1.Processamento de Linguagem Natural e LLM

Davi Bezerra Barros

Introdução

NLP (Processamento de Linguagem Natural) é um campo da inteligência artificial que permite a máquinas entender, interpretar e gerar linguagem humana, facilitando a interação entre computadores e humanos por meio de texto ou fala.

Dificuldades da NLP

- As linguagens são ambíguas, a nlp tem dificuldade em identificar ironia, por exemplo.
- As palavras tem significados diferentes, dependendo do contexto e posição da palavra na frase.
- modelos de NLP processam letras, palavras, sentenças, textos, documentos e corpos compostos de muitos documentos

Gramática:
não é possível descrever formalmente as regras de uma linguagem, devido à sua complexidade e natureza dinâmica. Como as linguagens estão em constante mudança, elas requerem atualizações frequentes nos modelos.

NLP com SpaCy

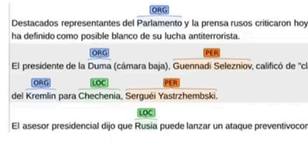
Os modelos pré treinados no spacy vem em três tamanhos:

- Pequeno: ~21MB
- Médio: ~48MB
- Grande: ~440 MB

Conceitos básicos

Corpus: Um conjunto de documentos, textos não estruturados.

Anotações: Metadados adicionados ao texto, como tags gramaticais, relações sintáticas ou identificação de entidades.




```
1\n# newpar\n# sent_id = 1\n# text = Nossa vida é
controlada por algoritmos, disse artista e professor de
artes digitais de uma universidade
americana\n1\tNossa\t_\tDET\tDET\t_\t2\tdet:poss\t_\
t_\n2\tvida\t_\tNOUN\tNOUN\t_\t4\tnsubj:pass\t_\t_\
n3\té\t_\tAUX\tAUX\t_\t4\taux:pass\t_\t_\n4\tcontrola
da\t_\tVERB\tVERB\t_\t8\tccomp\t_\t_\n5\tpor\t_\tAD
P\tADP\t_\t6\tcase\t_\t_\n6\talgoritmos\t_\tNOUN\tN
OUN\t_
```

Tokenização: Processo de separar a sentença em suas partes constituintes: palavras, pontos, símbolos etc.

Parts-of-Speech Tagging(POS): Adiciona tags a cada token, o caracterizando. Ferramentas são capazes de anotar tokens dentro do contexto, atribuindo significados diferentes.

Lemma: Traz a palavra na sua flexão, de modo que possam ser analisadas juntas. Cria uma representação única daquela palavra.

Stemming: Redução de palavras a suas raízes (ex: "correndo" → "corr"), menos preciso que a lematização.

Dependency Parsing: Encontra a relação de dependência hierárquica entre palavras

Ngram: Trata de palavras consecutivas. Bigrama trata duas palavras e trigrama, três. Mais palavras não serão processadas devido a esparsidade.

Modelo: É um banco de dados linguístico, específico de cada idioma. As bibliotecas de NLP utilizam seus próprios modelos, ou de terceiros.

Word Embedding

Por conta da incapacidade do computador de processar informação não estruturada, é necessário criar formas de representar o texto para que ele entenda. Embeddinga técnica utilizada para mapear cada token a um vetor espacial, permitindo o processamento semântico das palavras ao relacionar seu significado com posições espaciais. É um processo fundamental para o processamento de linguagem natural, por que transforma dados categóricos, como palavras, em uma forma que pode ser processada por uma rede neural.

One-hot encoding é utilizado para criar vetores de características para identificar as palavras dentro de um conjunto de dados de um vocabulário / linguagem. O problema desta solução é que os vetores precisariam ser do tamanho do corpus, e não há nenhuma representação de contexto.

TF-IDF: É uma forma de representar as palavras de acordo com sua frequência no texto, trazendo o peso e a importância da palavra no documento.

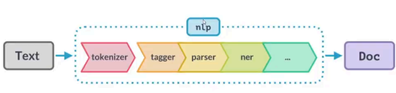
Word2Vec: Mostra além da frequência, a relação entre as palavras. através de um treinamento, a word2vec produz um vetor que demonstra matematicamente a relação entre as palavras. Tem duas formas principais:

- CBOW: busca prever uma palavra central no contexto
- Skip-gram: busca prever o contexto a partir de uma palavra central

O word2vec cria um vetor para cada palavra, que indiac o nível de semelhaça semântica entre as palavras representadas por aquele vetor.

Pipeline de processamento do SpaCy

Sequência de etapas como tokenização, POS tagging, parsing sintático e reconhecimento de entidades. Componentes podem ser personalizados.

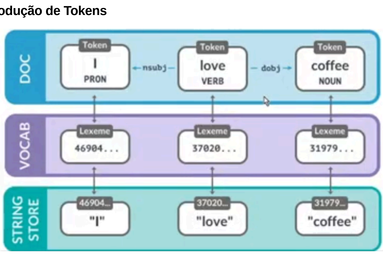


Os componentes do pipeline podem ser adicionados, removidos, e personalizados.

```
1  print(nlp.pipe_names)
```

```
['tok2vec', 'morphologizer', 'parser', 'lemmatizer', 'attribute_ruler', 'ner']
```

Produção de Tokens



Propriedades do Token:

- **text:** O texto original do token.
- **lemma**: Forma canônica (lema) da palavra, baseada em dicionário.
- **pos**: Classe gramatical no contexto.
- **tag**: Tag detalhada (ex: tempo verbal, número).
- **dep**: Relação sintática na árvore de dependências.
- **enttype**: Tipo de entidade nomeada.
- **isalpha**: Indica se o token contém apenas caracteres alfabéticos.
- **isstop**: Indica se o token é uma "stop word".
- **shape**: Forma do token .
- **vector**: Vetor de embedding .
- **similarity**(): Método para calcular similaridade semântica com outro token/doc.
- **etc..**

pratica:

```
for token in doc:
  print(token.text)
```

```
As
ações
do
Magazine
Luiza
S.A.
,
Franca
,
Brasil
,
acumularam
baixa
de
70
%
ao
ano
.
Assim
já
devolveram
todos
os
ganhos
do
período
da
pandemia
.
```

Exemplos de atributos:

```
#atributos
print("Tokens:",[token.text for token in doc])
print("Stop word",[token.is_stop for token in doc])
print("Alfanumérico",[token.is_alpha for token in doc])
print("Maiúculo:",[token.is_upper for token in doc])
print("Pontuação:",[token.is_punct for token in doc])
print("Número:",[token.like_num for token in doc])
print("Sentença inicial:",[token.is_sent_start for token in doc])


```

```
Tokens: ['As', 'ações', 'do', 'Magazine', 'Luiza', 'S.A.', ',', 'Franca', ',', 'Brasi
Stop word [True, False, True, False, False, False, False, False, False, False, False,
Alfanumérico [True, True, True, True, True, False, False, True, False, True, True, False, T
Maiúculo: [False, False, False, False, False, True, True, False, False, False, False, False
Pontuação: [False, False, False, False, False, False, True, False, True, False, True, False,
Número: [False, False, False, False, False, False, False, False, False, False, False, False,
Sentença inicial: [True, False, False, False, False, False, False, False, False, Fals
```

Pos-Taggin e Dependências

- **POS:** Classifica palavras em categorias .
- **Dependências:** Define relações hierárquicas entre palavras.

```
for token in doc:
  print(token.text, "-", token.pos_, "-", token.dep_, "-", token.lemma_, "-", token.shape_)
```

```
As - DET - det - o - Xx
ações - NOUN - nsubj - ação - xxxx
do - ADP - case - de o - xx
Magazine - PROPN - nmod - Magazine - Xxxxx
Luiza - PROPN - flat:name - Luiza - Xxxxx
S.A. - PROPN - flat:name - S.A. - X.X.
, - PUNCT - punct - , - ,
Franca - PROPN - conj - Franca - Xxxxx
, - PUNCT - punct - , - ,
Brasil - PROPN - conj - Brasil - Xxxxx
, - PUNCT - punct - , - ,
acumularam - VERB - ROOT - acumular - xxxx
baixa - NOUN - obj - baixa - xxxx
de - ADP - case - de - xx
70 - NUM - nummod - 70 - dd
% - SYM - nmod - % - %
ao - ADP - case - a o - xx
ano - NOUN - nmod - ano - xxx
. - PUNCT - punct - . - .
Assim - ADV - mark - assim - Xxxxx
já - ADV - advmod - já - xx
devolveram - VERB - ROOT - devolver - xxxx
todos - DET - det - todo - xxxx
os - DET - fixed - o - xx
ganhos - NOUN - obj - ganho - xxxx
do - ADP - case - de o - xx
período - NOUN - nmod - período - xxxx
da - ADP - case - de o - xx
pandemia - NOUN - nmod - pandemia - xxxx
. - PUNCT - punct - . - .
```

Listando Entidades Nomeadas:

Identifica e classifica entidades como pessoas, locais ou datas no texto.

```
#Tag
for ent in doc.ents:
  print(ent.text, " - ", ent.label_)
```

```
Magazine Luiza S.A.  -  ORG
Franca  -  LOC
Brasil  -  LOC
```

Gerenciamento de stop Words

Permite adicionar ou remover palavras irrelevantes

```
for token in doc:
  if token.is_stop:
    print("Stop word:", token.text)
```

```
Stop word: As
Stop word: do
Stop word: de
Stop word: ao
Stop word: Assim
Stop word: já
Stop word: todos
Stop word: os
Stop word: do
Stop word: da
```

Vocabulário

Conjunto de palavras conhecidas pelo modelo, com atributos como vetores e frequência. As palavras armazenadas tem seu proprio hash, e sao armazenadas como lexemas

```
print("hash:", nlp.vocab.strings["dados"])
print("hash:", doc.vocab.strings["dados"])
print("string:", nlp.vocab.strings[6813846809874238634"])
```

```
hash: 6813846809874238634
hash: 6813846809874238634
string: 15189847787399396957
```

```
#lexemas, palavras que não eestão representadas com uma dependencia de contexto
lex = nlp.vocab["dados"]
print(lex.text, "-", lex.orth, "-", lex.is_alpha,"-",lex.is_lower)
```

```
dados - 6813846809874238634 - True - True
```

Busca de Similaridade

Compara vetores de palavras/documentos para medir proximidade semântica, com base na semelhança cosenoidal entre os vetores.

```
doc1 = nlp("ele viaja regularmente de carro")
doc2 = nlp("ele viaja, nunca de avião")

print(doc1.similarity(doc2))
```

```
0.9020987749099731
```

```
doc3 = nlp("é pave ou pacome")

tokenA = doc3[1]
tokenB = doc3[3]

print(tokenA.similarity(tokenB))
```

```
0.0
```

Busca de expressões com matching

Encontra padrões específicos usando regras baseadas em tokens para identificar sequências específicas. Parece um pouco com expressões regulares.

```
from spacy.matcher import Matcher

doc5 = nlp("Você pode ligar para (63) - 984021557 ou (41) 81322096")

#inicializando matcher
matcher = Matcher(nlp.vocab)
padrao = [{"ORTH": "("},
          {"SHAPE": "dd"},
          {"ORTH": ")"},
          {"ORTH": "-", "OP":"?"},
          {"IS_DIGIT": True}]

matcher.add("telefone", [padrao])
matches = matcher(doc5)

for match_id, start, end in matches:
  print(doc5[start:end])
```

```
(63) - 984021557
(41) 81322096
```

Visualização com Displacy

Gera gráficos interativos para dependências sintáticas e entidades.

visualização de entidades:

```
from spacy import displacy
displacy.render(doc,style="ent", jupyter=True)
```

As ações do Magazine Luiza S.A. ORG Franca LOC Brasil LOC , acumularam baixa de 70% ao ano. Assim já devolveram todos os ganhos do período da pandemia.

visualizando dependências:

```
doc.user_data['title'] = "Exemplo"
displacy.render(doc,style='dep', jupyter=True,
                options={'compact':True, 'distance': 60, 'color': 'FFFFFF', 'bg': '#000000'})
```

Gerenciando Pipelines

Personalização de fluxos de processamento

```
print('Pipeline Normal:', nlp.pipe_names)
```

```
Pipeline Normal: ['morphologizer', 'parser', 'lemmatizer', 'attribute_ruler', 'ner']
```

```
nlp.remove_pipe('tok2vec')
```

```
('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7de483a2fc58>)
```

```
print('Pipeline sem tokenizer:', nlp.pipe_names)
```

```
Pipeline sem tokenizer: ['morphologizer', 'parser', 'lemmatizer', 'attribute_ruler', 'ner']
```

```
nlp.add_pipe("tok2vec", after='morphologizer')
```

```
<spacy.pipeline.tok2vec.Tok2Vec at 0x7de483970478>
```

NLP com NLTK

Biblioteca **Natual Language Toolkit**. No NLTK, o pipeline de processamento não vem pronto como no spacy, as etapas tem de ser feitas manualmete ou definidas em um pipeline personalizado.

Teste com a música "O papa perdoa tom zé, de Tom Zé:"

Produção de tokens

Divide textos em unidades menores usando métodos como word_tokenize.

```
sentencas = sent_tokenize(texto,language='portuguese')
print(type(sentencas))
print(sentencas)
```

```
<class 'list'>
['Papa francisco vem perdoar.', 'O tipo de pecado que acabaram de inventar.', 'O povo, querida, com pedras na mão Voltadas contra o imperialismo pagão.
```

texto dividido em 215 tokens de sentenças.

Stop words e Pontuação

Oferece listas pré-definidas de palavras irrelevantes e métodos para filtrar pontuação.

```
stops = stopwords.words("portuguese")
print(len(stops))
print(stops)
```

```
207
['a', 'à', 'ao', 'aos', 'aquela', 'aquelas', 'aquele', 'aqueles', 'aquilo', '
```

Produzindo métricas

Calcula frequência de palavras, diversidade lexical e índices de legibilidade.

```
palavras_sem_stopwords = [p for p in tokens if p not in stops]
print(texto)
print(palavras_sem_stopwords)
print("quantidade de palavras sem stopwords:",len(palavras_sem_stopwords))
print("quantidade de palavras com stopwords:",len(tokens))
```

```
Papa francisco vem perdoar. O tipo de pecado que acabaram de inventar. O povo, querida, com pedras n
['Papa', 'francisco', 'vem', 'perdoar', '.', 'O', 'tipo', 'pecado', 'acabaram', 'inventar', '.', 'O
quantidade de palavras sem stopwords: 165
quantidade de palavras com stopwords: 215
```

Stemming

Algoritmos como Porter e Lancaster para reduzir palavras a radicais.

```
#Porter Stemming
stemmer = PorterStemmer()
stem1 = [stemmer.stem(word) for word in palavras_sem_pontuacao]
print(palavras_sem_pontuacao)
print(stem1)
```

```
['Papa', 'francisco', 'vem', 'perdoar', 'O', 'tipo', 'pecado', 'acabaram', 'inventa
['papa', 'francisco', 'vem', 'perdoar', 'o', 'tipo', 'pecado', 'acabaram', 'inventa
```

```
stemmer2 = SnowballStemmer('portuguese')
stem2 = [stemmer2.stem(word) for word in palavras_sem_pontuacao]
print(palavras_sem_pontuacao)
print(stem2)
```

```
['Papa', 'francisco', 'vem', 'perdoar', 'O', 'tipo', 'pecado', 'acabaram', 'inventa
['pap', 'francisc', 'vem', 'perdo', 'o', 'tip', 'pec', 'acab', 'invent', 'o', 'pov
```

Pós-Taggin

Atribui tags gramaticais usando modelos como Penn Treebank. O método nltk.help.upenn_tagset() traz uma descrição de todas as tags disponíveis no modelo:

```
# descrição de todas as tags
nltk.help.upenn_tagset()
```

```
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
RBR: adverb, comparative
    further gloomier grander graver greater grimmer harder harsher
    healthier heavier higher however larger later leaner lengthier less-
    perfectly lesser lonelier longer louder lower more ...
RBS: adverb, superlative
    best biggest bluntest earliest farthest first furthest hardest
    heartiest highest largest least less most nearest second tightest worst
RP: particle
    aboard about across along apart around aside at away back before behind
    by crop down ever fast for forth from go high i.e. in into just later
    low more off on open out over per pie raising start teeth that through
    under unto up up-pp upon whole with you
SYM: symbol
    % & ' '' ''. ) ). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker
    to
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey gosh
    man baby diddle hush sonuvabitch ...
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated ...
```

tags do texto, dividido por sentença:

```
token2 = sent_tokenize(texto)

ntokens = []
for tokensentenca in token2:
  ntokens.append(word_tokenize(tokensentenca))
print(ntokens)

#pos tag feito a nivel de sentença
possentenca = pos_tag_sents(ntokens)
print(possentenca)
```

```
[['Papa', 'francisco', 'vem', 'perdoar', '.'], ['O', 'tipo', 'de', 'pecado
[[('Papa', 'NNP'), ('francisco', 'NN'), ('vem', 'NN'), ('perdoar', 'NN'),
```

Lemmatization

Reduz palavras à forma canônica

```
lemmatizer = WordNetLemmatizer()
resultado = [lemmatizer.lemmatize(palavra) for palavra in palavras_sem_pontuacao]
print(palavras_sem_pontuacao)
print(resultado)
```

```
['Papa', 'francisco', 'vem', 'perdoar', 'O', 'tipo', 'pecado', 'acabaram', 'inventar', 'O', 'po
['Papa', 'francisco', 'vem', 'perdoar', 'O', 'tipo', 'pecado', 'acabaram', 'inventar', 'O', 'po
```

Entidades Nomeadas

Identifica entidades assim como o spacy.

```
texto_en = "Barack Obama foi um presidente dos EUA"
token3 = word_tokenize(texto_en)
tags = pos_tag(token3)
en = nltk.ne_chunk(tags)
print(en)
```

```
(S
  (PERSON Barack/NNP)
  (ORGANIZATION Obama/NNP)
  foi/NN
  um/JJ
  presidente/NN
  dos/NN
  (ORGANIZATION EUA/NNP))
```