EpiTator Documentation

Release 1.3.3

EcoHealth Alliance

Jun 18, 2019

Contents

Annotators for extracting epidemiological information from text.

CHAPTER 1

Installation

```
pip install epitator
python -m spacy download en_core_web_md
```

CHAPTER 2

Annotators

2.1 Geoname Annotator

The geoname annotator uses the geonames.org dataset to resolve mentions of geonames. A classifier is used to disambiguate geonames and rule out false positives.

To use the geoname annotator run the following command to import geonames.org data into epitator's embedded sqlite3 database:

You should review the geonames license before using this data.

```
python -m epitator.importers.import_geonames
```

2.2 Usage

```
from epitator.annotator import AnnoDoc
from epitator.geoname_annotator import GeonameAnnotator
doc = AnnoDoc("Where is Chiang Mai?")
doc.add_tiers(GeonameAnnotator())
annotations = doc.tiers["geonames"].spans
geoname = annotations[0].geoname
geoname['name']
# = 'Chiang Mai'
geoname['geonameid']
# = '1153671'
geoname['latitude']
# = 18.79038
geoname['longitude']
# = 98.98468
```

2.3 Resolved Keyword Annotator

The resolved keyword annotator uses an sqlite database of entities to resolve mentions of multiple synonyms for an entity to a single id. This project includes scripts for importing infectious diseases and animal species into that database. The following commands can be used to invoke them:

The scripts import data from the Disease Ontology, Wikidata and ITIS. You should review their licenses and terms of use before using this data. Currently the Disease Ontology is under public domain and ITIS requests citation.

```
python -m epitator.importers.import_species
# By default entities under the disease by infectious agent class will be
# imported from the disease ontology, but this can be altered by supplying
# a --root-uri parameter.
python -m epitator.importers.import_disease_ontology
python -m epitator.importers.import_wikidata
```

2.4 Usage

```
from epitator.annotator import AnnoDoc
from epitator.resolved_keyword_annotator import ResolvedKeywordAnnotator
doc = AnnoDoc("5 cases of smallpox")
doc.add_tiers(ResolvedKeywordAnnotator())
annotations = doc.tiers["resolved_keywords"].spans
annotations[0].metadata["resolutions"]
# = [{'entity': <sqlite3.Row>, 'entity_id': u'http://purl.obolibrary.org/obo/DOID_8736
↪', 'weight': 3}]
```

2.5 Count Annotator

The count annotator identifies counts, and case counts in particular. The count's value is extracted and parsed. Attributes such as whether the count refers to cases or deaths, or whether the value is approximate are also extracted.

2.6 Usage

```
from epitator.annotator import AnnoDoc
from epitator.count_annotator import CountAnnotator
doc = AnnoDoc("5 cases of smallpox")
doc.add_tiers(CountAnnotator())
annotations = doc.tiers["counts"].spans
annotations[0].metadata
# = {'count': 5, 'text': '5 cases', 'attributes': ['case']}
```

2.7 Date Annotator

The date annotator identifies and parses dates and date ranges. All dates are parsed into datetime ranges. For instance, a date like "11-6-87" would be parsed as a range from the start of the day to the start of the next day, while a month like "December 2011" would be parsed as a range from the start of December 1st to the start of the next month.

2.8 Usage

```
from epitator.annotator import AnnoDoc
from epitator.date_annotator import DateAnnotator
doc = AnnoDoc("From March 5 until April 7 1988")
doc.add_tiers(DateAnnotator())
annotations = doc.tiers["dates"].spans
annotations[0].metadata["datetime_range"]
# = [datetime.datetime(1988, 3, 5, 0, 0), datetime.datetime(1988, 4, 7, 0, 0)]
```

2.9 Structured Data Annotator

The structured data annotator identifies and parses embedded tables.

2.10 Usage

```
from epitator.annotator import AnnoDoc
from epitator.structured_data_annotator import StructuredDataAnnotator
doc = AnnoDoc("""
species | cases | deaths
Cattle  | 0     | 0
Dogs    | 2     | 1
""")
doc.add_tiers(StructuredDataAnnotator())
annotations = doc.tiers["structured_data"].spans
annotations[0].metadata
# = {'data': [
#       [AnnoSpan(1-8, species), AnnoSpan(11-16, cases), AnnoSpan(19-25, deaths)],
#       [AnnoSpan(26-32, Cattle), AnnoSpan(36-37, 0), AnnoSpan(44-45, 0)],
#       [AnnoSpan(46-50, Dogs), AnnoSpan(56-57, 2), AnnoSpan(64-65, 1)]],
#    'delimiter': '|',
#    'type': 'table'}
```

2.11 Structured Incident Annotator

The structured incident annotator identifies and parses embedded tables that describe case counts paired with location, date, disease and species metadata. Metadata is also extracted from the text around the table.

2.12 Usage

```
from epitator.annotator import AnnoDoc
from epitator.structured_incident_annotator import StructuredIncidentAnnotator
doc = AnnoDoc("""
Fictional October 2015 rabies cases in Svalbard

species | cases | deaths
Cattle  | 0     | 0
```

(continues on next page)

(continued from previous page)

```
Dogs    | 4     | 1
""")
doc.add_tiers(StructuredIncidentAnnotator())
annotations = doc.tiers["structured_incidents"].spans
annotations[-1].metadata
# = {'location': {'name': u'Svalbard', ...},
#    'species': {'label': u'Canidae', ...},
#    'attributes': [],
#    'dateRange': [datetime.datetime(2015, 10, 1, 0, 0), datetime.datetime(2015, 11,␣
↪1, 0, 0)],
#    'type': 'deathCount',
#    'value': 1,
#    'resolvedDisease': {'label': u'rabies', ...}}
```

CHAPTER 3

Architecture

EpiTator provides the following classes for organizing annotations.

AnnoDoc - The document being annotated. The AnnoDoc links to the tiers of annotations applied to it.

AnnoTier - A group of AnnoSpans. Each annotator creates one or more tiers of annotations.

AnnoSpan - A span of text with an annotation applied to it.

CHAPTER 4

License

Copyright 2016 EcoHealth Alliance

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER 5

Navigation

5.1 Getting Started

Annotators for extracting epidemiological information from text.

5.1.1 Installation

```
pip install epitator
python -m spacy download en_core_web_md
```

5.1.2 Annotators

Geoname Annotator

The geoname annotator uses the geonames.org dataset to resolve mentions of geonames. A classifier is used to disambiguate geonames and rule out false positives.

To use the geoname annotator run the following command to import geonames.org data into epitator's embedded sqlite3 database:

You should review the geonames license before using this data.

```
python -m epitator.importers.import_geonames
```

Usage

```
from epitator.annotator import AnnoDoc
from epitator.geoname_annotator import GeonameAnnotator
doc = AnnoDoc("Where is Chiang Mai?")
```

(continues on next page)

(continued from previous page)

```
doc.add_tiers(GeonameAnnotator())
annotations = doc.tiers["geonames"].spans
geoname = annotations[0].geoname
geoname['name']
# = 'Chiang Mai'
geoname['geonameid']
# = '1153671'
geoname['latitude']
# = 18.79038
geoname['longitude']
# = 98.98468
```

Resolved Keyword Annotator

The resolved keyword annotator uses an sqlite database of entities to resolve mentions of multiple synonyms for an entity to a single id. This project includes scripts for importing infectious diseases and animal species into that database. The following commands can be used to invoke them:

The scripts import data from the Disease Ontology, Wikidata and ITIS. You should review their licenses and terms of use before using this data. Currently the Disease Ontology is under public domain and ITIS requests citation.

```
python -m epitator.importers.import_species
# By default entities under the disease by infectious agent class will be
# imported from the disease ontology, but this can be altered by supplying
# a --root-uri parameter.
python -m epitator.importers.import_disease_ontology
python -m epitator.importers.import_wikidata
```

Usage

```
from epitator.annotator import AnnoDoc
from epitator.resolved_keyword_annotator import ResolvedKeywordAnnotator
doc = AnnoDoc("5 cases of smallpox")
doc.add_tiers(ResolvedKeywordAnnotator())
annotations = doc.tiers["resolved_keywords"].spans
annotations[0].metadata["resolutions"]
# = [{'entity': <sqlite3.Row>, 'entity_id': u'http://purl.obolibrary.org/obo/DOID_8736
↪', 'weight': 3}]
```

Count Annotator

The count annotator identifies counts, and case counts in particular. The count's value is extracted and parsed. Attributes such as whether the count refers to cases or deaths, or whether the value is approximate are also extracted.

Usage

```
from epitator.annotator import AnnoDoc
from epitator.count_annotator import CountAnnotator
doc = AnnoDoc("5 cases of smallpox")
doc.add_tiers(CountAnnotator())
```

(continues on next page)

(continued from previous page)

```
annotations = doc.tiers["counts"].spans
annotations[0].metadata
# = {'count': 5, 'text': '5 cases', 'attributes': ['case']}
```

Date Annotator

The date annotator identifies and parses dates and date ranges. All dates are parsed into datetime ranges. For instance, a date like "11-6-87" would be parsed as a range from the start of the day to the start of the next day, while a month like "December 2011" would be parsed as a range from the start of December 1st to the start of the next month.

Usage

```
from epitator.annotator import AnnoDoc
from epitator.date_annotator import DateAnnotator
doc = AnnoDoc("From March 5 until April 7 1988")
doc.add_tiers(DateAnnotator())
annotations = doc.tiers["dates"].spans
annotations[0].metadata["datetime_range"]
# = [datetime.datetime(1988, 3, 5, 0, 0), datetime.datetime(1988, 4, 7, 0, 0)]
```

Structured Data Annotator

The structured data annotator identifies and parses embedded tables.

Usage

```
from epitator.annotator import AnnoDoc
from epitator.structured_data_annotator import StructuredDataAnnotator
doc = AnnoDoc("""
species | cases | deaths
Cattle  | 0     | 0
Dogs    | 2     | 1
""")
doc.add_tiers(StructuredDataAnnotator())
annotations = doc.tiers["structured_data"].spans
annotations[0].metadata
# = {'data': [
#       [AnnoSpan(1-8, species), AnnoSpan(11-16, cases), AnnoSpan(19-25, deaths)],
#       [AnnoSpan(26-32, Cattle), AnnoSpan(36-37, 0), AnnoSpan(44-45, 0)],
#       [AnnoSpan(46-50, Dogs), AnnoSpan(56-57, 2), AnnoSpan(64-65, 1)]],
#    'delimiter': '|',
#    'type': 'table'}
```

Structured Incident Annotator

The structured incident annotator identifies and parses embedded tables that describe case counts paired with location, date, disease and species metadata. Metadata is also extracted from the text around the table.

Usage

```
from epitator.annotator import AnnoDoc
from epitator.structured_incident_annotator import StructuredIncidentAnnotator
doc = AnnoDoc("""
Fictional October 2015 rabies cases in Svalbard

species | cases | deaths
Cattle  | 0     | 0
Dogs    | 4     | 1
""")
doc.add_tiers(StructuredIncidentAnnotator())
annotations = doc.tiers["structured_incidents"].spans
annotations[-1].metadata
# = {'location': {'name': u'Svalbard', ...},
#    'species': {'label': u'Canidae', ...},
#    'attributes': [],
#    'dateRange': [datetime.datetime(2015, 10, 1, 0, 0), datetime.datetime(2015, 11,
↪1, 0, 0)],
#    'type': 'deathCount',
#    'value': 1,
#    'resolvedDisease': {'label': u'rabies', ...}}
```

5.2 AnnoDoc

class epitator.annodoc.**AnnoDoc**(*text=None*, *date=None*)

Bases: object

A document to be annotated. The tiers property links to the annotations applied to it.

add_tier(*annotator*, ***kwargs*)

add_tiers(*annotator*, ***kwargs*)

create_regex_tier(*regex*, *label=None*)

Create an AnnoTier from all the spans of text that match the regex.

filter_overlapping_spans(*tiers=None*, *tier_names=None*, *score_func=None*)

Remove the smaller of any overlapping spans.

require_tiers(**tier_names*, ***kwargs*)

Return the specified tiers or add them using the via annotator.

to_dict()

Convert the document into a json serializable dictionary. This does not store all the document's data. For a complete serialization use pickle.

```
>>> from .annospan import AnnoSpan
>>> from .annotier import AnnoTier
>>> import datetime
>>> doc = AnnoDoc('one two three', date=datetime.datetime(2011, 11, 11))
>>> doc.tiers = {
...     'test': AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(4, 7, doc)])}
>>> d = doc.to_dict()
>>> str(d['text'])
'one two three'
>>> str(d['date'])
```

(continues on next page)

(continued from previous page)

```
'2011-11-11T00:00:00Z'
>>> sorted(d['tiers']['test'][0].items())
[('label', None), ('textOffsets', [[0, 3]])]
>>> sorted(d['tiers']['test'][1].items())
[('label', None), ('textOffsets', [[4, 7]])]
```

5.3 AnnoTier

class epitator.annotier.**AnnoTier**(*spans=None*, *presorted=False*)

Bases: object

A group of AnnoSpans stored sorted by start offset.

chains(*at_least=1*, *at_most=None*, *max_dist=1*)

Create a new tier from all chains of spans within max_dist of eachother.

combined_adjacent_spans(*max_dist=1*)

Create a new tier from groups of spans within max_dist of eachother.

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three four')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc),
...                  AnnoSpan(8, 13, doc),
...                  AnnoSpan(14, 18, doc)])
>>> tier.combined_adjacent_spans()
AnnoTier([SpanGroup(text=one, label=None, AnnoSpan(0-3, one)),␣
↪SpanGroup(text=three four, label=None, AnnoSpan(8-13, three), AnnoSpan(14-
↪18, four))])
```

group_spans_by_containing_span(*other_tier*, *allow_partial_containment=False*)

Group spans in other_tier by the spans that contain them in this one.

Parameters

- **other_tier** (AnnoTier) – The spans to be grouped together
- **allow_partial_containment** – Include spans in groups for spans that partially overlap them.

Returns An iterator that returns pairs of values, the first of which is the containing span from this tier, the second is an array of spans from other_tier that the span from this tier contans.

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
>>> tier_a = AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(4, 7, doc)])
>>> tier_b = AnnoTier([AnnoSpan(0, 1, doc)])
>>> list(tier_a.group_spans_by_containing_span(tier_b))
[(AnnoSpan(0-3, one), [AnnoSpan(0-1, o)]), (AnnoSpan(4-7, two), [])]
```

label_spans(*label*)

Create a new tier based on this one with labeled spans that can be looked up by groupdict.

match_subspans(*regex*)

Create a new tier from the components of spans matching the given regular expression.

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three four')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc),
...                  AnnoSpan(4, 13, doc),
...                  AnnoSpan(14, 18, doc)])
>>> tier.match_subspans(r"two")
AnnoTier([AnnoSpan(4-7, two)])
```

nearest_to(*target_span*)
Find the nearest span to the target span.

optimal_span_set(*prefer='text_length'*)

Parameters **perfer** – A function that takes a span and returns a numeric tuple score. The following predefined functions may be specified via string: text_length, text_length_min_spans, num_spans, and num_spans_and_no_linebreaks

Returns A tier with the set of non-overlapping spans from this tier that maximizes the prefer function.

Return type *AnnoTier*

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc, 'odd'),
...                  AnnoSpan(4, 7, doc, 'even'),
...                  AnnoSpan(3, 13, doc, 'long_span'),
...                  AnnoSpan(8, 13, doc, 'odd')])
>>> tier.optimal_span_set()
AnnoTier([AnnoSpan(0-3, odd), AnnoSpan(3-13, long_span)])
```

search_spans(*regex*, *label=None*)
Search spans for ones matching the given regular expression.

span_after(*target_span*)
Find the nearest span that comes after the target span.

span_before(*target_span*, *allow_overlap=True*)
Find the nearest span that comes before the target span.

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three four')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc),
...                  AnnoSpan(8, 13, doc),
...                  AnnoSpan(14, 18, doc)])
>>> tier.span_before(AnnoSpan(4, 7, doc))
AnnoSpan(0-3, one)
```

spans_contained_by_span(*selector_span*)
Return a list of spans that are contained by a "selector span".

```
>>> from epitator.annospan import AnnoSpan
>>> from epitator.annodoc import AnnoDoc
>>> from epitator.annotier import AnnoTier
>>> doc = AnnoDoc('one two three')
>>> tier1 = AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(4, 7, doc)])
```

(continues on next page)

(continued from previous page)

```
>>> span1 = AnnoSpan(3, 9, doc)
>>> tier1.spans_contained_by_span(span1)
AnnoTier([AnnoSpan(4-7, two)])
```

spans_overlapped_by_span(*selector_span*)

Return a list of spans that overlap a "selector span".

```
>>> from epitator.annospan import AnnoSpan
>>> from epitator.annodoc import AnnoDoc
>>> from epitator.annotier import AnnoTier
>>> doc = AnnoDoc('one two three')
>>> tier1 = AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(4, 7, doc)])
>>> span1 = AnnoSpan(0, 1, doc)
>>> tier1.spans_overlapped_by_span(span1)
AnnoTier([AnnoSpan(0-3, one)])
```

subtract_overlaps(*other_tier*)

Parameters **other_tier** (AnnoTier) – The spans to be removed from the territory of this tier

Returns A copy of this tier with spans truncated and split so that none of the new spans overlap a span in other_tier

Return type *AnnoTier*

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three four')
>>> tier_a = AnnoTier([AnnoSpan(0, 18, doc)])
>>> tier_b = AnnoTier([AnnoSpan(3, 8, doc), AnnoSpan(13, 18, doc)])
>>> tier_a.subtract_overlaps(tier_b)
AnnoTier([AnnoSpan(0-3, one), AnnoSpan(8-13, three)])
```

with_contained_spans_from(*other_tier*, *allow_partial_containment=False*)

Create a new tier from pairs spans in this tier and the other tier where the span in this tier contains one in the other tier.

with_following_spans_from(*other_tier*, *max_dist=1*, *allow_overlap=False*)

Create a new tier from pairs of spans where the one in the other tier follows a span from this tier.

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three four')
>>> tier1 = AnnoTier([AnnoSpan(0, 3, doc),
...                   AnnoSpan(8, 13, doc)])
>>> tier2 = AnnoTier([AnnoSpan(14, 18, doc)])
>>> tier1.with_following_spans_from(tier2)
AnnoTier([SpanGroup(text=three four, label=None, AnnoSpan(8-13, three),␣
↪AnnoSpan(14-18, four))])
```

with_label(*label*)

Create a tier from the spans which have the given label

```
>>> from .annospan import AnnoSpan
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
```

(continues on next page)

(continued from previous page)

```
>>> tier = AnnoTier([AnnoSpan(0, 3, doc, 'odd'),
...                  AnnoSpan(4, 7, doc, 'even'),
...                  AnnoSpan(8, 13, doc, 'odd')])
>>> tier.with_label("odd")
AnnoTier([AnnoSpan(0-3, odd), AnnoSpan(8-13, odd)])
```

with_nearby_spans_from(*other_tier*, *max_dist=100*)
Create a new tier from pairs spans in this tier and the other tier that are near eachother.

without_overlaps(*other_tier*)
Create a copy of this tier without spans that overlap a span in the other tier.

5.4 AnnoSpan

class epitator.annospan.**AnnoSpan**(*start*, *end*, *doc*, *label=None*, *metadata=None*)
Bases: object

A span of text with an annotation applied to it.

adjacent_to(*other_span*, *max_dist=1*)
Return true if the span comes before or after other_span with at most max_dist charaters between them.

base_spans

comes_before(*other_span*, *max_dist=1*, *allow_overlap=False*)
Return True if the span comes before the other_span and there are max_dist or fewer charaters between them.

```
>>> from .annotier import AnnoTier
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(4, 7, doc)])
>>> tier.spans[0].comes_before(tier.spans[1])
True
>>> tier.spans[1].comes_before(tier.spans[0])
False
```

contains(*other_span*)
Return true if the span completely contains other_span.

distance(*other_span*)
The number of characters between this span and the other one. If the spans overlap the distance is the negative length of their overlap.

```
>>> from .annotier import AnnoTier
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
>>> tier = AnnoTier([AnnoSpan(0, 3, doc), AnnoSpan(8, 13, doc)])
>>> tier.spans[0].distance(tier.spans[1])
5
```

doc

end

extended_through(*other_span*)
Create a new span that includes this one and the other span.

groupdict()
Return a dict with all the labeled matches.

```
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two wolf')
>>> number_span_g = SpanGroup([AnnoSpan(0, 3, doc, 'number'),
...                            AnnoSpan(4, 7, doc, 'number'),
...                            AnnoSpan(8, 12, doc, 'animal')])
>>> number_span_g.groupdict()['number']
[AnnoSpan(0-3, number), AnnoSpan(4-7, number)]
>>> number_span_g.groupdict()['animal']
[AnnoSpan(8-12, animal)]
```

iterate_base_spans()
Recursively iterate over all base_spans including base_spans of child SpanGroups.

iterate_leaf_base_spans()
Return the leaf base spans in a SpanGroup tree.

label

metadata

overlaps(*other_span*)
Return true if the span overlaps other_span.

start

text

to_dict()
Return a json serializable dictionary.

trimmed()
Create a new AnnoSpan based on this one with the offsets adjusted so that there is no white space at the beginning or end.

```
>>> from .annodoc import AnnoDoc
>>> doc = AnnoDoc('one two three')
>>> original_span = AnnoSpan(3, 8, doc)
>>> original_span.trimmed()
AnnoSpan(4-7, two)
```

class epitator.annospan.**SpanGroup**(*base_spans*, *label=None*, *metadata=None*)
Bases: *epitator.annospan.AnnoSpan*

A AnnoSpan that extends through a group of AnnoSpans.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

T

W