Forte Wrapper Documentation

Release 0.0.3

Forte

Jan 19, 2023

CONTENTS

Forte is a toolkit for building Natural Language Processing pipelines. This repository tries to wrap the fantastic collections of NLP libraries built by the community.

This project is part of the CASL Open Source family.

CHAPTER

ONE

GET STARTED

- First, install the library along with the desired tools. Let's take AllenNLP as an example:

```
git clone https://github.com/asyml/forte-wrappers.git
cd forte-wrappers
pip install ."[allennlp]"
```

CHAPTER

TWO

LIBRARIES AND WRAPPER SOURCE

- NLTK (Processors)
  - POS Tagger
  - Sentence Segmenter
  - Tokenizer
  - Lemmatizer
  - NER
- spaCy (Processors)
  - Tokenizer, Lemmatizer and POS Tagging
  - NER
- AllenNLP (Processors)
  - Tokenizer, POS Tagging
  - Semantic Role Labeling
  - Dependency Parsing
- Stanza (Processors)
  - Tokenization, POS Tagging, Lemmatizer
  - Dependency Parsing
- HuggingFace Models
  - BioBERT NER (Processors)
- Vader Sentiment (Processors)
  - Sentiment Analysis
- Elastic Search (Processors)
  - Elastic Indexer
  - Elastic Search
- Faiss (Processors)
  - Faiss Indexer
- GPT2 (Processors)
  - GPT2 Text Generation

- Tweepy (Processors)
  - TwitterAPI Search

CHAPTER

THREE

LICENSE

Apache License 2.0

CHAPTER

FOUR

AVAILABLE WRAPPERS

4.1 Processors

4.1.1 AllenNLP

AllenNLP Processors

4.1.2 SpaCy

SpaCy Processors

class fortex.spacy.**SpacyProcessor**

This processor wraps spaCy(v2.3.x) and ScispaCy(v0.3.0) models, providing functions including sentence parsing, tokenize, POS tagging, lemmatization, NER, and medical entity linking.

This processor will do user defined tasks according to configs. The supported tasks includes:

- *sentence*: sentence segmentation
- *tokenize*: word tokenize
- *pos*: Part-of-speech tagging
- *lemma*: word lemmatization
- *ner*: named entity recognition
- *dep*: dependency parsing
- *umls_link*: medical entity linking to UMLS concepts

spaCy is a library for advanced Natural Language Processing in Python and Cython. spaCy github page: https://github.com/explosion/spaCy/tree/v2.3.1

ScispaCy is a Python package containing spaCy models for processing biomedical, scientific or clinical text. ScispaCy github page: https://github.com/allenai/scispacy/tree/v0.3.0

Citation:

- spaCy: Industrial-strength Natural Language Processing in Python
- ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing.

initialize(*resources*, *configs*)

The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()

This defines a basic config structure for spaCy.

Following are the keys for this dictionary:

- *processors*: List of strings that defines which components will be included and will be performed on the input pack, default value is *["sentence", "tokenize", "pos", "lemma"]* which performs the basic operations included in spaCy models like *en_core_web_sm*, *sentence* performs segmentation, *tokenize* will perform tokenization and pos tagging, *ner* will perform named entity recognition, *lemma* will perform lemmatization.

  Additional values for this list further includes: *ner* for named entity and *dep* for dependency parsing.

- ***medical_onto_type*: defines which entry type in the input pack** that the medical entity mentions should be saved as output.
- ***umls_onto_type*: defines which entry type in the input pack** that the UMLS concept links should be saved as part of output.
- *lang*: language model, default is spaCy *en_core_web_sm* model. The pipeline support spaCy and ScispaCy models. A list of available spaCy models could be found at https://spacy.io/models. For UMLS entity linking task, ScispaCy model trained on biomedical dataset is preferred. A list of available models could be found at https://github.com/allenai/scispacy/tree/v0.3.0.
- *require_gpu*: whether GPU is required, default value is False. This value is directly used by https://spacy.io/api/top-level#spacy.require_gpu
- *prefer_gpu*: whether gpu is preferred, default value is False. This value is directly used by https://spacy.io/api/top-level#spacy.prefer_gpu
- *gpu_id*: the GPU device index to use when GPU is enabled. Default is 0.
- *testing*: states whether or not the processor is being used in a test case.

Returns: A dictionary with the default config for this processor.

record(*record_meta*)

Method to add output type record of current processor to `forte.data.data_pack.Meta.record`. The processor produce different types with different settings of *processors* in config.

Parameters **record_meta** – the field in the data pack for type record that need to fill in for consistency checking.

class fortex.spacy.**SpacyBatchedProcessor**

This processor wraps spaCy(v2.3.x) and ScispaCy(v0.3.0) models, providing most models included in the SpaCy pipeline, such as including sentence parsing, tokenize, POS tagging, lemmatization, NER, and medical entity linking. This is the batch processing version for *SpacyProcessor*, where it supports to batching across different data packs.

This processor will do user defined tasks according to configs. The supported tasks includes:

- *sentence*: sentence segmentation
- *tokenize*: word tokenize
- *pos*: Part-of-speech tagging
- *lemma*: word lemmatization

- *ner*: named entity recognition
- *dep*: dependency parsing
- *umls_link*: medical entity linking to UMLS concepts

Citation:

- spaCy: Industrial-strength Natural Language Processing in Python
- ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod define_batcher()
The batcher take raw text from a fixed number of data packs.

predict(*data_batch*)
The function that task processors should implement. Make predictions for the input `data_batch`.

Parameters **data_batch** (*dict*) – A batch of instances in our `dict` format.

Returns The prediction results in dictionary form.

pack(*pack*, *predict_results*, *_=None*)
The function that task processors should implement. It is the custom function on how to add the predicted output back to the data pack.

Parameters

- **pack** – The pack to add entries or fields to.
- **predict_results** – The prediction results returned by `predict()`. This processor will add these results to the provided *pack* as entry and attributes.
- **context** – The context entry that the prediction is performed, and the pack operation should be performed related to this range annotation. If None, then we consider the whole data pack is used as the context.

record(*record_meta*)
Method to add output type record of current processor to `forte.data.data_pack.Meta.record`. The processor produce different types with different settings of *processors* in config.

Parameters **record_meta** – the field in the data pack for type record that need to fill in for consistency checking.

classmethod default_configs()
Specify additional parameters for SpaCy processor.

The available parameters are:

- *medical_onto_type*: defines which entry type in the input pack that the medical entity mentions should be saved as output.
- *umls_onto_type*: defines which entry type in the input pack that the UMLS concept links should be saved as part of output.

- *batcher.batch_size*: max size of the batch (in terms of number of data packs).
- *processors*: List of strings that defines which components will be included and will be performed on the input pack, default value is *["sentence", "tokenize", "pos", "lemma"]* which performs the basic operations included in spaCy models like *en_core_web_sm*, *sentence* performs segmentation, *tokenize* will perform tokenization and pos tagging, *ner* will perform named entity recognition, *lemma* will perform lemmatization. Additional values for this list further includes: *ner* for named entity and *dep* for dependency parsing.
- *lang*: language model, default is spaCy *en_core_web_sm* model. The pipeline support spaCy and ScispaCy models. A list of available spaCy models could be found at https://spacy.io/models. For UMLS entity linking task, ScispaCy model trained on biomedical dataset is preferred. A list of available models could be found at https://github.com/allenai/scispacy/tree/v0.3.0
- *require_gpu*: whether GPU is required, default value is False. This value is directly used by https://spacy.io/api/top-level#spacy.require_gpu
- *prefer_gpu*: whether gpu is preferred, default value is False. This value is directly used by https://spacy.io/api/top-level#spacy.prefer_gpu
- *gpu_id*: the GPU device index to use when GPU is enabled. Default is 0.
- *num_processes*: number of processes to run when using *spacy.pipe*. Default is 1. This will be passed directly to the *n_process* option.
- *testing*: states whether or not the processor is being used in a test case.

4.1.3 NLTK

NLTK Processors

class fortex.nltk.**NLTKPOSTagger**

A wrapper of NLTK pos tagger.

initialize(*resources*, *configs*)

The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

record(*record_meta*)

Method to add output type record of *NLTKPOSTagger*, which adds attribute *pos* to *ft.onto.base_ontology.Token* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

expected_types_and_attributes()

Method to add expected type *ft.onto.base_ontology.Token* for input which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

class fortex.nltk.**NLTKSentenceSegmenter**

A wrapper of NLTK sentence tokenizer.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

record(*record_meta*)
Method to add output type record of *NLTKSentenceSegmenter*, which is *ft.onto.base_ontology.Sentence* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

class fortex.nltk.**NLTKWordTokenizer**
A wrapper of NLTK word tokenizer.

record(*record_meta*)
Method to add output type record of *NLTKWordTokenizer*, which is *ft.onto.base_ontology.Token*, to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

class fortex.nltk.**NLTKLemmatizer**
A wrapper of NLTK lemmatizer.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

record(*record_meta*)
Method to add output type record of *NLTKLemmatizer* which adds attribute *lemma* to *ft.onto.base_ontology.Token* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

expected_types_and_attributes()
Method to add expected type *ft.onto.base_ontology.Token* with attribute *pos* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

class fortex.nltk.**NLTKChunker**
A wrapper of NLTK chunker.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()
This defines a basic config structure for NLTKChunker.

record(*record_meta*)
Method to add output type record of *NLTKChunker* which adds *ft.onto.base_ontology.Phrase* with attribute *phrase_type* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

expected_types_and_attributes()
Method to add expected type ft.onto.base_ontology.Token` with attribute *pos* and *ft.onto.base_ontology.Sentence* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

class fortex.nltk.**NLTKNER**
A wrapper of NLTK NER.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

record(*record_meta*)
Method to add output type record of *NLTKNER* which is *ft.onto.base_ontology.EntityMention* with attribute *phrase_type* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

expected_types_and_attributes()
Method to add expected type ft.onto.base_ontology.Token` with attribute *pos* and *ft.onto.base_ontology.Sentence* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

4.1.4 Stanza

Stanza Processors

class fortex.stanza.**StandfordNLPProcessor**

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()
This defines a basic config structure for StanfordNLP.

record(*record_meta*)
Method to add output type record of current processor to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

4.1.5 HuggingFace

HuggingFace Processors

class fortex.huggingface.**ZeroShotClassifier**
Wrapper of the models on HuggingFace platform with pipeline tag of *zero-shot-classification*. https://huggingface.co/models?pipeline_tag=zero-shot-classification This wrapper could take any model name on HuggingFace platform with pipeline tag of *zero-shot-classification* in configs to make prediction on the user specified entry type in the input pack and the prediction result goes to the user specified attribute name of that entry type in the output pack. User could input the prediction labels in the config with any word or phrase.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()
This defines a basic config structure for ZeroShotClassifier.

Following are the keys for this dictionary:

- *entry_type*: defines which entry type in the input pack to make prediction on. The default makes prediction on each *Sentence* in the input pack.
- *attribute_name*: defines which attribute of the *entry_type* in the input pack to save prediction to. The default saves prediction to the *classification* attribute for each *Sentence* in the input pack.
- *multi_class*: whether to allow multiple class true
- *model_name*: language model, default is *"valhalla/distilbart-mnli-12-1"*. The wrapper supports Hugging Face models with pipeline tag of *zero-shot-classification*.
- *candidate_labels*: The set of possible class labels to classify each sequence into. Can be a single label, a string of comma-separated labels, or a list of labels. Note that for the model with a specific language, the candidate_labels need to be of that language.
- *hypothesis_template*: The template used to turn each label into an NLI-style hypothesis. This template must include a {} or similar syntax for the candidate label to be inserted into the template.

For example, the default template is "`This example is {}.`" Note that for the model with a specific language, the hypothesis_template need to be of that language.

- *cuda_device*: Device ordinal for CPU/GPU supports. Setting this to -1 will leverage CPU, a positive will run the model on the associated CUDA device id.

Returns: A dictionary with the default config for this processor.

expected_types_and_attributes()
Method to add expected type *ft.onto.base_ontology.Sentence* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

record(*record_meta*)
Method to add output type record of *ZeroShotClassifier* which is user specified entry type with user specified attribute name to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

class fortex.huggingface.**QuestionAnsweringSingle**
Wrapper of the models on HuggingFace platform with pipeline tag of *question-answering* (reading comprehension). https://huggingface.co/models?pipeline_tag=question-answering This wrapper could take any model name on HuggingFace platform with pipeline tag of *question-answering* in configs to make prediction on the context of user specified entry type in the input pack and the prediction result would be annotated as *Phrase* in the output pack. User could input the question in the config.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()
This defines a basic config structure for *QuestionAnsweringSingle*.

Following are the keys for this dictionary:

- *entry_type*: defines which entry type in the input pack to make prediction on. The default makes prediction on each *Document* in the input pack.
- *model_name*: language model, default is *"ktrapeznikov/biobert_v1.1_pubmed_squad_v2"*. The wrapper supports Hugging Face models with pipeline tag of *question-answering*.
- *question*: One question to retrieve answer from the input pack context.
- *max_answer_len*: The maximum length of predicted answers (e.g., only answers with a shorter length are considered).
- *cuda_device*: Device ordinal for CPU/GPU supports. Setting this to -1 will leverage CPU, a positive will run the model on the associated CUDA device id.
- *handle_impossible_answer*: Whether or not we accept impossible as an answer.

Returns: A dictionary with the default config for this processor.

expected_types_and_attributes()
Method to add user specified expected type which would be checked before running the processor if the

pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

record(*record_meta*)
Method to add output type record of *QuestionAnsweringSingle* which is *"ft.onto.base_ontology.Phrase"* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

class fortex.huggingface.**BERTTokenizer**
A wrapper of BERT tokenizer.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()
Returns a *dict* of configurations of the processor with default values. Used to replace the missing values of input `configs` during pipeline construction.

record(*record_meta*)
Method to add output type *ft.onto.base_ontology.Subword* of current processor *BERTTokenizer* to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

class fortex.huggingface.**BioBERTNERPredictor**
An Named Entity Recognizer fine-tuned on BioBERT

Note that to use *BioBERTNERPredictor*, the `ontology` of `Pipeline` must be an ontology that include `ft.onto.base_ontology.Subword` and `ft.onto.base_ontology.Sentence`.

initialize(*resources*, *configs*)
The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

predict(*data_batch*)
The function that task processors should implement. Make predictions for the input `data_batch`.

Parameters **data_batch** (*dict*) – A batch of instances in our `dict` format.

Returns The prediction results in dictionary form.

pack(*pack*, *predict_results=None*, *context=None*)
Write the prediction results back to datapack by aggregating subwords into named entity mentions.

classmethod default_configs()
Default config for NER Predictor

record(*record_meta*)
Method to add output type record of current processor to `forte.data.data_pack.Meta.record`.

Parameters **record_meta** – the field in the datapack for type record that need to fill in for consistency checking.

expected_types_and_attributes()
Method to add expected type ft.onto.base_ontology.Subword` with attribute *is_first_segment* and *ft.onto.base_ontology.Sentence* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

4.1.6 Twitter

Twitter Processors

class fortex.tweepy.**TweetSearchProcessor**
TweetSearchProcessor is designed to query tweets with Tweepy and Twitter API. Tweets will be returned as datapacks in input multipack.

classmethod default_configs()
This defines a basic config structure for TweetSearchProcessor. For more details about the parameters, refer to https://docs.tweepy.org/en/latest/api.html#tweepy.API.search_tweets and https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets

Returns A dictionary with the default config for this processor.

Following are the keys for this dictionary:

- ***"credential_file"*:** Defines the path of credential file needed for Twitter API usage.
- ***"num_tweets_returned"*:** Defines the number of tweets returned by processor.
- ***"lang"*:** Language, restricts tweets to the given language, default is 'en'.
- ***"date_since"*:** Restricts tweets created after the given date.
- ***"result_type"*:** Defines what type of search results to receive. The default is "recent." Valid values include:

  mixed : include both popular and real time results in the response

  recent : return only the most recent results in the response

  popular : return only the most popular results in the response.
- ***"query_pack_name"*:** The query pack's name, default is "query".
- ***"response_pack_name_prefix"*:** The pack name prefix to be used in response data packs.

4.1.7 Vader

Vader Processors

class fortex.vader.**VaderSentimentProcessor**

A wrapper of a sentiment analyzer: Vader (Valence Aware Dictionary and Sentiment Reasoner). Vader needs to be installed to use this package

> *pip install vaderSentiment*

or

> *pip install –upgrade vaderSentiment*

This processor will add assign sentiment label to each sentence in the document. If the input pack contains no sentence then no processing will happen. If the data pack has multiple set of sentences, one can specify the set of sentences to tag by setting the *sentence_component* attribute.

Vader URL: (https://github.com/cjhutto/vaderSentiment)

Citation: VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text (by C.J. Hutto and Eric Gilbert)

initialize(*resources*, *configs*)

The pipeline will call the initialize method at the start of a processing. The processor and reader will be initialized with `configs`, and register global resources into `resource`. The implementation should set up the states of the component.

Parameters

- **resources** – A global resource register. User can register shareable resources here, for example, the vocabulary.
- **configs** – The configuration passed in to set up this component.

classmethod default_configs()

This defines a basic config structure for VaderSentimentProcessor.

Returns A dictionary with the default config for this processor.

Following are the keys for this dictionary:

- ***"entry_type"*:** Defines which entry type in the input pack to make prediction on. The default makes prediction on each *Sentence* in the input pack.
- ***"attribute_name"*:** Defines which attribute of the *entry_type* in the input pack to save score to. The default saves prediction to the *sentiment* attribute for each *Sentence* in the input pack.
- ***"sentence_component"*:** str. If not None, the processor will process sentence with the provided component name. If None, then all sentences will be processed.

expected_types_and_attributes()

Method to add expected type *ft.onto.base_ontology.Sentence* which would be checked before running the processor if the pipeline is initialized with *enforce_consistency=True* or `enforce_consistency()` was enabled for the pipeline.

INDEX