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

Higor Miller Grassi

Descrição da Atividade: O processamento de linguagem natural (NLP) é uma área da inteligência artificial que permite às máquinas entenderem, interpretarem e processarem a linguagem humana, onde utilizamos técnicas como tokenização, reconhecimento de entidades nomeadas (NER) e análise sintática, ele facilita muito as tarefas como extração de informações, categorização de texto e tradução automatica. Nesta área, a biblioteca spaCy se destaca nesse campo por sua eficiência e modelos pré-treinados, permitindo o desenvolvimento de aplicações escaláveis.

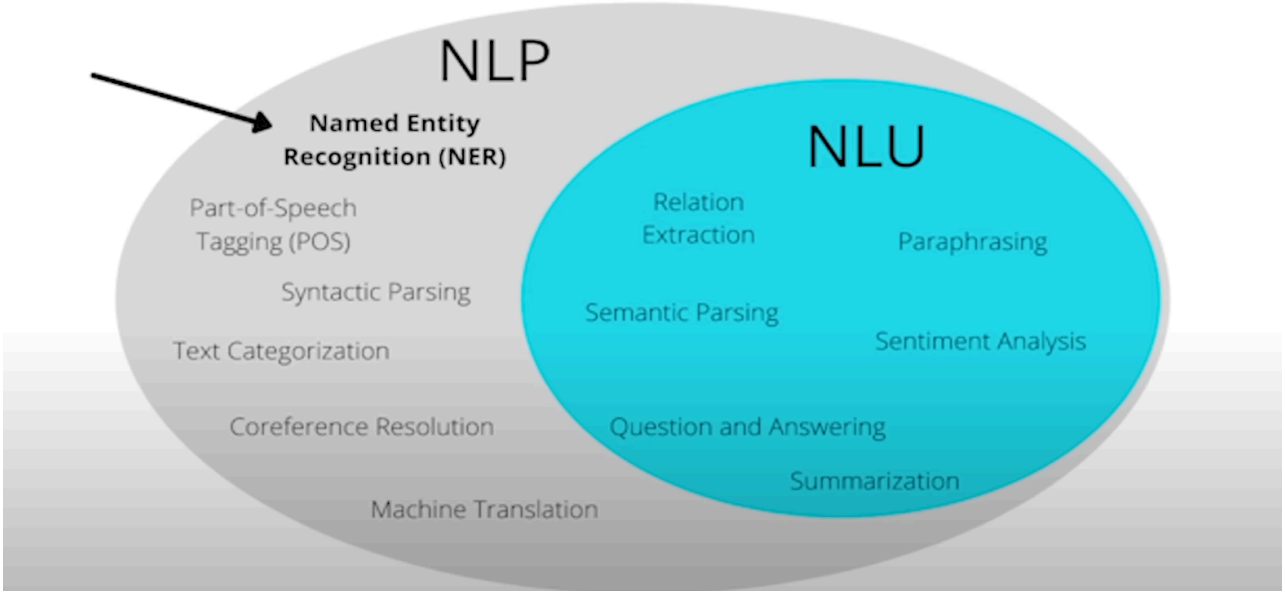
Tokenização: As palavras são formadas por letras, onde as mesmas podem ser representadas por um conjunto de números, assim, tendo o famoso código ASCII. No exemplo dado no vídeo, apenas interpretar as palavras pelas letras pode não ser tão eficaz, já que muitas palavras têm as mesmas letras mas em formações diferentes, então, para diferenciar é numerado as palavras para diferenciar mais facilmente, na imagem podemos descobrir qual a palavra mais frequente de um texto pela tokenização.

```
In [12]: tokenizer = Tokenizer(num_words = 100)
         tokenizer.fit_on_texts(sentences) #tokenizer percorre todo o texto, descobrindo as palavras mais frequentes
         word_index = tokenizer.word_index #frase completa
         print(word_index) #imprime o dicionario

{'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}
```

NLP (Natural Language Processing): Processo em que a máquina entende, analisa e extrai a linguagem, na maior parte das vezes com um texto bruto. Usando a marcação de classes gramaticais, análise sintática, categorização de texto, resolução de correferências e tradução automática.

LUN (Language Understanding and Normalization): Reduz a complexidade e o tamanho do texto sem perder seu significado essencial. Usado principalmente para realizar extração de relações, análise semântica, perguntas e respostas.



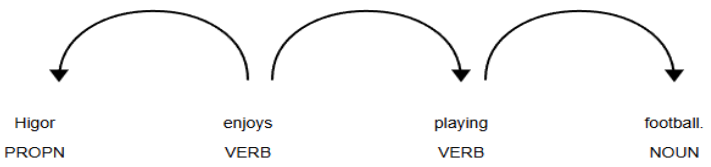
SpaCy: Utilizaremos esta biblioteca pois ela é conhecida por sua rapidez, eficiência e facilidade de uso, tendo modelos pré-treinados para análise sintática, reconhecimento de entidades, categorização de texto e outros recursos avançados, tornando-se ideal para aplicações que exigem processamento rápido e preciso de linguagem natural, escalando muito bem.

Há uma estrutura hierárquica de um objeto Doc na biblioteca spaCy, usada para processamento de linguagem natural. No topo da hierarquia está o Doc, que representa um texto completo, esse Doc é composto por sentenças (Sent), que, por sua vez, são formadas por tokens (Token), que representam as menores unidades do texto, como palavras ou pontuações.

Além disso, tokens podem ser agrupados em Span, que representam sequências contínuas de tokens dentro do texto. Vários Span podem ser organizados em SpanGroup, permitindo a categorização de diferentes partes do texto para tarefas específicas, como reconhecimento de entidades nomeadas ou análise sintática.

Named Entity Recognition (NER): Identifica e classifica as entidades mencionadas em textos, como nomes de pessoas, organizações, locais, datas, entre outros.

```
In [43]: from spacy import displacy #ferramenta de visualizacao
         displacy.render(doc2, style = "dep") #renderiza e imprime a arvore de dependencias
```



Word Vectors: Eles capturam o significado das palavras com base no contexto em que aparecem, na imagem a seguir mostra a comparação das frases e a porcentagem que tem em comum.

```
In [22]:
doc4 = nlp("I like playing video games.") #processa a terceira frase
doc5 = nlp("I like playing soccer.") #processa a terceira frase
print(doc4, "<->", doc5, doc4.similarity(doc5)) # Exibe as duas frases e calcula a similaridade entre elas
```

```
I like playing video games. <-> I like playing soccer. 0.948816379541094
```

Pipelines: É uma sequência de processos aplicados a um texto para transformá-lo em um objeto Doc, que contém tokens, informações linguísticas e anotações, na imagem a seguir mostra o código onde visualizamos os componentes da pipeline.

```
In [7]:
nlp2.analyze_pipes() #visualiza os componentes do pipeline
```

```
Out[7]: {'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'],
```

Entity Ruler: permite adicionar regras personalizadas para identificar e rotular entidades em um texto com base em padrões definidos, onde na imagem a seguir mostra como é feito para definir os padrões.

```
In [36]:
patterns = [ #padroes definidos para identificar o nome GPE e o mr deeds como film
    {"label": "GPE", "pattern": "West Chestertenfieldville"},
    {"label": "FILM", "pattern": "Mr. Deeds"}
]
```

Matcher: Identifica sequências de tokens (palavras ou partes do texto) com base em padrões definidos, diferentemente do entity permite criar padrões complexos para encontrar combinações específicas de tokens, levando em consideração atributos como texto, lema, parte do discurso (POS), dependências sintáticas, no código a seguir temos a parte em que o matcher e aplicado e quais sao os padroes do texto.

In [35]:
```
matcher = Matcher(nlp.vocab) #inicializa o Matcher com o vocabulario do modelo

# define os padroes que o Matcher vai procurar
pattern1 = [{'ORTH': "'"}, {'IS_ALPHA': True, "OP": "+"}, {'IS_PUNCT': True, "OP": "*"}, {'ORTH': "'"}]
pattern2 = [{'ORTH': "'"}, {'IS_ALPHA': True, "OP": "+"}, {'IS_PUNCT': True, "OP": "*"}, {'ORTH': "'"}]
pattern3 = [{"POS": "PROPN", "OP": "+"}, {"POS": "VERB", "LEMMA": {"IN": ["speak", "talk"]}}, {'ORTH': "'"}]

matcher.add("PROPER_NOUNS", [pattern1, pattern2, pattern3], greedy='LONGEST') # adiciona os padroes ao Matcher

data = [
    ["Exemplo de texto", "Outro texto", ["'Hello, world!'", "'This is a test.'", "'John speaks.'"]]
]

# Itera sobre os textos e aplica o Matcher
for text in data[0][2]:
    text = text.replace('"', "'") # substitui aspas duplas por aspas simples
    doc = nlp(text)# processa o texto com o modelo de linguagem
    matches = matcher(doc)# aplica o Matcher ao documento
    matches.sort(key=lambda x: x[1])# ordena as correspondências pelo inicio do match
    print(len(matches))# exibe o nemero de correspondencias encontradas
    for match in matches[:10]: #exibe as primeiras 10 correspondencias
        print(match, doc[match[1]:match[2]])
```

```
0
1
(3232560085755078826, 0, 7) 'This is a test.'
1
(3232560085755078826, 0, 5) 'John speaks.'
```

Custom Components: são funções ou classes que você pode adicionar ao pipeline de processamento de texto para realizar tarefas específicas, podendo manipular o objeto Doc (que representa o texto processado) e adicionar ou modificar atributos, como entidades, tokens, extensões personalizadas, etc

In [7]:
```
doc = nlp("Britain is a place. Mary is as doctor.")
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Britain GPE
Mary PERSON
```

RegEX_aula: Usados para buscar combinações específicas em textos como palavras números ou sequências de caracteres no spacy o regex pode ser integrado usando ferramentas como o matcher ou o phrase matcher que permitem identificar padrões complexos diretamente no pipeline de processamento de texto.