iobes: A Library for Span-Level Processing

Brian Lester

To Appear in and be presented at NLPOSS 2020

Span-Level NLP

- The classic NLP/ML problem is unstructured classification
  - Mapping a sequence of tokens to single label
  - Text classification
- The next classic NLP is tagging
  - Mapping a sequence of tokens to a sequence of labels
  - Part of speech tagging
  - A class of problems we solve with taggers
- But often we care about subsequences of tokens
- Tasks like NER or Slot-Filling
  - You want the whole name for people

Span-Level NLP

- Jack White was born in Detroit on July 9th 1975
  - “Jack White” is a person
  - “Detroit” is a location
  - “July 9th 1975” is a date

What are Taggers?

- A sequence transduction task

  `X = (x_1, x_2, …, x_n)`

  `Y = (y_1, y_2, …, y_n)`

- Additional constraint that the input and output sequences have the same length

Span Encoding

B-PER

Each tag is made from two parts

- The second part is the type of the span. A person, location, etc.
- The first part is the function of this token in the span
  - B is the Beginning of the span
  - I is the inside of the span
  - O is outside the span
  - Different encoding schemes have different function prefixes.

Encoding Schemes

- IOB
- BIO
- IOBES
  - BMEWO
  - BILOU

Span Encoding

```
Above-normal O
summer       O
rainfall     O
in           O
the          O
U.S.         I-LOC
High         B-LOC
Plains       I-LOC
```

Span Encoding

Jack	I-PER	B-PER	B-PER
White	I-PER	I-PER	E-PER
was	O	O	O
born	O	O	O
in	O	O	O
Detroit	I-LOC	B-LOC	S-LOC
on	O	O	O
July	I-DATE	B-DATE	B-DATE
9th	I-DATE	I-DATE	I-DATE
1975	I-DATE	I-DATE	E-DATE

Evaluation - Span Level F1

- Predicted
  - “Jack White”, Person, 0 -> 2
  - “Detroit”, Location, 5 -> 6
  - “July 9th”, Date, 7 -> 9
  - “1975”, Date, 9 -> 10

- Gold
  - “Jack White”, Person, 0 -> 2
  - “Detroit”, Location, 5 -> 6
  - “July 9th 1975”, Date 7 -> 10

- The “Jack White” span is correct
- The “Detroit” span is correct
- The “July 9th” span is incorrect
- The “1975” span is incorrect

- Precision: 0.5
- Recall: 0.66
- F1: 0.57

Errors

- What is your policy when the tags don’t follow the encoding scheme?

```
O O O B-LOC I-LOC I-ORG E-LOC O O O
```

- Shift entities when you shift type?
  - LOC 3 -> 5
  - ORG 5 -> 6
  - LOC 6 -> 7
- Wait for the correct ending?
  - LOC 3 -> 7

Our Library

- `iobes`
- Pure Python (python >= 3.6)
- No dependencies
- Property based testing
- Multiple formats
- Multiple APIs
- Documentation

Parsing

- The input to most everything is a list of strings representing the tags
- The result of parsing is our Span data structure

```
class Span(NamedTuple):
    type: str
    start: int
    end: int
    tokens: Tuple[int]
```

Validation

- Given a sequence of tags, verify that the span encoding scheme is followed.
- Returns an `Error` data structure that describe the type of error it found

Conversion

- Convert a tag sequence (`List[str]`) from one span encoding to another (`List[str]`)
- It uses the parsing functions to convert the tokens into a list of Spans and then, based on those, writes out a second list of tags
- When an error is encountered we raise and error, an error means the actual spans are ambiguous so making a choice would be overly prescriptive

Writing

- Given `List[Span]` return a `List[str]` so that these tags would be parsed back to these spans.
- If your sequence doesn’t end with a span (there are outside tokens after the last span) you can pass in a length it will generate out to, otherwise it will make an output just long enough to cover the spans.

Transition Legality

- The encoding schemes actually have rule about what kind of tokens are allowed to touch.
- For example, in the `IOBES` scheme the spans *must* begin with a `B-`, this means that a tag beginning with `I-` cannot follow an `O`

- We provide these legalities in a variety of formats including as a mask (with the numpy optional dependency installed) that is ready to use with a CRF.

APIs

- There are two main ways to interact with the library
  - Dedicated functions
    - `parse_spans_iob`
    - `parse_spans_bio`
    - `bio_to_iobes`
  - Dispatch functions
    - `convert_tags`
    - `parse_spans`
    - Based on the `SpanEncoding` Enum

Code Reuse

- Conversion uses the parsing and writing code
- IOBES, BILOU, and BMEWO all use the same code

```
class SpanFormat:
    BEGIN = None
    INSIDE = None
    END = None
    SINGLE = None
```

```
class IOBES(SpanFormat):
    BEGIN = “B”
    INSIDE = “I”
    END = “E”
    SINGLE = “S”
```

```
if extract_function(token) == span_format.BEGIN:
```

Installation

```
pip install iobes
```

Contact

- The Library
  - Github: https://github.com/blester125/iobes
  - Docs: https://iobes.readthedocs.io
  - PyPI: https://pypi.org/project/iobes/
- Me
  - Twitter: https://twitter.com/blester125
  - Web: https://blester125.com/