Morfessor FlatCat Documentation

Release 1.0.5

Stig-Arne Grönroos

Aug 14, 2017

Contents

Note: The Morfessor FlatCat documentation is still a work in progress and contains some unfinished parts.

Contents:

CHAPTER 1

License

CHAPTER 2

General

Morfessor FlatCat is a method for learning of morphological segmentation of natural language. FlatCat uses a Hidden Markov Model with the states (morph categories) `{prefix, stem, suffix, non-morpheme}` for modeling the word internal ordering of morphs. The non-morpheme category (marked ZZZ) is intended to catch segments that are not proper morphs because they are fragments of a larger morph. FlatCat can be trained in an unsupervised or semi-supervised fashion.

Publications

The Morfessor FlatCat method is presented in the paper *[Gronroos2014a]*. The paper is available for download from http://www.aclweb.org/anthology/C/C14/C14-1111.pdf

The Morfessor FlatCat method is described in detail in the Master's Thesis *[Gronroos2014b]*.

Morfessor FlatCat was developed together with the Morfessor 2.0 software package. The work done in Morfessor 2.0 is described in detail in the Morfessor 2.0 Technical Report *[TechRep]*. The report is available for download from http://urn.fi/URN:ISBN:978-952-60-5501-5.

Citing

The authors do kindly ask that you cite the Morfessor FlatCat paper *[Gronroos2014a]* when using this tool in academic publications.

In addition, when you refer to other Morfessor algorithms, you should cite the respective publications where they have been introduced. For example, the first Morfessor algorithm was published in *[Creutz2002]* and the semi-supervised extension in *[Kohonen2010]*. See *[TechRep]* for further information on the relevant publications.

CHAPTER 3

Quickstart guide

Setting up a virtual environment

The Morfessor packages are created using the current Python packaging standards, as described on http://docs.python.org/install/. Morfessor packages are fully compatible with, and recommended to run in, virtual environments as described on http://virtualenv.org.

If you don't want to use virtualenv, you can skip this step.

```
virtualenv flatcat
cd flatcat
source bin/activate
```

Installation instructions

Morfessor FlatCat installation packages can be obtained from the Morpho project homepage (latest stable version) or the FlatCat Github page (all versions).

Morfessor FlatCat 1.0 is installed using setuptools library for Python.

Unpack the tarball or zip file, change into the newly created directory, and then run:

```
python setup.py install
```

The setup command also installs Morfessor Baseline, which is a dependency of FlatCat.

Basic workflow

This quickstart workflow assumes that your training corpus is in `data/train.txt`, your gold standard segmentation annotations are in `data/annotations.txt`, and your testing wordlist is in `data/test.txt`. All of these

files are encoded with the encoding of your shell locale. We also assume that you wish to use Morfessor Baseline as initialization method. If you have different needs, see command-line-tools for details.

We will be using the values 100 for the perplexity threshold, 1.0 for the unannotated corpus likelihood weight (alpha), and 0.1 for the annotated corpus likelihood weight (beta). You will need to adjust these values for your data set.

Perform the Morfessor Baseline segmentation:

```
morfessor-train data/train.txt -S baseline.gz -w 1.0 -A data/annotations.txt -W 0.1
```

Perform Morfessor FlatCat training:

```
flatcat-train baseline.gz -p 100 -w 1.0 -A data/annotations.txt -W 0.1 -s analysis.
↪tar.gz
```

Segment the test data with the trained Morfessor FlatCat model:

```
flatcat-segment analysis.tar.gz data/test.txt -o test_corpus.segmented --remove-
↪nonmorphemes --output-categories
```

CHAPTER 4

Morfessor file types

Morfessor FlatCat 1.0 style text model

The recommended format for long-term storage of Morfessor FlatCat 1.0 models is as a compressed plain-text analysis of the corpus (model.segmentation.gz) and a separate hyper-parameter file (parameters.txt). In semi-supervised training the annotated corpus is not included in these files, so the annotation file must be stored alongside the analyzed corpus. The plain-text format ensures that the model is human-readable for inspection, and that later versions of Morfessor FlatCat are able to load the model.

Model file

Specification:

```
<int><space><CONSTRUCTION>/<CATEGORY>[<space>+<space><CONSTRUCTION>/<CATEGORY>]*
```

Example:

```
10 kahvi/STM + kakku/STM
5 kahvi/STM + kilo/STM + n/SUF
24 kahvi/STM + ko/ZZZ + ne/ZZZ + emme/SUF
```

Parameter file

Specification:

```
<KEY><colon><tab><VALUE>
```

Example:

```
corpusweight:   1.0
min-perplexity-length:  4
perplexity-threshold:   10.0
perplexity-slope:       1.0
length-threshold:       3.0
type-perplexity:        False
length-slope:   2.0
```

Binary model

Warning: Pickled models are sensitive to bitrot. Sometimes incompatibilities exist between Python versions that prevent loading a model stored by a different version. Also, next versions of Morfessor are not guaranteed to be able to load models of older versions.

For short-term development use a binary model might be more convenient. The binary model is generated by pickling the *FlatcatModel* object. This ensures that all training-data, annotation-data, weights and other hyper-parameters are exactly the same as when the model was saved.

Morfessor 1.0 style text model

Morfessor FlatCat is initialized using Morfessor 1.0 style text models. These files consists of one segmentation per line, preceded by a count, where the constructions are separated by ' + '. In short, these are identical to a FlatCat text model without the category tags.

Specification:

```
<int><space><CONSTRUCTION>[<space>+<space><CONSTRUCTION>]*
```

Example:

```
10 kahvi + kakku
5 kahvi + kilo + n
24 kahvi + kone + emme
```

You can load files with slightly different format by using the format arguments `--compound-separator` and `--construction-separator` (The arguments `--analysis-separator` and `--category-separator` are useful for annotations and category-tagged analyses respectively). If you are loading a file using the wrong construction separator, you may see the error message:

```
#################### WARNING ####################
The input does not seem to be segmented.
Are you using the correct construction separator?
```

Text corpus file

A text corpus file is a free format text-file. All lines are split into compounds using the compound-separator (default <space>). The compounds then are split into atoms using the atom-separator. Compounds can occur multiple times and will be counted as such.

Example:

```
kahvikakku kahvikilon kahvikilon
kahvikoneemme kahvikakku
```

Word list file

A word list corpus file contains one compound per line, possibly preceded by a count. If multiple entries of the same word occur there counts are summed. If no count is given, a count of one is assumed (per entry).

Specification:

```
[<int><space>]<COMPOUND>
```

Example 1:

```
10 kahvikakku
5 kahvikilon
24 kahvikoneemme
```

Example 2:

```
kahvikakku
kahvikilon
kahvikoneemme
```

Annotation file

An annotation file contains one compound and one or more annotations per compound on each line. The separators between the annotations (default ‘, ‘) and between the constructions (default ‘ ‘) are configurable.

Annotations can also be category tagged, by appending a slash ‘/’ and the category to each morph. If category tags are used, all morphs within the file must be tagged.

Specification:

```
<compound> <analysis1construction1>[ <analysis1constructionN>][,
↪<analysis2construction1> [<analysis2constructionN>]*]*

<compound> <analysis1construction1>/<analysis1category1>[ <analysis1constructionN></
↪analysis1categoryN>][, <analysis2construction1>/<analysis2category2> [
↪<analysis2constructionN>/<analysis2categoryN>]*]*
```

Example:

```
kahvikakku kahvi kakku, kahvi kak ku
kahvikilon kahvi kilon
kahvikoneemme kahvi konee mme, kah vi ko nee mme

kahvikakku kahvi/STM kakku/STM, kahvi/STM kak/SUF ku/SUF
```

CHAPTER 5

Command line tools

The installation process installs 5 scripts in the appropriate PATH.

flatcat

The flatcat command is a full-featured script for training, updating models and segmenting test data.

Initializing or loading existing model

The FlatCat model needs to be initialized with an initial segmentation, which can be produced e.g. using Morfessor Baseline. Loading an existing FlatCat model is done by initializing using the saved model. If you don't want to perform training on the loaded model, make sure to use *flatcat-segment* or specify `-m none` to specify that no training needs to be done.

positional argument <init file> Initialize or load a model from the specified file. By default a segmentation file is expected. The segmentation can be compressed using bz2/gzip. If the file name ends in `.pickled`, it is treated as a saved binary FlatCat model.

-L <file>, --load-parameters <file> load hyper-parameters from the specified file. Alternatively the hyper-parameters can be given on the command line. A binary model stores the hyper-parameter values used in training, so you do not need to load or specify them separately.

Extending the model with additional data

--extend <file> Input data to extend the model with. This can be an untagged corpus file, or an already tagged segmentation. Binary FlatCat models can not be used to extend a model. Data can be read from standard input by specifying '-' as the file name. You can use –extend several times in order to append multiple files).

-A <file>, --annotations <file> Add annotated data from the specified file.

-T <file>, --testdata <file> Input corpus file(s) to analyze (text or bz2/gzipped text; use '-' for standard input; add several times in order to append multiple files). The file is read in the same manner as an input corpus file. See *flatcat-reformat* for changing the delimiters used for separating compounds and atoms.

Training model options

-m <mode>, --mode <mode> Morfessor FlatCat can run in different modes, each doing different actions on the model. The modes are:

none Do initialize or train a model. Can be used when just loading a model for segmenting new data

batch Loads an existing model (which is already initialized with training data) and run Can be used to retrain an already trained model, after extending it with more data.

online Initialize a model, and then extend it in an on-line fashion.

online+batch Initialization and online training followed by batch training.

--max-epochs <int> Hard maximum of epochs in training

--online-epochint <int> Epoch interval for online training (default 10000)

-f <list>, --forcesplit <list> A list of atoms that would always cause the compound to be split. By default only hyphens (-) would force a split. Note the notation of the argument list. To have no force split characters, use as an empty string as argument (`-f ""`). To split, for example, both hyphen (-) and apostrophe (') use `-f "-'"`

--nosplit-re <regexp> If the expression matches the two surrounding characters, do not allow splitting (default None)

-p <float>, --perplexity-threshold <float> Threshold value for sigmoid used to calculate class conditional probabilities for prefixes and suffixes, from left and right perplexities. There is no default value, to make forgotten hyperparameters more explicit, so you must specify a value either on the command line or in a hyper-parameter file. If you do not know what value to use, try something between 10 (small corpus / low morphological complexity) and 400 (large corpus, high complexity).

--skips Use random skips for frequently seen compounds to speed up oneline training. Has no effect on batch training.

-d <type>, --dampening <type> Method for changing the compound counts in the input data. Note, that if the counts in the intialization file are already dampened (e.g. by Morfessor Baseline), you should not specify dampening again. Options:

none Do not alter the counts of compounds (token based training)

log Change the count x of a compound to $\log(x)$ (log-token based training)

ones Treat all compounds as if they only occured once (type based training)

--batch-minfreq <int> Compound frequency threshold for batch training (default 1)

Saving model

-s <file> save the model as an analysis (can be compressed by specifying a file ending .gz or .bz2). Use together with `-S <file>`.

-S <file>, --save-parameters <file> save hyper-parameters into file.

--save-binary-model save *Binary model*. Not recommended for long-term storage of models, due to bit-rot.

-x <file>, --lexicon <file> save the morph lexicon

Examples

Initialize a model from the Morfessor Baseline segmentation baseline_segmentation.txt, batch train the model using a perplexity threshold of 10, save the model as an analysis file analysis.gz and a hyper-parameter file parameters.txt, and segment the test.txt set:

```
flatcat baseline_segmentation.txt -p 10 -s analysis.gz -S parameters.txt -T test.txt -
↪-remove-nonmorphemes -o test.segmentation
```

flatcat-train

The flatcat-train command is a convenience command that enables easier training for Morfessor FlatCat models.

The basic command structure is:

```
flatcat-train [arguments] initialization-file
```

The arguments are identical to the ones for the *flatcat* command. The most relevant are:

-s <file> save the model as an analysis (can be compressed by specifying a file ending .gz or .bz2). Use together with `-S <file>`.

-S <file>, --save-parameters <file> save hyper-parameters into file.

Examples

Train a Morfessor FlatCat model from a Morfessor Baseline segmentation in ISO_8859-15 encoding, writing the log to logfile, and saving the model as a binary file model.pickled:

```
flatcat-train baseline_segmentation.txt --encoding=ISO_8859-15 --perplexity-threshold␣
↪10 --logfile=log.log --save-binary-model model.pickled
```

flatcat-segment

The flatcat-segment command is a convenience command that enables easier segmentation of test data with a Morfessor FlatCat model.

The basic command structure is:

```
flatcat-segment [arguments] model-file test-data [test-data ...]
```

The arguments are identical to the ones for the *flatcat* command. The most relevant are:

-L <file> Load hyper-parameters from file. Not necessary if the model is saved in binary format.

-o <file> Output the segmentation of the test data into this file.

--remove-nonmorphemes Apply heuristics for non-morpheme removal to the segmentation output, to ensure that no morphemes categorized as non-morphemes (ZZZ) remain.

--output-categories Include the categories in the segmentation output. Default is to only output the surface form of the morphs.

Examples

Loading a model from analysis.gz, hyper-parameters from parameters.txt and segmenting the file test_corpus.txt:

```
flatcat-segment analysis.gz -L parameters.txt --remove-nonmorphemes -o test_corpus.
↪segmented test_corpus.txt
```

Include the categories in the output:

```
flatcat-segment analysis.gz -L parameters.txt --output-categories -o test_corpus.
↪segmented test_corpus.txt
```

Use FlatCat as a stemmer by removing prefixes and suffixes:

```
flatcat-segment analysis.gz -L parameters.txt --filter-categories PRE,SUF -o test_
↪corpus.segmented test_corpus.txt
```

flatcat-diagnostics

The flatcat-diagnostics command is used to plot the diagnostic statistics collected by giving the parameters `--statsfile <file>` and `--stats-annotations <file>` to *flatcat* or *flatcat-train*.

Examples

Collect statistics during training, using development set devset.segmentation:

```
flatcat-train baseline_segmentation.txt --perplexity-threshold 100 --save-binary-
↪model model.pickled --statsfile stats.pickled --stats-annotations devset.
↪segmentation
```

Plot the statistics:

```
flatcat-diagnostics stats.pickled
```

flatcat-reformat

A reformatting tool which makes format conversion operations on category tagged data a bit more convenient. Work in progress.

Data format command line options

`--encoding <encoding>` Encoding of input and output files (if none is given, both the local encoding and UTF-8 are tried).

`--compound-separator <regexp>` compound separator regexp (default 's+')

`--analysis-separator <str>` separator for different analyses in an annotation file. Use NONE for only allowing one analysis per line.

--output-format <format> format string for –output file (default: '{analysis}\n'). Valid keywords are: `{analysis}` = constructions of the compound, `{compound}` = compound string, {count} = count of the compound (currently always 1), `{logprob}` = log-probability of the analysis, Valid escape sequences are `\n` (newline) and `\t` (tabular).

--output-format-separator <str> construction separator for analysis in `--output` file (default: ' ').

--output-newlines for each newline in input, print newline in `--output` file (default: 'False').

Universal command line options

--verbose <int> -v verbose level; controls what is written to the standard error stream or log file (default 1)

--logfile <file> write log messages to file in addition to standard error stream

--progressbar Force the progressbar to be displayed (possibly lowers the log level for the standard error stream)

--help -h show this help message and exit

--version show version number and exit

CHAPTER 6

Python library interface to Morfessor FlatCat

Morfessor FlatCat 1.0 contains a library interface in order to be integrated in other python applications. The public members are documented below and should remain relatively the same between Morfessor FlatCat versions. Private members are documented in the code and can change anytime in releases.

The classes are documented below.

IO class

class flatcat.io.**FlatcatIO**(*encoding=None*, *construction_separator=u' + '*, *comment_start=u'#'*, *compound_separator=u'\s+'*, *analysis_separator=u','*, *category_separator=u'/'*, *strict=True*)

Definition for all input and output files. Also handles all encoding issues.

The only state this class has is the separators used in the data. Therefore, the same class instance can be used for initializing multiple files.

Extends Morfessor Baseline data file formats to include category tags.

read_annotations_file(*file_name*, *construction_sep=u' '*, *analysis_sep=None*)

Read an annotations file.

Each line has the format: <compound> <constr1> <constr2>... <constrN>, <constr1>...<constrN>, ...

Returns a defaultdict mapping a compound to a list of analyses.

read_any_model(*file_name*)

Read a complete model in either binary or tarball format. This method can NOT be used to initialize from a Morfessor 1.0 style segmentation

read_combined_file(*file_name*, *annotation_prefix=u'<'*, *construction_sep=u' '*, *analysis_sep=u','*)

Reads a file that combines unannotated word tokens and annotated data. The formats are the same as for files containing only one of the mentioned types of data, except that lines with annotations are additionally prefixed with a special symbol.

read_corpus_file(*file_name*)
Read one corpus file.

For each compound, yield (1, compound, compound_atoms). After each line, yield (0, "n", ()).

read_corpus_files(*file_names*)
Read one or more corpus files.

Yield for each compound found (1, compound, compound_atoms).

read_corpus_list_file(*file_name*)
Read a corpus list file.

Each line has the format: <count> <compound>

Yield tuples (count, compound, compound_atoms) for each compound.

read_corpus_list_files(*file_names*)
Read one or more corpus list files.

Yield for each compound found (count, compound, compound_atoms).

read_parameter_file(*file_name*)
Read learned or estimated parameters from a file

read_segmentation_file(*file_name*)
Read segmentation file. see docstring for write_segmentation_file for file format.

read_tarball_model_file(*file_name*, *model=None*)
Read model from a tarball.

write_formatted_file(*file_name*, *line_format*, *data*, *data_func*, *newline_func=None*, *output_newlines=False*, *output_tags=False*, *construction_sep=None*, *analysis_sep=None*, *category_sep=None*, *filter_tags=None*, *filter_len=3*)
Writes a file in the specified format.

Formatting is flexible: even formats that cannot be read by FlatCat can be specified.

write_lexicon_file(*file_name*, *lexicon*)
Write to a Lexicon file all constructions and their emission counts.

write_segmentation_file(*file_name*, *segmentations*, *construction_sep=None*, *output_tags=True*, *comment_string=u''*)
Write segmentation file.

File format (single line, wrapped only for pep8): <count> <construction1><cat_sep><category1><cons_sep>...

<constructionN><cat_sep><categoryN>

class flatcat.io.**TarGzMember**(*arcname*, *tarmodel*)
File-like object that writes itself into the tarfile on closing

class flatcat.io.**TarGzModel**(*filename*, *mode*)
A wrapper to hide the ugliness of the tarfile API.

Both TarGzModel itself and the method newmember are context managers: Writing a model requires a nested with statement.

members()
Generates the (name, contents) pairs for each file in the archive.

The contents are in the form of file-like objects. The files are generated in the order they are in the archive: the recipient must be able to handle them in an arbitrary order.

newmember(*arcname*)
Receive a new member to the .tar.gz archive.

Parameters **– the name of the file within the archive.** (*arcname*) –

Returns a file-like object into which the contents can be written. This is a context manager: use a "with" statement.

Model classes

Morfessor 2.0 FlatCat variant.

class flatcat.flatcat.**FlatcatModel**(*morph_usage=None*, *forcesplit=None*, *nosplit=None*, *corpusweight=1.0*, *use_skips=False*, *ml_emissions_epoch=-1*)
Morfessor FlatCat model class.

Parameters

- **morph_usage** – A MorphUsageProperties object describing how the usage of a morph affects the category.
- **forcesplit** – Force segmentations around the characters in the given list. The same value should be used in Morfessor Baseline or other initialization, to guarantee results.
- **nosplit** – Prevent splitting between character pairs matching this regular expression. The same value should be used in Morfessor Baseline or other initialization, to guarantee results.
- **corpusweight** – Multiplicative weight for the (unannotated) corpus cost.
- **use_skips** – Randomly skip frequently occurring constructions to speed up online training. Has no effect on batch training.
- **ml_emissions_epoch** – The number of epochs of resegmentation using Maximum Likelihood estimation for emission probabilities, instead of using the morph property based probability. These are performed after the normal training. Default -1 means do not switch over to ML estimation.

add_annotations(*annotations*, *annotatedcorpusweight=None*)
Adds data to the annotated corpus.

add_corpus_data(*segmentations*, *freqthreshold=1*, *count_modifier=None*)
Adds the given segmentations (with counts) to the corpus data. The new data can be either untagged or tagged.

If the added data is untagged, you must call viterbi_tag_corpus to tag the new data.

You should also call reestimate_probabilities and consider calling initialize_hmm.

Parameters

- **segmentations** – Segmentations of format: (count, (morph1, morph2, ...)) where the morphs can be either strings or CategorizedMorphs.
- **freqthreshold** – discard words that occur less than given times in the corpus (default 1).
- **count_modifier** – function for adjusting the counts of each word.

cost_breakdown(*segmentation*, *penalty=0.0*, *index=0*)
Returns breakdown of costs for the given tagged segmentation.

cost_comparison(*segmentations*, *retag=True*)
Diagnostic function. (Re)tag the given segmentations, calculate their cost and return the sorted breakdowns of the costs. Can be used to analyse reasons for a segmentation choice.

generate_focus_samples(*num_sets*, *num_samples*)
Generates subsets of the corpus by weighted sampling.

get_cost()
Return current model encoding cost.

get_lexicon()
Returns morphs in lexicon, with emission counts

get_params()
Returns a dict of hyperparameters.

initialize_baseline(*min_difference_proportion=0.005*)
Initialize emission and transition probabilities without changing the segmentation, using Viterbi EM, from a previously added (see add_corpus_data) segmentation produced by a morfessor baseline model.

initialize_hmm(*min_difference_proportion=0.005*)
Initialize emission and transition probabilities without changing the segmentation.

num_compounds
Compound (word) types

num_constructions
Construction (morph) types

rank_analyses(*choices*)
Choose the best analysis of a set of choices.

Observe that the call and return signatures are different from baseline: this method is more versatile.

Parameters **choices** – a sequence of AnalysisAlternative(analysis, penalty) namedtuples. The analysis must be a sequence of CategorizedMorphs, (segmented and tagged). The penalty is a float that is added to the cost for this choice. Use 0 to disable.

Returns

A sorted (by cost, ascending) list of SortedAnalysis(cost, analysis, index, breakdown) namedtuples.

```
cost :  the contribution of this analysis to the corpus cost.
analysis :  as in input.
breakdown :  A CostBreakdown object, for diagnostics
```

reestimate_probabilities()
Re-estimates model parameters from a segmented, tagged corpus.

theta(t) = arg min { L(theta, Y(t), D) }

set_focus_sample(*set_index*)
Select one pregenerated focus sample set as active.

set_params(*params*)
Sets hyperparameters to loaded values.

toggle_callbacks(*callbacks=None*)
Callbacks are not saved in the pickled model, because pickle is unable to restore instance methods. If you need callbacks in a loaded model, you have to readd them after loading.

train_batch(*min_iteration_cost_gain=0.0025*, *min_epoch_cost_gain=0.005*, *max_epochs=5*, *max_iterations_first=1*, *max_iterations=1*, *max_resegment_iterations=1*, *max_shift_distance=2*, *min_shift_remainder=2*)
Perform batch training.

Parameters

- **min_iteration_cost_gain** – Do not repeat iteration if the gain in cost was less than this proportion. No effect if max_iterations is 1. Set to None to disable.
- **min_epoch_cost_gain** – Stop before max_epochs, if the gain in cost of the previous epoch was less than this proportion. Set to None to disable.
- **max_epochs** – Maximum number of training epochs.
- **max_iterations_first** – Number of iterations of each operator, in the first epoch.
- **max_iterations** – Number of iterations of each operator, in later epochs.
- **max_resegment_iterations** – Number of resegment iterations in any epoch.
- **max_shift_distance** – Limit on the distance (in characters) that the shift operation can move a boundary.
- **min_shift_remainder** – Limit on the shortest morph allowed to be produced by the shift operation.

train_online(*data*, *count_modifier=None*, *epoch_interval=10000*, *max_epochs=None*, *result_callback=None*)
Adapt the model in online fashion.

violated_annotations()
Yields all segmentations which have an associated annotation, but woud currently not be naturally segmented in a way that is included in the annotation alternatives,

viterbi_analyze_list(*corpus*)
Convenience wrapper around viterbi_analyze for a list of word strings or segmentations with attached counts. Segmented input can be with or without tags. This function can be used to analyze previously unseen data.

viterbi_tag_corpus()
(Re)tags the corpus segmentations using viterbi_tag

words_with_morph(*morph*)
Diagnostic function. Returns all segmentations using the given morph. Format: (index_to_segmentations, count, analysis)

class flatcat.flatcat.**FlatcatLexiconEncoding**(*morph_usage*)
Extends LexiconEncoding to include the coding costs of the encoding cost of morph usage (context) features.

Parameters **morph_usage** – A MorphUsageProperties object, or something that quacks like it.

clear()
Resets the cost variables. Use before fully reprocessing a segmented corpus.

class flatcat.flatcat.**FlatcatEncoding**(*morph_usage*, *lexicon_encoding*, *weight=1.0*)
Class for calculating the encoding costs of the grammar and the corpus. Also stores the HMM parameters.

tokens: the number of emissions observed. boundaries: the number of word tokens observed.

clear_emission_cache()
Clears the cache for emission probability values. Use if an incremental change invalidates cached values.

clear_emission_counts()
Resets emission counts and costs. Use before fully reprocessing a tagged segmented corpus.

clear_transition_cache()
Clears the cache for emission probability values. Use if an incremental change invalidates cached values.

clear_transition_counts()
Resets transition counts, costs and cache. Use before fully reprocessing a tagged segmented corpus.

get_cost()
Override for the Encoding get_cost function.

This is P(D_W | theta, Y)

log_emissionprob(*category*, *morph*, *extrazero=False*)
-Log of posterior emission probability P(morph|category)

log_transitionprob(*prev_cat*, *next_cat*)
-Log of transition probability P(next_cat|prev_cat)

logtransitionsum()
Returns the term of the cost function associated with the transition probabilities. This term is recalculated on each call to get_cost, as the transition matrix is small and each segmentation change is likely to modify a large part of the transition matrix, making cumulative updates unnecessary.

transit_emit_cost(*prev_cat*, *next_cat*, *morph*)
Cost of transitioning from prev_cat to next_cat and emitting the morph.

update_emission_count(*category*, *morph*, *diff_count*)
Updates the number of observed emissions of a single morph from a single category, and the logtokensum (which is category independent). Updates logcondprobsum.

Parameters

- **category** – name of category from which emission occurs.
- **morph** – string representation of the morph.
- **diff_count** – the change in the number of occurences.

update_transition_count(*prev_cat*, *next_cat*, *diff_count*)
Updates the number of observed transitions between categories. OBSERVE! Clearing the cache is left to the caller.

Parameters

- **prev_cat** – The name (not index) of the category transitioned from.
- **next_cat** – The name (not index) of the category transitioned to.
- **diff_count** – The change in the number of transitions.

class flatcat.flatcat.**FlatcatAnnotatedCorpusEncoding**(*corpus_coding*, *weight=None*)
Class for calculating the cost of encoding the annotated corpus

get_cost()
Returns the cost of encoding the annotated corpus

modify_contribution(*morph*, *direction*)
Removes or readds the complete contribution of a morph to the cost function. The contribution must be removed using the same probability value as was used when adding it, making ordering of operations important.

reset_contributions()
Recalculates the contributions of all morphs.

set_counts(*counts*)
Sets the counts of emissions and transitions occurring in the annotated corpus to precalculated values.

transition_cost()
Returns the term of the cost function associated with the transition probabilities. This term is recalculated on each call to get_cost, as the transition matrix is small and each segmentation change is likely to modify a large part of the transition matrix, making cumulative updates unnecessary.

update_counts(*counts*)
Updates the counts of emissions and transitions occurring in the annotated corpus, building on earlier counts.

update_weight()
Update the weight of the Encoding by taking the ratio of the corpus boundaries and annotated boundaries. Does not scale by corpus weight,, unlike Morfessor Baseline.

CHAPTER 7

Code Examples for using library interface

Initialize a semi-supervised model from a given segmentation and annotations

```
import flatcat

io = flatcat.FlatcatIO()

morph_usage = flatcat.categorizationscheme.MorphUsageProperties()

model = flatcat.FlatcatModel(morph_usage, corpusweight=1.0)

model.add_corpus_data(io.read_segmentation_file('segmentation.txt'))

model.add_annotations(io.read_annotations_file('annotations.txt'),
                      annotatedcorpusweight=1.0)

model.initialize_hmm()
```

The model is now ready to be trained.

Segmenting new data using an existing model

First printing only the segmentations, followed by the analysis with morph categories.

```
import flatcat

io = flatcat.FlatcatIO()

model = io.read_binary_model_file('model.pickled')

words = ['words', 'segmenting', 'morfessor', 'categories', 'semisupervised']
```

```
for word in words:
    print(model.viterbi_segment(word))

for word in words:
    print(model.viterbi_analyze(word))
```

CHAPTER 8

Frequently asked questions

UnicodeError: Can not determine encoding of input files

If the encoding of the input data is not detected automatically, you need to specify it on the command line, e.g.

```
--encoding=ISO_8859-15
```

The word counts in the output are smaller than expected

If the counts in the input data are already dampened, you should not specify a dampening to FlatCat.

```
--dampening none
```

The input does not seem to be segmented

If you receive the following warning when loading a segmentation:

```
#################### WARNING ####################
The input does not seem to be segmented.
Are you using the correct construction separator?
```

the reason might be that the data is in an unexpected format. By default Morfessor FlatCat assumes that morphs are separated by a plus sign surrounded on both sides by a space ' + '. If your data uses e.g. only a space to separate morphs, you need to specify:

```
--construction-separator ' '
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[Gronroos2014a] Grönroos, S.-A. and Virpioja, S. (2014). Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. In Proceedings of the 25th International Conference on Computational Linguistics. Pages 1177–1185, Dublin, Ireland, August 2014, Association for Computational Linguistics.

[Gronroos2014b] Grönroos, S.-A. (2014). Semi-supervised induction of a concatenative morphology with simple morphotactics: A model in the Morfessor family. Master's thesis, Aalto University, Helsinki.

[TechRep] Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline. Aalto University publication series SCIENCE + TECHNOLOGY, 25/2013. Aalto University, Helsinki, 2013. ISBN 978-952-60-5501-5.

[Creutz2002] Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In Proceedings of the Workshop on Morphological and Phonological Learning of ACL-02, pages 21-30, Philadelphia, Pennsylvania, 11 July, 2002.

[Kohonen2010] Oskar Kohonen, Sami Virpioja and Krista Lagus. Semi-supervised learning of concatenative morphology. In Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, pages 78-86, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Python Module Index

f

Index

S

T

U

V

W