antu Documentation

Release 0.0.1

AntNLP

May 24, 2020

Contents:

Universal data IO and neural network modules in NLP tasks.

- *data IO is an universal module in Natural Language Processing system and not based on any framework (like TensorFlow, PyTorch, MXNet, Dynet...).
- *neural network module contains the neural network structures commonly used in NLP tasks. We want to design commonly used structures for each neural network framework. We will continue to develop this module.

CHAPTER 1

antu.io package

1.1 Subpackages

1.1.1 antu.io.dataset_readers package

Submodules

antu.io.dataset_readers.dataset_reader module

class antu.io.dataset_readers.dataset_reader.**DatasetReader**
Bases: object

Methods

input_to_instance	
read	

input_to_instance(*inputs: str*) → antu.io.instance.Instance

read(*file_path: str*) → List[antu.io.instance.Instance]

Module contents

1.1.2 antu.io.datasets package

Submodules

antu.io.datasets.dataset module

class antu.io.datasets.dataset.**Dataset**
Bases: object

Methods

build_dataset	

build_dataset()

datasets = {}

vocabulary_set = {}

Module contents

1.1.3 antu.io.fields package

Submodules

antu.io.fields.field module

class antu.io.fields.field.**Field**
Bases: object

A `Field` is an ingredient of a data instance. In most NLP tasks, `Field` stores data of string types. It contains one or more indexers that map string data to the corresponding index. Data instances are collections of fields.

Methods

count_vocab_items(counter, Dict[str, int]])	We count the number of strings if the string needs to be mapped to one or more integers.
index(vocab)	Gets one or more index mappings for each element in the Field.

count_vocab_items(*counter: Dict[str, Dict[str, int]]*) → None
We count the number of strings if the string needs to be mapped to one or more integers. You can pass directly if there is no string that needs to be mapped.

Parameters

counter [`Dict[str, Dict[str, int]]`]

"counter" is used to count the number of each item. The first key

represents the namespace of the vocabulary, and the second key represents

the string of the item.

index(*vocab: antu.io.vocabulary.Vocabulary*) → None
Gets one or more index mappings for each element in the Field.

Parameters

vocab [Vocabulary]

‘‘vocab‘‘ is used to get the index of each item.

antu.io.fields.index_field module

class antu.io.fields.index_field.**IndexField**(*name: str, tokens: List[str]*)

Bases: *antu.io.fields.field.Field*

A IndexField is an integer field, and we can use it to store data ID.

Parameters

name [str] Field name. This is necessary and must be unique (not the same as other field names).

tokens [List[str]] Field content that contains a list of string.

Methods

count_vocab_items(counters, Dict[str, int]])	IndexField doesn't need index operation.
index(vocab)	IndexField doesn't need index operation.

count_vocab_items(*counters: Dict[str, Dict[str, int]]*) → None

IndexField doesn't need index operation.

index(*vocab: antu.io.vocabulary.Vocabulary*) → None

IndexField doesn't need index operation.

antu.io.fields.sequence_label_field module

class antu.io.fields.sequence_label_field.**SequenceLabelField**(*name: str, tokens: List[str], indexers: List[antu.io.token_indexers.token_indexer.T*

Bases: *antu.io.fields.field.Field*

Methods

count_vocab_items(counters, Dict[str, int]])	We count the number of strings if the string needs to be mapped to one or more integers.
index(vocab)	Gets one or more index mappings for each element in the Field.

count_vocab_items(*counters: Dict[str, Dict[str, int]]*) → None

We count the number of strings if the string needs to be mapped to one or more integers. You can pass directly if there is no string that needs to be mapped.

Parameters

counter [Dict[str, Dict[str, int]]]

‘‘counter‘‘ is used to count the number of each item. The first key

represents the namespace of the vocabulary, and the second key represents

the string of the item.

index (*vocab: antu.io.vocabulary.Vocabulary*) → None
Gets one or more index mappings for each element in the Field.

Parameters

vocab [Vocabulary]

‘‘vocab‘‘ is used to get the index of each item.

antu.io.fields.text_field module

class antu.io.fields.text_field.**TextField**(*name: str, tokens: List[str], indexers: List[antu.io.token_indexers.token_indexer.TokenIndexer] = []*)
Bases: *antu.io.fields.field.Field*

A TextField is a data field that is commonly used in NLP tasks, and we can use it to store text sequences such as sentences, paragraphs, POS tags, and so on.

Parameters

name [str] Field name. This is necessary and must be unique (not the same as other field names).

tokens [List[str]] Field content that contains a list of string.

indexers [List[TokenIndexer], optional (default=‘‘list()‘‘)] Indexer list that defines the vocabularies associated with the field.

Methods

count_vocab_items(counters, Dict[str, int]])	We count the number of strings if the string needs to be counted to some
index(vocab)	Gets one or more index mappings for each element in the Field.

count_vocab_items (*counters: Dict[str, Dict[str, int]]*) → None

We count the number of strings if the string needs to be counted to some counters. You can pass directly if there is no string that needs

to be counted.

Parameters

counters [Dict[str, Dict[str, int]]] Element statistics for datasets. if field indexers indicate that this field is related to some counters, we use field content to update the counters.

index (*vocab: antu.io.vocabulary.Vocabulary*) → None
Gets one or more index mappings for each element in the Field.

Parameters

vocab [Vocabulary] vocab is used to get the index of each item.

Module contents

1.1.4 antu.io.token_indexers package

Submodules

antu.io.token_indexers.char_token_indexer module

class antu.io.token_indexers.char_token_indexer.**CharTokenIndexer**(*related_vocabs: List[str], transform: Callable[[str], str] = <function CharTokenIndexer.<lambda>>*)

Bases: *antu.io.token_indexers.token_indexer.TokenIndexer*

A `CharTokenIndexer` determines how string token get represented as arrays of list of character indices in a model.

Parameters

related_vocabs [`List[str]`] Which vocabularies are related to the indexer.

transform [`Callable[[str,], str]`, optional (default=‘‘lambda x:x‘‘)] What changes need to be made to the token when counting or indexing. Commonly used are lowercase transformation functions.

Methods

count_vocab_items(token, counters, Dict[str, ...)	Each character in the token is counted directly as an element.
tokens_to_indices(tokens, vocab)	Takes a list of tokens and converts them to one or more sets of indices.

count_vocab_items(*token: str, counters: Dict[str, Dict[str, int]]*) → None

Each character in the token is counted directly as an element.

Parameters

counter [`Dict[str, Dict[str, int]]`] We count the number of strings if the string needs to be counted to some counters.

tokens_to_indices(*tokens: List[str], vocab: antu.io.vocabulary.Vocabulary*) → Dict[str, List[List[int]]]

Takes a list of tokens and converts them to one or more sets of indices. During the indexing process, each token item corresponds to a list of index in the vocabulary.

Parameters

vocab [`Vocabulary`] `vocab` is used to get the index of each item.

antu.io.token_indexers.single_id_token_indexer module

class antu.io.token_indexers.single_id_token_indexer.**SingleIdTokenIndexer**(*related_vocabs: List[str], transform: Callable[[str], str] = <function SingleId-TokenIndexer.<lambda>>*)

Bases: *antu.io.token_indexers.token_indexer.TokenIndexer*

A `SingleIdTokenIndexer` determines how string token get represented as arrays of single id indices in a model.

Parameters

related_vocabs [`List[str]`] Which vocabularies are related to the indexer.

transform [`Callable[[str,], str]`, optional (default=‘‘lambda x:x‘‘)] What changes need to be made to the token when counting or indexing. Commonly used are lowercase transformation functions.

Methods

count_vocab_items(token, counters, Dict[str, ...)	The token is counted directly as an element.
tokens_to_indices(tokens, vocab)	Takes a list of tokens and converts them to one or more sets of indices.

count_vocab_items(*token: str, counters: Dict[str, Dict[str, int]]*) → None

The token is counted directly as an element.

Parameters

counter [`Dict[str, Dict[str, int]]`] We count the number of strings if the string needs to be counted to some counters.

tokens_to_indices(*tokens: List[str], vocab: antu.io.vocabulary.Vocabulary*) → Dict[str, List[int]]

Takes a list of tokens and converts them to one or more sets of indices. During the indexing process, each item corresponds to an index in the vocabulary.

Parameters

vocab [`Vocabulary`] `vocab` is used to get the index of each item.

Returns

res [`Dict[str, List[int]]`] if the token and index list is [w1:5, w2:3, w3:0], the result will be {‘vocab_name’ : [5, 3, 0]}

antu.io.token_indexers.token_indexer module

class antu.io.token_indexers.token_indexer.**TokenIndexer**
Bases: object

A TokenIndexer determines how string tokens get represented as arrays of indices in a model.

Methods

count_vocab_items(token, counter, Dict[str, ...)	Defines how each token in the field is counted.
tokens_to_indices(tokens, vocab)	Takes a list of tokens and converts them to one or more sets of indices.

count_vocab_items (*token: str, counter: Dict[str, Dict[str, int]]*) → None
Defines how each token in the field is counted. In most cases, just use the string as a key. However, for character-level TokenIndexer, you need to traverse each character in the string.

Parameters

counter [Dict[str, Dict[str, int]]] We count the number of strings if the string needs to be counted to some counters.

tokens_to_indices (*tokens: List[str], vocab: antu.io.vocabulary.Vocabulary*) → Dict[str, Indices]
Takes a list of tokens and converts them to one or more sets of indices. This could be just an ID for each token from the vocabulary.

Parameters

vocab [Vocabulary] vocab is used to get the index of each item.

Module contents

1.2 Submodules

1.3 antu.io.instance module

class antu.io.instance.**Instance** (*fields: List[antu.io.fields.field.Field] = None*)
Bases: collections.abc.Mapping, typing.Generic

An Instance is a collection (list) of multiple data fields.

Parameters

fields [List[Field], optional (default=‘‘None‘‘)] A list of multiple data fields.

Methods

add_field(field)	Add the field to the existing Instance.
count_vocab_items(counter, Dict[str, int]])	Increments counts in the given counter for all of the vocabulary items in all of the Fields in this Instance.

Continued on next page

Table 8 – continued from previous page

dynamic_index_fields(vocab, dynamic_fields)	Indexes all fields in this `Instance` using the provided `Vocabulary`.
`get`(k[,d])	
index_fields(vocab)	Indexes all fields in this `Instance` using the provided `Vocabulary`.
`items`()	
`keys`()	
`values`()	

add_field (*field: antu.io.fields.field.Field*) → None
Add the field to the existing `Instance`.

Parameters

field [`Field`] Which field needs to be added.

count_vocab_items (*counter: Dict[str, Dict[str, int]]*) → None
Increments counts in the given `counter` for all of the vocabulary items in all of the `Fields` in this `Instance`.

Parameters

counter [`Dict[str, Dict[str, int]]`] We count the number of strings if the string needs to be counted to some counters.

dynamic_index_fields (*vocab: antu.io.vocabulary.Vocabulary, dynamic_fields: Set[str]*) → Dict[str, Dict[str, Indices]]
Indexes all fields in this `Instance` using the provided `Vocabulary`. This *mutates* the current object, it does not return a new `Instance`. A `DataIterator` will call this on each pass through a dataset; we use the `indexed` flag to make sure that indexing only happens once. This means that if for some reason you modify your vocabulary after you've indexed your instances, you might get unexpected behavior.

Parameters

vocab [`Vocabulary`] `vocab` is used to get the index of each item.

Returns

res [`Dict[str, Dict[str, Indices]]`] Returns the Indices corresponding to the instance. The first key is field name and the second key is the vocabulary name.

index_fields (*vocab: antu.io.vocabulary.Vocabulary*) → Dict[str, Dict[str, Indices]]
Indexes all fields in this `Instance` using the provided `Vocabulary`. This *mutates* the current object, it does not return a new `Instance`. A `DataIterator` will call this on each pass through a dataset; we use the `indexed` flag to make sure that indexing only happens once. This means that if for some reason you modify your vocabulary after you've indexed your instances, you might get unexpected behavior.

Parameters

vocab [`Vocabulary`] `vocab` is used to get the index of each item.

Returns

res [`Dict[str, Dict[str, Indices]]`] Returns the Indices corresponding to the instance. The first key is field name and the second key is the vocabulary name.

1.4 antu.io.vocabulary module

class antu.io.vocabulary.**Vocabulary**(*counters: Dict[str, Dict[str, int]] = {}, min_count: Dict[str, int] = {}, pretrained_vocab: Dict[str, List[str]] = {}, intersection_vocab: Dict[str, str] = {}, no_pad_namespace: Set[str] = {}, no_unk_namespace: Set[str] = {}*)

Bases: object

Parameters

counters [Dict[str, Dict[str, int]], optional (default= dict())] Element statistics for datasets.

min_count [Dict[str, int], optional (default= dict())] Defines the minimum number of occurrences when some counter are converted to vocabulary.

pretrained_vocab [Dict[str, List[str]], optional (default= dict()] External pre-trained vocabulary.

intersection_vocab [Dict[str, str], optional (default= dict())] Defines the intersection with which vocabulary takes, when loading some oversized pre-trained vocabulary.

no_pad_namespace [Set[str], optional (default= set())] Defines which vocabularies do not have *pad* token.

no_unk_namespace [Set[str], optional (default= set())] Defines which vocabularies do not have *oov* token.

Methods

add_token_to_namespace(token, namespace)	Extend the vocabulary by add token to vocabulary namespace.
extend_from_counter(counters, Dict[str, ...)	Extend the vocabulary from the dataset statistic counters after defining the vocabulary.
extend_from_pretrained_vocab(...)	Extend the vocabulary from the pre-trained vocabulary after defining the vocabulary.
get_token_from_index(index, vocab_name)	Gets the token of a index in the vocabulary.
get_token_index(token, vocab_name)	Gets the index of a token in the vocabulary.
get_vocab_size(namespace)	Gets the size of a vocabulary.

get_padding_index	
get_unknow_index	

add_token_to_namespace(*token: str, namespace: str*) → None

Extend the vocabulary by add token to vocabulary namespace.

Parameters

token [str] The token that needs to be added.

namespace [str] Which vocabulary needs to be added to.

extend_from_counter(*counters: Dict[str, Dict[str, int]], min_count: Union[int, Dict[str, int]] = {}, no_pad_namespace: Set[str] = {}, no_unk_namespace: Set[str] = {}*) → None

Extend the vocabulary from the dataset statistic counters after defining the vocabulary.

Parameters

counters [`Dict[str, Dict[str, int]]`] Element statistics for datasets.

min_count [`Dict[str, int]`, optional (default= `dict()`)] Defines the minimum number of occurrences when some counter are converted to vocabulary.

no_pad_namespace [`Set[str]`, optional (default= `set()`)] Defines which vocabularies do not have *pad* token.

no_unk_namespace [`Set[str]`, optional (default= `set()`)] Defines which vocabularies do not have *oov* token.

extend_from_pretrained_vocab(*pretrained_vocab: Dict[str, List[str]], intersection_vocab: Dict[str, str] = {}, no_pad_namespace: Set[str] = {}, no_unk_namespace: Set[str] = {}*) → None

Extend the vocabulary from the pre-trained vocabulary after defining the vocabulary.

Parameters

pretrained_vocab [`Dict[str, List[str]]`] External pre-trained vocabulary.

intersection_vocab [`Dict[str, str]`, optional (default= `dict()`)] Defines the intersection with which vocabulary takes, when loading some oversized pre-trained vocabulary.

no_pad_namespace [`Set[str]`, optional (default= `set()`)] Defines which vocabularies do not have *pad* token.

no_unk_namespace [`Set[str]`, optional (default= `set()`)] Defines which vocabularies do not have *oov* token.

get_padding_index(*namespace: str*) → int

get_token_from_index(*index: int*, *vocab_name: str*) → str

Gets the token of a index in the vocabulary.

Parameters

index [`int`] Gets the token of which index.

namespace [`str`] Which vocabulary this index belongs to.

Returns

Token [`str`]

get_token_index(*token: str*, *vocab_name: str*) → int

Gets the index of a token in the vocabulary.

Parameters

token [`str`] Gets the index of which token.

namespace [`str`] Which vocabulary this token belongs to.

Returns

Index [`int`]

get_unknow_index(*namespace: str*) → int

get_vocab_size(*namespace: str*) → int

Gets the size of a vocabulary.

Parameters

namespace [str] Which vocabulary.

Returns

Vocabulary size [int]

1.5 Module contents

CHAPTER 2

antu.nn package

2.1 Subpackages

2.1.1 antu.nn.dynet package

Submodules

antu.nn.dynet.attention_mechanism module

antu.nn.dynet.char2word_embedder module

antu.nn.dynet.initializer module

antu.nn.dynet.multi_layer_perception module

antu.nn.dynet.nn_classifier module

antu.nn.dynet.rnn_builder module

Module contents

2.2 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

G

I

R

S

T

V