kindred Documentation

Release =2.8.3

Jake Lever

Mar 12, 2023

Contents

CHAPTER 1

File Formats

Kindred can load several different file formats that contain text and their annotations. Below are examples of the different file formats with code for loading them.

1.1 BioNLP Shared Task format

This format, used in BioNLP Shared Tasks, is a standoff format. This means that the text is stored in one file and the annotations in other files. The text is stored in the .txt file, the entity annotations in the .a1 file and the relations in the .a2 file. For a project, you may have a directory with many .txt files, perhaps one per document or one per sentence. Then each file has its corresponding annotation files. If no relations annotations exist, the .a2 file may be missing.

Example file: example.txt

```
The colorectal cancer was caused by mutations in APC
```

Example file: example.a1

```
T1   disease 4 21    colorectal cancer
T2   gene 49 52      APC
```

Example file: example.a2

```
E1   causes subj:T2 obj:T1
```

The .txt file contains Unicode text and no annotations. The .a1 file contains entity annotations. Each line is a new annotation and contains three tab-delimited columns. The first column is the unique identifier which is a T with a number. The second column contains the entity type, start and end position in the text with spaces in between. And the third column has a copy of the text for this entity. The .a2 file contains the relation annotations and contains tab-delimited columns. The first column is a unique identifier of the relation. The second column is the relation type and then the arguments of the relation, in the form of name:entityid. The entity identifier corresponds to the identifier in the .a1 file. Kindred supports relations with two or more arguments in the relation.

The identifiers for an entity annotation (in the .a1 file) must start with a T. The T stands for trigger. The identifiers for a relation annotation (in the .a2 file) must start with an E or R. For Kindred, these are synonymous. Note, that Kindred

doesn't support "complex" relations, which are relations where one of the arguments is another relation. All relations must be between entities.

The following code would load these files to create a *kindred.Corpus* with a single document.

```
corpus = kindred.load('standoff','example.txt')
```

Perhaps more useful, to load a whole corpus with multiple files in the format, use the following code assuming that the files are in the example directory. This will create a *kindred.Corpus* object.

```
corpus = kindred.load('standoff','example')
```

1.2 JSON format

This format, used by PubAnnotation and PubTator, stores the text and annotation data all together in a single file. Furthermore, multiple documents can be stored in a single document.

The format is standard JSON and is either a dictionary (for a single document) or a list of dictionaries (for multiple documents). Each dictionary needs to have three fields: text, denotations, and relations. The text is the text of the document. The denotations are the entity annotations and provide the unique identifier, entity type and location (span) in the text. The relations are the relation annotations.

Example file: example.json

```
{
  "text": "The colorectal cancer was caused by mutations in APC",
  "denotations":
    [{"id":"T1", "obj":"disease",
      "span":{"begin":4,"end":21}},
     {"id":"T2", "obj":"gene",
      "span":{"begin":49,"end":52}}],
  "relations":
    [{"id":"R1","pred":"causes",
      "subj":"T2", "obj":"T1"}]
}
```

To load a whole corpus with multiple files in the format, use the following code assuming that the files are in the example directory. This will create a *kindred.Corpus* object.

```
corpus = kindred.load('json','example')
```

1.3 BioC XML format

The BioC XML format contains text and annotations together in a single file. Furthermore, it is designed to store more than one document. It stores each document as "document" within a larger "collection". Each document contains passages (e.g. sections of a paper) which then contain the text, entity annotations, and relations. In loading this, each passage is turned into a single *kindred.Document*. An example of the format is outlined below.

```
<?xml version='1.0' encoding='UTF-8'?><!DOCTYPE collection SYSTEM 'BioC.dtd'>
<collection>
  <source></source>
  <date></date>
  <key></key>
```

(continues on next page)

(continued from previous page)

```
  <document>
    <id></id>
    <passage>
      <offset>0</offset>
      <text>The colorectal cancer was caused by mutations in APC</text>
      <annotation id="T1">
        <infon key="type">disease</infon>
        <location offset="4" length="17"/>
        <text>colorectal cancer</text>
      </annotation>
      <annotation id="T2">
        <infon key="type">gene</infon>
        <location offset="49" length="3"/>
        <text>APC</text>
      </annotation>
      <relation id="R1">
        <infon key="type">causes</infon>
        <node refid="T2" role="subj"/>
        <node refid="T1" role="obj"/>
      </relation>
    </passage>
  </document>
</collection>
```

To load a whole directory of BioC XML files, use the code below. This will create a single *kindred.Corpus* file with each passage found in all XML files in the directory turned a *kindred.Document* entity.

```
corpus = kindred.load('bioc','example')
```

1.4 Simple Tag format

This format is not designed for production-use but for illustration and testing purposes. It is Kindred-specific. It is an XML-based format that keeps all annotations inline, to make it easier to see which entities are annotated. A relation tag provides a relation annotation and must have a type attribute. All other attributes are assumed to be relation argument. Any non-relation tag is assumed to be an entity annotation and must wrap around text. It must also have an id attribute.

Example file: example.simple

```
The <disease id="T1">colorectal cancer</disease> was caused by mutations in <gene id=
↪"T2">APC</gene>
<relation type="causes" subj="T2" obj="T1" />
```

It is most useful for quickly creating examples for testing. For example, the code below creates a *kindred.Corpus* with a single document of the associated text and annotations.

```
text = '<drug id="1">Erlotinib</drug> is a common treatment for <cancer id="2">NSCLC</
↪cancer>. <drug id="3">Aspirin</drug> is the main cause of <disease id="4">boneitis</
↪disease>. <relation type="treats" subj="1" obj="2" />'

corpus = kindred.Corpus(text,loadFromSimpleTag=True)
```

If you do need to load a directory of these files (with suffix: .simple), the following command will load them into a *kindred.Corpus* file.

```
corpus = kindred.load('simpletag','example')
```

1.5 Streaming

Some corpora are too large to load into memory in a single go. Kindred supports streaming in chunks of a corpus in the BioC format. The code below uses an iterator to load smaller *kindred.Corpus* objects that contain a subset of the documents each time.

```
for corpus in kindred.iterLoad('example.bioc.xml',corpusSizeCutoff=3):
        pass
```

CHAPTER 2

Overview

Kindred is a Python package specifically designed for binary relation extraction from biomedical texts (e.g. PubMed abstracts). It takes a supervised learning approach, and therefore requires training data in order to build a model.

Kindred can do simple dictionary-based entity extraction. It also has integration with Pubtator to automatically pull out PubMed abstracts with a number of entities tagged and with PubAnnotation and can easily load annotation data.

CHAPTER 3

Installation

Kindred is distributed through PyPI. Hence you should be able to install it with the shell command below.

```
pip install kindred
```

If you need to upgrade to a newer release, use the following shell command.

```
pip install --upgrade kindred
```

And if you want to install directly from source, use this shell command.

```
python setup.py install
```

Once it is installed, Kindred can be imported in Python with:

```
>>> import kindred
```

3.1 Installing a Spacy language model

As of v2, Kindred uses the Spacy python package for parsing. A language model needs to be installed for the corresponding language using a command similar to below.

```
python -m spacy download en_core_web_sm
```

CHAPTER 4

Tutorial with a mini annotation problem

There is a tutorial with sample code that steps through a small annotation task for extracting capital cities from text. It's on Github and may give you an understanding of the annotations that Kindred needs and how you might go about getting them. Once you've understood the input data, you might want to dive more into the code and the below examples will give you some ideas.

CHAPTER 5

Getting started with code

Let's walk through a basic example for the BioNLP Shared Task. This will involve loading a corpus of data to train a classifier and a corpus to make predictions on and for evaluation. We will then train the classifier, make the predictions and evaluate how we did. The smaller steps (parsing, candidate building & vectorizing) are done behind the scenes.

First, we need to load the data. We want the training and development corpus and use the commands below

```
>>> trainCorpus = kindred.bionlpst.load('2016-SeeDev-binary-train')
>>> devCorpus = kindred.bionlpst.load('2016-SeeDev-binary-dev')
```

We're going to build a model for the relations in the training corpus and make predictions on the development corpus. We are going to keep the devCorpus object to make comparisons against, but need a copy of it that doesn't have any relations attached to it. Hence we will clone it and remove the relations. This will contain all the same text and entity annotations as the devCorpus, but no relations.

```
>>> predictionCorpus = devCorpus.clone()
>>> predictionCorpus.removeRelations()
```

Now we're going to build the model on the training data with default settings.

```
>>> classifier = kindred.RelationClassifier()
>>> classifier.train(trainCorpus)
```

Now we will use this classifier to predict relations in the predictionCorpus object. These new relations will be added to the corpus.

```
>>> classifier.predict(predictionCorpus)
```

Lastly, we will evaluate how well we have done. The common measure is F1-score.

```
>>> f1score = kindred.evaluate(devCorpus, predictionCorpus, metric='f1score')
```

CHAPTER 6

Specific Examples

Here we will show some of the individual steps that might be needed.

6.1 Loading data from files

To load a corpus from a directory, you can use the load function, providing the format of the data.

```
>>> corpus = kindred.load('biocxml','/home/user/data/')
```

And if it was in another format, you change the dataFormat parameter. Options include: 'standoff' for the standoff format used in the BioNLP Shared Tasks, 'biocxml' for BioC XML files and 'simpletag' if there are a set of SimpleTag XML files. Note that we only use SimpleTag for generating easy test data and not for any large problems.

6.2 Loading data from online resources

Kindred integrates with several online resources to make it easy to import data. For BioNLP Shared Tasks, you can use the command below:

```
>>> corpus = kindred.bionlpst.load('2016-SeeDev-binary-train')
```

You can currently import data from the '2016-SeeDev-binary' shared tasks as the files for '2016-BB3-event' are no longer available. Add 'train', 'dev' or 'test' to them. The 'train' and 'dev' corpora contain relations while the 'test' corpus does not.

You can import PubMed abstracts annotated by Pubtator with a list of PubMed IDs (or PMIDs for short). These will contain entity annotations but no relations. The command below will import the two articles with those PMIDs.

```
>>> corpus = kindred.pubtator.load([19894120,19894121])
```

You can also import text and annotation data from PubAnnotation. In this case, you provide the project name and Kindred will download all the annotations and associated text. For the 'bionlp-st-gro-2013-development' project, the command to import is below. These annotations may include relation information

```
>>> corpus = kindred.pubannotation.load('bionlp-st-gro-2013-development')
```

6.3 Parsing

If you want to parse a corpus, you use a Parser object.

```
>>> parser = kindred.Parser()
>>> parser.parse(corpus)
```

6.4 Candidate Building

Given a corpus with annotated entities, one may want to generate the set of all candidate relations between two entities within the same text. One can do this for the first set with the command below. Each Sentence object within the corpus will now have a set of candidate relations attached to it.

```
>>> candidateBuilder = kindred.CandidateBuilder()
>>> candidateBuilder.fit_transform(corpus)
```

You can easily extract all the candidate relations using the command below:

```
>>> candidateRelations = corpus.getCandidateRelations()
```

The corpus contains a list of relation types contained within.

```
>>> print(corpus.relationTypes)
```

And if the corpus contains annotated relations, the candidate relations will be assigned a non-zero class index. Hence a candidate relation with class 0 has not been annotated, but a candidate relation with class 1 is of the first relation type in corpus.relationTypes.

6.5 Vectorizing

You may want to generate vectors for each candidate relation. The command below will produce the vectorized matrix with the default set of feature types.

```
>>> vectorizer = kindred.Vectorizer()
>>> trainMatrix = vectorizer.fit_transform(trainCorpus)
```

Once you've fit the vectorizer to the training set, remember to only use transform for the test set.

```
>>> testMatrix = vectorizer.transform(testCorpus)
```

Want to use only specific feature types (of which the options are: entityTypes, unigramsBetweenEntities, bigrams, dependencyPathEdges, dependencyPathEdgesNearEntities)? Use a command like below:

```
>>> vectorizer = kindred.Vectorizer(featureChoice=['entityTypes','bigrams'])
```

CHAPTER 7

Frequently Asked Questions

Does Kindred handle multiple relations that contain the same entities?

At the moment, no. Kindred will only use the first annotation of a relation.

CHAPTER 8

Release Notes

CHAPTER 9

Version 2.8.0

- Updates for newer version of bioc library
- Dealing with BioNLP 2016 task files that are unavailable

CHAPTER 10

Version 2.7.0

- Added support to save to PubAnnotation format

CHAPTER 11

Version 2.6.0

- Release v2.6.1 is the final Python2 compatible version
- Added option for metadata associated with entities
- Added option to load any Spacy model for parsing

CHAPTER 12

Version 2.5.0

- Added MultiLabelClassifier and changed behaviour when multiple relation types are present. They are now predicted independently using separate classifiers. This allows overlapping relations (where the same entities are part of multiple relations).

CHAPTER 13

Version 2.4.0

- Updates to the loading and saving functionality so that everything is done through kindred.load or kindred.save
- Changed EntityRecognizer logic to use token boundaries and exact string matching instead of matching tokenization (for faster wordlist loading)

CHAPTER 14

Version 2.3.0

- Add manuallyAnnotate for a simple mechanism to annotate candidate relations
- Add splitIntoSentences for a parsed corpus/document

CHAPTER 15

Version 2.2.0

- Add CandidateRelation class to distinguish from Relation
- Reworking of API so that Candidate Relations are no longer stored in corpus. Changes across API that will break backwards compatibility
- Fixes to PubTator input

CHAPTER 16

Version 2.1.0

- Added EntityRecognizer for basic entity extraction
- Relations can now be n-ary, not just binary

CHAPTER 17

Version 2.0.0

- Large overhaul to replace CoreNLP with Spacy package for easier integration and installation
- Simplified relation classifier functionality by removing feature building and multiclassifier options
- Add functionality for streaming BioC files

17.1 Version 1.1.0

- Upgraded to new version of Stanford CoreNLP (3.8.0) and added code to manage upgrade
- Changed dependency parsing to use standard CoreNLP dep parser (instead of constituency with a conversion).
- Changed evaluation function to not output specific details by default
- You can now parse with every language in CoreNLP (arabic,chinese,english,french,german,spanish)
- Improved error display for CoreNLP failures

17.2 Version 1.0.0

- Original release (corresponding to original paper)

CHAPTER 18

Citing

If your work makes use of Kindred, it'd be really nice if you cited us.

```
@article{lever2017painless,
         title={Painless {R}elation {E}xtraction with {K}indred},
         author={Lever, Jake and Jones, Steven JM},
         journal={Bio{NLP} 2017},
         pages={176},
         year={2017}
         }
```

CHAPTER 19

Reference

19.1 Main components

EntityRecognizer	Annotates entities in a Corpus using an exact-dictionary matching scheme with additional heuristics.
CandidateBuilder	Generates set of all possible relations in corpus.
Parser	Runs Spacy on corpus to get sentences and associated tokens
RelationClassifier	Manages binary classifier(s) for relation classification.
Vectorizer	Vectorizes set of candidate relations into scipy sparse matrix.

19.1.1 kindred.EntityRecognizer

class kindred.**EntityRecognizer**(*lookup*, *detectFusionGenes=False*, *detectMicroRNA=False*, *acronymDetectionForAmbiguity=False*, *mergeTerms=False*, *detectVariants=False*, *variantStopwords=None*, *detectPolymorphisms=False*, *removePathways=False*)

Annotates entities in a Corpus using an exact-dictionary matching scheme with additional heuristics. These heuristics include detecthing fusion gene mentions, microRNA, identifying acronyms to reduce ambiguity, identifying variants and more. All the options are parameters for the constructor of this class.

Variables

- **lookup** – Used for the dictionary matching. A dictionary of terms (tuple of parsed words) to a list of (entityType,externalID).
- **detectFusionGenes** – Whether it will try to identify fusion gene terms (e.g. BCR-ABL1). Lookup must contain terms of type 'gene'
- **detectMicroRNA** – Whether it will identify microRNA terms (added as 'gene' entities)
- **acronymDetectionForAmbiguity** – Whether it will try to identify acronyms and

use this to deal with ambiguity (by removing incorrect matches to acronyms or the longer terms)

- **mergeTerms** – Whether it will merge neighbouring terms that refer to the same external entity (e.g. HER2/neu as one term instead of two)
- **detectVariants** – Whether it will identify a variant (e.g. V600E) and create an entity of type 'variant'
- **variantStopwords** – Variant terms to be ignored (e.g. S100P) if detectVariants is used
- **detectPolymorphisms** – Whether it will identify a SNP (using a dbSNP ID) and create an entity of type 'variant'
- **removePathways** – Whether it will remove genes that are actually naming a signalling pathway (e.g. MTOR pathway)

Methods

__init__(*lookup*, *detectFusionGenes=False*, *detectMicroRNA=False*, *acronymDetectionForAmbiguity=False*, *mergeTerms=False*, *detectVariants=False*, *variantStopwords=None*, *detectPolymorphisms=False*, *removePathways=False*)

Create an EntityRecognizer and provide the lookup table for terms and additional flags for what to identify in text

Parameters

- **lookup** (*dict*) – A dictionary of terms (tuple of parsed words) to a list of (entityType,externalID).
- **detectFusionGenes** (*bool*) – Whether to try to identify fusion gene terms (e.g. BCR-ABL1). Lookup must contain terms of type 'gene'
- **detectMicroRNA** (*bool*) – Whether to identify microRNA terms (added as 'gene' entities)
- **acronymDetectionForAmbiguity** (*bool*) – Whether to try to identify acronyms and use this to deal with ambiguity (by removing incorrect matches to acronyms or the longer terms)
- **mergeTerms** (*bool*) – Whether to merge neighbouring terms that refer to the same external entity (e.g. HER2/neu as one term instead of two)
- **detectVariants** (*bool*) – Whether to identify a variant (e.g. V600E) and create an entity of type 'variant'
- **variantStopwords** (*list*) – Variant terms to be ignored (e.g. S100P) if detectVariants is used
- **detectPolymorphisms** (*bool*) – Whether to identify a SNP (using a dbSNP ID) and create an entity of type 'variant'
- **removePathways** (*bool*) – Remove genes that are actually naming a signalling pathway (e.g. MTOR pathway)

annotate(*corpus*)

Annotate a parsed corpus with the wordlist lookup and other entity types

Parameters **corpus** (kindred.Corpus) – Corpus to annotate

static loadWordlists(*entityTypesWithFilenames*, *idColumn=0*, *termsColumn=1*, *columnSeparator='\t'*, *termSeparator='|'*)

Load a wordlist from multiple files. By default, each file should be a tab-delimited file with the first

column is the ID and the second column containing all the terms separated by 'l'. This can be modified by the parameters.

As each term is parsed, this can take a long time. It is recommended to run this one time and save the output as a Python pickle file and load in.

Parameters

- **entityTypesWithFilenames** (*dict*) – Dictionary of entityType => filename
- **idColumn** (*int*) – The column containing the ID for the term (starts from 0)
- **termsColumn** (*int*) – The column containing the list of terms (starts from 0)
- **columnSeparator** (*str*) – The column separator for the file (default is a tab)
- **termSeparator** (*str*) – The separator for the list of terms (default is a 'l')

Returns Dictionary of lookup values

Return type dict

19.1.2 kindred.CandidateBuilder

class kindred.**CandidateBuilder**(*entityCount=2*, *acceptedEntityTypes=None*)

Generates set of all possible relations in corpus.

Variables

- **entityCount** – Number of entities in each relation (default=2)
- **acceptedEntityTypes** – Tuples of entities that candidate relations must match. Each entity should be the same length as entityCount. None will match all candidate relations.

Methods

__init__(*entityCount=2*, *acceptedEntityTypes=None*)

Constructor

Parameters

- **entityCount** (*int*) – Number of entities in each relation (default=2)
- **acceptedEntityTypes** (*list of tuples*) – Tuples of entities that candidate relations must match. Each entity should be the same length as entityCount. None will match all candidate relations.

build(*corpus*)

Creates the set of all possible relations that exist within the given corpus. Each relation will be contained within a single sentence.

Parameters **corpus** (kindred.Corpus) – Corpus of text with which to build relation candidates

Returns List of candidate relations matching entityCount and acceptedEntityTypes

Return type List of kindred.Relation

19.1.3 kindred.Parser

class kindred.**Parser**(*model='en_core_web_sm'*)

Runs Spacy on corpus to get sentences and associated tokens

Variables

- **model** – Model for parsing (e.g. en/de/es/pt/fr/it/nl)
- **nlp** – The underlying Spacy language model to use for parsing

Methods

__init__(*model='en_core_web_sm'*)

Create a Parser object that will use Spacy for parsing. It offers all the same languages that Spacy offers. Check out: https://spacy.io/usage/models. Note that the language model needs to be downloaded first (e.g. python -m spacy download en)

Parameters **model** (*str*) – Name of an available Spacy language model for parsing (e.g. en/de/es/pt/fr/it/nl)

parse(*corpus*)

Parse the corpus. Each document will be split into sentences which are then tokenized and parsed for their dependency graph. All parsed information is stored within the corpus object.

Parameters **corpus** (kindred.Corpus) – Corpus to parse

19.1.4 kindred.RelationClassifier

class kindred.**RelationClassifier**(*classifierType='SVM'*, *tfidf=True*, *features=None*, *threshold=None*, *entityCount=2*, *acceptedEntityTypes=None*, *model='en_core_web_sm'*)

Manages binary classifier(s) for relation classification.

Parameters

- **classifierType** – Which classifier is used ('SVM' or 'LogisticRegression')
- **tfidf** – Whether it will use tfidf for the vectorizer
- **features** – A list of specific features. Valid features are "entityTypes", "unigramsBetweenEntities", "bigrams", "dependencyPathEdges", "dependencyPathEdgesNearEntities"
- **threshold** – A specific threshold to use for classification (which will then use a logistic regression classifier)
- **entityCount** – Number of entities in each relation (default=2). Passed to the CandidateBuilder (if needed)
- **acceptedEntityTypes** – Tuples of entity types that relations must match. None will match allow relations of any entity types. Passed to the CandidateBuilder (if needed)
- **isTrained** – Whether the classifier has been trained yet. Will throw an error if predict is called before it is trained.

Methods

__init__(*classifierType='SVM'*, *tfidf=True*, *features=None*, *threshold=None*, *entityCount=2*, *acceptedEntityTypes=None*, *model='en_core_web_sm'*)
Constructor for the RelationClassifier class

Parameters

- **classifierType** (*str*) – Which classifier to use (must be 'SVM' or 'LogisticRegression')
- **tfidf** (*bool*) – Whether to use tfidf for the vectorizer
- **features** (*list of str*) – A list of specific features. Valid features are "entityTypes", "unigramsBetweenEntities", "bigrams", "dependencyPathEdges", "dependencyPathEdgesNearEntities"
- **threshold** (*float*) – A specific threshold to use for classification (which will then use a logistic regression classifier)
- **entityCount** (*int*) – Number of entities in each relation (default=2). Passed to the CandidateBuilder (if needed)
- **acceptedEntityTypes** (*list of tuples*) – Tuples of entity types that relations must match. None will match allow relations of any entity types. Passed to the CandidateBuilder (if needed)
- **model** (*str*) – Name of an available Spacy language model for any parsing needed (e.g. en/de/es/pt/fr/it/nl)

predict(*corpus*)
Use the relation classifier to predict new relations for a corpus. The new relations will be added to the Corpus.

Parameters **corpus** (kindred.Corpus) – Corpus to make predictions on

train(*corpus*)
Trains the classifier using this corpus. All relations in the corpus will be used for training.

Parameters **corpus** (kindred.Corpus) – Corpus to use for training

19.1.5 kindred.Vectorizer

class kindred.**Vectorizer**(*entityCount=2*, *featureChoice=None*, *tfidf=True*)
Vectorizes set of candidate relations into scipy sparse matrix.

Variables

- **entityCount** – Number of entities in candidate relations to vectorize
- **featureChoice** – List of features (can be one or a set of the following: 'entityTypes', 'unigramsBetweenEntities', 'bigrams', 'dependencyPathEdges', 'dependencyPathEdgesNearEntities'). Set as None to use all of them.
- **tfidf** – Whether it will normalize n-gram based features using term frequency-inverse document frequency
- **fitted** – Whether it has been fit on data first (before transforming).
- **dictVectorizers** – Dictionary vectorizers used for each feature
- **tfidfTransformers** – TFIDF transformers used for each feature (if appropriate and selected)

Methods

__init__(*entityCount=2*, *featureChoice=None*, *tfidf=True*)
Constructor for vectorizer class with options for what features to use and whether to normalize using TFIDF

Parameters

- **entityCount** (*int*) – Number of entities in candidate relations to vectorize
- **featureChoice** (*list of str*) – List of features (can be one or a set of the following: 'entityTypes', 'unigramsBetweenEntities', 'bigrams', 'dependencyPathEdges', 'dependencyPathEdgesNearEntities'). Set as None to use all of them.
- **tfidf** (*bool*) – Whether to normalize n-gram based features using term frequency-inverse document frequency

fit_transform(*candidates*)
Fit the vectorizer to a list of candidate relations found in a corpus and vectorize them to generate the feature matrix.

Parameters **candidates** (*list of kindred.CandidateRelation*) – Relation candidates to vectorize

Returns Feature matrix (# rows = number of candidate relations, # cols = number of features)

Return type scipy.sparse.csr.csr_matrix

getFeatureNames()
Get the names for each feature (i.e. each column in matrix generated by the fit_transform() and transform() functions. Fit_transform() must have already been used, i.e. the vectorizer needs to have been fit to training data.

Returns List of names for each feature (column of the vectorized data)

Return type List of str

transform(*candidates*)
Vectorize the candidate relations to generate the feature matrix. Must already have been fit.

Parameters **candidates** (*list of kindred.CandidateRelation*) – Relation candidates to vectorize

Returns Feature matrix (# rows = number of candidate relations, # cols = number of features)

Return type scipy.sparse.csr.csr_matrix

19.2 Data types

CandidateRelation	Describes a candidate relation between entities (i.e.
Corpus	Collection of text documents.
Document	Span of text with associated tagged entities and relations between entities.
Entity	Biomedical entity with information of location in text
Relation	Describes relationship between entities (including relation type and argument names if applicable).
Sentence	Set of tokens for a sentence after parsing

Continued on next page

Table 2 – continued from previous page

Token	Individual word with lemma, part-of-speech and location in text.

19.2.1 kindred.CandidateRelation

class kindred.**CandidateRelation**(*entities=None*, *knownTypesAndArgNames=None*, *sentence=None*)

Describes a candidate relation between entities (i.e. one that could exist but has not yet been predicted). Contains information about known relation types and arg names associated with this candidate (from training data) and also a link to the sentence containing this candidate.

Variables

- **entities** – List of entities in relation
- **knownTypesAndArgNames** – List of tuples with known relation types and argument names associated with this candidate relation
- **sentence** – Parsed sentence containing the candidate relation

Methods

__init__(*entities=None*, *knownTypesAndArgNames=None*, *sentence=None*)

Constructor for Candidate Relation class

Parameters

- **entities** (*list of kindred.Entity*) – List of entities in relation
- **knownTypesAndArgNames** (*list of tuples (str, list of str)*) – List of tuples with known relation types and argument names associated with this candidate relation
- **sentence** (kindred.Sentence) – Parsed sentence containing the candidate relation

19.2.2 kindred.Corpus

class kindred.**Corpus**(*text=None*, *loadFromSimpleTag=False*)

Collection of text documents.

Variables

- **documents** – List of *kindred.Document*
- **parsed** – Boolean of whether it has been parsed yet. A kindred.parser can parse it.

Methods

__init__(*text=None*, *loadFromSimpleTag=False*)

Create an empty corpus with no documents, or quickly load one with a single document using optional SimpleTag

Parameters

- **text** (*String (with SimpleTag format XML)*) – Optional SimpleTag text to initalize a single document

- **loadFromSimpleTag** (*bool*) – If text is provided, whether the text parameter is in the SimpleTag format and will extract entities and relations accordingly

addDocument(*doc*)
Add a single document to the corpus

Parameters **doc** (kindred.Document) – Document to add

clone()
Clone the corpus

Returns Clone of the corpus

Return type *kindred.Corpus*

getRelations()
Get all relations in this corpus

Returns List of relations

Return type list

nfold_split(*folds*)
Method for splitting up the corpus multiple times and is used for an n-fold cross validation approach (as a generator). Each iteration, the training and test set for that fold are provided.

Parameters **folds** (*int*) – Number of folds to create

Returns Tuple of training and test corpus (for iterations=folds)

Return type (*kindred.Corpus*,*kindred.Corpus*)

removeEntities()
Remove all entities in this corpus

removeRelations()
Remove all relations in this corpus

split(*trainFraction*)
Randomly split the corpus into two corpus for use as a training and test set

Parameters **trainFraction** (*float*) – Fraction of documents to use in training set

Returns Tuple of training and test corpus

Return type (*kindred.Corpus*,*kindred.Corpus*)

splitIntoSentences()
Create a new corpus with one document for each sentence in this corpus.

Returns Corpus with one document per sentence

Return type *kindred.Corpus*

19.2.3 kindred.Document

class kindred.**Document**(*text*, *entities=None*, *relations=None*, *sourceFilename=None*, *metadata=None*, *loadFromSimpleTag=False*)
Span of text with associated tagged entities and relations between entities.

Variables

- **text** – Text in document (plain text or SimpleTag)
- **entities** – Entities in document

- **relations** – Relations in document
- **sourceFilename** – Filename that this document came from
- **metadata** – IDs and other information associated with the source (e.g. PMID)
- **sentences** – List of sentences (*kindred.Sentence*) if the document has been parsed

Methods

__init__(*text*, *entities=None*, *relations=None*, *sourceFilename=None*, *metadata=None*, *loadFromSimpleTag=False*)

Constructor for a Document that can take text using the SimpleTag XML format, or a set of Entities and Relations with associated text.

Parameters

- **text** (*str*) – Text in document (plain text or SimpleTag)
- **entities** (*list of kindred.Entity*) – Entities in document
- **relations** (*list of kindred.Relation*) – Relations in document
- **sourceFilename** (*str*) – Filename that this document came from
- **metadata** (*dict*) – IDs and other information associated with the source (e.g. PMID)
- **loadFromSimpleTag** (*bool*) – Assumes the text parameter is in the SimpleTag format and will extract entities and relations accordingly

addEntity(*entity*)

Add an entity to this document. If document has been parsed, it will add the entity into the sentence structure and associated with tokens.

Parameters **entity** (kindred.Entity) – Entity to add

addRelation(*relation*)

Add a relation to this document

Parameters **relation** (kindred.Relation) – Relation to add

addSentence(*sentence*)

Add a sentence to this document

Parameters **sentence** (kindred.Sentence) – Sentence to add

clone()

Clones the document

Returns Clone of the document

Return type *kindred.Document*

removeEntities()

Remove all entities in this document

removeRelations()

Remove all relations in this document

splitIntoSentences()

Create a new corpus with one document for each sentence in this document.

Returns Corpus with one document per sentence

Return type *kindred.Corpus*

19.2.4 kindred.Entity

class kindred.**Entity**(*entityType*, *text*, *position*, *sourceEntityID=None*, *externalID=None*, *metadata=None*)

Biomedical entity with information of location in text

Variables

- **entityType** – Type of the entity
- **text** – Text of the entity
- **position** – Position within the text passage at which point entity appears. Entity may be non-contigious
- **sourceEntityID** – Entity ID used in source document
- **externalID** – ID associated with external ontology (e.g. Hugo Gene ID)
- **metadata** – Additional metadata about the the entity

Methods

__init__(*entityType*, *text*, *position*, *sourceEntityID=None*, *externalID=None*, *metadata=None*)

Constructor for Entity class

Parameters

- **entityType** (*str*) – Type of the entity
- **text** (*str*) – Text of the entity
- **position** (*list of tuples of two integers*) – Position within the text passage at which point entity appears. Entity may be non-contigious
- **sourceEntityID** (*str*) – Entity ID used in source document
- **externalID** (*str*) – ID associated with external ontology (e.g. Hugo Gene ID)
- **metadata** (*dict*) – Additional metadata about the the entity

clone()

Clones the entity

Returns Clone of the entity

Return type *kindred.Entity*

19.2.5 kindred.Relation

class kindred.**Relation**(*relationType=None*, *entities=None*, *argNames=None*, *probability=None*, *sourceRelationID=None*)

Describes relationship between entities (including relation type and argument names if applicable).

Variables

- **relationType** – Type of relation
- **entities** – List of entities in relation
- **argNames** – Names of relation argument associated with each entity
- **probability** – Optional probability for predicted relations

- **sourceRelationID** – Relation ID used in source document

Methods

__init__(*relationType=None*, *entities=None*, *argNames=None*, *probability=None*, *sourceRelationID=None*)
Constructor for Relation class

Parameters

- **relationType** (*str*) – Type of relation
- **entities** (*list of kindred.Entity*) – List of entities in relation
- **argNames** (*list of str*) – Names of relation argument associated with each entity
- **probability** (*float*) – Optional probability for predicted relations
- **sourceRelationID** (*str*) – Relation ID used in source document

19.2.6 kindred.Sentence

class kindred.**Sentence**(*text*, *tokens*, *dependencies*, *sourceFilename=None*)
Set of tokens for a sentence after parsing

Variables

- **text** – Text of the sentence
- **tokens** – List of tokens in sentence
- **dependencies** – List of dependencies from dependency path. Should be a list of tuples with form (tokenindex1,tokenindex2,dependency_type)
- **sourceFilename** – Filename of the source document
- **entityAnnotations** – List of entities associated with token indices

Methods

__init__(*text*, *tokens*, *dependencies*, *sourceFilename=None*)
Constructor for Sentence class

Parameters

- **text** (*str*) – Text of the sentence
- **tokens** (*list of kindred.Token*) – List of tokens in sentence
- **dependencies** (*list of tuples*) – List of dependencies from dependency path. Should be a list of tuples with form (tokenindex1,tokenindex2,dependency_type)
- **sourceFilename** (*str*) – Filename of the source document

addEntityAnnotation(*entity*, *tokenIndices*)
Add an entity annotation to this sentence. Associated a specific entity with the indices of specific tokens

Parameters

- **entity** (kindred.Entity) – Entity to add to sentence
- **tokenIndices** (*List of ints*) – List of token indices

extractMinSubgraphContainingNodes(*minSet*)
Find the minimum subgraph of the dependency graph that contains the provided set of nodes. Useful for finding dependency-path like structures

Parameters **minSet** (*List of ints*) – List of token indices

Returns All the nodes and edges in the minimal subgraph

Return type Tuple of nodes,edges where nodes is a list of token indices, and edges are the associated dependency edges between those tokens

19.2.7 kindred.Token

class kindred.**Token**(*word*, *lemma*, *partofspeech*, *startPos*, *endPos*)
Individual word with lemma, part-of-speech and location in text.

Variables

- **word** – Unprocessed word
- **lemma** – Lemmatized word
- **partofspeech** – Part-of-speech of word
- **startPos** – Start position of token in document text (note: not the sentence text)
- **endPos** – End position of token in document text (note: not the sentence text)

Methods

__init__(*word*, *lemma*, *partofspeech*, *startPos*, *endPos*)
Constructor for Token class

Parameters

- **word** (*str*) – Unprocessed word
- **lemma** (*str*) – Lemmatized word
- **partofspeech** (*str*) – Part-of-speech of word
- **startPos** (*int*) – Start position of token in document text (note: not the sentence text)
- **endPos** (*int*) – End position of token in document text (note: not the sentence text)

19.3 Machine Learning Components

LogisticRegressionWithThreshold	A modified Logistic Regression classifier that will filter calls by a custom threshold, instead of the default 0.5.
MultiLabelClassifier	Wrapper for a set of classifiers that can behave as a multi-label classifier.

19.3.1 kindred.LogisticRegressionWithThreshold

class kindred.**LogisticRegressionWithThreshold**(*threshold=0.5*)
A modified Logistic Regression classifier that will filter calls by a custom threshold, instead of the default 0.5. This allows for control of the precision-recall tradeoff, e.g. false positives versus false negatives.

Variables

- **clf** – The underlying LogisticRegression classifier
- **threshold** – Threshold to use, should be between 0 and 1

Methods

__init__(*threshold=0.5*)

Set up a Logistic Regression classifier that can use a different threshold for predictions and thereby be more lenient (lower threshold, false positives increase, false negatives decrease) or more conservative (higher threshold, false positives decrease, false negative increase).

Parameters **threshold** (*float*) – Threshold to use, should be between 0 and 1

fit(*X*, *Y*)

Train the classifier using the associated matrix X and classes Y. Class zero should represent no associated class.

Parameters

- **X** (*sparse matrix*) – Training vector
- **Y** (*matrix*) – Associated class for each row of X

predict(*X*)

Make predictions for the class of each row in X. Class zero should represent no prediction.

Parameters **X** (*sparse matrix*) – Testing vector

Returns Predictions of classes for each row in X

Return type matrix

predict_proba(*X*)

Calculate probabilities for the class of each row in X. Class zero should represent no prediction. Returns a matrix of probabilities

Parameters **X** (*sparse matrix*) – Testing vector

Returns Probabilities of classes for each row in X

Return type matrix

19.3.2 kindred.MultiLabelClassifier

class kindred.**MultiLabelClassifier**(*classifier*, ***kwargs*)

Wrapper for a set of classifiers that can behave as a multi-label classifier. Multi-label means that each data point can have multiple labels (or belong to multiple classes). This is particularly relevant in text mining where two words can belong to multiple relations. This class just creates a classifier for each label and runs then together, concatenating the results into a nice matrix form

Methods

__init__(*classifier*, ***kwargs*)

Create a classifier that can handle multiple labels using multiple instance of the supplied classifier class. Any additional parameters are passed onto the classifier.

Parameters **classifier** (*class with fit/predict*) – The type of classifier to use

fit(*X*, *Y*)
Fit multiple classifiers for the number of labels provided

Parameters

- **X** (*matrix*) – Training matrix (with n_samples rows and n_features columns)
- **Y** (*matrix*) – Target matrix (with n_samples rows and n_labels columns)

has_predict_proba()
Returns whether the underlying classifier has the predict_proba method

Returns Whether classifier has predict_proba method

Return type bool

predict(*X*)
Predict for multiple labels and return a matrix with predicted labels

Parameters **X** (*matrix*) – Testing matrix (with n_samples rows and n_features columns)

Returns Predicted binary matrix (with n_samples rows and n_labels columns)

Return type matrix

predict_proba(*X*)
Predict for multiple labels and return a matrix with predicted labels. Returns for the probability for the positive class (for each label column) only.

Parameters **X** (*matrix*) – Testing matrix (with n_samples rows and n_features columns)

Returns Predicted probability matrix (with n_samples rows and n_labels columns)

Return type matrix

19.4 Data sources

bionlpst	Importer for BioNLP Shared Task data
pubannotation	Importer for PubAnnotation data
pubtator	Importer for PubTator data

19.4.1 kindred.bionlpst

Importer for BioNLP Shared Task data

Functions

kindred.bionlpst.**listTasks**()
List the names of the BioNLP Shared Task datasets that can be loaded. These values can be passed to the kindred.bionlpst.load function as the taskName argument

Returns List of valid taskNames

Return type str

kindred.bionlpst.**load**(*taskName*, *ignoreEntities=[]*)
Download and load the corresponding corpus from the BioNLP Shared Task

Parameters

- **taskName** (*str*) – The name of the shared task to download (e.g. 'BioNLP-ST-2016_BB-event_train'). Use kindred.bionlpst.listTasks() to get a list of valid options
- **ignoreEntities** (*list of str*) – A list of any entities that should be ignored during loading

Returns The loaded corpus

Return type *kindred.Corpus*

19.4.2 kindred.pubannotation

Importer for PubAnnotation data

Functions

`kindred.pubannotation.load`(*projectName*)

Download and load the corresponding corpus from the PubAnnotation resource

Parameters **projectName** (*str*) – The name of the PubAnnotation project to download

Returns The loaded corpus

Return type *kindred.Corpus*

19.4.3 kindred.pubtator

Importer for PubTator data

Functions

`kindred.pubtator.load`(*pmids*)

Load a set of documents with annotations from Pubmed given a list of Pubmed IDs (PMIDs)

```
>>> corpus = load(19894120)
>>> len(corpus.documents)
1
```

Parameters **pmids** (*List of ints*) – the list of Pubmed IDs

Returns a kindred corpus object

Return type *kindred.Corpus*

19.5 Essential functions

load	Load a corpus from a variety of formats.
iterLoad	Iteratively load sections of a (presumably large) corpus.
save	Save a corpus to a directory
evaluate	Compares the gold corpus with the test corpus and calculate appropriate metrics.

Continued on next page

Table 5 – continued from previous page

manuallyAnnotate	Provides a method for basic manual annotation of a series of candidate relations.

19.5.1 kindred.load

kindred.**load**(*dataFormat*, *path*, *ignoreEntities=[]*, *ignoreComplexRelations=True*)

Load a corpus from a variety of formats. If path is a directory, it will try to load all files of the corresponding data type. For standoff format, it will use any associated annotations files (with suffixes .ann, .a1 or .a2)

Parameters

- **dataFormat** (*str*) – Format of the data files to load ('standoff','biocxml','pubannotation','simpletag')
- **path** (*str*) – Path to data. Can be directory or an individual file. Should be the txt file for standoff.
- **ignoreEntities** (*list*) – List of entity types to ignore while loading
- **ignoreComplexRelations** (*bool*) – Whether to filter out relations where one argument is another relation (must be True as kindred doesn't currently support complex relations)

Returns Corpus of loaded documents

Return type *kindred.Corpus*

19.5.2 kindred.iterLoad

kindred.**iterLoad**(*dataFormat*, *path*, *corpusSizeCutoff=500*)

Iteratively load sections of a (presumably large) corpus. This will create a generator that provides kindred.Corpus objects that are subsets of the larger corpus. This should be used to lower the memory requirements (so that the entire file doesn't need to be loaded into memory at one time).

Parameters

- **dataFormat** (*str*) – Format of the data files to load (only 'biocxml' is currently supported)
- **path** (*str*) – Path to data. Can be directory or an individual file (for bioc, json or simpletag)
- **corpusSizeCutoff** (*int*) – Approximate maximum number of documents to be in each corpus subset

Returns Subsets of the BioC file

Return type A kindred.Corpus generator

19.5.3 kindred.save

kindred.**save**(*corpus*, *dataFormat*, *path*)

Save a corpus to a directory

Parameters

- **corpus** (kindred.Corpus) – The corpus of documents to save

- **dataFormat** (*str*) – Format of data to save (only 'standoff', 'biocxml', 'pubannotation' and 'csv' are supported currently)
- **path** (*str*) – Path where corpus should be saved. Must be an existing directory for 'standoff'.

19.5.4 kindred.evaluate

kindred.**evaluate**(*goldCorpus*, *testCorpus*, *metric='f1score'*, *display=False*)

Compares the gold corpus with the test corpus and calculate appropriate metrics.

Parameters

- **goldCorpus** (kindred.Corpus) – The gold standard set of data
- **testCorpus** (kindred.Corpus) – The test set for comparison
- **metric** (*str*) – Which metric to use (precision/recall/f1score). 'all' will provide all three as a tuple
- **display** (*bool*) – Whether to print (to stdout) specific statistics for each relation type

Returns The value of the corresponding metric (or metrics)

Return type float (or tuple of floats)

19.5.5 kindred.manuallyAnnotate

kindred.**manuallyAnnotate**(*corpus*, *candidateRelations*)

Provides a method for basic manual annotation of a series of candidate relations. Deals with a corpus, sentence by sentence, and prompts the user to annotate each candidate relation in turn. Can be exited before completion of the full list and the resulting annotations are split into an annotated corpus and unannotated corpus. Each document in the new corpora are individual sentences.

Parameters

- **corpus** (kindred.Corpus) – Corpus of text for annotation
- **candidateRelations** (*List of kindred.CandidateRelation*) – List of candidate relations (created using CandidateBuilder) to manually review and annotate

Returns a tuple of an annotated corpus and unannotated corpus

Return type two kindred.Corpus

Python Module Index

k

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

V