python-crfsuite Documentation

Release 0.9

Terry Peng, Mikhail Korobov

December 19, 2016

Contents

python-crfsuite is a python binding to CRFsuite.

CHAPTER 1

Installation

```
pip install python-crfsuite
```

CHAPTER 2

Usage

- *API Reference*
- Example: building a Named Entity Recognition system with python-crfsuite.

python-crfsuite is licensed under MIT license. CRFsuite C/C++ library is licensed under BSD license.

Development happens at github: https://github.com/scrapinghub/python-crfsuite

2.1 API Reference

class `pycrfsuite.`**`ItemSequence`**

A wrapper for crfsuite ItemSequence - a class for storing features for all items in a single sequence.

Using this class is an alternative to passing data to *`Trainer`* and *`Tagger`* directly. By using this class it is possible to save some time if the same input sequence is passed to trainers/taggers more than once - features won't be processed multiple times. It also allows to get "processed" features/attributes that are sent to CRFsuite - they could be helpful e.g. to check which attributes (returned by *`info()`*) are active for a given observation.

Initialize ItemSequence with a list of item features:

```
>>> ItemSequence([{'foo': 1, 'bar': 0}, {'foo': 1.5, 'baz': 2}])
<ItemSequence of size 2>
```

Item features could be in one of the following formats:

- {"string_key": float_weight, ...} dict where keys are observed features and values are their weights;
- {"string_key": bool, ...} dict; True is converted to 1.0 weight, False - to 0.0;
- {"string_key": "string_value", ...} dict; that's the same as {"string_key=string_value": 1.0, ...}
- ["string_key1", "string_key2", ...] list; that's the same as {"string_key1": 1.0, "string_key2": 1.0, ...}
- {"string_prefix": {...}} dicts: nested dict is processed and "string_prefix" s prepended to each key.
- {"string_prefix": [...]} dicts: nested list is processed and "string_prefix" s prepended to each key.
- {"string_prefix": set([...])} dicts: nested list is processed and "string_prefix" s prepended to each key.

Dict-based features can be mixed, i.e. this is allowed:

```
{"key1": float_weight,
 "key2": "string_value",
 "key3": bool_value,
```

```
    "key4: {"key5": ["x", "y"], "key6": float_value},
    }
```

items (*self*)
Return a list of prepared item features: a list of `{unicode_key: float_value}` dicts.

```
    >>> ItemSequence([["foo"], {"bar": {"baz": 1}}]).items()
    [{'foo': 1.0}, {'bar:baz': 1.0}]
```

2.1.1 Training

class pycrfsuite.**Trainer**
Bases: pycrfsuite._pycrfsuite.BaseTrainer

The trainer class.

This class maintains a data set for training, and provides an interface to various training algorithms.

Parameters **algorithm** : {'lbfgs', 'l2sgd', 'ap', 'pa', 'arow'}

The name of the training algorithm. See *Trainer.select()*.

params : dict, optional

Training parameters. See *Trainer.set_params()* and *Trainer.set()*.

verbose : boolean

Whether to print debug messages during training. Default is True.

append (*self*, *xseq*, *yseq*, *int group=0*)
Append an instance (item/label sequence) to the data set.

Parameters **xseq** : a sequence of item features

The item sequence of the instance. `xseq` should be a list of item features or an *ItemSequence* instance. Allowed item features formats are the same as described in *ItemSequence* docs.

yseq : a sequence of strings

The label sequence of the instance. The number of elements in yseq must be identical to that in xseq.

group : int, optional

The group number of the instance. Group numbers are used to select subset of data for heldout evaluation.

clear (*self*)
Remove all instances in the data set.

get (*self*, *name*)
Get the value of a training parameter. This function gets a parameter value for the graphical model and training algorithm specified by *Trainer.select()* method.

Parameters **name** : string

The parameter name.

get_params (*self*)
Get training parameters.

Returns dict

A dictionary with `{parameter_name: parameter_value}` with all trainer parameters.

help (*self*, *name*)
Get the description of a training parameter. This function obtains the help message for the parameter specified by the name. The graphical model and training algorithm must be selected by *Trainer.select()* method before calling this method.

Parameters **name** : string

The parameter name.

Returns string

The description (help message) of the parameter.

logparser = None

message (*self*, *message*)

on_end (*self*, *log*)

on_featgen_end (*self*, *log*)

on_featgen_progress (*self*, *log*, *percent*)

on_iteration (*self*, *log*, *info*)

on_optimization_end (*self*, *log*)

on_prepare_error (*self*, *log*)

on_prepared (*self*, *log*)

on_start (*self*, *log*)

params (*self*)
Obtain the list of parameters.

This function returns the list of parameter names available for the graphical model and training algorithm specified in Trainer constructor or by *Trainer.select()* method.

Returns list of strings

The list of parameters available for the current graphical model and training algorithm.

select (*self*, *algorithm*, *type='crf1d'*)
Initialize the training algorithm.

Parameters **algorithm** : {'lbfgs', 'l2sgd', 'ap', 'pa', 'arow'}

The name of the training algorithm.

- 'lbfgs' for Gradient descent using the L-BFGS method,
- 'l2sgd' for Stochastic Gradient Descent with L2 regularization term
- 'ap' for Averaged Perceptron
- 'pa' for Passive Aggressive
- 'arow' for Adaptive Regularization Of Weight Vector

type : string, optional

The name of the graphical model.

set (*self*, *name*, *value*)
Set a training parameter. This function sets a parameter value for the graphical model and training algorithm specified by *Trainer.select()* method.

Parameters **name** : string

The parameter name.

value : string

The value of the parameter.

set_params (*self*, *params*)
Set training parameters.

Parameters **params** : dict

A dict with parameters `{name: value}`

train (*self*, *model*, *int holdout=-1*)
Run the training algorithm. This function starts the training algorithm with the data set given by *Trainer.append()* method.

Parameters **model** : string

The filename to which the trained model is stored. If this value is empty, this function does not write out a model file.

holdout : int, optional

The group number of holdout evaluation. The instances with this group number will not be used for training, but for holdout evaluation. Default value is -1, meaning "use all instances for training".

verbose
verbose: object

2.1.2 Tagging

class pycrfsuite.**Tagger**
The tagger class.

This class provides the functionality for predicting label sequences for input sequences using a model.

close (*self*)
Close the model.

dump (*self*, *filename=None*)
Dump a CRF model in plain-text format.

Parameters **filename** : string, optional

File name to dump the model to. If None, the model is dumped to stdout.

info (*self*)
Return a *ParsedDump* structure with model internal information.

labels (*self*)
Obtain the list of labels.

Returns list of strings

The list of labels in the model.

marginal (*self*, *y*, *pos*)
Compute the marginal probability of the label `y` at position `pos` for the current input sequence (i.e. a sequence set using *Tagger.set()* method or a sequence used in a previous *Tagger.tag()* call).

Parameters **y** : string

The label.

t : int

The position of the label.

Returns float

The marginal probability of the label `y` at position `t`.

open (*self*, *name*)
Open a model file.

Parameters **name** : string

The file name of the model file.

probability (*self*, *yseq*)
Compute the probability of the label sequence for the current input sequence (a sequence set using *Tagger.set()* method or a sequence used in a previous *Tagger.tag()* call).

Parameters **yseq** : list of strings

The label sequence.

Returns float

The probability `P(yseq|xseq)`.

set (*self*, *xseq*)
Set an instance (item sequence) for future calls of *Tagger.tag()*, *Tagger.probability()* and *Tagger.marginal()* methods.

Parameters **xseq** : item sequence

The item sequence of the instance. `xseq` should be a list of item features or an *ItemSequence* instance. Allowed item features formats are the same as described in *ItemSequence* docs.

tag (*self*, *xseq=None*)
Predict the label sequence for the item sequence.

Parameters **xseq** : item sequence, optional

The item sequence. If omitted, the current sequence is used (a sequence set using *Tagger.set()* method or a sequence used in a previous *Tagger.tag()* call).

`xseq` should be a list of item features or an *ItemSequence* instance. Allowed item features formats are the same as described in *ItemSequence* docs.

Returns list of strings

The label sequence predicted.

2.1.3 Debugging

class `pycrfsuite._dumpparser.`**ParsedDump**
CRFsuite model parameters. Objects of this type are returned by *pycrfsuite.Tagger.info()* method.

Attributes

transitions	(dict) `{(from_label, to_label): weight}` dict with learned transition weights
state_features	(dict) `{(attribute, label): weight}` dict with learned `(attribute, label)` weights
header	(dict) Metadata from the file header
labels	(dict) `{name: internal_id}` dict with model labels
attributes	(dict) `{name: internal_id}` dict with known attributes

CHAPTER 3

See Also

sklearn-crfsuite is a python-crfsuite wrapper which provides API similar to scikit-learn.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Changes

5.1 0.9.1 (2016-12-19)

This is a release without changes in functionality.

- Repository is moved to https://github.com/scrapinghub/python-crfsuite;
- We're now providing Windows wheels for Python 2.7, 3.3. and 3.4.

5.2 0.9 (2016-12-08)

- Python 2.6 support is dropped;
- CRFSuite C++ library is updated to a more recent commit;
- improved Windows support (thanks @fgregg);
- fixed building with gcc < 5.0.0 (thanks @kantan2015);
- extension is rebuilt with Cython 0.25.1; this improves PyPy compatibility (but we're not quite there yet).
- docs: trainer.logparser example is added to the notebook (thanks @samgalen).

5.3 0.8.4 (2015-11-25)

- the wrapper is rebuilt with Cython 0.23.4;
- declared Python 3.5 compatibility;
- fixed an issue with feature names ending with white spaces.

5.4 0.8.3 (2015-04-24)

- fix build on Windows. (thanks @fgregg)

5.5 0.8.2 (2015-02-04)

- memory leak is fixed by updating the bundled CRFsuite C++ library;
- the wrapper is rebuilt with Cython 0.21.2.

5.6 0.8.1 (2014-10-10)

- fix packaging issues with 0.8 release.

5.7 0.8 (2014-10-10)

- `ItemSequence` wrapper is added;
- tox tests are fixed.

5.8 0.7 (2014-08-11)

- More data formats for `xseq`: `{"prefix": {feature_dict}}` and `{"key": set(["key1",...])}` feature dicts are now accepted by *pycrfsuite.Trainer* and *pycrfsuite.Tagger*;
- feature separator changed from "=" to ":" (it looks better in case of multi-level features);
- small doc and README fixes.

5.9 0.6.1 (2014-06-06)

- Switch to setuptools;
- wheels are uploaded to pypi for faster installation.

5.10 0.6 (2014-05-29)

- More data formats for `xseq`: `{"key": "value"}` and `{"key": bool_value}` feature dicts are now accepted by *pycrfsuite.Trainer* and *pycrfsuite.Tagger*.

5.11 0.5 (2014-05-27)

- Exceptions in logging message handlers are now propogated and raised. This allows, for example, to stop training earlier by pressing Ctrl-C.
- It is now possible to customize *pycrfsuite.Trainer* logging more easily by overriding the following methods: *pycrfsuite.Trainer.on_start()*, *pycrfsuite.Trainer.on_featgen_progress()*, *pycrfsuite.Trainer.on_featgen_end()*, *pycrfsuite.Trainer.on_prepared()*, *pycrfsuite.Trainer.on_prepare_error()*,

pycrfsuite.Trainer.on_iteration(), *pycrfsuite.Trainer.on_optimization_end()* *pycrfsuite.Trainer.on_end()*. The feature is implemented by parsing CRFsuite log. There is `pycrfsuite.BaseTrainer` that is not doing this.

5.12 0.4.1 (2014-05-18)

- *pycrfsuite.Tagger.info()* is fixed.

5.13 0.4 (2014-05-16)

- (backwards-incompatible) training parameters are now passed using `params` argument of *pycrfsuite.Trainer* constructor instead of `**kwargs`;
- (backwards-incompatible) logging support is dropped;
- *verbose* argument for *pycrfsuite.Trainer* constructor;
- *pycrfsuite.Trainer.get_params()* and *pycrfsuite.Trainer.set_params()* for getting/setting multiple training parameters at once;
- string handling in Python 3.x is fixed by rebuilding the wrapper with Cython 0.21dev;
- algorithm names are normalized to support names used by crfsuite console utility and documented in crfsuite manual;
- type conversion for training parameters is fixed: `feature.minfreq` now works, and boolean arguments become boolean.

5.14 0.3 (2014-05-14)

python-crfsuite now detects the featue format (dict vs list of strings) automatically - it turns out the performance overhead is negligible.

- `Trainer.append_stringslists` and `Trainer.append_dicts` methods are replaced with a single *pycrfsuite.Trainer.append()* method;
- `Tagger.set_stringlists` and `Tagger.set_dicts` methods are removed in favor of *pycrfsuite.Tagger.set()* method;
- `feature_format` arguments in *pycrfsuite.Tagger* methods and constructor are dropped.

5.15 0.2 (2014-05-14)

- *pycrfsuite.Tagger.dump()* and *pycrfsuite.Tagger.info()* methods for model debugging;
- a memory leak in Trainer is fixed (trainer instances were never garbage collected);
- documentation and testing improvements.

5.16 0.1 (2014-04-30)

Many changes; python-crfsuite is almost rewritten.

5.17 0.0.1 (2014-04-24)

Initial release.

p