CKIPNLP

Release v0.8.0

Mu Yang

Apr 27, 2020

OVERVIEW

CHAPTER

ONE

INTRODUCTION

Official CKIP CoreNLP Toolkits

1.1 Features

- Sentence Segmentation
- Word Segmentation
- Part-of-Speech Tagging
- Sentence Parsing
- Named-Entity Recognition
- Co-Reference Delectation

1.2 Git

https://github.com/ckiplab/ckipnlp

1.3 PyPI

https://pypi.org/project/ckipnlp

1.4 Documentation

https://ckipnlp.readthedocs.io/

1.5 Contributers

- Mu Yang at CKIP (Author & Maintainer)
- Wei-Yun Ma at CKIP (Maintainer)
- DouglasWu

1.6 External Links

- Online Demo

CHAPTER

TWO

INSTALLATION

2.1 Requirements

- Python 3.6+
- TreeLib 1.5+
- CkipTagger 0.1.1+ [Optional, Recommended]
- CkipClassic 1.0+ [Optional]

2.2 Tool Requirements

Tool	Built-in	CkipTagger	CkipClassic
Sentence Segmentation	✓		
Word Segmentation†		✓	✓
Part-of-Speech Tagging†		✓	✓
Sentence Parsing			✓
Named-Entity Recognition		✓	
Co-Reference Delectation‡	✓	✓	✓

- † These tools require only one of either backends.
- ‡ Co-Reference implementation does not require any backend, but requires results from word segmentation, part-of-speech tagging, sentence parsing, and named-entity recognition.

2.3 Installation via Pip

- No backend (not recommended): `pip install ckipnlp`.
- With CkipTagger backend (recommended): `pip install ckipnlp[tagger]`
- With CkipClassic backend: Please refer https://ckip-classic.readthedocs.io/en/latest/src/readme.html#installation for CkipClassic installation guide.

CHAPTER

THREE

USAGE

See https://ckipnlp.readthedocs.io/en/latest/_api/ckipnlp.html for API details.

3.1 Pipelines

3.1.1 Core Pipeline

```
from ckipnlp.pipeline import CkipPipeline, CkipDocument

pipeline = CkipPipeline()
doc = CkipDocument(raw='')

# Word Segmentation
pipeline.get_ws(doc)
print(doc.ws)
for line in doc.ws:
    print(line.to_text())

# Part-of-Speech Tagging
pipeline.get_pos(doc)
print(doc.pos)
for line in doc.pos:
    print(line.to_text())

# Named-Entity Recognition
pipeline.get_ner(doc)
print(doc.ner)

# Sentence Parsing
pipeline.get_parsed(doc)
print(doc.parsed)

################################################################

from ckipnlp.container.util.wspos import WsPosParagraph

# Word Segmentation & Part-of-Speech Tagging
for line in WsPosParagraph.to_text(doc.ws, doc.pos):
    print(line)
```

3.1.2 Co-Reference Pipeline

```
from ckipnlp.pipeline import CkipCorefPipeline, CkipDocument

pipeline = CkipCorefPipeline()
doc = CkipDocument(raw='')

# Co-Reference
corefdoc = pipeline(doc)
print(corefdoc.coref)
for line in corefdoc.coref:
    print(line.to_text())
```

3.2 Containers

The container objects provides following methods:

- `from_text()`, `to_text()` for plain-text format conversions;
- `from_dict()`, `to_dict()` for dictionary-like format conversions;
- `from_list()`, `to_list()` for list-like format conversions;
- `from_json()`, `to_json()` for JSON format conversions (based-on dictionary-like format conversions).

The following are the interfaces, where `CONTAINER_CLASS` refers to the container class.

```
obj = CONTAINER_CLASS.from_text(plain_text)
plain_text = obj.to_text()

obj = CONTAINER_CLASS.from_dict({ key: value })
dict_obj = obj.to_dict()

obj = CONTAINER_CLASS.from_list([ value1, value2 ])
list_obj = obj.to_list()

obj = CONTAINER_CLASS.from_json(json_str)
json_str = obj.to_json()
```

Note that not all container provide all above methods. Here is the table of implemented methods. Please refer the documentation of each container for detail formats.

Container	Item	from/to text	from/to dict, list, json
TextParagraph	str	✓	✓
SegSentence	str	✓	✓
SegParagraph	*SegSentence*	✓	✓
NerToken			✓
NerSentence	*NerToken*		✓
NerParagraph	*NerSentence*		✓
ParsedParagraph	str	✓	✓
CorefToken		only to	✓
CorefSentence	*CorefToken*	only to	✓
CorefParagraph	*CorefSentence*	only to	✓

3.2.1 WS with POS

There are also conversion routines for word-segmentation and POS containers jointly. For example, *WsPosToken* provides routines for a word (`str`) with POS-tag (`str`):

```
ws_obj, pos_obj = WsPosToken.from_text('(Na)')
plain_text = WsPosToken.to_text(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_dict({ 'word': '', 'pos': 'Na', })
dict_obj = WsPosToken.to_dict(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_list([ '', 'Na' ])
list_obj = WsPosToken.to_list(ws_obj, pos_obj)

ws_obj, pos_obj = WsPosToken.from_json(json_str)
json_str = WsPosToken.to_json(ws_obj, pos_obj)
```

Similarly, *WsPosSentence*/*WsPosParagraph* provides routines for word-segmented and POS sentence/paragraph (*SegSentence*/*SegParagraph*) respectively.

3.2.2 Parsed Tree

In addition to *ParsedParagraph*, we have implemented tree utilities base on TreeLib.

ParsedTree is the tree structure of a parsed sentence. One may use `from_text()` and `to_text()` for plain-text conversion; `from_dict()`, `to_dict()` for dictionary-like object conversion; and also `from_json()`, `to_json()` for JSON string conversion.

The *ParsedTree* is a TreeLib tree with *ParsedNode* as its nodes. The data of these nodes is stored in a *ParsedNodeData* (accessed by `node.data`), which is a tuple of `role` (semantic role), `pos` (part-of-speech tagging), `word`.

ParsedTree provides useful methods: `get_heads()` finds the head words of the sentence; `get_relations()` extracts all relations in the sentence; `get_subjects()` returns the subjects of the sentence.

```
from ckipnlp.container import ParsedTree

#
tree_text =
↪'S(goal:NP(possessor:N(head:Nhaa:|Head:DE:)|Head:Nab(DUMMY1:Nab(DUMMY1:Nab:|Head:Caa:|DUMMY2:Naa:)
↪'

tree = ParsedTree.from_text(tree_text, normalize=False)

print('Show Tree')
tree.show()

print('Get Heads of {}'.format(tree[5]))
print('-- Semantic --')
for head in tree.get_heads(5, semantic=True): print(repr(head))
print('-- Syntactic --')
for head in tree.get_heads(5, semantic=False): print(repr(head))
print()

print('Get Relations of {}'.format(tree[0]))
print('-- Semantic --')
for rel in tree.get_relations(0, semantic=True): print(repr(rel))
```

(continues on next page)

(continued from previous page)

```
print('-- Syntactic --')
for rel in tree.get_relations(0, semantic=False): print(repr(rel))
print()

#
tree_text =
↪'S(theme:NP(DUMMY1:NP(Head:Nhaa:)|Head:Caa:|DUMMY2:NP(Head:Naa:))|evaluation:Dbb:|quantity:Dab:|deg
↪'

tree = ParsedTree.from_text(tree_text, normalize=False)

print('Show Tree')
tree.show()

print('Get get_subjects of {}'.format(tree[0]))
print('-- Semantic --')
for subject in tree.get_subjects(0, semantic=True): print(repr(subject))
print('-- Syntactic --')
for subject in tree.get_subjects(0, semantic=False): print(repr(subject))
print()
```

CHAPTER

FOUR

LICENSE

CHAPTER

FIVE

CKIPNLP PACKAGE

The Official CKIP CoreNLP Toolkits.

Subpackages

5.1 ckipnlp.container package

This module implements specialized container datatypes for CKIPNLP.

Subpackages

5.1.1 ckipnlp.container.util package

This module implements specialized utilities for CKIPNLP containers.

Submodules

ckipnlp.container.util.parsed_tree module

This module provides tree containers for sentence parsing.

class ckipnlp.container.util.parsed_tree.**ParsedNodeData**

Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.util.parsed_tree._ParsedNodeData

A parser node.

Variables

- **role** (*str*) – the semantic role.
- **pos** (*str*) – the POS-tag.
- **word** (*str*) – the text term.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for *from_text()* and *to_text()*.

```
'Head:Na:'  # role / POS-tag / text-term
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'role': 'Head',    # role
    'pos': 'Na',       # POS-tag
    'word': '',  # text term
}
```

List format Not implemented.

---

from_list = NotImplemented

to_list = NotImplemented

classmethod from_text(*data*)

Construct an instance from text format.

Parameters **data** (*str*) – text such as `'Head:Na:'`.

Note:

- `'Head:Na:'` -> **role** = `'Head'`, **pos** = `'Na'`, **word** = `''`
- `'Head:Na'` -> **role** = `'Head'`, **pos** = `'Na'`, **word** = `None`
- `'Na'` -> **role** = `None`, **pos** = `'Na'`, **word** = `None`

---

to_text()

class ckipnlp.container.util.parsed_tree.**ParsedNode**(*tag=None*, *identifier=None*, *expanded=True*, *data=None*)

Bases: *ckipnlp.container.base.Base*, `treelib.node.Node`

A parser node for tree.

Variables **data** (*ParsedNodeData*) –

See also:

treelib.tree.Node Please refer https://treelib.readthedocs.io/ for built-in usages.

Data Structure Examples

Text format Not implemented.

Dict format Used for *to_dict()*.

```
{
    'role': 'Head',    # role
    'pos': 'Na',       # POS-tag
    'word': '',  # text term
}
```

List format Not implemented.

---

data_class
alias of *ParsedNodeData*

from_dict = NotImplemented

from_text = NotImplemented

to_text = NotImplemented

from_list = NotImplemented

to_list = NotImplemented

to_dict()

class ckipnlp.container.util.parsed_tree.**ParsedRelation**
Bases: *ckipnlp.container.base.Base*, ckipnlp.container.util.parsed_tree._ParsedRelation

A parser relation.

Variables

- **head** (*ParsedNode*) – the head node.
- **tail** (*ParsedNode*) – the tail node.
- **relation** (*ParsedNode*) – the relation node. (the semantic role of this node is the relation.)

Notes

The parent of the relation node is always the common ancestor of the head node and tail node.

Data Structure Examples

Text format Not implemented.

Dict format Used for *to_dict()*.

```
{
    'tail': { 'role': 'Head', 'pos': 'Nab', 'word': '' }, # head node
    'tail': { 'role': 'particle', 'pos': 'Td', 'word': '' }, # tail node
    'relation': 'particle',  # relation
}
```

List format Not implemented.

---

from_dict = NotImplemented

from_text = NotImplemented

to_text = NotImplemented

from_list = NotImplemented

to_list = NotImplemented

property head_first

to_dict()

class ckipnlp.container.util.parsed_tree.**ParsedTree**(*tree=None*, *deep=False*, *node_class=None*, *identifier=None*)

Bases: *ckipnlp.container.base.Base*, treelib.tree.Tree

A parsed tree.

See also:

treereelib.tree.Tree Please refer https://treelib.readthedocs.io/ for built-in usages.

Data Structure Examples

Text format Used for *from_text()* and *to_text()*.

```
'S(Head:Nab:|particle:Td:)'
```

Dict format Used for *from_dict()* and *to_dict()*. A dictionary such as `{ 'id': 0, 'data': { ... }, 'children': [ ... ] }`, where `'data'` is a dictionary with the same format as `ParsedNodeData.to_dict()`, and `'children'` is a list of dictionaries of subtrees with the same format as this tree.

```
{
    'id': 0,
    'data': {
        'role': None,
        'pos': 'S',
        'word': None,
    },
    'children': [
        {
            'id': 1,
            'data': {
                'role': 'Head',
                'pos': 'Nab',
                'word': '',
            },
            'children': [],
        },
        {
            'id': 2,
            'data': {
                'role': 'particle',
                'pos': 'Td',
                'word': '',
            },
            'children': [],
        },
    ],
}
```

List format Not implemented.

---

node_class
alias of *ParsedNode*

from_list = NotImplemented

to_list = NotImplemented

static normalize_text(*tree_text*)
Text normalization.

Remove leading number and trailing #.

classmethod from_text(*data*, *, *normalize=True*)
Construct an instance from text format.

Parameters

- **data** (*str*) – A parsed tree in text format.
- **normalize** (*bool*) – Do text normalization using *normalize_text()*.

to_text(*node_id=None*)
Transform to plain text.

Parameters **node_id** (*int*) – Output the plain text format for the subtree under **node_id**.

Returns *str*

classmethod from_dict(*data*)
Construct an instance a from python built-in containers.

Parameters **data** (*str*) – A parsed tree in dictionary format.

to_dict(*node_id=None*)
Construct an instance a from python built-in containers.

Parameters **node_id** (*int*) – Output the plain text format for the subtree under **node_id**.

Returns *str*

show(*, *key=<function ParsedTree.<lambda>>*, *idhidden=False*, **kwargs*)
Show pretty tree.

get_children(*node_id*, *, *role*)
Get children of a node with given role.

Parameters

- **node_id** (*int*) – ID of target node.
- **role** (*str*) – the target role.

Yields *ParsedNode* – the children nodes with given role.

get_heads(*root_id=None*, *, *semantic=True*, *deep=True*)
Get all head nodes of a subtree.

Parameters

- **root_id** (*int*) – ID of the root node of target subtree.
- **semantic** (*bool*) – use semantic/syntactic policy. For semantic mode, return `DUMMY` or `head` instead of syntactic `Head`.
- **deep** (*bool*) – find heads recursively.

Yields *ParsedNode* – the head nodes.

get_relations(*root_id=None*, *, *semantic=True*)
Get all relations of a subtree.

Parameters

- **root_id** (*int*) – ID of the subtree root node.
- **semantic** (*bool*) – please refer *get_heads()* for policy detail.

Yields *ParsedRelation* – the relations.

get_subjects(*root_id=None*, *, *semantic=True*, *deep=True*)
Get the subject node of a subtree.

Parameters

- **root_id** (*int*) – ID of the root node of target subtree.
- **semantic** (*bool*) – please refer *get_heads()* for policy detail.
- **deep** (*bool*) – please refer *get_heads()* for policy detail.

Yields *ParsedNode* – the subject node.

Notes

A node can be a subject if either:

1. is a head of *NP*
2. is a head of a subnode of *S* with subject role
3. is a head of a subnode of *S* with neutral role and precede the head of *S*

---

ckipnlp.container.util.wspos module

This module provides containers for word-segmented sentences with part-of-speech-tags.

class ckipnlp.container.util.wspos.**WsPosToken**
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.util.wspos._WsPosToken

A word with POS-tag.

Variables

- **word** (*str*) – the word.
- **pos** (*str*) – the POS-tag.

Note: This class is an subclass of *tuple*. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for *from_text()* and *to_text()*.

```
'(Na)'  # word / POS-tag
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '', # word
    'pos': 'Na',     # POS-tag
}
```

List format Used for `from_list()` and `to_list()`.

```
[
    '', # word
    'Na',    # POS-tag
]
```

---

classmethod from_text(*data*)

Construct an instance from text format.

Parameters **data** (*str*) – text such as `'(Na)'`.

Note:

- `'(Na)'` -> word = `''`, pos = `'Na'`
- `''` -> word = `''`, pos = `None`

---

to_text()

class ckipnlp.container.util.wspos.**WsPosSentence**

Bases: `object`

A helper class for data conversion of word-segmented and part-of-speech sentences.

classmethod from_text(*data*)

Convert text format to word-segmented and part-of-speech sentences.

Parameters **data** (*str*) – text such as `'(Na)\u3000(T)'`.

Returns

- *SegSentence* – the word sentence
- *SegSentence* – the POS-tag sentence.

static to_text(*word*, *pos*)

Convert text format to word-segmented and part-of-speech sentences.

Parameters

- **word** (*SegSentence*) – the word sentence
- **pos** (*SegSentence*) – the POS-tag sentence.

Returns *str* – text such as `'(Na)\u3000(T)'`.

class ckipnlp.container.util.wspos.**WsPosParagraph**

Bases: `object`

A helper class for data conversion of word-segmented and part-of-speech sentence lists.

classmethod from_text(*data*)

Convert text format to word-segmented and part-of-speech sentence lists.

Parameters **data** (*Sequence[str]*) – list of sentences such as `'(Na)\u3000(T)'`.

Returns

- *SegParagraph* – the word sentence list
- *SegParagraph* – the POS-tag sentence list.

static to_text(*word*, *pos*)
Convert text format to word-segmented and part-of-speech sentence lists.

Parameters

- **word** (*SegParagraph*) – the word sentence list
- **pos** (*SegParagraph*) – the POS-tag sentence list.

Returns *List[str]* – list of sentences such as `'(Na)\u3000(T)'`.

Submodules

5.1.2 ckipnlp.container.base module

This module provides base containers.

class ckipnlp.container.base.**Base**
Bases: object

The base CKIPNLP container.

abstract classmethod from_text(*data*)
Construct an instance from text format.

Parameters **data** (*str*) –

abstract to_text()
Transform to plain text.

Returns *str*

abstract classmethod from_dict(*data*)
Construct an instance a from python built-in containers.

abstract to_dict()
Transform to python built-in containers.

abstract classmethod from_list(*data*)
Construct an instance a from python built-in containers.

abstract to_list()
Transform to python built-in containers.

classmethod from_json(*data*, ***kwargs*)
Construct an instance from JSON format.

Parameters **data** (*str*) – please refer *from_dict()* for format details.

to_json(***kwargs*)
Transform to JSON format.

Returns *str*

class ckipnlp.container.base.**BaseTuple**
Bases: *ckipnlp.container.base.Base*

The base CKIPNLP tuple.

abstract classmethod from_text(*data*)

abstract to_text()

classmethod from_dict(*data*)
Construct an instance from python built-in containers.

Parameters **data** (*dict*) –

to_dict()
Transform to python built-in containers.

Returns *dict*

classmethod from_list(*data*)
Construct an instance from python built-in containers.

Parameters **data** (*list*) –

to_list()
Transform to python built-in containers.

Returns *list*

class ckipnlp.container.base.**BaseList**(*initlist=None*)
Bases: ckipnlp.container.base._BaseList, ckipnlp.container.base._InterfaceItem

The base CKIPNLP list.

item_class = Not Implemented
Must be a CKIPNLP container class.

class ckipnlp.container.base.**BaseList0**(*initlist=None*)
Bases: ckipnlp.container.base._BaseList, ckipnlp.container.base._InterfaceBuiltInItem

The base CKIPNLP list with built-in item class.

item_class = Not Implemented
Must be a built-in type.

class ckipnlp.container.base.**BaseSentence**(*initlist=None*)
Bases: ckipnlp.container.base._BaseSentence, ckipnlp.container.base._InterfaceItem

The base CKIPNLP sentence.

item_class = Not Implemented
Must be a CKIPNLP container class.

class ckipnlp.container.base.**BaseSentence0**(*initlist=None*)
Bases: ckipnlp.container.base._BaseSentence, ckipnlp.container.base._InterfaceBuiltInItem

The base CKIPNLP sentence with built-in item class.

item_class = Not Implemented
Must be a built-in type.

5.1.3 ckipnlp.container.coref module

This module provides containers for co-reference sentences.

class ckipnlp.container.coref.**CorefToken**
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.coref._CorefToken

A co-reference token.

Variables

- **word** (*str*) – the token word.
- **coref** (*Tuple[int, str]*) – the co-reference ID and type. *None* if not a co-reference source or target.
  - **type:**
    - *'source'*: co-reference source.
    - *'target'*: co-reference target.
    - *'zero'*: null element co-reference target.
- **idx** (*int*) – the node index in parsed tree.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Used for `to_list()`.

```
'_0'
```

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '',         # token word
    'coref': (0, 'source'), # coref ID and type
    'idx': 2,               # node index
}
```

List format Used for `from_list()` and `to_list()`.

```
[
    '',         # token word
    (0, 'source'), # coref ID and type
    2,             # node index
]
```

---

from_text = NotImplemented

to_text()

class ckipnlp.container.coref.**CorefSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseSentence*

A list of co-reference sentence.

Data Structure Examples

Text format Used for `to_list()`.

```
'_0_0' # Token segmented by \u3000 (full-width space)
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    { 'word': '', 'coref': (0, 'source'), 'idx': 2, }, # coref-token 1
    { 'word': '', 'coref': (0, 'target'), 'idx': 3, },    # coref-token 2
    { 'word': '', 'coref': None, 'idx': 4, },             # coref-token 3
]
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ '', (0, 'source'), 2, ], # coref-token 1
    [ '', (0, 'target'), 3, ],    # coref-token 2
    [ '', None, 4, ],             # coref-token 3
]
```

---

item_class
alias of *CorefToken*

from_text = NotImplemented

to_text()

class ckipnlp.container.coref.**CorefParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of co-reference sentence.

Data Structure Examples

Text format Used for `to_list()`.

```
[
    '_0_0', # Sentence 1
    'None_0',  # Sentence 2
]
```

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    [ # Sentence 1
        { 'word': '', 'coref': (0, 'source'), 'idx': 2, },
        { 'word': '', 'coref': (0, 'target'), 'idx': 3, },
        { 'word': '', 'coref': None, 'idx': 4, },
    ],
    [ # Sentence 2
        { 'word': None, 'coref': (0, 'zero'), None, },
        { 'word': '', 'coref': None, 'idx': 1, },
        { 'word': '', 'coref': None, 'idx': 2, },
    ],
]
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ # Sentence 1
        [ '', (0, 'source'), 2, ],
        [ '', (0, 'target'), 3, ],
        [ '', None, 4, ],
    ],
    [ # Sentence 2
        [ None, (0, 'zero'), None, ],
        [ '', None, 1, ],
        [ '', None, 2, ],
    ],
]
```

---

item_class
alias of *CorefSentence*

from_text = NotImplemented

5.1.4 ckipnlp.container.ner module

This module provides containers for NER sentences.

class ckipnlp.container.ner.**NerToken**
Bases: *ckipnlp.container.base.BaseTuple*, ckipnlp.container.ner._NerToken

A named-entity recognition token.

Variables

- **word** (*str*) – the token word.
- **ner** (*str*) – the NER-tag.
- **idx** (*Tuple[int, int]*) – the starting / ending index.

Note: This class is an subclass of `tuple`. To change the attribute, please create a new instance instead.

Data Structure Examples

Text format Not implemented

Dict format Used for `from_dict()` and `to_dict()`.

```
{
    'word': '',    # token word
    'ner': 'LANGUAGE', # NER-tag
    'idx': (0, 3),     # starting / ending index.
}
```

List format Used for `from_list()` and `to_list()`.

```
[
    ''      # token word
    'LANGUAGE', # NER-tag
```

(continues on next page)

(continued from previous page)

```
    (0, 3),      # starting / ending index.
]
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
(
    0,          # starting index
    3,          # ending index
    'LANGUAGE', # NER-tag
    '',     # token word
)
```

---

to_text = NotImplemented

from_text = NotImplemented

classmethod from_tagger(*data*)
Construct an instance a from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

class ckipnlp.container.ner.**NerSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseSentence*

A named-entity recognition sentence.

Data Structure Examples

Text format Not implemented

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    { 'word': '', 'ner': 'GPE', 'idx': (0, 2), },    # name-entity 1
    { 'word': '', 'ner': 'ORG', 'idx': (3, 5), }, # name-entity 2
]
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ '', 'GPE', (0, 2), ],    # name-entity 1
    [ '', 'ORG', (3, 5), ], # name-entity 2
]
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
[
    ( 0, 2, 'GPE', '', ),    # name-entity 1
    ( 3, 5, 'ORG', '', ), # name-entity 2
]
```

---

item_class
alias of *NerToken*

to_text = NotImplemented

from_text = NotImplemented

classmethod from_tagger(*data*)
Construct an instance a from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

class ckipnlp.container.ner.**NerParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of named-entity recognition sentence.

Data Structure Examples

Text format Not implemented

Dict format Used for `from_dict()` and `to_dict()`.

```
[
    [ # Sentence 1
        { 'word': '', 'ner': 'LANGUAGE', 'idx': (0, 3), },
    ],
    [ # Sentence 2
        { 'word': '', 'ner': 'GPE', 'idx': (0, 2), },
        { 'word': '', 'ner': 'ORG', 'idx': (3, 5), },
    ],
]
```

List format Used for `from_list()` and `to_list()`.

```
[
    [ # Sentence 1
        [ '', 'LANGUAGE', (0, 3), ],
    ],
    [ # Sentence 2
        [ '', 'GPE', (0, 2), ],
        [ '', 'ORG', (3, 5), ],
    ],
]
```

CkipTagger format Used for *from_tagger()* and *to_tagger()*.

```
[
    [ # Sentence 1
        ( 0, 3, 'LANGUAGE', '', ),
    ],
    [ # Sentence 2
        ( 0, 2, 'GPE', '', ),
        ( 3, 5, 'ORG', '', ),
    ],
]
```

---

item_class
alias of *NerSentence*

to_text = NotImplemented

from_text = NotImplemented

classmethod from_tagger(*data*)
Construct an instance a from CkipTagger format.

to_tagger()
Transform to CkipTagger format.

5.1.5 ckipnlp.container.parsed module

This module provides containers for parsed sentences.

class ckipnlp.container.parsed.**ParsedParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList0*

A list of parsed sentence.

Data Structure Examples

Text/Dict/List format Used for `from_text()`, `to_text()`, `from_dict()`, `to_dict()`, `from_list()`, and `to_list()`.

```
[
    'S(Head:Nab:|particle:Td:)',                      # Sentence 1
    '%(particle:I:|manner:Dh:|manner:Dh:|time:Dh:)', # Sentence 2
]
```

---

item_class
alias of `builtins.str`

5.1.6 ckipnlp.container.seg module

This module provides containers for word-segmented sentences.

class ckipnlp.container.seg.**SegSentence**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseSentence0*

A word-segmented sentence.

Data Structure Examples

Text format Used for *`from_text()`* and *`to_text()`*.

```
'' # Words segmented by \u3000 (full-width space)
```

Dict/List format Used for `from_dict()`, `to_dict()`, `from_list()`, and `to_list()`.

```
[ '', '', ]
```

Note: This class is also used for part-of-speech tagging.

---

item_class
alias of `builtins.str`

classmethod from_text(*data*)

to_text()

class ckipnlp.container.seg.**SegParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList*

A list of word-segmented sentences.

Data Structure Examples

Text format Used for `from_text()` and `to_text()`.

```
[
    '',     # Sentence 1
    '', # Sentence 2
]
```

Dict/List format Used for `from_dict()`, `to_dict()`, `from_list()`, and `to_list()`.

```
[
    [ '', '', ],          # Sentence 1
    [ '', '', '', '', ], # Sentence 2
]
```

Note: This class is also used for part-of-speech tagging.

---

item_class
alias of *SegSentence*

5.1.7 ckipnlp.container.text module

This module provides containers for text sentences.

class ckipnlp.container.text.**TextParagraph**(*initlist=None*)
Bases: *ckipnlp.container.base.BaseList0*

A list of text sentence.

Data Structure Examples

Text/Dict/List format Used for `from_text()`, `to_text()`, `from_dict()`, `to_dict()`, `from_list()`, and `to_list()`.

```
[
    '', # Sentence 1
    '', # Sentence 2
]
```

---

item_class
alias of `builtins.str`

5.2 ckipnlp.driver package

This module implements CKIPNLP drivers.

Submodules

5.2.1 ckipnlp.driver.base module

This module provides base drivers.

class ckipnlp.driver.base.**DriverType**
Bases: enum.IntEnum

The enumeration of driver types.

SENTENCE_SEGMENTER = 1
Sentence segmentation

WORD_SEGMENTER = 2
Word segmentation

POS_TAGGER = 3
Part-of-speech tagging

NER_CHUNKER = 4
Named-entity recognition

SENTENCE_PARSER = 5
Sentence parsing

COREF_CHUNKER = 6
Co-reference delectation

class ckipnlp.driver.base.**DriverKind**
Bases: enum.IntEnum

The enumeration of driver backend kinds.

BUILTIN = 1
Built-in Implementation

TAGGER = 2
CkipTagger Backend

CLASSIC = 3
CkipClassic Backend

class ckipnlp.driver.base.**DriverRegister**
Bases: object

The driver registering utility.

static get(*driver_type*, *driver_kind*)

class ckipnlp.driver.base.**BaseDriver**(*, *lazy=False*)
Bases: object

The base CKIPNLP driver.

is_dummy = False

init()

abstract driver_type()

abstract driver_kind()

class ckipnlp.driver.base.**DummyDriver**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The dummy driver.

driver_type = None

driver_kind = None

is_dummy = True

5.2.2 ckipnlp.driver.classic module

This module provides drivers with CkipClassic backend.

class ckipnlp.driver.classic.**CkipClassicWordSegmenter**(*, *do_pos=False*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP word segmentation driver with CkipClassic backend.

driver_type = 2

driver_kind = 3

class ckipnlp.driver.classic.**CkipClassicSentenceParser**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP sentence parsing driver with CkipClassic backend.

driver_type = 5

driver_kind = 3

5.2.3 ckipnlp.driver.coref module

This module provides built-in co-reference detection driver.

class ckipnlp.driver.coref.**CkipCorefChunker**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP co-reference detection driver.

driver_type = 6

driver_kind = 1

static transform_ws(*, *text*, *ws*, *ner*)
Transform word-segmented sentence lists (create a new instance).

static transform_pos(*, *ws*, *pos*, *ner*)
Transform pos-tag sentence lists (modify in-place).

5.2.4 ckipnlp.driver.ss module

This module provides built-in sentence segmentation driver.

class ckipnlp.driver.ss.**CkipSentenceSegmenter**(*, *delims=',!?:;\n'*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP sentence segmentation driver.

driver_type = 1

driver_kind = 1

5.2.5 ckipnlp.driver.tagger module

This module provides drivers with CkipTagger backend.

class ckipnlp.driver.tagger.**CkipTaggerWordSegmenter**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP word segmentation driver with CkipTagger backend.

driver_type = 2

driver_kind = 2

class ckipnlp.driver.tagger.**CkipTaggerPosTagger**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP part-of-speech tagging driver with CkipTagger backend.

driver_type = 3

driver_kind = 2

class ckipnlp.driver.tagger.**CkipTaggerNerChunker**(*, *lazy=False*)
Bases: *ckipnlp.driver.base.BaseDriver*

The CKIP named-entity recognition driver with CkipTagger backend.

driver_type = 4

driver_kind = 2

5.3 ckipnlp.pipeline package

This module implements CKIPNLP pipelines.

Submodules

5.3.1 ckipnlp.pipeline.core module

This module provides core CKIPNLP pipeline.

class ckipnlp.pipeline.core.**CkipDocument**(*, *raw=None*, *text=None*, *ws=None*, *pos=None*, *ner=None*, *parsed=None*)
Bases: collections.abc.Mapping

The core document.

Variables

- **raw** (*str*) – The unsegmented text input.
- **text** (*TextParagraph*) – The sentences.
- **ws** (*SegParagraph*) – The word-segmented sentences.
- **pos** (*SegParagraph*) – The part-of-speech sentences.
- **ner** (*NerParagraph*) – The named-entity recognition results.
- **parsed** (*ParsedParagraph*) – The parsed-sentences.

class ckipnlp.pipeline.core.**CkipPipeline**(*, *sentence_segmenter_kind=<DriverKind.BUILTIN: 1>*, *word_segmenter_kind=<DriverKind.TAGGER: 2>*, *pos_tagger_kind=<DriverKind.TAGGER: 2>*, *sentence_parser_kind=<DriverKind.CLASSIC: 3>*, *ner_chunker_kind=<DriverKind.TAGGER: 2>*, *lazy=True*)

Bases: object

The core pipeline.

Parameters

- **sentence_segmenter_kind** (*DriverKind*) – The type of sentence segmenter.
- **word_segmenter_kind** (*DriverKind*) – The type of word segmenter.
- **pos_tagger_kind** (*DriverKind*) – The type of part-of-speech tagger.
- **ner_chunker_kind** (*DriverKind*) – The type of named-entity recognition chunker.
- **sentence_parser_kind** (*DriverKind*) – The type of sentence parser.

Other Parameters **lazy** (*bool*) – Lazy initialize the drivers.

get_text(*doc*)

Apply sentence segmentation.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.text** (*TextParagraph*) – The sentences.

Note: This routine modify **doc** inplace.

get_ws(*doc*)

Apply word segmentation.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.ws** (*SegParagraph*) – The word-segmented sentences.

Note: This routine modify **doc** inplace.

get_pos(*doc*)

Apply part-of-speech tagging.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.pos** (*SegParagraph*) – The part-of-speech sentences.

Note: This routine modify **doc** inplace.

get_ner(*doc*)
Apply named-entity recognition.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.ner** (*NerParagraph*) – The named-entity recognition results.

Note: This routine modify **doc** inplace.

get_parsed(*doc*)
Apply sentence parsing.

Parameters **doc** (*CkipDocument*) – The input document.

Returns **doc.parsed** (*ParsedParagraph*) – The parsed sentences.

Note: This routine modify **doc** inplace.

5.3.2 ckipnlp.pipeline.coref module

This module provides co-reference detection pipeline.

class ckipnlp.pipeline.coref.**CkipCorefDocument**(*, *ws=None*, *pos=None*, *parsed=None*, *coref=None*)

Bases: collections.abc.Mapping

The co-reference document.

Variables

- **ws** (*SegParagraph*) – The word-segmented sentences.
- **pos** (*SegParagraph*) – The part-of-speech sentences.
- **parsed** (*ParsedParagraph*) – The parsed sentences.
- **coref** (*CorefParagraph*) – The co-reference sentences.

class ckipnlp.pipeline.coref.**CkipCorefPipeline**(*, *coref_chunker_kind=<DriverKind.BUILTIN: 1>*, *lazy=True*, ***kwargs*)

Bases: *ckipnlp.pipeline.core.CkipPipeline*

The co-reference detection pipeline.

Parameters

- **sentence_segmenter_kind** (*DriverKind*) – The type of sentence segmenter.
- **word_segmenter_kind** (*DriverKind*) – The type of word segmenter.
- **pos_tagger_kind** (*DriverKind*) – The type of part-of-speech tagger.
- **ner_chunker_kind** (*DriverKind*) – The type of named-entity recognition chunker.
- **sentence_parser_kind** (*DriverKind*) – The type of sentence parser.

- **coref_chunker_kind** (*DriverKind*) – The type of co-reference detection chunker.

Other Parameters **lazy** (*bool*) – Lazy initialize the drivers.

get_coref(*doc*, *corefdoc*)
Apply co-reference delectation.

Parameters

- **doc** (*CkipDocument*) – The input document.
- **corefdoc** (*CkipCorefDocument*) – The input document for co-reference.

Returns **corefdoc.coref** (*CorefParagraph*) – The co-reference results.

Note: This routine modify **corefdoc** inplace.

5.4 ckipnlp.util package

This module implements extra utilities for CKIPNLP.

Submodules

5.4.1 ckipnlp.util.data module

This module implements data loading utilities for CKIPNLP.

class ckipnlp.util.data.**TaggerData**
Bases: ckipnlp.util.data._DataBase

name = 'tagger'

fullname = 'CkipTagger'

env = 'CKIPTAGGER_DATA'

extra_dirs = ('./data',)

classmethod user_data_dir()

classmethod site_data_dir()

ckipnlp.util.data.**get_tagger_data**()
Get CkipTagger data directory.

ckipnlp.util.data.**install_tagger_data**(*src_dir*, *, *copy=False*)
Link/Copy CkipTagger data directory.

ckipnlp.util.data.**download_tagger_data**()
Download CkipTagger data directory.

5.4.2 ckipnlp.util.logger module

This module implements logging utilities for CKIPNLP.

ckipnlp.util.logger.**get_logger**()
Get the CKIPNLP logger.

CHAPTER

SIX

TABLES OF TAGS

6.1 Part-of-Speech Tags

Tag	Description
A	
Caa	
Cab	
Cba	
Cbb	
D	
Da	
Dfa	
Dfb	
Di	
Dk	
DM	
I	
Na	
Nb	
Nc	
Ncd	
Nd	
Nep	
Neqa	
Neqb	
Nes	
Neu	
Nf	
Ng	
Nh	
Nv	
P	
T	
VA	
VAC	
VB	
VC	
VCL	

continues on next page

Table 1 – continued from previous page

Tag	Description
VD	
VF	
VE	
VG	
VH	
VHC	
VI	
VJ	
VK	
VL	
V_2	
DE	
SHI	
FW	
COLONCATEGORY	
COMMACATEGORY	
DASHCATEGORY	
DOTCATEGORY	
ETCCATEGORY	
EXCLAMATIONCATEGORY	
PARENTHESISCATEGORY	
PAUSECATEGORY	
PERIODCATEGORY	
QUESTIONCATEGORY	
SEMICOLONCATEGORY	
SPCHANGECATEGORY	
WHITESPACE	

6.2 Parsing Tree Tags

Tag	Description
S	SNP
VP	V
NP	N
GP	NgDUMMY1
PP	PDUMMY
XP	CXXPVPVPNP
DM	

6.3 Parsing Tree Roles

Role	Description
#	
apposition	
possessor	
predication	
property	
quantifier	
#–	
agent	
benefactor	
causer	
companion	
comparison	
experiencer	
goal	
range	
source	
target	
theme	
topic	
#–	
aspect	
degree	
deixis	
deontics	
duration	
evaluation	
epistemics	
frequency	
instrument	
interjection	
location	
manner	
negation	
particle	
quantity	
standard	
time	
#–	
addition	
alternative	
avoidance	
complement	
conclusion	
condition	
concession	
contrast	
conversion	

continues on next page

Table 2 – continued from previous page

Role	Description
exclusion	
hypothesis	
listing	
purpose	
reason	
rejection	
result	
restriction	
selection	
uncondition	
whatever	
#	
DUMMY	
DUMMY1	
DUMMY2	
Head	Head
head	
nominal	

CHAPTER

SEVEN

INDEX

CHAPTER

EIGHT

MODULE INDEX

PYTHON MODULE INDEX

C

INDEX

B

C

D

E

F

G

H

I

M

N

P

S

T

U

W