spaczz

Grant Andersen

May 01, 2023

CONTENTS

CHAPTER

ONE

REFERENCE

1.1 spaczz.matcher

Module for matchers.

class spaczz.matcher.**FuzzyMatcher**(*vocab*, ***defaults*)

spaCy-like matcher for finding fuzzy phrase matches in *Doc* objects.

Fuzzy matches patterns against the *Doc* it is called on. Accepts labeled patterns in the form of *Doc* objects with optional, per-pattern match settings.

name

Class attribute - the name of the matcher.

Type str

defaults

Keyword arguments to be used as default match settings. Per-pattern match settings take precedence over defaults.

Type dict[str, bool|int|str|Literal['default', 'min', 'max']]

Match Settings

- **ignore_case** (*bool*) – Whether to lower-case text before matching. Default is *True*.
- **min_r** (*int*) – Minimum match ratio required.
- **thresh** (*int*) – If this ratio is exceeded in initial scan, and *flex* > *0*, no optimization will be attempted. If *flex* == *0*, *thresh* has no effect. Default is *100*.
- **fuzzy_func** (*str*) – Key name of fuzzy matching function to use. All rapidfuzz matching functions with default settings are available. Additional fuzzy matching functions can be registered by users. Included functions are:

- *"simple"* = *ratio*
- *"partial"* = *partial_ratio*
- *"token"* = *token_ratio*
- *"token_set"* = *token_set_ratio*
- *"token_sort"* = *token_sort_ratio*
- *"partial_token"* = *partial_token_ratio*
- *"partial_token_set"* = *partial_token_set_ratio*
- *"partial_token_sort"* = *partial_token_sort_ratio*
- *"weighted"* = *WRatio*
- *"quick"* = *QRatio*
- *"partial_alignment"* = *partial_ratio_alignment* (Requires *rapidfuzz>=2.0.3*)

Default is *"simple"*.

- **flex** (*int|Literal['default', 'min', 'max']*) – Number of tokens to move match boundaries left and right during optimization. Can be an *int* with a max of *len(pattern)* and a min of *0*, (will warn and change if higher or lower). *"max"*, *"min"*, or *"default"* are also valid. Default is *"default"*: *len(pattern) // 2*.
- **min_r1** (*int|None*) – Optional granular control over the minimum match ratio required for selection during the initial scan. If *flex == 0*, *min_r1* will be overwritten by *min_r2*. If *flex > 0*, *min_r1* must be lower than *min_r2* and "low" in general because match boundaries are not flexed initially. Default is *None*, which will result in *min_r1* being set to *round(min_r / 1.5)*.
- **min_r2** (*int|None*) – Optional granular control over the minimum match ratio required for selection during match optimization. Needs to be higher than *min_r1* and "high" in general to ensure only quality matches are returned. Default is *None*, which will result in *min_r2* being set to *min_r*.

`__call__`(*doc*)

Finds matches in *doc* given the matchers patterns.

Parameters **doc** (`Doc`) – The *Doc* object to match over.

Return type `List[Tuple[str, int, int, int, str]]`

Returns A list of *MatchResult* tuples, (label, start index, end index, match ratio, pattern).

Example

```
>>> import spacy
>>> from spaczz.matcher import FuzzyMatcher
>>> nlp = spacy.blank("en")
>>> matcher = FuzzyMatcher(nlp.vocab)
>>> doc = nlp("Rdley Scott was the director of Alien.")
>>> matcher.add("NAME", [nlp.make_doc("Ridley Scott")])
>>> matcher(doc)
[('NAME', 0, 2, 96, 'Ridley Scott')]
```

__contains__(*label*)
Whether the matcher contains patterns for a label.

Return type bool

__len__()
The number of labels added to the matcher.

Return type int

__reduce__()
Interface for pickling the matcher.

Return type Tuple[Any, Any]

add(*label*, *patterns*, *kwargs=None*, *on_match=None*)
Add a rule to the matcher, consisting of a label and one or more patterns.

Patterns must be a list of *Doc* objects and if *kwargs* is not *None*, *kwargs* must be a list of dicts.

Parameters

- **label** (str) – Name of the rule added to the matcher.
- **patterns** (List[Doc]) – *Doc* objects that will be matched against the *Doc* object the matcher is called on.
- **kwargs** (Optional[List[Dict[str, Any]]]) – Optional settings to modify the matching behavior. If supplying *kwargs*, one per pattern should be included. Empty dicts will use the matcher instances default settings. Default is *None*.
- **on_match** (Optional[Callable[[TypeVar(PMT, bound= PhraseMatcher), Doc, int, List[Tuple[str, int, int, int, str]]], None]]) – Optional callback function to modify the *Doc* object the matcher is called on after matching. Default is *None*.

Warning:

KwargsWarning:

- If there are more patterns than kwargs default matching settings will be used for extra patterns.
- If there are more kwargs dicts than patterns, the extra kwargs will be ignored.

Example

```
>>> import spacy
>>> from spaczz.matcher import FuzzyMatcher
>>> nlp = spacy.blank("en")
>>> matcher = FuzzyMatcher(nlp.vocab)
>>> matcher.add("SOUND", [nlp.make_doc("mooo")])
>>> "SOUND" in matcher
True
```

Return type None

property labels: Tuple[str, ...]
All labels present in the matcher.

Return type Tuple[str, ...]

Returns The unique labels as a tuple of strings.

Example

```
>>> import spacy
>>> from spaczz.matcher import FuzzyMatcher
>>> nlp = spacy.blank("en")
>>> matcher = FuzzyMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [nlp.make_doc("Kerouac")])
>>> matcher.labels
('AUTHOR',)
```

property patterns: List[Dict[str, Any]]

Get all patterns and match settings that were added to the matcher.

Return type List[Dict[str, Any]]

Returns The patterns and their respective match settings as a list of dicts.

Example

```
>>> import spacy
>>> from spaczz.matcher import FuzzyMatcher
>>> nlp = spacy.blank("en")
>>> matcher = FuzzyMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [nlp.make_doc("Kerouac")],
    [{"ignore_case": False}])
>>> matcher.patterns == [
    {
        "label": "AUTHOR",
        "pattern": "Kerouac",
        "type": "fuzzy",
        "kwargs": {"ignore_case": False}
        },
        ]
True
```

remove(*label*)

Remove a label and its respective patterns from the matcher.

Parameters **label** (str) – Name of the rule added to the matcher.

Example

```
>>> import spacy
>>> from spaczz.matcher import FuzzyMatcher
>>> nlp = spacy.blank("en")
>>> matcher = FuzzyMatcher(nlp.vocab)
>>> matcher.add("SOUND", [nlp.make_doc("mooo")])
>>> matcher.remove("SOUND")
>>> "SOUND" in matcher
False
```

Return type None

property type: Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']
Getter for the matchers *SpaczzType*.

Return type Literal[‘fuzzy’, ‘regex’, ‘token’, ‘similarity’, ‘phrase’]

property vocab: spacy.vocab.Vocab
Getter for the matchers *Vocab*.

Return type Vocab

class spaczz.matcher.**RegexMatcher**(*vocab*, ***defaults*)
spaCy-like matcher for finding regex phrase matches in *Doc* objects.

Regex matches patterns against the *Doc* it is called on. Accepts labeled patterns in the form of strings with optional, per-pattern match settings.

To utilize regex flags, use inline flags.

name
Class attribute - the name of the matcher.

Type str

defaults
Keyword arguments to be used as default match settings. Per-pattern match settings take precedence over defaults.

Type dict[str, bool|int|str]

Match Settings

- **ignore_case** (*bool*) – Whether to lower-case text before matching. Default is *True*.
- **min_r** (*int*) – Minimum match ratio required.
- **fuzzy_weights** (*str*) – Name of weighting method for regex insertion, deletion, and substituion counts. Additional weighting methods can be registered by users. Included weighting methods are:
  - *“indel” = (1, 1, 2)*
  - *“lev” = (1, 1, 1)*

  Default is *“indel”*.
- **partial** – (bool): Whether partial matches should be extended to *Token* or *Span* boundaries in *doc* or not. For example, the regex only matches part of a *Token* or *Span* in *doc*. Default is *True*.

- **predef** (*string*) – Whether the regex string should be interpreted as a key to a predefined regex pattern or not. Additional predefined regex patterns can be registered by users. The included predefined regex patterns are:
  - *"dates"*
  - *"times"*
  - *"phones"*
  - *"phones_with_exts"*
  - *"links"*
  - *"emails"*
  - *"ips"*
  - *"ipv6s"*
  - *"prices"*
  - *"hex_colors"*
  - *"credit_cards"*
  - *"btc_addresses"*
  - *"street_addresses"*
  - *"zip_codes"*
  - *"po_boxes"*
  - *"ssn_numbers"*

  Default is *False*.

__call__(*doc*)

Finds matches in *doc* given the matchers patterns.

Parameters **doc** (Doc) – The *Doc* object to match over.

Return type List[Tuple[str, int, int, int, str]]

Returns A list of *MatchResult* tuples, (label, start index, end index, match ratio, pattern).

Example

```
>>> import spacy
>>> from spaczz.matcher import RegexMatcher
>>> nlp = spacy.blank("en")
>>> matcher = RegexMatcher(nlp.vocab)
>>> doc = nlp.make_doc("I live in the united states, or the US")
>>> matcher.add("GPE", ["[Uu](nited|\.?) ?[Ss](tates|\.?)"])
>>> matcher(doc)[0]
('GPE', 4, 6, 100, '[Uu](nited|\\.?) ?[Ss](tates|\\.?)')
```

__contains__(*label*)

Whether the matcher contains patterns for a label.

Return type bool

__len__()
The number of labels added to the matcher.

Return type int

__reduce__()
Interface for pickling the matcher.

Return type Tuple[Any, Any]

add(*label*, *patterns*, *kwargs=None*, *on_match=None*)
Add a rule to the matcher, consisting of a label and one or more patterns.

Patterns must be a list of *Doc* objects and if *kwargs* is not *None*, *kwargs* must be a list of dicts.

Parameters

- **label** (str) – Name of the rule added to the matcher.
- **patterns** (List[str]) – *Doc* objects that will be matched against the *Doc* object the matcher is called on.
- **kwargs** (Optional[List[Dict[str, Any]]]) – Optional settings to modify the matching behavior. If supplying *kwargs*, one per pattern should be included. Empty dicts will use the matcher instances default settings. Default is *None*.
- **on_match** (Optional[Callable[[*RegexMatcher*, Doc, int, List[Tuple[str, int, int, int, str]]], None]]) – Optional callback function to modify the *Doc* object the matcher is called on after matching. Default is *None*.

Raises

- **TypeError** – If *patterns* is not a list of strings.
- **TypeError** – If *kwargs* is not a list of dictionaries.

Warning:

KwargsWarning:

- If there are more patterns than kwargs default matching settings will be used for extra patterns.
- If there are more kwargs dicts than patterns, the extra kwargs will be ignored.

Example

```
>>> import spacy
>>> from spaczz.matcher import RegexMatcher
>>> nlp = spacy.blank("en")
>>> matcher = RegexMatcher(nlp.vocab)
>>> matcher.add("GPE", ["[Uu](nited|\.?) ?[Ss](tates|\.?)"])
>>> "GPE" in matcher
True
```

Return type None

property labels: Tuple[str, ...]
All labels present in the matcher.

Return type Tuple[str, ...]

Returns The unique labels as a tuple of strings.

Example

```
>>> import spacy
>>> from spaczz.matcher import RegexMatcher
>>> nlp = spacy.blank("en")
>>> matcher = RegexMatcher(nlp.vocab)
>>> matcher.add("ZIP", ["zip_codes"], [{"predef": True}])
>>> matcher.labels
('ZIP',)
```

property patterns: List[Dict[str, Any]]

Get all patterns and match settings that were added to the matcher.

Return type List[Dict[str, Any]]

Returns The patterns and their respective match settings as a list of dicts.

Example

```
>>> import spacy
>>> from spaczz.matcher import RegexMatcher
>>> nlp = spacy.blank("en")
>>> matcher = RegexMatcher(nlp.vocab)
>>> matcher.add("ZIP", ["zip_codes"], [{"predef": True}])
>>> matcher.patterns == [
    {
        "label": "ZIP",
        "pattern": "zip_codes",
        "type": "regex",
        "kwargs": {"predef": True},
        }
        ]
True
```

remove(*label*)

Remove a label and its respective patterns from the matcher.

Parameters **label** (str) – Name of the rule added to the matcher.

Raises **ValueError** – If *label* does not exist in the matcher.

Example

```
>>> import spacy
>>> from spaczz.matcher import RegexMatcher
>>> nlp = spacy.blank("en")
>>> matcher = RegexMatcher(nlp.vocab)
>>> matcher.add("GPE", ["[Uu](nited|\.?) ?[Ss](tates|\.?)"])
>>> matcher.remove("GPE")
>>> "GPE" in matcher
False
```

Return type None

property type: Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']
Getter for the matchers *SpaczzType*.

Return type Literal[‘fuzzy’, ‘regex’, ‘token’, ‘similarity’, ‘phrase’]

property vocab: spacy.vocab.Vocab
Getter for the matchers *Vocab*.

Return type Vocab

class spaczz.matcher.**SimilarityMatcher**(*vocab*, ***defaults*)
spaCy-like matcher for finding phrase similarity matches in *Doc* objects.

Similarity matches patterns against the *Doc* it is called on. Accepts labeled patterns in the form of *Doc* objects with optional, per-pattern match settings.

Similarity matching uses spaCy word vectors if available, therefore spaCy vocabs without word vectors may not produce useful results. The spaCy medium and large English models provide word vectors that will work for this purpose.

Searching over/with *Doc* objects that do not have vectors will always return a similarity score of 0.

Warnings from spaCy about the above two scenarios are suppressed for convenience. However, spaczz will still warn about the former.

name
Class attribute - the name of the matcher.

Type str

defaults
Keyword arguments to be used as default match settings. Per-pattern match settings take precedence over defaults.

Type dict[str, bool|int|str|Literal[‘default’, ‘min’, ‘max’]]

Match Settings

- **ignore_case** (*bool*) – Whether to lower-case text before fuzzy matching. Default is *True*.
- **min_r** (*int*) – Minimum match ratio required.
- **thresh** (*int*) – If this ratio is exceeded in initial scan, and *flex* > *0*, no optimization will be attempted. If *flex* == *0*, *thresh* has no effect. Default is *100*.
- **flex** (*int|Literal[‘default’, ‘min’, ‘max’]*) – Number of tokens to move match boundaries left and right during optimization. Can be an *int* with a max of *len(pattern)* and a min of *0*, (will

warn and change if higher or lower). *"max"*, *"min"*, or *"default"* are also valid. Default is *"default"*: *len(pattern) // 2*.

- **min_r1** (*int|None*) – Optional granular control over the minimum match ratio required for selection during the initial scan. If *flex == 0*, *min_r1* will be overwritten by *min_r2*. If *flex > 0*, *min_r1* must be lower than *min_r2* and "low" in general because match boundaries are not flexed initially. Default is *None*, which will result in *min_r1* being set to *round(min_r / 1.5)*.
- **min_r2** (*int|None*) – Optional granular control over the minimum match ratio required for selection during match optimization. Needs to be higher than *min_r1* and "high" in general to ensure only quality matches are returned. Default is *None*, which will result in *min_r2* being set to *min_r*.

Warning:

MissingVectorsWarning: If *vocab* does not contain any word vectors.

`__call__`(*doc*)

Finds matches in *doc* given the matchers patterns.

Parameters **doc** (Doc) – The *Doc* object to match over.

Return type List[Tuple[str, int, int, int, str]]

Returns A list of *MatchResult* tuples, (label, start index, end index, match ratio, pattern).

Example

```
>>> import spacy
>>> from spaczz.matcher import SimilarityMatcher
>>> nlp = spacy.load("en_core_web_md")
>>> matcher = SimilarityMatcher(nlp.vocab)
>>> doc = nlp("I like apples.")
>>> matcher.add("FRUIT", [nlp("fruit")], [{'min_r': 60}])
>>> matcher(doc)
[('FRUIT', 2, 3, 70, 'fruit')]
```

`__contains__`(*label*)

Whether the matcher contains patterns for a label.

Return type bool

`__len__`()

The number of labels added to the matcher.

Return type int

`__reduce__`()

Interface for pickling the matcher.

Return type Tuple[Any, Any]

add(*label*, *patterns*, *kwargs=None*, *on_match=None*)

Add a rule to the matcher, consisting of a label and one or more patterns.

Patterns must be a list of *Doc* objects and if *kwargs* is not *None*, *kwargs* must be a list of dicts.

Parameters

- **label** (str) – Name of the rule added to the matcher.
- **patterns** (List[Doc]) – *Doc* objects that will be matched against the *Doc* object the matcher is called on.
- **kwargs** (Optional[List[Dict[str, Any]]]) – Optional settings to modify the matching behavior. If supplying *kwargs*, one per pattern should be included. Empty dicts will use the matcher instances default settings. Default is *None*.
- **on_match** (Optional[Callable[[TypeVar(PMT, bound= PhraseMatcher), Doc, int, List[Tuple[str, int, int, int, str]]], None]]) – Optional callback function to modify the *Doc* object the matcher is called on after matching. Default is *None*.

Warning:

KwargsWarning:

- If there are more patterns than kwargs default matching settings will be used for extra patterns.
- If there are more kwargs dicts than patterns, the extra kwargs will be ignored.

Example

```
>>> import spacy
>>> from spaczz.matcher import SimilarityMatcher
>>> nlp = spacy.load("en_core_web_md")
>>> matcher = SimilarityMatcher(nlp.vocab)
>>> matcher.add("SOUND", [nlp("mooo")])
>>> "SOUND" in matcher
True
```

Return type None

property labels: Tuple[str, ...]

All labels present in the matcher.

Return type Tuple[str, ...]

Returns The unique labels as a tuple of strings.

Example

```
>>> import spacy
>>> from spaczz.matcher import SimilarityMatcher
>>> nlp = spacy.load("en_core_web_md")
>>> matcher = SimilarityMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [nlp("Kerouac")])
>>> matcher.labels
('AUTHOR',)
```

property patterns: List[Dict[str, Any]]

Get all patterns and match settings that were added to the matcher.

Return type List[Dict[str, Any]]

Returns The patterns and their respective match settings as a list of dicts.

Example

```
>>> import spacy
>>> from spaczz.matcher import SimilarityMatcher
>>> nlp = spacy.load("en_core_web_md")
>>> matcher = SimilarityMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [nlp("Kerouac")],
    [{"ignore_case": False}])
>>> matcher.patterns == [
    {
        "label": "AUTHOR",
        "pattern": "Kerouac",
        "type": "similarity",
        "kwargs": {"ignore_case": False}
        },
        ]
True
```

remove(*label*)

Remove a label and its respective patterns from the matcher.

Parameters **label** (str) – Name of the rule added to the matcher.

Example

```
>>> import spacy
>>> from spaczz.matcher import SimilarityMatcher
>>> nlp = spacy.load("en_core_web_md")
>>> matcher = SimilarityMatcher(nlp.vocab)
>>> matcher.add("SOUND", [nlp("mooo")])
>>> matcher.remove("SOUND")
>>> "SOUND" in matcher
False
```

Return type None

property type: Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']

Getter for the matchers *SpaczzType*.

Return type Literal[‘fuzzy’, ‘regex’, ‘token’, ‘similarity’, ‘phrase’]

property vocab: spacy.vocab.Vocab

Getter for the matchers *Vocab*.

Return type Vocab

class spaczz.matcher.**TokenMatcher**(*vocab*, ***defaults*)

spaCy-like matcher for finding fuzzy token matches in *Doc* objects.

Fuzzy matches added patterns against the *Doc* object it is called on. Accepts labeled patterns in the form of lists of dictionaries where each list describes an individual pattern and each dictionary describes an individual token.

Uses extended spaCy token matching patterns. "FUZZY" and "FREGEX" are the two additional spaCy token pattern options.

For example:

```
[
    {"TEXT": {"FREGEX": "(database){e<=1}"}},
    {"LOWER": {"FUZZY": "access", "MIN_R": 85, "FUZZY_FUNC": "partial"}},
]
```

Make sure to use uppercase dictionary keys in patterns.

name
Class attribute - the name of the matcher.

Type str

defaults
Keyword arguments to be used as default match settings. Per-pattern match settings take precedence over defaults.

Type dict[str, bool|int|str]

Match Settings

- **ignore_case** (*bool*) – Whether to lower-case text before matching. Can only be set at the pattern level. For "FUZZY" and "FREGEX" patterns. Default is *True*.
- **min_r** (*int*) – Minimum match ratio required. For "FUZZY" and "FREGEX" patterns.
- **fuzzy_func** (*str*) – Key name of fuzzy matching function to use. Can only be set at the pattern level. For "FUZZY" patterns only. All rapidfuzz matching functions with default settings are available, however any token-based functions provide no utility at the individual token level. Additional fuzzy matching functions can be registered by users. Included, and useful, functions are:
  - *"simple" = ratio*
  - *"partial" = partial_ratio*
  - *"quick" = QRatio*
  - ***"partial_alignment" = partial_ratio_alignment*** (Requires *rapidfuzz>=2.0.3*)

  Default is *"simple"*.
- **fuzzy_weights** – Name of weighting method for regex insertion, deletion, and substituion counts. Can only be set at the pattern level. For "FREGEX" patterns only. Included weighting methods are:
  - *"indel" = (1, 1, 2)*
  - *"lev" = (1, 1, 1)*

  Default is *"indel"*.
- **predef** – Whether regex should be interpreted as a key to a predefined regex pattern or not. Can only be set at the pattern level. For "FREGEX" patterns only. Default is *False*.

__call__(*doc*)
Finds matches in *doc* given the matchers patterns.

Parameters **doc** (Doc) – The *Doc* object to match over.

Return type List[Tuple[str, int, int, int, str]]

Returns A list of *MatchResult* tuples, (label, start index, end index, match ratio, pattern).

Example

```
>>> import spacy
>>> from spaczz.matcher import TokenMatcher
>>> nlp = spacy.blank("en")
>>> matcher = TokenMatcher(nlp.vocab)
>>> doc = nlp("Rdley Scot was the director of Alien.")
>>> matcher.add("NAME", [
    [{"TEXT": {"FUZZY": "Ridley"}},
    {"TEXT": {"FUZZY": "Scott"}}]
    ])
>>> matcher(doc)[0][:4]
('NAME', 0, 2, 90)
```

__contains__(*label*)
Whether the matcher contains patterns for a label.

Return type bool

__len__()
The number of labels added to the matcher.

Return type int

__reduce__()
Interface for pickling the matcher.

Return type Tuple[Any, Any]

add(*label*, *patterns*, *on_match=None*)
Add a rule to the matcher, consisting of a label and one or more patterns.

Patterns must be a list of lists of dicts where each list of dicts represent an individual pattern and each dictionary represents an individual token.

Uses extended spaCy token matching patterns. "FUZZY" and "FREGEX" are the two additional spaCy token pattern options.

For example:

```
[
    {"TEXT": {"FREGEX": "(database){e<=1}"}},
    {"LOWER": {"FUZZY": "access", "MIN_R": 85, "FUZZY_FUNC": "partial"}},
]
```

Make sure to use uppercase dictionary keys in patterns.

Parameters

- **label** (str) – Name of the rule added to the matcher.
- **patterns** (List[List[Dict[str, Any]]]) – List of lists of dicts that will be matched against the *Doc* object the matcher is called on.

- **on_match** (Optional[Callable[[*TokenMatcher*, Doc, int, List[Tuple[str, int, int, int, str]]], None]]) – Optional callback function to modify the *Doc* object the matcher is called on after matching. Default is *None*.

Raises

- **TypeError** – If patterns is not a list of *Doc* objects.
- **ValueError** – Patterns cannot have zero tokens.

Example

```
>>> import spacy
>>> from spaczz.matcher import TokenMatcher
>>> nlp = spacy.blank("en")
>>> matcher = TokenMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [[{"TEXT": {"FUZZY": "Kerouac"}}]])
>>> "AUTHOR" in matcher
True
```

Return type None

property labels: Tuple[str, ...]

All labels present in the matcher.

Return type Tuple[str, ...]

Returns The unique labels as a tuple of strings.

Example

```
>>> import spacy
>>> from spaczz.matcher import TokenMatcher
>>> nlp = spacy.blank("en")
>>> matcher = TokenMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [[{"TEXT": {"FUZZY": "Kerouac"}}]])
>>> matcher.labels
('AUTHOR',)
```

property patterns: List[Dict[str, Any]]

Get all patterns and match settings that were added to the matcher.

Return type List[Dict[str, Any]]

Returns The patterns and their respective match settings as a list of dicts.

Example

```
>>> import spacy
>>> from spaczz.matcher import TokenMatcher
>>> nlp = spacy.blank("en")
>>> matcher = TokenMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [[{"TEXT": {"FUZZY": "Kerouac"}}]])
>>> matcher.patterns == [
    {
        "label": "AUTHOR",
        "pattern": [{"TEXT": {"FUZZY": "Kerouac"}}],
        "type": "token",
        },
        ]
True
```

remove(*label*)

Remove a label and its respective patterns from the matcher.

Parameters **label** (str) – Name of the rule added to the matcher.

Raises **ValueError** – If label does not exist in the matcher.

Example

```
>>> import spacy
>>> from spaczz.matcher import TokenMatcher
>>> nlp = spacy.blank("en")
>>> matcher = TokenMatcher(nlp.vocab)
>>> matcher.add("AUTHOR", [[{"TEXT": {"FUZZY": "Kerouac"}}]])
>>> matcher.remove("AUTHOR")
>>> "AUTHOR" in matcher
False
```

Return type None

property type: Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']

Getter for the matchers *SpaczzType*.

Return type Literal[‘fuzzy’, ‘regex’, ‘token’, ‘similarity’, ‘phrase’]

property vocab: spacy.vocab.Vocab

Getter for the matchers *Vocab*.

Return type Vocab

1.2 spaczz.pipeline

Module for pipeline components.

class spaczz.pipeline.**SpaczzRuler**(*nlp*, *name='spaczz_ruler'*, ***, *overwrite_ents=False*, *ent_id_sep='||'*, *fuzzy_defaults={}*, *regex_defaults={}*, *token_defaults={}*, *patterns=None*, *scorer=<function spaczz_ruler_scorer>*)

The *SpaczzRuler* adds fuzzy matches to spaCy *Doc.ents*.

It can be combined with other spaCy NER components like the statistical *EntityRecognizer*, and/or the *EntityRuler* it is inspired by, to boost accuracy. After initialization, the component is typically added to the pipeline using *nlp.add_pipe*.

nlp
The shared *Language* object that passes its *Vocab* to the matchers (not currently used by spaczz matchers) and processes fuzzy patterns.

Type Language

name
Instance name of the current pipeline component. Typically passed in automatically from the factory when the component is added. Used to disable the current entity ruler while creating phrase patterns with the nlp object.

Type str

overwrite_ents
If existing entities are present, e.g. entities added by the model, overwrite them by matches if necessary.

Type bool

ent_id_sep
Separator used internally for entity IDs.

Type str

scorer
The scoring method for the ruler.

Type Optional[Callable]

fuzzy_matcher
The *FuzzyMatcher* instance the spaczz ruler will use for fuzzy phrase matching.

Type *FuzzyMatcher*

regex_matcher
The *RegexMatcher* instance the spaczz ruler will use for regex phrase matching.

Type *RegexMatcher*

token_matcher
The *TokenMatcher* instance the spaczz ruler will use for fuzzy token matching.

Type *TokenMatcher*

defaults
Default match settings for their respective matchers.

Type Dict[str, Any]

__call__(*doc*)
Find matches in document and add them as entities.

Parameters **doc** (Doc) – The *Doc* object in the pipeline.

Return type Doc

Returns The *Doc* with added entities, if available.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> doc = nlp.make_doc("My name is Anderson, Grunt")
>>> ruler.add_patterns([{"label": "NAME", "pattern": "Grant Andersen",
    "type": "fuzzy", "kwargs": {"fuzzy_func": "token_sort"}}])
>>> doc = ruler(doc)
>>> "Anderson, Grunt" in [ent.text for ent in doc.ents]
True
```

__contains__(*label*)

Whether a label is present in the patterns.

Return type bool

__len__()

The number of all patterns added to the ruler.

Return type int

add_patterns(*patterns*)

Add patterns to the ruler.

A pattern must be a spaczz pattern: *{label (str), pattern (str or list), type (str), optional kwargs (dict[str, Any]), and optional id (str)}*.

For example, a fuzzy phrase pattern:

```
{
    'label': 'ORG',
    'pattern': 'Apple',
    'kwargs': {'min_r2': 90},
    'type': 'fuzzy',
}
```

Or, a token pattern:

```
{
    'label': 'ORG',
    'pattern': [{'TEXT': {'FUZZY': 'Apple'}}],
    'type': 'token',
}
```

To utilize regex flags, use inline flags.

Parameters **patterns** (List[Dict[str, Union[str, Dict[str, Any], List[Dict[str, Any]]]]]) – The spaczz patterns to add.

Raises

- **TypeError** – If *patterns* is not a list of dicts.
- **ValueError** – If one or more patterns do not conform the spaczz pattern structure.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> "AUTHOR" in ruler.labels
True
```

Return type None

clear()
Reset all patterns.

Return type None

property ent_ids: Tuple[Optional[str], ...]
All entity ids present in the match patterns id properties.

Return type Tuple[Optional[str], ...]

Returns The unique string entity ids as a tuple.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy", "id": "BEAT"}])
>>> ruler.ent_ids
('BEAT',)
```

from_bytes(*patterns_bytes*, *, *exclude=[]*)
Load the spaczz ruler from a bytestring.

Parameters

- **patterns_bytes** (bytes) – The bytestring to load.
- **exclude** (Iterable[str]) – For spaCy consistency.

Return type *SpaczzRuler*

Returns The loaded spaczz ruler.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> ruler_bytes = ruler.to_bytes()
>>> new_ruler = SpaczzRuler(nlp)
>>> new_ruler = new_ruler.from_bytes(ruler_bytes)
>>> "AUTHOR" in new_ruler
True
```

from_disk(*path*, *, *exclude=[]*)

Load the spaczz ruler from a file.

Expects a file containing newline-delimited JSON (JSONL) with one entry per line.

Parameters

- **path** (Union[str, Path]) – The JSONL file to load.
- **exclude** (Iterable[str]) – For spaCy consistency.

Return type *SpaczzRuler*

Returns The loaded spaczz ruler.

Raises **ValueError** – If *path* does not exist or cannot be accessed.

Example

```
>>> import os
>>> import tempfile
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> with tempfile.TemporaryDirectory() as tmpdir:
>>>     ruler.to_disk(f"{tmpdir}/ruler")
>>>     new_ruler = SpaczzRuler(nlp)
>>>     new_ruler = new_ruler.from_disk(f"{tmpdir}/ruler")
>>> "AUTHOR" in new_ruler
True
```

initialize(*get_examples*, *, *nlp=None*, *patterns=None*)

Initialize the pipe for training.

Parameters

- **get_examples** (Callable[[], Iterable[Example]]) – Function that returns a representative sample of gold-standard Example objects.
- **nlp** (Optional[Language]) – The current nlp object the component is part of.

- **patterns** (Optional[Sequence[Dict[str, Union[str, Dict[str, Any], List[Dict[str, Any]]]]]]) – The list of patterns.

Return type None

property labels: Tuple[str, ...]

All labels present in the ruler.

Return type Tuple[str, ...]

Returns The unique string labels as a tuple.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> ruler.labels
('AUTHOR',)
```

match(*doc*)

Used in call to find matches in *doc*.

Return type List[Tuple[str, int, int, int, str, Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']]]

property patterns: List[Dict[str, Union[str, Dict[str, Any], List[Dict[str, Any]]]]]

Get all patterns and kwargs that were added to the ruler.

Return type List[Dict[str, Union[str, Dict[str, Any], List[Dict[str, Any]]]]]

Returns The original patterns and kwargs, one dictionary for each combination.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "STREET", "pattern": "street_addresses",
    "type": "regex", "kwargs": {"predef": True}}])
>>> ruler.patterns == [
    {
        "label": "STREET",
        "pattern": "street_addresses",
        "type": "regex",
        "kwargs": {"predef": True},
        },
        ]
True
```

remove(*ent_id*)
Remove patterns by their *ent_id*.

Return type None

score(*examples*, ***kwargs*)
Pipeline scoring for spaCy >= 3.0, < 3.2 compatibility.

Return type Any

set_annotations(*doc*, *matches*)
Modify the document in place.

Return type None

to_bytes(*, *exclude=[]*)
Serialize the spaczz ruler patterns to a bytestring.

Parameters **exclude** (Iterable[str]) – For spaCy consistency.

Return type bytes

Returns The serialized patterns.

Example

```
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> ruler_bytes = ruler.to_bytes()
>>> isinstance(ruler_bytes, bytes)
True
```

to_disk(*path*, *, *exclude=[]*)
Save the spaczz ruler patterns to a directory.

The patterns will be saved as newline-delimited JSON (JSONL).

Parameters

- **path** (Union[str, Path]) – The JSONL file to save.
- **exclude** (Iterable[str]) – For spaCy consistency.

Example

```
>>> import os
>>> import tempfile
>>> import spacy
>>> from spaczz.pipeline import SpaczzRuler
>>> nlp = spacy.blank("en")
>>> ruler = SpaczzRuler(nlp)
>>> ruler.add_patterns([{"label": "AUTHOR", "pattern": "Kerouac",
    "type": "fuzzy"}])
>>> with tempfile.TemporaryDirectory() as tmpdir:
```

(continues on next page)

(continued from previous page)

```
>>>     ruler.to_disk(f"{tmpdir}/ruler")
>>>     isdir = os.path.isdir(f"{tmpdir}/ruler")
>>> isdir
True
```

Return type None

1.3 spaczz.registry

Function and object registries.

spaczz.registry.**get_fuzzy_func**(*name*)
Get the registered function for a given name.

name (str): The name. RETURNS (Any): The registered function.

Return type Any

spaczz.registry.**get_re_pattern**(*name*)
Get the registered function for a given name.

name (str): The name. RETURNS (Any): The registered function.

Return type Any

spaczz.registry.**get_re_weights**(*name*)
Get the registered function for a given name.

name (str): The name. RETURNS (Any): The registered function.

Return type Any

1.4 spaczz.customattrs

Custom spaCy attributes for spaczz.

class spaczz.customattrs.**SpaczzAttrs**
Adds spaczz custom attributes to spaCy.

static get_doc_types(*doc*)
Getter for spaczz_types *Doc* attribute.

Return type Set[Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']]

classmethod get_pattern(*span*)
Getter for spaczz_pattern *Span* attribute.

Return type Optional[str]

classmethod get_ratio(*span*)
Getter for spaczz_ratio *Span* attribute.

Return type Optional[int]

static get_spaczz_doc(*doc*)
Getter for spaczz_doc *Doc* attribute.

Return type bool

static get_spaczz_ent(*span*)
Getter for spaczz_ent *Span* attribute.

Return type bool

classmethod get_span_type(*span*)
Getter for spaczz_type *Span* attribute.

Return type Optional[Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']]

static get_span_types(*span*)
Getter for spaczz_types *Span* attribute.

Return type Set[Literal['fuzzy', 'regex', 'token', 'similarity', 'phrase']]

classmethod initialize()
Initializes and registers custom attributes.

Return type None

1.5 spaczz.customtypes

Custom spaczz types.

1.6 spaczz.exceptions

Module for custom exceptions and warnings.

exception spaczz.exceptions.**AttrOverwriteWarning**
It warns if custom attributes are being overwritten.

exception spaczz.exceptions.**FlexWarning**
It warns if flex value is changed if too large.

exception spaczz.exceptions.**KwargsWarning**
It warns if there are more kwargs than patterns or vice versa.

exception spaczz.exceptions.**MissingVectorsWarning**
It warns if the spaCy Vocab does not have word vectors.

exception spaczz.exceptions.**PatternTypeWarning**
It warns if the spaczz pattern does not have a valid pattern type.

exception spaczz.exceptions.**RatioWarning**
It warns if match ratio values are incompatible with each other.

exception spaczz.exceptions.**RegexParseError**
General error for errors that may happen during regex compilation.

CHAPTER

TWO

LICENSE

MIT License

Copyright (c) 2020 Grant Andersen (gandersen101)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Spaczz provides fuzzy matching and multi-token regex matching functionality to spaCy. Spaczz's components have similar APIs to their spaCy counterparts and spaczz pipeline components can integrate into spaCy pipelines where they can be saved/loaded as models.

While this website will eventually be the home for definitive spaczz documentation, for now it is kind of a placeholder. Please visit spaczz's GitHub page for now to see usage documentation and more.

CHAPTER

THREE

INSTALLATION

Spaczz can be installed using pip.

```
pip install spaczz
```

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

G