Relatório 24 - Prática: Processamento de Linguagem Natural (NLP) (III)

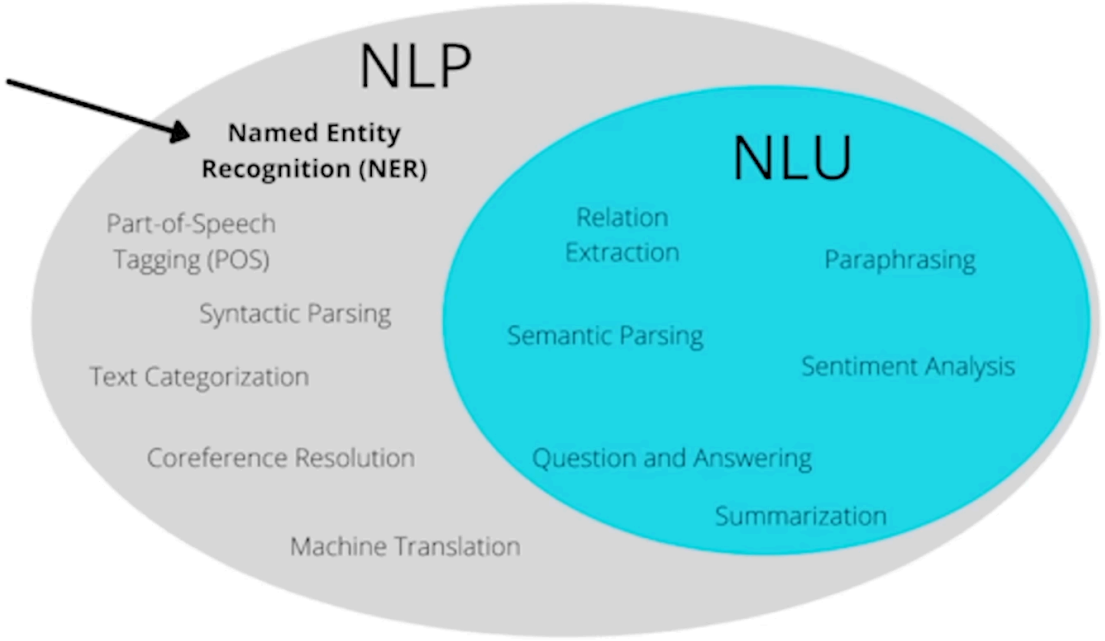
Lucas Scheffer Hundsdorfer

Descrição da atividade

No primeiro vídeo da playlist é ensinado sobre o tokenizer que é transformar palavras inteiras em algo que o computador consiga entender, números. É explicado porque é mais eficiente tokenizar palavras inteiras e não letra por letra, pois existe casos onde palavras diferentes possuem as mesmas letras mas em ordens diferentes:

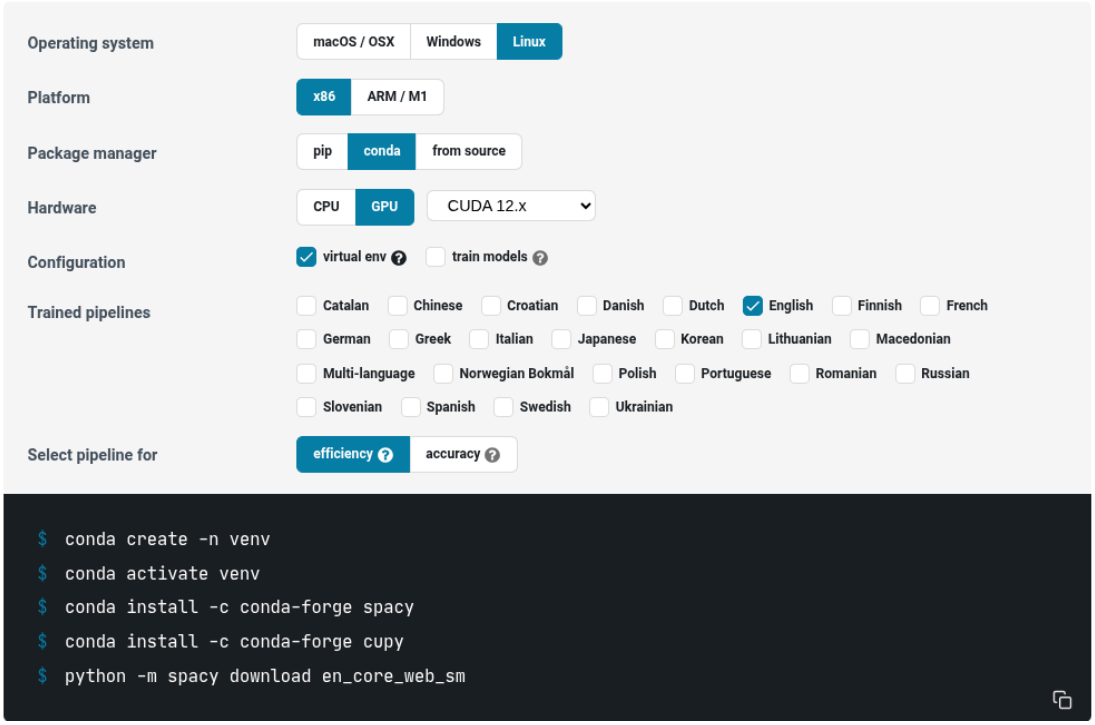
Após as demonstrações práticas da tokenização ele fala brevemente sobre como o modelo consegue reconhecer o significado de cada palavra utilizando uma base de dados do Kaggle de sentenças sarcásticas e não sarcásticas. E também mostra bem pouco sobre uma rede neural recorrente. A última aula do minicurso é sobre como criar uma IA para gerar poesia, e ele deixa bem claro que não é necessário uma base de testes para isso pois agora só queremos tentar descobrir padrões e como as palavras ocorrem.

O outro minicurso começa com a explicação mais técnica e um tom mais técnico sobre o que é realmente o processamento de linguagem natural:

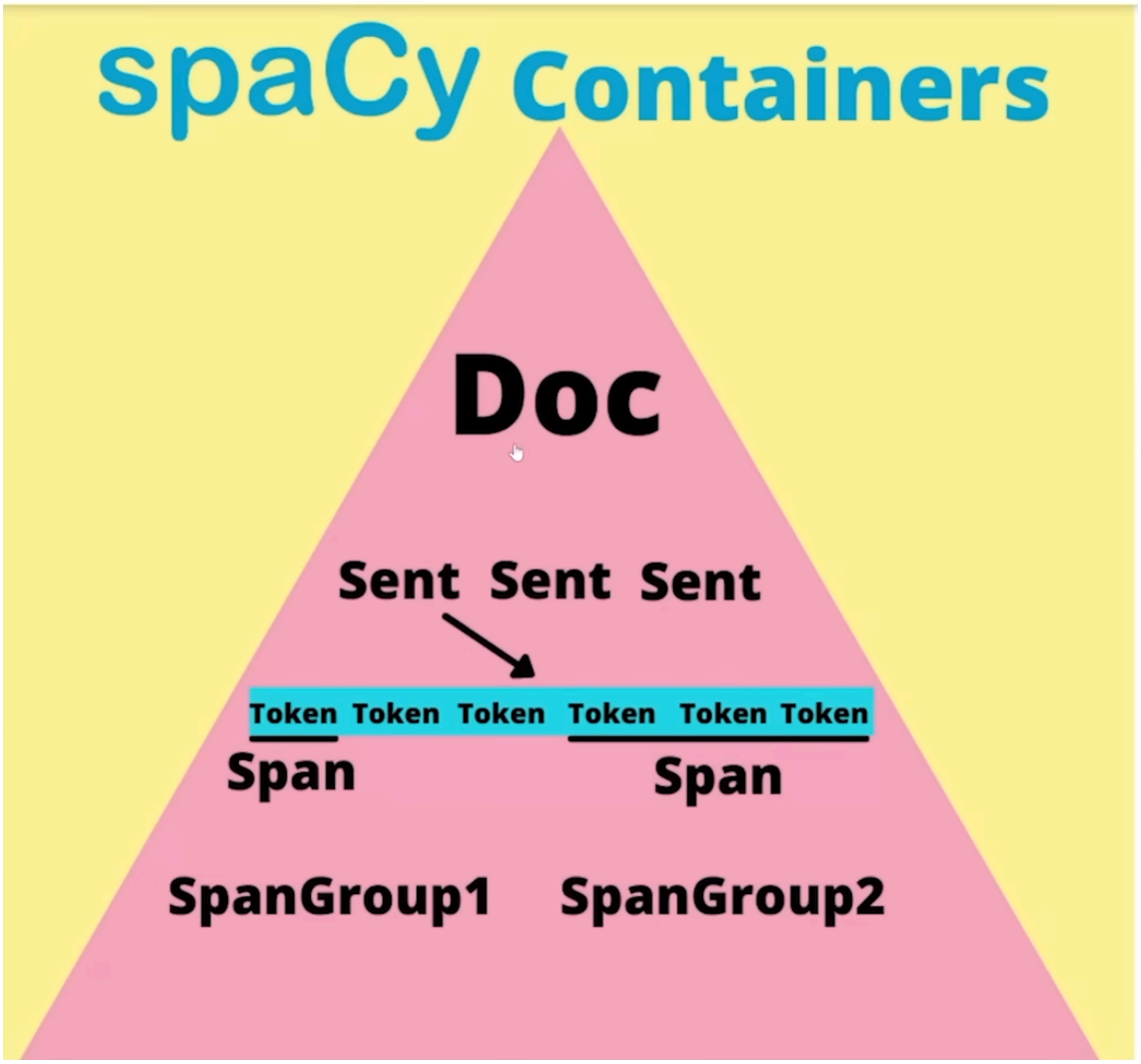


Ele fala sobre algumas aplicações no dia a dia do NLP, como o reconhecimento de spam em emails, e diz também sobre como o framework do spaCy é útil e da existência de outros frameworks para o mesmo tipo de problema.

Ele explica o passo a passo da instalação do spaCy, que na verdade é bem simples por ser muito bem explicado dentro do próprio site deles:



Você escolhe o SO que está sendo utilizado, a plataforma o gerenciador de pacotes, se vai utilizar a cpu ou a gpu, se está em um ambiente virtual ou não e a até a língua que você deseja treinar, a instalação é bem simples devida a isso. Após a instalação ele mostra o que são contêineres dentro do contexto do spaCy, containers são objetos que contém uma enorme quantidade de dados sobre texto:



É utilizado essa imagem para exemplificar, pensando no spaCy como uma pirâmide um sistema hierárquico. A imagem mostra que um Doc (o documento inteiro) é o container de nível superior. Ele é segmentado em Sents (sentenças), que por sua vez são compostas por Tokens (palavras/pontuação). A partir desses tokens, você pode criar Spams (fatias de tokens) e, finalmente, agrupar esses Spams em Span Groups. Essa estrutura organizada e eficiente é um dos principais pontos fortes da biblioteca spaCy.

Sentence Boundary Detection (SBD) é basicamente a identificação das sentenças dentro do texto, parece ser algo simples porém se levar em conta as pontuações e as regras semânticas de cada língua pode dificultar isso, o

exemplo dentro do vídeo utilizado é a sigla U.S.A onde o '.' não determina o fim da frase mas sim a abreviação de United States of America, e por isso se tem um Doc container para entender isso de forma mais fácil e rápida:

```
for token in text[0:10]:
    print(token)
```

```
T
h
e

U
n
i
t
e
d
```

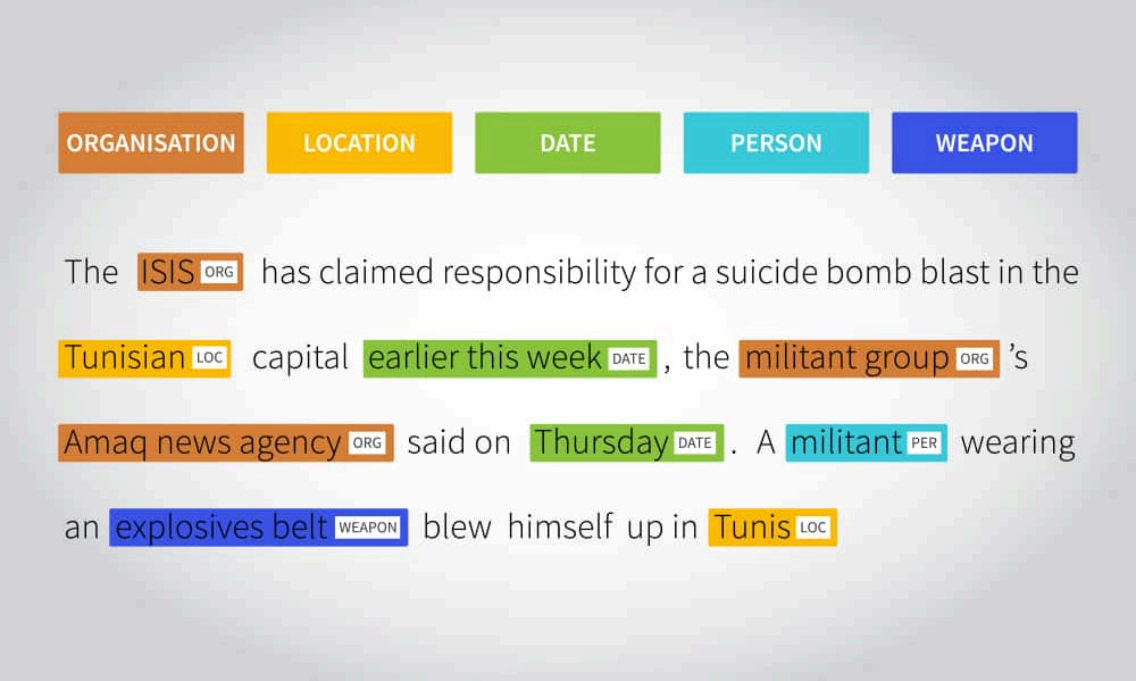
```
for token in doc[0:10]:
    print(token)
```

```
The
United
States
of
America
(
U.S.A.
or
USA
)
```

```
#Aqui a demonstração de porque utilizar o doc, o python consegue
for token in text.split()[0:10]:
    print(token)
```

```
The
United
States
of
America
(U.S.A.
or
USA),
commonly
known
```

Esse código mostra o texto sem ter sido passado para o objeto doc, que reconhece cada letra como um caractere, o doc que reconhece cada palavra como um carácter separado de maneira correta e a separação utilizando o split do python o que levaria a prováveis erros. Depois de algumas explicações de algumas funções dentro da biblioteca do spaCy, como funções para conseguir a classe gramatical e qual a dependência semântica de cada palavra o curso se encaminha para o reconhecimento de entidades dentro do texto:



Que cada palavra tem sua entidade, bem como organização, localização e entre outros. Após isso é explicado um pouco mais sobre 'word vectors' ou vetores de palavras. Esses vetores de palavras são utilizados para o computador compreender o sentido da palavra, mas os computadores não conseguem interpretar palavras, então eles dão um número para cada palavra. Porém não é tão simples como {gato : 2} é algo mais complicado já que esse número é representado por um array de 300 floats:

BANANA.VETOR

```
array([2.02280000e-01,  -7.66180009e-02,   3.70319992e-01,
       3.28450017e-02,  -4.19569999e-01,   7.20689967e-02,
      -3.74760002e-01,   5.74599989e-02,  -1.24009997e-02,
       5.29489994e-01,  -5.23800015e-01,  -1.97710007e-01,
      -3.41470003e-01,   5.33169985e-01,  -2.53309999e-02,
       1.73800007e-01,   1.67720005e-01,   8.39839995e-01,
       5.51070012e-02,   1.05470002e-01,   3.78719985e-01,
       2.42750004e-01,   1.47449998e-02,   5.59509993e-01,
       1.25210002e-01,  -6.75960004e-01,   3.58420014e-01,
       # ... and so on ...
       3.66849989e-01,   2.52470002e-03,  -6.40089989e-01,
      -2.97650009e-01,   7.89430022e-01,   3.31680000e-01,
      -1.19659996e+00,  -4.71559986e-02,   5.31750023e-01], dtype=float32)
```

Esses vetores de palavras podem ser utilizados para medir a similaridade de cada palavra como no exemplo abaixo:

```
doc4 = nlp("I enjoy oranges.")
doc5 = nlp("I enjoy apples.")
print(doc4, "<->", doc5, doc4.similarity(doc5))
```

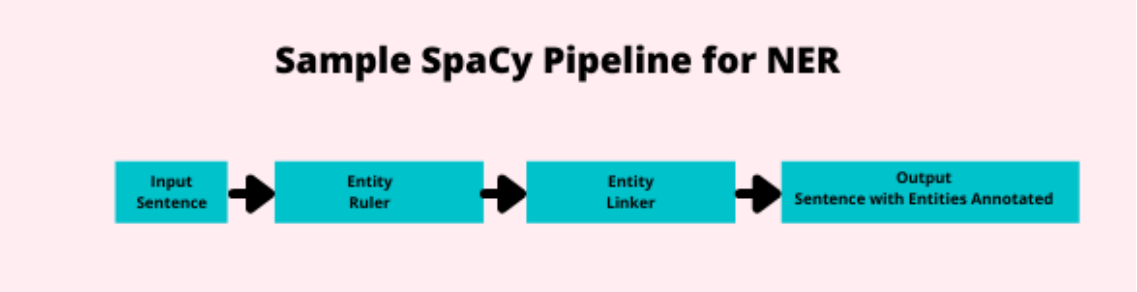
```
I enjoy oranges. <-> I enjoy apples. 0.9522808194160461
```

```
Uma similiaridade menor porém bem alta ainda
```

```
doc6 = nlp("I enjoy burguers.")
print(doc4, "<->", doc6, doc4.similarity(doc6))
```

```
I enjoy oranges. <-> I enjoy burguers. 0.9250046014785767
```

Dentro do contexto da biblioteca spaCy existem as pipelines, que é basicamente como os dados fluem e são tratados:



No exemplo acima é feito a entrada de dados, seguido da ‘entity ruler’ que encontra as entidades e o ’entity liker’ que identifica qual entidade realizará a solução e por fim a saída com as entidades encontradas. Uma pipeline vai ter essa cara:

```
{'summary': {'tok2vec': {'assigns': ['doc.tensor'],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'tagger': {'assigns': ['token.tag'],
   'requires': [],
   'scores': ['tag_acc'],
   'retokenizes': False},
  'parser': {'assigns': ['token.dep',
    'token.head',
    'token.is_sent_start',
    'doc.sents'],
   'requires': [],
   'scores': ['dep_uas',
    'dep_las',
    'dep_las_per_type',
    'sents_p',
    'sents_r',
    'sents_f'],
   'retokenizes': False},
  'attribute_ruler': {'assigns': [],
   'requires': [],
   'scores': [],
   'retokenizes': False},
  'lemmatizer': {'assigns': ['token.lemma'],
   'requires': [],
   'scores': ['lemma_acc'],
   'retokenizes': False},
  'ner': {'assigns': ['doc.ents', 'token.ent_iob', 'token.ent_type'],
   'requires': [],
   'scores': ['ents_f', 'ents_p', 'ents_r', 'ents_per_type'],
   'retokenizes': False}},
 'problems': {'tok2vec': [],
  'tagger': [],
  'parser': [],
  'attribute_ruler': [],
  'lemmatizer': [],
  'ner': []},
 'attrs': {'token.lemma': {'assigns': ['lemmatizer'], 'requires': []},
  'doc.sents': {'assigns': ['parser'], 'requires': []},
  'token.is_sent_start': {'assigns': ['parser'], 'requires': []},
  'token.dep': {'assigns': ['parser'], 'requires': []},
  'token.tag': {'assigns': ['tagger'], 'requires': []},
  'doc.ents': {'assigns': ['ner'], 'requires': []},
  'token.ent_iob': {'assigns': ['ner'], 'requires': []},
  'token.head': {'assigns': ['parser'], 'requires': []},
  'doc.tensor': {'assigns': ['tok2vec'], 'requires': []},
  'token.ent_type': {'assigns': ['ner'], 'requires': []}}}
```

Importante perceber que além de mostrar o que se tem, mostra que tudo tem uma ordem.

Novamente dentro do contexto do spaCy outro conceito que é mostrado importante é o EntityRuler, é uma forma de se criar um conjunto de padrões para reconhecer e rotular entidades após a criação do EntityRuler é possível adicioná-lo a pipeline. O exemplo dado dentro do minicurso em código foi esse:

```
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
West Chestertenfieldville LOC
Deeds PERSON
```

E após a criação dos EntityRuler e colocá-los no lugar correto da pipeline o resultado fica assim:

```
doc = nlp3(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
West Chestertenfieldville GPE
Mr. Deeds FILM
```

Outro conceito muito utilizado dentro do minicurso é o matcher, algo muito útil. Basicamente como se fosse um ‘pesquisador’, você passa um padrão de uma sequência específica dentro do texto e ele retorna todas as ocorrências desse tal padrão dentro do texto analisado:

```
speak_lemmas = ["think", "say"]
matcher = Matcher(nlp.vocab)
pattern = [{"ORTH" : "'"},
           {"IS_ALPHA": True, "OP" : "+"},
           {"IS_PUNCT": True, "OP" : "*"},
           {"ORTH" : "'"},
           {"POS": "VERB", "LEMMA": {"IN": speak_lemmas}},
           {"POS" : "PROPN", "OP": "+"},
           {"ORTH" : "'"},
           {"IS_ALPHA": True, "OP" : "+"},
           {"IS_PUNCT": True, "OP" : "*"},
           {"ORTH" : "'"},
           ]
matcher.add("PROPER_NOUN", [pattern], greedy = "LONGEST")
doc = nlp(text)
matches = matcher(doc)
matches.sort(key = lambda x: x[1])
print(len(matches))
for match in matches[:10]:
    print(match, doc[match[1]:match[2]])
```

```
1
(451313080118390996, 47, 67) 'and what is the use of a book,' thought Alice 'without pictures or conversation?'
```

Aqui o padrão estabelecido foi uma frase entre aspas simples seguida de um verbo conjugado de say ou think seguido de um substantivo com mais uma frase em aspas simples.

O que dá para fazer também com o spaCy é customizar toda a pipeline, criando funções que fazem o que você deseja, o exemplo dentro do minicurso foi para retirar a entidade GPE que é utilizada para países, estados e cidades.

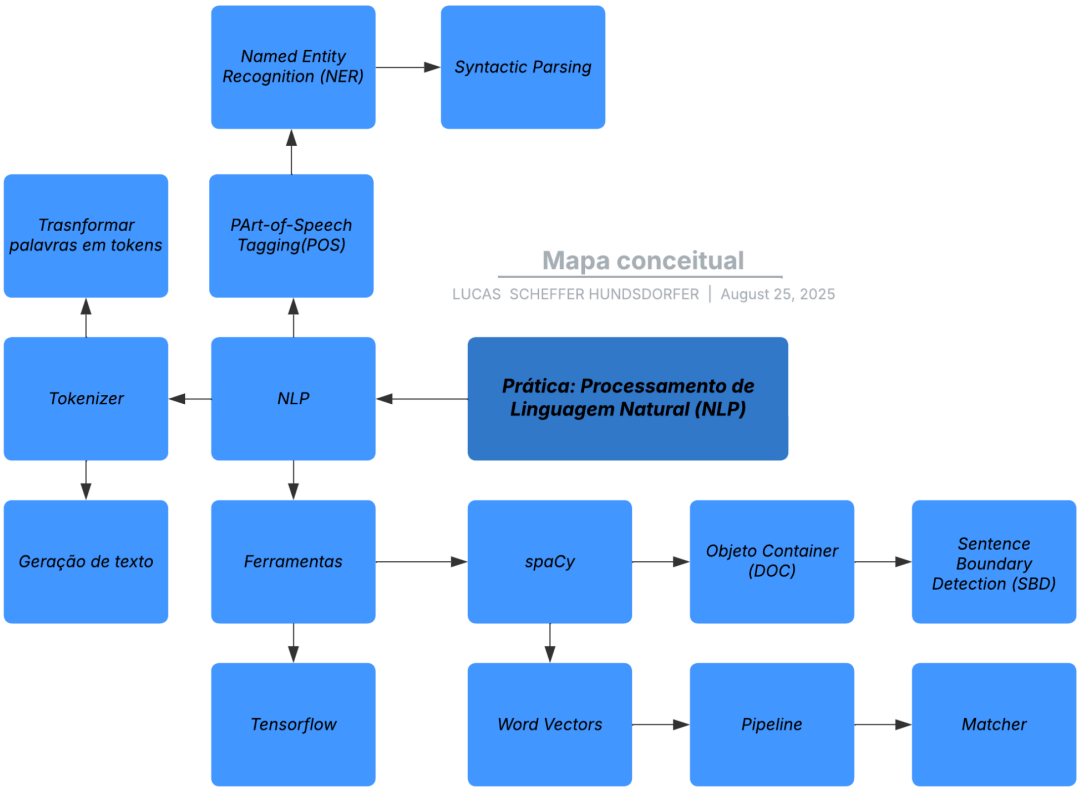
Por último, dentro do minicurso é ensinado o RegEx ou Expressões Regulares. É uma maneira de procurar correspondências dentro de uma string, é bem parecido com o matcher, porém o matcher atua em tokens e o regex atua em strings, sequências de caracteres. RegEx não é algo novo, foi criado na década de 1950 por Stephen Cole Kleene e utilizado até hoje. Mas é algo complexo de ser utilizado, pois o que você pode estar procurando pode aparecer em diferentes padrões. Exemplo:

```python
#Importção biblioteca regex
import re

#Texto exemplo
text = "Paul Newman was an American actor, but Paul Hollywood is a British TV Host. The name Paul is quite comm

#Padrão par encontrar 'Paul' e um possível sobrenome
pattern = r"Paul [A-Z]\w+"

#printar as correspondencias
matches = re.finditer(pattern, text)
for match in matches:
    print(match)
```

```
<re.Match object; span=(0, 11), match='Paul Newman'>
<re.Match object; span=(39, 53), match='Paul Hollywood'>
```

Aqui o padrão procurou ‘Paul’ + Sobrenome, porém o nome inteiro de algum Paul pode aparecer de maneiras diferentes, como em citações onde o sobrenome vem em primeiro, por isso é necessário ter total compreendimento do assunto e dos possíveis padrões.

Insight visual original:



Conclusões

Após a playlist do Tensor Flow e do mini curso focado em spaCy fica claro a importância do NLP ser uma ferramenta tão interessante e útil para diversas formas de utilizá-la, tanto na geração de texto como também no compreendimento deles. E o mais interessante para mim que ficou claro no

curso de spaCy é das múltiplas maneiras que da para se utilizar, indo além da geração de texto e partindo para pesquisas de padrões dentro do texto.

Referências

https://spacy.pythonhumanities.com/intro.html