
elit Documentation

Gary Lai

Jul 31, 2018

Contents:

1	Installation	1
2	Documentation	3
2.1	SDK	3
2.2	Benchmark	3
2.3	Structure	3
2.4	Component	10
2.5	State	10
2.6	Model	10
2.7	Util	10
3	Indices and tables	11
	Python Module Index	13

CHAPTER 1

Installation

The elit project only supports Python with version ≥ 3.4 . Please make sure you have installed correct version of the Python on the machine.

It is as simple as one line to install elit by:

```
pip install elit
```


2.1 SDK

2.2 Benchmark

class `elit.benchmark.Timer` (*timer=<built-in function perf_counter>*)

Bases: `object`

Timer is an context manager measures elapsed time of running function or process.

Example:

```
from elitsdk.benchmark import Timer
from example.example import SpaceTokenizer
space_tokenizer = SpaceTokenizer()
with Timer() as t:
    space_tokenizer.decode("hello world")
print(t.runtime)
```

2.3 Structure

class `elit.structure.Sentence` (*d=None, **kwargs*)

Bases: `dict`

tokens

Returns the list of tokens in the sentence.

Return type list of str

part_of_speech_tags

Returns the list of part-of-speech tags corresponding to the tokens in this sentence.

Return type list of str

class `elit.structure.Document` (*d=None, **kwargs*)

Bases: `dict`

sentences

Returns the list of sentences in the document.

Return type list of `Sentence`

tokens

Returns list of tokens across all sentences.

Return type list of str

class `elit.structure.Node`

Bases: `object`

Node is used for constructing the NLP structure.

parent

Returns parent of current node

Return type `Node`

left_sibling

Returns left libling of current node

Return type `Node`

right_sibling

Returns right libling of current node

Return type `Node`

children

the list of children

Returns the list of children

Return type list

index_of (*child*)

the index of the child

Parameters **child** (`Node`) – a child node

Returns the index of the child node. return None if the given node is not a child

Return type int or None

child (*index*)

Parameters **index** (*int*) – index of a child node

Returns the index'th child of this node if exists; otherwise, None

Return type `Node` or None

first_child ()

Returns the first child of this node if exists; otherwise, None

Return type `Node` or None

last_child()

Returns the last child of this node if exists; otherwise, None

Return type *Node* or None

find_child_by (*matcher=<function Node.<lambda>>*, *index=0*)

Parameters

- **matcher** (*lambda*) – the condition
- **index** (*int*) – displacement (0: 1st, 1: 2nd, etc.)

Returns the order'th child matching the specific condition of this node if exists; otherwise, None

Return type *Node* or None

add_child (*node*, *index=None*)

Adds a node as the index'th child of this node if it is not already a child of this node.

Parameters

- **node** (*Node*) – node is added to children
- **index** (*int*) – add child to this position

Returns if the node is added successfully, return True; otherwise, False

Return type *bool*

set_child (*node*, *index*)

Sets a node as the index'th child of this node if it is not already a child of this node.

Parameters

- **node** (*Node*) – node is set at the index position
- **index** (*int*) – set a child to this position

Returns the previously index'th node if added; otherwise, None

Return type *Node* or None

remove_child (*node*)

Removes a child from this node.

Parameters **node** (*Node*) – remove node

Returns True if the node exist; otherwise, False

Return type *bool*

replace_child_from (*old_node*, *new_node*)

Replaces the child from old_node to new_node

Parameters

- **old_node** (*Node*) – old child
- **new_node** (*Node*) – new child

Returns True if the child is replaced successfully; otherwise, False

Return type *bool*

remove ()

Removes this node from its parent. If this is the only child, removeToken its parent from its grandparent, and applies this logic recursively to the ancestors.

has_child()

Whether current node has any child or not

Returns True if this node has any child; otherwise, False

Return type `bool`

is_child_of(*node*)

Whether current node is a child of the node or not

Parameters

- **node** – Node
- **node** – Node

Returns True if this node is a child of the specific node; otherwise, False

Return type `bool`

contains_child(*matcher*=<function Node.<lambda>>)

Whether current node contains a child with the matcher or not

Parameters **matcher** (*lambda*) – The matcher of condition

Returns True if this node contains a child of the matcher; otherwise, False

Return type `bool`

num_of_children()

Returns numbers of children

Return type `int`

find_children_in(*first_child_index*=0, *last_child_index*=None)

The sublist begins at the specific position and extends to the end.

Parameters

- **first_child_index** (*int*) – the ID of the first child (inclusive)
- **last_child_index** (*int*) – the ID of the last child (exclusive)

Returns an immutable list of sub-children

Return type `list`

find_children_by(*matcher*=<function Node.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the list of children matching the specific condition.

Return type `list`

grand_children()

Returns the list of grand-children

Return type `list`

find_first_descendant_by(*matcher*=<function Node.<lambda>>)

Parameters **matcher** (*lambda*) – the condition

Returns the first descendant matching the specific condition

Return type `Node` or `None`

find_first_lowest_chained_descendant_by (*matcher*)

Parameters *matcher* (*lambda*) – the condition

Returns the first lowest descendant whose intermediate ancestors to this node all match the specific condition

Return type *Node* or *None*

descendants ()

Returns the list of all descendents (excluding this node).

Return type *list*

find_descendants_in (*depth*)

Parameters *depth* (*int*) – the level of the descendents to be retrieved (1: children, 2: childrenNLPNode + grand-children, etc.).

Returns the list of descendents (excluding this node).

Return type *list*

find_single_chained_by (*matcher*=<function *Node*.<lambda>>)

Parameters *matcher* (*lambda*) – the condition

Returns a node in this node's subtree (including self) matching the specific condition if it is single-chained.

Return type *Node* or *None*

adapt_descendants_from (*node*)

Parameters *node* (*Node*) –

is_descendants_of (*node*)

Parameters *node* (*Node*) – the node

Returns True if the node is a descendant of the specific node

Return type *bool*

grand_parent ()

Returns the grandparent of this node if exists; otherwise, *None*

Return type *Node* or *None*

ancestor (*height*)

Parameters *height* (*int*) – height of the ancestor from this node (1: parent, 2: grandparent, etc.).

Returns the lowest ancestor matching the specific condition

Return type *Node*

find_lowest_ancestor_by (*matcher*=<function *Node*.<lambda>>)

Parameters *matcher* (*lambda*) – the condition

Returns the lowest ancestor matching the specific condition

Return type *Node*

find_highest_ancestor_by (*matcher*=<function *Node*.<lambda>>)

Parameters `matcher` (*lambda*) – the condition

Returns the highest ancestor where all the intermediate ancestors match the specific condition

Return type *Node*

`lowest_common_ancestor` (*node*)

Parameters `node` (*Node*) – the node

Returns the lowest common ancestor of this node and the specified node

Return type *Node* or *None*

`ancestor_set` ()

Returns the setToken of all ancestors of this node

Return type *set*

`is_parent_of` (*node*)

Parameters `node` (*Node*) – the node

Returns True if this node is the parent of the specific node; otherwise, False

Return type *bool*

`is_ancestor_of` (*node*)

Parameters `node` (*Node*) – the node

Returns True if the node is a descendant of the specific node; otherwise, False

Return type *bool*

`has_parent` (*matcher*=<function *Node*.<lambda>>)

Parameters `matcher` – the condition

Returns True if this node has a parent; otherwise, False

Return type *bool*

`has_grand_parent` ()

Returns True if this node has a grand parent; otherwise, False

Return type *bool*

`siblings` ()

Returns the list of all siblings

Return type *list*

`left_nearest_sibling` (*order*=0)

Parameters `order` (*int*) – displacement (0: left-nearest, 1: 2nd left-nearest, etc.)

Returns the order'th left-nearest sibling of this node if exists; otherwise, *None*

Return type *Node* or *None*

`find_left_nearest_sibling_by` (*matcher*=<function *Node*.<lambda>>)

Parameters `matcher` (*lambda*) – the condition

Returns the left nearest sibling that matches the specific condition

Return type *Node*

right_nearest_sibling (*order=0*)

Parameters **order** (*int*) – displacement (0: left-nearest, 1: 2nd left-nearest, etc.)

Returns the order'th right-nearest sibling of this node if exists; otherwise, None

Return type *Node* or None

find_right_nearest_sibling_by (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns the right nearest sibling that matches the specific condition

Return type *Node*

has_left_sibling (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns Trur if this node has a left sibling; otherwise, False.

has_right_sibling (*matcher=<function Node.<lambda>>*)

Parameters **matcher** (*lambda*) – the condition

Returns Trur if this node has a right sibling; otherwise, False.

is_sibling_of (*node*)

Parameters **node** (*Node*) – the node

Returns True if this node is a sibling of the specific node.

Return type *bool*

is_left_sibling_of (*node*)

Parameters **node** (*Node*) –

Returns True if this node is a left sibling of the specific node

Return type *bool*

is_right_sibling_of (*node*)

Parameters **node** (*Node*) –

Returns True if this node is a right sibling of the specific node

Return type *bool*

find_node_in (*order, getter*)

Parameters

- **order** (*int*) – 0: self, 1: nearest, 2: second nearest, etc.
- **getter** (*str*) – takes a node and returns a node.

Returns the order'th node with respect to the getter.

Return type *Node*

find_nearest_node_by (*getter, matcher=<function Node.<lambda>>*)

Parameters

- **getter** (*str*) – takes a node and returns a node.

- **matcher** (*lambda*) – takes a node and the supplement, and returns true if its field matches to the specific predicate.

Returns the first node matching the specific condition

Return type *Node* or *None*

isolate()

Isolates this node from its parent, children, and siblings

second_order (*getter*)

Parameters **getter** (*str*) – takes a node and returns a list of nodes.

Returns the list of second order elements according to the getter.

Return type *list*

2.4 Component

2.5 State

2.6 Model

2.7 Util

`elit.util.to_gold(key: str) → str`

`elit.util.to_out(key: str) → str`

class `elit.util.BILOU`

Bases: `object`

`B = 'B'`

`I = 'I'`

`L = 'L'`

`O = 'O'`

`U = 'U'`

classmethod `to_chunks(tags: List[str], fix: bool = False) → List[Tuple[int, int, str]]`

Parameters

- **tags** – a list of tags encoded by BILOU.
- **fix** – if True, fixes potential mismatches in BILOU (see `heuristic_fix()`).

Returns a list of tuples where each tuple contains (begin index (inclusive), end index (exclusive), label) of the chunk.

classmethod `heuristic_fix(tags)`

Use heuristics to fix potential mismatches in BILOU. :param tags: a list of tags encoded by BLIOU.

`elit.util.get_default_args(func)`

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

e

`elit.benchmark`, 3
`elit.structure`, 3
`elit.util`, 10
`elitsdk.benchmark`, 3

A

adapt_descendants_from() (elit.structure.Node method),
7
 add_child() (elit.structure.Node method), 5
 ancestor() (elit.structure.Node method), 7
 ancestor_set() (elit.structure.Node method), 8

B

B (elit.util.BILOU attribute), 10
 BILOU (class in elit.util), 10

C

child() (elit.structure.Node method), 4
 children (elit.structure.Node attribute), 4
 contains_child() (elit.structure.Node method), 6

D

descendants() (elit.structure.Node method), 7
 Document (class in elit.structure), 4

E

elit.benchmark (module), 3
 elit.structure (module), 3
 elit.util (module), 10
 elitsdk.benchmark (module), 3

F

find_child_by() (elit.structure.Node method), 5
 find_children_by() (elit.structure.Node method), 6
 find_children_in() (elit.structure.Node method), 6
 find_descendants_in() (elit.structure.Node method), 7
 find_first_descendant_by() (elit.structure.Node method),
6
 find_first_lowest_chained_descendant_by()
(elit.structure.Node method), 6
 find_highest_ancestor_by() (elit.structure.Node method),
7
 find_left_nearest_sibling_by() (elit.structure.Node
method), 8

find_lowest_ancestor_by() (elit.structure.Node method),
7
 find_nearest_node_by() (elit.structure.Node method), 9
 find_node_in() (elit.structure.Node method), 9
 find_right_nearest_sibling_by() (elit.structure.Node
method), 9
 find_single_chained_by() (elit.structure.Node method), 7
 first_child() (elit.structure.Node method), 4

G

get_default_args() (in module elit.util), 10
 grand_children() (elit.structure.Node method), 6
 grand_parent() (elit.structure.Node method), 7

H

has_child() (elit.structure.Node method), 5
 has_grand_parent() (elit.structure.Node method), 8
 has_left_sibling() (elit.structure.Node method), 9
 has_parent() (elit.structure.Node method), 8
 has_right_sibling() (elit.structure.Node method), 9
 heuristic_fix() (elit.util.BILOU class method), 10

I

I (elit.util.BILOU attribute), 10
 index_of() (elit.structure.Node method), 4
 is_ancestor_of() (elit.structure.Node method), 8
 is_child_of() (elit.structure.Node method), 6
 is_descendants_of() (elit.structure.Node method), 7
 is_left_sibling_of() (elit.structure.Node method), 9
 is_parent_of() (elit.structure.Node method), 8
 is_right_sibling_of() (elit.structure.Node method), 9
 is_sibling_of() (elit.structure.Node method), 9
 isolate() (elit.structure.Node method), 10

L

L (elit.util.BILOU attribute), 10
 last_child() (elit.structure.Node method), 4
 left_nearest_sibling() (elit.structure.Node method), 8
 left_sibling (elit.structure.Node attribute), 4

lowest_common_ancestor() (elit.structure.Node method),
8

N

Node (class in elit.structure), 4

num_of_children() (elit.structure.Node method), 6

O

O (elit.util.BILOU attribute), 10

P

parent (elit.structure.Node attribute), 4

part_of_speech_tags (elit.structure.Sentence attribute), 3

R

remove() (elit.structure.Node method), 5

remove_child() (elit.structure.Node method), 5

replace_child_from() (elit.structure.Node method), 5

right_nearest_sibling() (elit.structure.Node method), 8

right_sibling (elit.structure.Node attribute), 4

S

second_order() (elit.structure.Node method), 10

Sentence (class in elit.structure), 3

sentences (elit.structure.Document attribute), 4

set_child() (elit.structure.Node method), 5

siblings() (elit.structure.Node method), 8

T

Timer (class in elit.benchmark), 3

to_chunks() (elit.util.BILOU class method), 10

to_gold() (in module elit.util), 10

to_out() (in module elit.util), 10

tokens (elit.structure.Document attribute), 4

tokens (elit.structure.Sentence attribute), 3

U

U (elit.util.BILOU attribute), 10