medaCy Documentation

Release 0.6

Andriy Mulyar, Corey Sutphin, Bobby Best, Steele Farnsworth, Bri

Feb 19, 2019

CONTENTS

[Info about what MedaCy is here]

CHAPTER

ONE

CONTENTS

1.1 medacy package

1.1.1 medacy.data package

medacy.data.dataset module

A Dataset facilities the management of data for both model training and model prediction. A Dataset object provides a wrapper for a unix file directory containing training/prediction data. If a Dataset, at training time, is fed into a pipeline requiring auxilary files (Metamap for instance) the Dataset will automatically create those files in the most efficient way possible.

Training When a directory contains **both** raw text files alongside annotation files, an instantiated Dataset detects and facilitates access to those files.

Prediction When a directory contains **only** raw text files, an instantiated Dataset object interprets this as a directory of files that need to be predicted. This means that the internal Datafile that aggregates meta-data for a given prediction file does not have fields for annotation_file_path set.

External Datasets An actual dataset can be versioned and distributed by interfacing this class as described in the Dataset examples. Existing Datasets can be imported by installing the relevant python packages that wrap them.

class medacy.data.dataset.**Dataset**(*data_directory*, *raw_text_file_extension='txt'*, *annotation_file_extension='ann'*, *metamapped_files_directory=None*, *data_limit=None*)

Bases: object

A facilitation class for data management.

_parallel_metamap(*files*, *i*)

Facilitates Metamapping in parallel by forking off processes to Metamap each file individually. :param files: an array of file paths to the file to map :param i: index in the array used to determine the file that the called process will be responsible for mapping :return: metamapped_files_directory now contains metamapped versions of the dataset files

get_data_directory()

Retrieves the directory this Dataset abstracts from. :return:

get_data_files()

Retrieves an list containing all the files registered by a Dataset. :return: a list of DataFile objects.

is_metamapped()

Verifies if all fil es in the Dataset are metamapped. :return: True if all data files are metamapped, False otherwise.

is_training()
Whether this Dataset can be used for training. :return: True if training dataset, false otherwise. A training dataset is a collection raw text and corresponding annotation files while a prediction dataset contains solely raw text files.

static load_external(*package_name*)
Loads an external medaCy compatible dataset. Requires the dataset's associated package to be installed. Alternatively, you can import the package directly and call it's .load() method. :param package_name: the package name of the dataset :return: A tuple containing a training set, evaluation set, and meta_data

metamap(*metamap*, *n_jobs=3*, *retry_possible_corruptions=True*)
Metamaps the files registered by a Dataset. Attempts to Metamap utilizing a max prune depth of 30, but on failure retries with lower max prune depth. A lower prune depth roughly equates to decreased MetaMap performance. More information can be found in the MetaMap documentation. :param metamap: an instance of MetaMap. :param n_jobs: the number of processes to spawn when metamapping. Defaults to one less core than available on your machine. :param retry_possible_corruptions: Re-Metamap's files that are detected as being possibly corrupt. Set to False for more control over what gets Metamapped or if you are having bugs with Metamapping. (default: True) :return: Inside *metamapped_files_directory* or by default inside a sub-directory of your *data_directory* named *metamapped* we have that for each raw text file, *file_name*, an auxiliary metamapped version is created and stored.

set_data_limit(*data_limit*)
A limit to the number of files in the Dataset that medaCy works with This is useful for preliminary experimentation when working with an entire Dataset takes time. :return:

1.1.2 medacy.model package

medacy.model.feature_extractor module

class medacy.model.feature_extractor.**FeatureExtractor**(*window_size=2*, *spacy_features=['pos_', 'shape_', 'prefix_', 'suffix_', 'like_num']*)

Bases: object

Extracting training data for use in a CRF. Features are given as rich dictionaries as described in: https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html#features

sklearn CRF suite is a wrapper for CRF suite that gives it a sci-kit compatability.

_sent_to_feature_dicts(*sent*)

_sent_to_labels(*sent*, *attribute='gold_label'*)

_token_to_feature_dict(*index*, *sentence*)

Parameters

- **index** – the index of the token in the sequence
- **sentence** – an array of tokens corresponding to a sequence

Returns

get_features_with_span_indices(*doc*)
Given a document this method orchestrates the organization of features and labels for the sequences to classify. Sequences for classification are determined by the sentence boundaries set by spaCy. These can be modified. :param doc: an annoted spacy Doc object :return: Tuple of parallel arrays - 'features' an array of feature dictionaries for each sequence (spaCy determined sentence) and 'indices' which are arrays of character offsets corresponding to each extracted sequence of features.

mapper_for_crf_wrapper(*text*)
CURRENTLY UNUSED. CRF wrapper uses regexes to extract the output of the underlying C++ code. The inclusion of n and space characters mess up these regexes, hence we map them to text here. :return:

medacy.model.model module

A medaCy named entity recognition model wraps together three functionalities

class medacy.model.model.**Model**(*medacy_pipeline=None*, *model=None*, *n_jobs=4*)
Bases: object

_extract_features(*data_file*, *medacy_pipeline*, *is_metamapped*)
A multi-processed method for extracting features from a given DataFile instance. :param conn: pipe to pass back data to parent process :param data_file: an instance of DataFile :return: Updates queue with features for this given file.

cross_validate(*num_folds=10*)
Performs k-fold stratified cross-validation using our model and pipeline. :param num_folds: number of folds to split training data into for cross validation :return: Prints out performance metrics

dump(*path*)
Dumps a model into a pickle file :param path: Directory path to dump the model :return:

fit(*dataset*)
Runs dataset through the designated pipeline, extracts features, and fits a conditional random field. :param training_data_loader: Instance of Dataset. :return model: a trained instance of a sklearn_crfsuite.CRF model.

get_info(*return_dict=False*)
Retrieves information about a Model including details about the feature extraction pipeline, features utilized, and learning model. :param return_dict: Returns a raw dictionary of information as opposed to a formatted string :return: Returns structured information

load(*path*)
Loads a pickled model. :param path: File path to directory where fitted model should be dumped :return:

static load_external(*package_name*)
Loads an external medaCy compatible Model. Require's the models package to be installed Alternatively, you can import the package directly and call it's .load() method. :param package_name: the package name of the model :return: an instance of Model that is configured and loaded - ready for prediction.

predict(*dataset*, *prediction_directory=None*)

Parameters

- **documents** – a string or Dataset to predict
- **prediction_directory** – the directory to write predictions if doing bulk prediction (default: */prediction* sub-directory of Dataset)

Returns

medacy.model.stratified_k_fold module

Partitions a data set of sequence labels and classifications into 10 stratified folds. See Dietterich, 1997 "Approximate Statistical Tests for Comparing Supervised Classification Algorithms" for in-depth analysis.

Each partition should have an evenly distributed representation of sequence labels. Without stratification, under-representated labels may not appear in some folds.

class medacy.model.stratified_k_fold.**SequenceStratifiedKFold**(*folds=10*)
Bases: object

1.1.3 medacy.pipeline_components package

medacy.pipeline_components.annotation package

medacy.pipeline_components.annotation.gold_annotator_component module

class medacy.pipeline_components.annotation.gold_annotator_component.**GoldAnnotatorComponent**

Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that overlays gold annotations. This pipeline component sets the attribute 'gold_label' to all tokens to be used as the class value of the token when fed into a supervised learning algorithm.

_abc_impl = <_abc_data object>

dependencies = []

find_span(*start*, *end*, *label*, *span*, *doc*)
Greedily searches characters around word to find a valid set of tokens the annotation likely corresponds to. :param start: index of token start :param end: index of token end :param label: :param span: :param doc: :return:

name = 'gold_annotator'

medacy.pipeline_components.base package

medacy.pipeline_components.base.base_component module

class medacy.pipeline_components.base.base_component.**BaseComponent**(*component_name='DEFAULT_CO dependencies=[]*)

Bases: abc.ABC

A base medacy pipeline component that wraps over a spacy component

_abc_impl = <_abc_data object>

get_component_dependencies()
Retrieves a list of dependencies this component has. :return: a list of component dependencies

get_component_name()

medacy.pipeline_components.lexicon package

medacy.pipeline_components.lexicon.lexicon_component module

class medacy.pipeline_components.lexicon.lexicon_component.**LexiconComponent**(*spacy_pipeline*, *lexicon*)

Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

```
_abc_impl = <_abc_data object>

dependencies = []

name = 'lexicon_component'
```

medacy.pipeline_components.metamap package

medacy.pipeline_components.metamap.metamap module

A utility class to Metamap medical text documents. Metamap a file and utilize it the output or manipulate stored metamap output

class medacy.pipeline_components.metamap.metamap.**MetaMap**(*metamap_path=None*, *cache_output=False*, *cache_directory=None*, *convert_ascii=True*)

Bases: object

_convert_to_ascii(*text*)

Takes in a text string and converts it to ASCII, keeping track of each character change

The changes are recorded in a list of objects, each object detailing the original non-ASCII character and the starting index and length of the replacement in the new string (keys `original`, `start`, and `length`, respectively).

Parameters **text** (*string*) – The text to be converted

Returns

tuple containing:

text (*string*): The converted text

diff (*list*): Record of all ASCII conversions

Return type tuple

_item_generator(*json_input*, *lookup_key*)

_restore_from_ascii(*text*, *diff*, *metamap_dict*)

Takes in non-ascii text and the list of changes made to it from the *convert()* function, as well as a dictionary of metamap taggings, converts the text back to its original state and updates the character spans in the metamap dict to match

Parameters

- **text** (*string*) – Output of `_convert_to_ascii()`
- **diff** (*list*) – Output of `_convert_to_ascii()`
- **metamap_dict** (*dict*) – Dictionary of metamap information obtained from `text`

Returns

tuple containing:

text (*string*): The input with all of the changes listed in `diff` reversed **metamap_dict** (*dict*): The input with all of its character spans updated to reflect the changes to `text`

Return type tuple

_run_metamap(*args*, *document*)
Runs metamap through bash and feeds in appropriate arguments :param args: arguments to feed into metamap :param document: the raw text to be metamapped :return:

extract_mapped_terms(*metamap_dict*)
Extracts an array of term dictionaries from metamap_dict :param metamap_dict: A dictionary containing the metamap output :return: an array of mapped_terms

get_semantic_types_by_term(*term*)
Returns an array of the semantic types of a given term :param term: :return:

get_span_by_term(*term*)
Takes a given utterance dictionary (term) and extracts out the character indices of the utterance

Parameters **term** – The full dictionary corresponding to a metamap term

Returns the span of the referenced term in the document

get_term_by_semantic_type(*mapped_terms*, *include=[]*, *exclude=None*)
Returns Metamapped utterances that all contain a given set of semantic types found in include

Parameters **mapped_terms** – An array of candidate dictionaries

Returns the dictionaries that contain a term with all the semantic types in semantic_types

load(*file_to_load*)

map_corpus(*documents*, *directory=None*, *n_job=-1*)
Metamaps a large amount of files quickly by forking processes and utilizing multiple cores

Parameters

- **documents** – an array of documents to map
- **directory** – location to map all files
- **n_job** – number of cores to utilize at once while mapping - this may use a large amount of memory

Returns

map_file(*file_to_map*, *max_prune_depth=10*)
Maps a given document from a file_path and returns a formatted dict :param file_to_map: the path of the file that will be metamapped :param max_prune_depth: set to larger for better results. See metamap specs about pruning depth. :return:

map_text(*text*, *max_prune_depth=10*)

mapped_terms_to_spacy_ann(*mapped_terms*, *entity_label=None*)
Transforms an array of mapped_terms in a spacy annotation object. Label for each annotation defaults to first semantic type in semantic_type array :param mapped_terms: an array of mapped terms :param label: the label to assign to each annotation, defaults to first semantic type of mapped_term :return: a annotation formatted to spacy's specifications

medacy.pipeline_components.metamap.metamap_component module

class medacy.pipeline_components.metamap.metamap_component.**MetaMapComponent**(*spacy_pipeline, metamap, cuis=True, semantic_type_labels=['orc 'phsu'], merge_tokens=False*)

Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component for SpaCy that overlays Metamap output as token attributes

_abc_impl = <_abc_data object>

dependencies = []

name = 'metamap_annotator'

medacy.pipeline_components.tokenization package

medacy.pipeline_components.tokenization.character_tokenizer module

class medacy.pipeline_components.tokenization.character_tokenizer.**CharacterTokenizer**(*nlp*)

Bases: object

A tokenizer that tokenizes on every character

_get_infix_regex()

Custom infix tokenization rules :return:

_get_prefix_regex()

Custom prefix tokenization rules :return:

_get_suffix_regex()

Custom suffix tokenization rules :return:

add_exceptions(*exceptions*)

Adds exception for tokenizer to ignore. :param exceptions: an array of terms to not split on during tokenization :return:

medacy.pipeline_components.tokenization.clinical_tokenizer module

class medacy.pipeline_components.tokenization.clinical_tokenizer.**ClinicalTokenizer**(*nlp*)

Bases: object

A tokenizer for clinical text

_get_infix_regex()

Custom infix tokenization rules :return:

_get_prefix_regex()

Custom prefix tokenization rules :return:

_get_suffix_regex()

Custom suffix tokenization rules :return:

add_exceptions(*exceptions*)
Adds exception for tokenizer to ignore. :param exceptions: an array of terms to not split on during tokenization :return:

medacy.pipeline_components.tokenization.systematic_review_tokenizer module

class medacy.pipeline_components.tokenization.systematic_review_tokenizer.**SystematicReview**
Bases: object

A tokenizer for clinical text

_get_infix_regex()
Custom infix tokenization rules :return:

_get_prefix_regex()
Custom prefix tokenization rules :return:

_get_suffix_regex()
Custom suffix tokenization rules :return:

add_exceptions(*exceptions*)
Adds exception for tokenizer to ignore. :param exceptions: an array of terms to not split on during tokenization :return:

medacy.pipeline_components.units package

medacy.pipeline_components.units.frequency_unit_component module

class medacy.pipeline_components.units.frequency_unit_component.**FrequencyUnitComponent**(*spacy*
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags Frequency units

_abc_impl = <_abc_data object>

dependencies = []

name = 'frequency_unit_annotator'

medacy.pipeline_components.units.mass_unit_component module

class medacy.pipeline_components.units.mass_unit_component.**MassUnitComponent**(*spacy_pipeline*)
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags mass units

_abc_impl = <_abc_data object>

dependencies = []

name = 'mass_unit_annotator'

medacy.pipeline_components.units.measurement_unit_component module

class medacy.pipeline_components.units.measurement_unit_component.**MeasurementUnitComponent**
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags Frequency units

_abc_impl = <_abc_data object>

dependencies = [<class 'medacy.pipeline_components.units.mass_unit_component.MassUnitC

name = 'measurement_unit_annotator'

medacy.pipeline_components.units.route_unit_component module

medacy.pipeline_components.units.time_unit_component module

class medacy.pipeline_components.units.time_unit_component.**TimeUnitComponent**(*spacy_pipeline*)
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags time units

_abc_impl = <_abc_data object>

dependencies = []

name = 'time_unit_annotator'

medacy.pipeline_components.units.unit_component module

class medacy.pipeline_components.units.unit_component.**UnitComponent**(*nlp*)
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags units. Begins by first tagging all mass, volume, time, and form units then aggregates as necessary.

_abc_impl = <_abc_data object>

dependencies = []

name = 'unit_annotator'

medacy.pipeline_components.units.volume_unit_component module

class medacy.pipeline_components.units.volume_unit_component.**VolumeUnitComponent**(*spacy_pipelin*
Bases: *medacy.pipeline_components.base.base_component.BaseComponent*

A pipeline component that tags volume units

_abc_impl = <_abc_data object>

dependencies = []

name = 'volume_unit_annotator'

1.1.4 medacy.pipelines package

medacy.pipelines.base package

medacy.pipelines.base.base_pipeline module

class medacy.pipelines.base.base_pipeline.**BasePipeline**(*pipeline_name*, *spacy_pipeline=None*, *description=None*, *creators=''*, *organization=''*)

Bases: abc.ABC

An abstract wrapper for a Medical NER Pipeline

_abc_impl = <_abc_data object>

add_component(*component*, **argv*, ***kwargs*)
Adds a given component to pipeline :param component: a subclass of BaseComponent

get_components()
Retrieves a listing of all components currently in the pipeline. :return: a list of components inside the pipeline.

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_language_pipeline()
Retrieves the associated spaCy Language pipeline that the medaCy pipeline wraps. :return: spacy_pipeline

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_pipeline_information()
Retrieves information about the current pipeline in a structured dictionary :return: a json dictionary containing information

get_tokenizer()
Returns an instance of a tokenizer :return:

medacy.pipelines.clinical_pipeline module

class medacy.pipelines.clinical_pipeline.**ClinicalPipeline**(*metamap=None*, *entities=[]*)

Bases: *medacy.pipelines.base.base_pipeline.BasePipeline*

A pipeline for clinical named entity recognition. A special tokenizer that breaks down a clinical document to character level tokens defines this pipeline.

_abc_impl = <_abc_data object>

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_tokenizer()
Returns an instance of a tokenizer :return:

medacy.pipelines.drug_event_pipeline module

class medacy.pipelines.drug_event_pipeline.**DrugEventPipeline**(*metamap=None*, *entities=[]*, *lexicon={}*)

Bases: *medacy.pipelines.base.base_pipeline.BasePipeline*

_abc_impl = <_abc_data object>

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_tokenizer()
Returns an instance of a tokenizer :return:

medacy.pipelines.fda_nano_drug_label_pipeline module

class medacy.pipelines.fda_nano_drug_label_pipeline.**FDANanoDrugLabelPipeline**(*metamap*, *entities=[]*)

Bases: *medacy.pipelines.base.base_pipeline.BasePipeline*

A pipeline for clinical named entity recognition. This pipeline was designed over-top of the TAC 2018 SRIE track challenge.

_abc_impl = <_abc_data object>

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_tokenizer()
Returns an instance of a tokenizer :return:

medacy.pipelines.systematic_review_pipeline module

class medacy.pipelines.systematic_review_pipeline.**SystematicReviewPipeline**(*metamap=None*, *entities=[]*)

Bases: *medacy.pipelines.base.base_pipeline.BasePipeline*

A pipeline for clinical named entity recognition. This pipeline was designed over-top of the TAC 2018 SRIE track challenge.

_abc_impl = <_abc_data object>

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_tokenizer()
Returns an instance of a tokenizer :return:

medacy.pipelines.testing_pipeline module

class medacy.pipelines.testing_pipeline.**TestingPipeline**(*entities=[]*)
Bases: *medacy.pipelines.base.base_pipeline.BasePipeline*

A pipeline for test running

_abc_impl = <_abc_data object>

get_feature_extractor()
Returns an instant of FeatureExtractor with all configs set. :return: An instant of FeatureExtractor

get_learner()
Retrieves an instance of a sci-kit learn compatible learning algorithm. :return: model

get_tokenizer()
Returns an instance of a tokenizer :return:

1.1.5 medacy.tools package

medacy.tools.con package

medacy.tools.con.brat_to_con module

Converts data from brat to con. Enter input and output directories as command line arguments. Each '.ann' file must have a '.txt' file in the same directory with the same name, minus the extension. Use '-c' (without quotes) as an optional final command-line argument to copy the text files used in the conversion process to the output directory.

Also possible to import 'convert_brat_to_con()' directly and pass the paths to the ann and txt files for individual conversion.

author Steele W. Farnsworth

date 30 December, 2018

medacy.tools.con.brat_to_con.**check_valid_line**(*item: str*)
Returns a boolean value for whether or not a given line is in the BRAT format. Tests are not comprehensive.

medacy.tools.con.brat_to_con.**convert_brat_to_con**(*brat_file_path*, *text_file_path=None*)
Takes a path to a brat file and returns a string representation of that file converted to the con format. :param brat_file_path: The path to the brat file; not the file itself. If the path is not valid, the argument

will be assumed to be text of the brat file itself.

Parameters **text_file_path** – The path to the text file; if not provided, assumed to be a file with the same path as the brat file ending in '.txt' instead of '.ann'. If neither file is found, raises error.

Returns A string (not a file) of the con equivalent of the brat file.

medacy.tools.con.brat_to_con.**find_line_num**(*text_*, *start*)

Parameters

- **text** – The text of the file, ex. f.read()

- **start** – The index at which the desired text starts

Returns The line index (starting at 0) containing the given start index

medacy.tools.con.brat_to_con.**get_end_word_index**(*data_item: str*, *start_index*, *end_index*)

Returns the index of the first char of the last word of **data_item_**; all parameters shadow the appropriate name in the final for loop

medacy.tools.con.brat_to_con.**get_relative_index**(*text_: str*, *line_*, *absolute_index*)

Takes the index of a phrase (the phrase itself is not a parameter) relative to the start of its file and returns its index relative to the start of the line that it's on. Assumes that the **line_** argument is long enough that (and thus so specific that) it only occurs once. :param **text_**: The text of the file, not separated by lines :param **line_**: The text of the line being searched for :param absolute_index: The index of a given phrase :return: The index of the phrase relative to the start of the line

medacy.tools.con.brat_to_con.**line_to_dict**(*item*)

Converts a string that is a line in brat format to a dictionary representation of that data. Keys are: T; data_type; start_ind; end_ind; data_type. :param item: The line of con text (str). :return: The dictionary containing that data.

medacy.tools.con.brat_to_con.**switch_extension**(*name*, *ext*)

Primarily for internal use. Takes the name of a file (str) and changes the extension to the one provided (str)

medacy.tools.con.con_to_brat module

Converts data from con to brat. Enter input and output directories as command line arguments. Each '.con' file must have a '.txt' file in the same directory with the same name, minus the extension. Use '-c' (without quotes) as an optional final command-line argument to copy the text files used in the conversion process to the output directory.

Function 'convert_con_to_brat()' can be imported independently and run on individual files.

author Steele W. Farnsworth

date 30 December, 2018

medacy.tools.con.con_to_brat.**check_valid_line**(*item: str*)

Non-comprehensive tests to see if a given line is valid for conversion. Returns respective boolean value. :param item: A string that is a line of text, hopefully in the con format. :return: Boolean of whether or not the line appears to be in con format.

medacy.tools.con.con_to_brat.**convert_con_to_brat**(*con_file_path*, *text_file_path=None*)

Converts a con file to a string representation of a brat file. :param con_file_path: Path to the con file being converted. If a valid path is not provided but the argument is a

string, it will be parsed as if it were a representation of the con file itself.

Parameters **text_file_path** – Path to the text file associated with the con file. If not provided, the function will look for a text file in the same directory with the same name except for the extention switched to 'txt'. Else, raises error. Note that no conversion can be performed without the text file.

Returns A string representation of the brat file, which can then be written to file if desired.

medacy.tools.con.con_to_brat.**get_absolute_index**(*txt*, *txt_lns*, *ind*)

Given one of the d+:d+ spans, which represent the index of a char relative to the start of the line it's on, returns the index of that char relative to the start of the file. :param txt: The text file associated with the annotation. :param txt_lns: The same text file as a list broken by lines :param ind: The string in format d+:d+ :return: The absolute index

medacy.tools.con.con_to_brat.**line_to_dict**(*item*)
Converts a string that is a line in con format to a dictionary representation of that data. Keys are: data_item; start_ind; end_ind; data_type. :param item: The line of con text (str). :return: The dictionary containing that data.

medacy.tools.con.con_to_brat.**switch_extension**(*name*, *ext*)
Takes the name of a file (str) and changes the extension to the one provided (str)

medacy.tools.ade_to_brat module

medacy.tools.annotations module

author Andriy Mulyar, Steele W. Farnsworth

date 12 January, 2019

class medacy.tools.annotations.**Annotations**(*annotation_data*, *annotation_type='ann'*, *source_text_path=None*)
Bases: object

A medaCy annotation. This stores all relevant information needed as input to medaCy or as output. The Annotation object is utilized by medaCy to structure input to models and output from models. This object wraps a dictionary containing two keys at the root level: 'entities' and 'relations'. This structured dictionary is designed to interface easily with the BRAT ANN format. The key 'entities' contains as a value a dictionary with keys T1, T2, ... ,TN corresponding each to a single entity. The key 'relations' contains a list of tuple relations where the first element of each tuple is the relation type and the last two elements correspond to keys in the 'entities' dictionary.

_Annotations__default_strict = 0.2

compare_by_entity(*gold_anno*)
Compares two Annotations for checking if an unverified annotation matches an accurate one by creating a data structure that looks like this:

{

'females': { 'this_anno': [('Sex', 1396, 1403), ('Sex', 295, 302), ('Sex', 3205, 3212)], 'gold_anno': [('Sex', 1396, 1403), ('Sex', 4358, 4365), ('Sex', 263, 270)] }

'SALDOX': { 'this_anno': [('GroupName', 5408, 5414)], 'gold_anno': [('TestArticle', 5406, 5412)] }

'MISSED_BY_PREDICTION': [('GroupName', 8644, 8660, 'per animal group'), ('CellLine', 1951, 1968, 'on control diet (')]

}

The object itself should be the predicted Annotations and the argument should be the gold Annotations.

Parameters **gold_anno** – the Annotations object for the gold data.

Returns The data structure detailed above.

compare_by_index(*gold_anno*, *strict=0.2*)
Similar to compare_by_entity, but organized by start index. The two data sets used in the comparison will often not have two annotations beginning at the same index, so the strict value is used to calculate within what margin a matched pair can be separated. :param gold_anno: The Annotation object representing an annotation set that is known to be accurate. :param strict: Used to calculate within what range a possible match can be. The length of the entity is

multiplied by this number, and the product of those two numbers is the difference that the entity can begin or end relative to the starting index of the entity in the gold dataset. Default is 0.2.

Returns

compare_by_index_stats(*gold_anno*, *strict=0.2*)
Runs compare_by_index() and returns a dict of related statistics. :param gold_anno: See compare_by_index() :param strict: See compare_by_index() :return: A dictionary with keys:

"num_not_matched": The number of entites in the predicted data that are not matched to an entity in the gold data,

"avg_accuracy": The average of all the decimal values representing how close to a 1:1 correlation there was between the start and end indices in the gold and predicted data.

diff(*other_anno*)
Identifies the difference between two Annotations objects. Useful for checking if an unverified annotation matches an annotation known to be accurate. :param other_anno: Another Annotations object. :return: A list of tuples of non-matching annotation pairs.

from_ann(*ann_file_path*)
Loads an ANN file given by ann_file :param ann_file_path: the system path to the ann_file to load :return: annotations object is loaded with the ann file.

from_con(*con_file_path*)
Converts a con file from a given path to an Annotations object. The conversion takes place through the from_ann() method in this class because the indices for the Annotations object must be those used in the BRAT format. The path to the source text for the annotations must be defined unless that file exists in the same directory as the con file. :param con_file_path: path to the con file being converted to an Annotations object.

get_entity_annotations(*return_dictionary=False*)
Returns a list of entity annotation tuples :param return_dictionary: returns the dictionary storing the annotation mappings. Useful if also working with relationship extraction :return: a list of entities or underlying dictionary of entities

get_entity_count()

stats()
Count the number of instances of a given entity type and the number of unique entities. :return: a dict with keys:

"entity_counts": a dict matching entities to the number of times that entity appears in the Annotations,

"unique_entity_num": an int of how many unique entities are in the Annotations, "entity_list": a list of all the entities that appear in the list; each only appears once.

to_ann(*write_location=None*)
Formats the Annotations object into a string representing a valid ANN file. Optionally writes the formatted string to a destination. :param write_location: path of location to write ann file to :return: returns string formatted as an ann file, if write_location is valid path also writes to that path.

to_con(*write_location=None*)
Formats the Annotation object to a valid con file. Optionally writes the string to a specified location. :param write_location: Optional path to an output file; if provided but not an existing file, will be

created. If this parameter is not provided, nothing will be written to file.

Returns A string representation of the annotations in the con format.

to_html(*output_file_path*, *title='medaCy'*)
Convert the Annotations to a displaCy-formatted HTML representation. The Annotations must have the path to the source file as one of its attributes. Does not return a value. :param output_file_path: Where to write the HTML to. :param title: What should appear in the header of the outputted HTML file; not very important

exception medacy.tools.annotations.**InvalidAnnotationError**
Bases: ValueError

Raised when a given input is not in the valid format for that annotation type.

medacy.tools.data_file module

class medacy.tools.data_file.**DataFile**(*file_name*, *raw_text_file_path*, *annotation_file_path*, *metamapped_path=None*)
Bases: object

DataFile wraps all relevent information needed to manage a text document and it's corresponding annotation

medacy.tools.unicode_to_ascii module

Extracted from the UTF-8 to ASCII conversion program found at https://metamap.nlm.nih.gov/ReplaceUTF8.shtml

PYTHON MODULE INDEX

m

INDEX

Symbols

T

U

V