*⟩ (control/baseline constiteval)

⟨*common-onset*⟩ '.' ⟨*distinct-test*⟩ '.' ⟨*common-coda*⟩ (test constiteval)

4.9 %.constitevaltable.txt

Generates human-readable summary table of multiple %.constiteval.txt results that can be used for plotting.

File format:

Specification:

⟨*row*⟩ → (⟨*index*⟩ (' ' ⟨*eval-measure*⟩)*)?

Example:

```
iter bracketingF1 segmentationF1 ...
1 0.5611 0.9175 ...
```

4.9.1 Recipe: %.constitevaltable.txt

Stem template:
⟨*corpus-specification*⟩ '.' ⟨*distinct-test*⟩ '.' ⟨*common-coda*⟩ '.' ⟨*constiteval-list*⟩ '.constitevaltable.txt'

Input(s):

⟨*basename*⟩ '.constitevallist' (file containing paths to %.constiteval files to aggregate, in most cases must be created by hand)

4.10 %.learning_curves

Convenience target. Generates plots of a variety of learning curves from a %.constitevaltable.txt file.

File format: *Plot images*

4.10.1 Recipe: %.learning_curves

Stem template:
⟨*constiteval-list-specification*⟩ '.learning_curves'

Input(s):

⟨*constiteval-list-specification*⟩ '.constitevallist.txt'

Chapter 5

RESOURCE-LCPARSE

RESOURCE-LCPARSE contains methods for generating representations of incremental left-corner parsing operations.

5.1 %.linedeps

Generates sorted set of dependencies for each 'line' (sentence, title, caption, etc.) in a corpus. If present, '0' dependencies are used to indicate vertex types.

File format:

Specification:

⟨*row*⟩ → (⟨*dependency*⟩ (' ' ⟨*dependency*⟩)*)?

⟨*dependency*⟩ → ⟨*(source-)vertex*⟩ ',' ⟨*dependency-label*⟩ ',' ⟨*(destination-)vertex*⟩

Example:

```
01,0,cats 02,0,sleep 02,1,01
```

TODO: Is this used? I don't see any recipes anywhere in MB with this extension.

5.2 %.tokdecs

Generates list of modeled decisions for each token position in a corpus.

File format:

Specification:

⟨*row*⟩ → ⟨*word*⟩ ' ' ⟨*(preterminal-)sign*⟩ ' ' ⟨*fork*⟩ ' ' ⟨*join*⟩ ' ' (⟨*deriv-frag*⟩ (';' ⟨*deriv-frag*⟩)*)?

⟨*deriv-frag*⟩ → ⟨*(top-)sign*⟩ '/' ⟨*(bottom-)sign*⟩

⟨*sign*⟩ → '[' (⟨*referential-context*⟩ (',' ⟨*referential-context*⟩)*)? ']:' ⟨*category-label*⟩

Example:

```
the [the_0]:D 1 0 [the_1]:NP/[the_1]:N
very [very_0]:Adv 1 0 [the_1]:NP/[the_1]:N;[very_1]:AP/[very_1]:A
```

5.2.1 Recipe: %.fromlinetrees.tokdecs

Stem template:
⟨*corpus-specification*⟩ '.fromlinetrees.tokdecs'

Input(s):

⟨*corpus-specification*⟩ '.linetrees'

5.3 %.decpars

Generates list of parameterizations for training decisions. There are separate formats for ordinary conditional probability and logistic regression parameters.

5.3.1 Recipe: %.cpt.decpars

Conditional probability decpars.

File format:

Specification:

⟨*row*⟩ → ⟨*decision-type*⟩ (' ' ⟨*predictor-value*⟩)* ' :' (' ' ⟨*result-value*⟩)+

Example:

```
P 2 0 1 VP : VT
```

Note: This ordering has the advantage that, when sorted, parameters are grouped by condition

Stem template:
⟨*training-set-specifications*⟩ '.cpt.decpars'

Input(s):

⟨*training-set-specification*⟩ '.tokdecs'

5.3.2 Recipe: %.mlr.decpars

Multinomial logistic regression decpars.

File format:

Specification:

⟨*row*⟩ → ⟨*decision-type*⟩ ' ' (⟨*predictor*⟩ (',' ⟨*predictor*⟩)*)? ' : ' ⟨*result-value*⟩

⟨*predictor*⟩ → ⟨*predictor-name*⟩ '=' ⟨*predictor-value*⟩

Example:

```
F d0&tT=1,d0&Top=1 :  f1&N-xX*:doctor_1
```

Note: This ordering has the advantage that, when sorted, parameters are grouped by condition

Stem template:
⟨*training-set-specifications*⟩ '`.mlr.decpars`'

Input(s):

⟨*training-set-specification*⟩ '`.tokdecs`'

5.4 `%.parweights` files

Generates list of estimated parameter weights of a model. There are separate formats for ordinary conditional probability and logistic regression parameters.

5.4.1 Recipe: `%.cpt.parweights`

Conditional probability parweights.

File format:

Specification:

⟨*row*⟩ → ⟨*decision-type*⟩ (' ' ⟨*predictor-value*⟩)* ' :' (' ' ⟨*result-value*⟩)+ ' = ' ⟨*weight*⟩

Example:
`P 2 0 1 VP : VT = 0.4237643`

Note: This ordering has the advantage that, when sorted, parameters are grouped by condition

Stem template:
⟨*training-set-specifications*⟩ '`.cpt.parweights`'

Input(s):

⟨*training-set-specification*⟩ '`.cpt.decpars`'

5.4.2 Recipe: `%.mlr.parweights`

Multinomial logistic regression parweights.

File format:

Specification:

⟨*row*⟩ → ⟨*decision-type*⟩ ' ' ⟨*predictor-name*⟩ ' : ' ⟨*result-value*⟩ ' = ' ⟨*weight*⟩

Example:
`F d0&tT : f0&N-aD:doctor_1 = 0.318317`

Note: This ordering has the advantage that, when sorted, parameters are grouped by condition

Stem template:
⟨*training-set-specifications*⟩ '`.mlr.parweights`'

Input(s):

⟨*training-set-specification*⟩ ‘`.decpars`’

Chapter 6

RESOURCE-RT

RESOURCE-RT contains methods for aggregating measures from time-series psycholinguistic experiments into a single data table. It also contains convenience recipes for a number of reading-time experiment types.

6.1 %.tokmeasures

Generates a data table containing space-delimited columns of measurements for each token in a corpus.

File format:

Specification:

$$\langle row \rangle \rightarrow \Big(\langle token \rangle \Big(\text{`` '' } \langle measure \rangle \Big)^{*} \Big)^{?}$$

Example:

```
word totsurp ...
Are 10.2871 ...
n't 2.1029 ...
```

6.2 %.itemmeasures

Generates a data table containing space-delimited columns of measurements for each item in a corpus.

File format:

Specification:

$$\langle row \rangle \rightarrow \Big(\langle item \rangle \Big(\text{`` '' } \langle measure \rangle \Big)^{*} \Big)^{?}$$

Example:

```
word fwprob5 ...
```

```
Aren't -11.0926 ...
```

6.2.1 Recipe: %.itemmeasures

6.3 %.evmeasures

Generates a data table containing space-delimited columns of measurements for each event (unit of stimulus) in a psycholinguistic experiment.

File format:

Specification:

$$\langle row \rangle \rightarrow \Big(\langle event \rangle \Big(\text{`\ '} \langle measure \rangle \Big)^{*} \Big)^{?}$$

Example:

```
word subject RT ...
Aren't A 187 ...
Aren't B 411 ...
```

6.3.1 Recipe: %.evmeasures

Chapter 7

RESOURCE-LMEFIT

RESOURCE-LMEFIT contains methods for statistical analysis of psycholinguistic measures. The methods are primarily designed for use in reading time studies but should be generalizable to other kinds of dependent measures, as long as those measures can be associated with words/tokens in the stimuli.

7.1 %.rdata files

Generates saved binary of fitted linear mixed-effects regression model, along with a human-readable summary file in %.lmefit (same stem, different extension).

File format: *Binary file*

7.2 %.lmefit files

Contains summary data from a fitted linear mixed-effects regression model. Cannot be generated directly, but is instead generated as a secondary output of %.rdata.

File format: *Human-readable*

7.3 %.lrtsignif files

Generates a human-readable summary of results of likelihood ratio testing (LRT) two linear mixed-effects regression models.

File format: *Human-readable*

7.3.1 %.diamond.lrtsignif files

Generates a special subtype of %.lrtsignif that contain results from a 'diamond' LRT comparing two predictors against a baseline and against each other (i.e. four LRT comparisons rather than two).

File format: *Human-readable*