spaCy pipelines

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

spaCy pipelines

- `spaCy` first tokenizes the text to produce a `Doc` object
- The `Doc` is processed in several different steps of **processing pipeline**

```
import spacy
nlp = spacy.load("en_core_web_sm")

doc = nlp(example_text)
```

spaCy pipelines

- A **pipeline** is a **sequence of pipes**, or **actors on data**
- A `spaCy` **NER** pipeline:
  - Tokenization
  - Named entity identification
  - Named entity classification


```
print([ent.text for ent in doc.ents])
```

Adding pipes

- `sentencizer` : `spaCy` pipeline component for sentence segmentation.

```
text = " ".join(["This is a test sentence."]*10000)
en_core_sm_nlp = spacy.load("en_core_web_sm")
start_time = time.time()
doc = en_core_sm_nlp(text)
print(f"Finished processing with en_core_web_sm model in
        {round((time.time() - start_time)/60.0 , 5)} minutes")
```

```
>>> Finished processing with en_core_web_sm model in 0.09332 minutes
```

Adding pipes

- Create a blank model and add a `sentencizer` pipe:

```
blank_nlp = spacy.blank("en")
blank_nlp.add_pipe("sentencizer")
start_time = time.time()
doc = blank_nlp(text)
print(f"Finished processing with blank model in
       {round((time.time() - start_time)/60.0 , 5)} minutes")
```

```
>>> Finished processing with blank model in 0.00091 minutes
```

Analyzing pipeline components

- `nlp.analyze_pipes()` analyzes a `spaCy` pipeline to determine:
  - Attributes that pipeline components set
  - Scores a component produces during training
  - Presence of all required attributes

- Setting `pretty` to `True` will print a table instead of only returning the structured data.

```
import spacy

nlp = spacy.load("en_core_web_sm")
analysis = nlp.analyze_pipes(pretty=True)
```

Analyzing pipeline components

```
============================= Pipeline Overview =============================

#   Component         Assigns               Requires         Scores             Retokenizes
-   ---------------   -------------------   --------------   ----------------   -----------
0   tok2vec           doc.tensor                                                False

1   tagger            token.tag                              tag_acc            False

2   parser            token.dep                              dep_uas            False
                      token.head                             dep_las
                      token.is_sent_start                    dep_las_per_type
                      doc.sents                              sents_p
                                                             sents_r
                                                             sents_f

3   attribute_ruler                                                             False

4   lemmatizer        token.lemma                            lemma_acc          False

5   ner               doc.ents                               ents_f             False
                      token.ent_iob                          ents_p
                      token.ent_type                         ents_r
                                                             ents_per_type

6   entity_linker     token.ent_kb_id       doc.ents         nel_micro_f        False
                                            doc.sents        nel_micro_r
                                            token.ent_iob    nel_micro_p
                                            token.ent_type

✓ No problems found.
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

spaCy EntityRuler

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

spaCy EntityRuler

- `EntityRuler` adds named-entities to a `Doc` container
- It can be used on its own or combined with `EntityRecognizer`
- **Phrase entity patterns** for exact string matches (string):

```
{"label": "ORG", "pattern": "Microsoft"}
```

- **Token entity patterns** with one dictionary describing one token (list):

```
{"label": "GPE", "pattern": [{"LOWER": "san"}, {"LOWER": "francisco"}]}
```

Adding EntityRuler to spaCy pipeline

- Using `.add_pipe()` method
- List of patterns can be added using `.add_patterns()` method

```
nlp = spacy.blank("en")
entity_ruler = nlp.add_pipe("entity_ruler")
patterns = [{"label": "ORG", "pattern": "Microsoft"},
            {"label": "GPE", "pattern": [{"LOWER": "san"}, {"LOWER": "francisco"}]}]
entity_ruler.add_patterns(patterns)
```

Adding EntityRuler to spaCy pipeline

- `.ents` store the results of an `EntityLinker` component

```
doc = nlp("Microsoft is hiring software developer in San Francisco.")
print([(ent.text, ent.label_) for ent in doc.ents])
```

```
[('Microsoft', 'ORG'), ('San Francisco', 'GPE')]
```

EntityRuler in action

- Integrates with `spaCy` pipeline components
- Enhances the named-entity recognizer
- `spaCy` model without `EntityRuler`:

```
nlp = spacy.load("en_core_web_sm")

doc = nlp("Manhattan associates is a company in the U.S.")
print([(ent.text, ent.label_) for ent in doc.ents])
```

```
>>> [('Manhattan', 'GPE'), ('U.S.', 'GPE')]
```

EntityRuler in action

- `EntityRuler` added after existing `ner` component:

```
nlp = spacy.load("en_core_web_sm")
ruler = nlp.add_pipe("entity_ruler", after='ner')
patterns = [{"label": "ORG", "pattern": [{"lower": "manhattan"}, {"lower": "associates"}]}]
ruler.add_patterns(patterns)

doc = nlp("Manhattan associates is a company in the U.S.")
print([(ent.text, ent.label_) for ent in doc.ents])
```

```
>>> [('Manhattan', 'GPE'), ('U.S.', 'GPE')]
```

EntityRuler in action

- `EntityRuler` added before existing `ner` component:

```
nlp = spacy.load("en_core_web_sm")
ruler = nlp.add_pipe("entity_ruler", before='ner')
patterns = [{"label": "ORG", "pattern": [{"lower": "manhattan"}, {"lower": "associates"}]}]
ruler.add_patterns(patterns)

doc = nlp("Manhattan associates is a company in the U.S.")
print([(ent.text, ent.label_) for ent in doc.ents])
```

```
>>> [('Manhattan associates', 'ORG'), ('U.S.', 'GPE')]
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

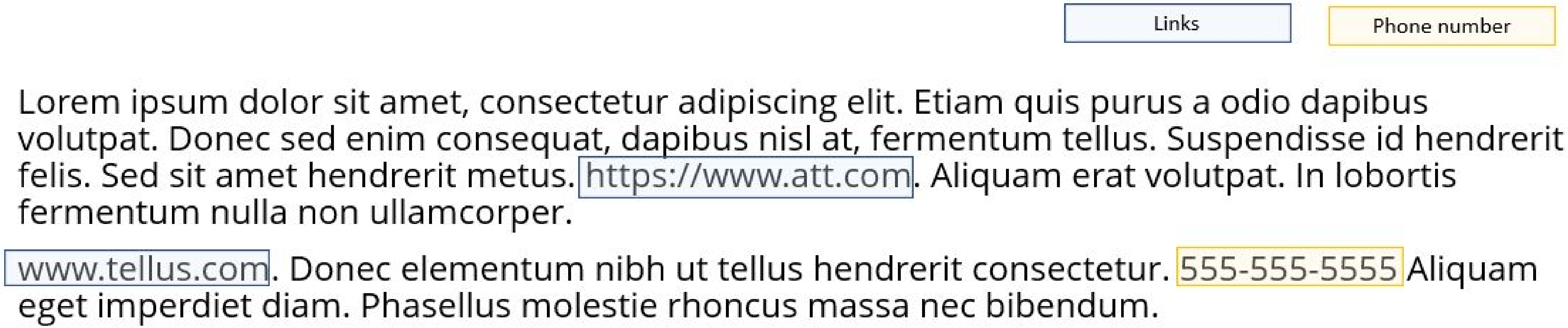
RegEx with spaCy

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

What is RegEx?

- **Rule-based information extraction** (IR) is useful for many NLP tasks
- **Regular expression (RegEx)** is used with complex string matching patterns
- RegEx **finds** and **retrieves** patterns or replace matching patterns



RegEx strengths and weaknesses

Pros:

- Enables writing robust rules to retrieve information
- Can allow us to find many types of variance in strings
- Runs fast
- Supported by programming languages

Cons:

- Syntax is challenging for beginners
- Requires knowledge of all the ways a pattern may be mentioned in texts

RegEx in Python

- Python comes prepackaged with a RegEx library, `re`.
- The first step in using `re` package is to define a `pattern`.
- The resulting pattern is used to find matching content.

```
import re

pattern = r"((\d){3}-(\d){3}-(\d){4})"
text = "Our phone number is 832-123-5555 and their phone number is 425-123-4567."
```

RegEx in Python

- We use `.finditer()` method from `re` package

```
iter_matches = re.finditer(pattern, text)
for match in iter_matches:
    start_char = match.start()
    end_char = match.end()
    print ("Start character: ", start_char, "| End character: ", end_char,
           "| Matching text: ", text[start_char:end_char])
```

```
>>> Start character:  20 | End character:  32 | Matching text:  832-123-5555
Start character:  59 | End character:  71 | Matching text:  425-123-4567
```

RegEx in spaCy

- **RegEx** in three pipeline components: `Matcher` , `PhraseMatcher` and `EntityRuler` .

```
text = "Our phone number is 832-123-5555 and their phone number is 425-123-4567."
nlp = spacy.blank("en")
patterns = [{"label": "PHONE_NUMBER", "pattern": [{"SHAPE": "ddd"},
            {"ORTH": "-"}, {"SHAPE": "ddd"},
            {"ORTH": "-"}, {"SHAPE": "dddd"}]}]
ruler = nlp.add_pipe("entity_ruler")
ruler.add_patterns(patterns)
doc = nlp(text)
print ([(ent.text, ent.label_) for ent in doc.ents])
```

```
>>> [('832-123-5555', 'PHONE_NUMBER'), ('425-123-4567', 'PHONE_NUMBER')]
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY

spaCy Matcher and PhraseMatcher

NATURAL LANGUAGE PROCESSING WITH SPACY

Azadeh Mobasher
Principal Data Scientist

Matcher in spaCy

- **RegEx** patterns can be complex, difficult to read and debug.
- `spaCy` provides a readable and production-level alternative, the `Matcher` class.

```
import spacy
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm")
doc = nlp("Good morning, this is our first day on campus.")
matcher = Matcher(nlp.vocab)
```

Matcher in spaCy

- Matching output include **start** and **end** token indices of the matched pattern.

```
pattern = [{"LOWER": "good"}, {"LOWER": "morning"}]
matcher.add("morning_greeting", [pattern])
matches = matcher(doc)
for match_id, start, end in matches:
    print("Start token: ", start, " | End token: ", end,
          "| Matched text: ", doc[start:end].text)
```

```
>>> Start token:  0  | End token:  2 | Matched text:  Good morning
```

Matcher extended syntax support

- Allows operators in defining the matching patterns.
- Similar operators to Python's `in`, `not in` and comparison operators

Attribute	Value type	Description
`IN`	any type	Attribute value is a member of a list
`NOT_IN`	any type	Attribute value is ***not*** a member of a list
`==`, `>=`, `<=`, `>`, `<`	int, float	Comparison operators for equality or inequality checks

Matcher extended syntax support

- Using `IN` operator to match both `good morning` and `good evening`

```
doc = nlp("Good morning and good evening.")
matcher = Matcher(nlp.vocab)
pattern = [{"LOWER": "good"}, {"LOWER": {"IN": ["morning", "evening"]}}]
matcher.add("morning_greeting", [pattern])
matches = matcher(doc)
```

- The output of matching using `IN` operator

```
for match_id, start, end in matches:
    print("Start token: ", start, " | End token: ", end,
          "| Matched text: ", doc[start:end].text)
```

```
>>> Start token:  0  | End token:  2 | Matched text:  Good morning
Start token:  3  | End token:  5 | Matched text:  good evening
```

PhraseMatcher in spaCy

- `PhraseMatcher` class matches a long list of phrases in a given text.

```
from spacy.matcher import PhraseMatcher
nlp = spacy.load("en_core_web_sm")
matcher = PhraseMatcher(nlp.vocab)
terms = ["Bill Gates", "John Smith"]
```

PhraseMatcher in spaCy

- PhraseMatcher outputs include **start** and **end** token indices of the matched pattern

```
patterns = [nlp.make_doc(term) for term in terms]
matcher.add("PeopleOfInterest", patterns)
doc = nlp("Bill Gates met John Smith for an important discussion regarding
           importance of AI.")
matches = matcher(doc)
for match_id, start, end in matches:
    print("Start token: ", start, " | End token: ", end,
          "| Matched text: ", doc[start:end].text)
```

```
>>> Start token:  0  | End token:  2 | Matched text:  Bill Gates
Start token:  3  | End token:  5 | Matched text:  John Smith
```

PhraseMatcher in spaCy

- We can use `attr` argument of the `PhraseMatcher` class

```
matcher = PhraseMatcher(nlp.vocab, attr = "LOWER")
terms = ["Government", "Investment"]
patterns = [nlp.make_doc(term) for term in terms]
matcher.add("InvestmentTerms", patterns)
doc = nlp("It was interesting to the investment division of the government.")
```

```
matcher = PhraseMatcher(nlp.vocab, attr = "SHAPE")
terms = ["110.0.0.0", "101.243.0.0"]
patterns = [nlp.make_doc(term) for term in terms]
matcher.add("IPAddresses", patterns)
doc = nlp("The tracked IP address was 234.135.0.0.")
```

Let's practice!

NATURAL LANGUAGE PROCESSING WITH SPACY