TI-2400 Processamento de Linguagem Natural

ANTONIO CARVALHO - TREINAMENTOS

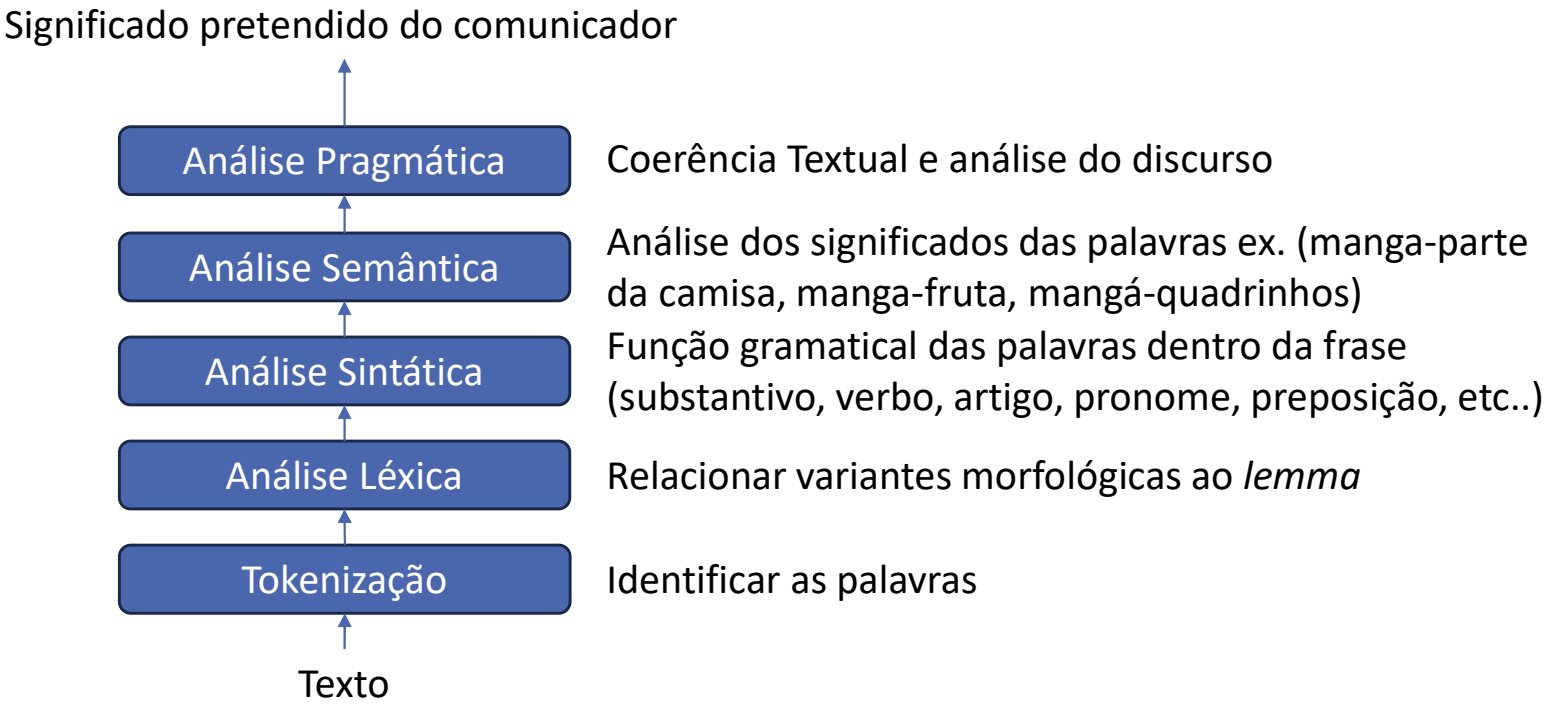
O que é PLN ?

Processamento de Linguagem Natural (PLN)

Processamento de Linguagem Natural é uma área da computação que tem como objetivo extrair representações e significados mais completos de textos livres escritos em linguagem natural. (Indurkhya and Damerau, 2010)

Refere-se por linguagem natural uma linguagem que é usada para comunicações do dia-a-dia feitas por humanos; línguas como português, Inglês ou outras.

Estágios da análise no PLN



Processamento de Linguagem Natural (Conceitos)

PLN – Corpus

Definição de Corpus é um conjunto de textos escritos ou falados numa língua, disponível para análise

Plural de corpus é **corpora**

Características:

- O corpus é um produto já **realizado** e não em processo de realização.
- O corpus é **finito**
- Representa uma **parte das possibilidades realizáveis** da língua e do discurso
- Não pode conter viés na sua composição, ou seja o linguística não pode adicionar ou remover termos

PLN – Corpus

Perspectivas:

- **Fonte de dados**: Pode estar disponível de maneira **oral** ou **escrita**, neste caso usaremos **escrita**
- **Tematização**: A tematização refere-se a qualquer traço de gênero literário, estilo, assunto, período histórico, e/ou outros aspectos que podem caracterizar o corpus do ponto de vista do conteúdo linguístico. Corpus **Geral** é quando não há nenhum traço temático
- **Anotado**: Pode ser dito como anotado quando paralelamente aos dados, há marcações feitas sobre os dados que sirvam para classifica-los, por exemplo, substantivo, verbo, adjetivo.

PLN – Corpus

Onde baixar alguns corpus:

- **NLTK**
- **Linguateca http://www.linguateca.pt**
- **Subtlex-pt-BR (Planilhas com palavras utilizadas em legendas de filmes)**

  http://crr.ugent.be/subtlex-pt-br/csv/SUBTLEX-BRPOR.zip

  http://crr.ugent.be/subtlex-pt-br/csv/SUBTLEX-BRPOR.cd-above2.zip
- **Projeto Guttenberg (Materiais bibliográficos diversos)**

  http://www.gutenberg.org

  Ex: https://www.gutenberg.org/cache/epub/38496/pg38496.txt (Ubirajara de Jose de Alencar)

PLN – Corpus

Exemplo simples de Corpus

```
texto = """rato roeu a roupa do rei de Roma,
O rato roeu a roupa do rei da Rússia,
O rato roeu a roupa do RodovaIho...
O rato a roer roía
E a rosa Rita Ramalho do rato a roer se ria.
O rato roeu a roupa do rei de roma
a rainha com raiva roeu o resto."""
```

PLN – Tokenização

Tokenização consiste no processo de dividir um texto em unidades menores chamadas tokens

Características:

- Normalmente a separação dos Tokens é feito por meio de caracteres como espaços, virgulas, pontos finais, e outros caracteres que não influenciam na mensagem.

```
tokens = ['rato', 'roeu', 'a', 'roupa', 'do', 'rei', 'de', 'Roma',
'O', 'rato', 'roeu', 'a', 'roupa', 'do', 'rei', 'da', 'Rússia',
'O', 'rato', 'roeu', 'a', 'roupa', 'do', 'RodovaIho...',
'O', 'rato', 'a', 'roer', 'roía', 'E', 'a', 'rosa', 'Rita', 'Ramalho',
'do', 'rato', 'a', 'roer', 'se', 'ria.', 'O', 'rato', 'roeu', 'a', 'roupa',
'do', 'rei', 'de', 'roma', 'a', 'rainha', 'com', 'raiva', 'roeu', 'o', 'resto.']
```

PLN – Vocabulário

Vocabulário consiste em uma lista de todas as palavras que aparecem no corpus, sem repetições

```
vocabulario = ['o', 'roía', 'se', 'RodovaIho...', 'E', 'rei',
'rosa', 'Rita', 'da', 'Roma', 'do', 'rato', 'roma', 'Rússia',
'roupa', 'roeu', 'a', 'de', 'resto.', 'raiva', 'Ramalho',
'O', 'com', 'roer', 'ria.', 'rainha']
```

Riqueza Lexical consiste no número de palavras distintas (tamanho do vocabulário) dividido pelo número total de palavras (tokens)

```
riqueza = len(vocabulario) / len(tokens)
```

PLN – Stopwords

Stopwords são palavras que não agregam informação ao sentido da sentença ou do corpus, tais como preposições, artigos, conjunções e outros

```
stopwords = ['a', 'agora', 'ainda', 'alguém', 'algum',
'alguma', 'alguns', ..., 'última', 'últimas', 'último',
'últimos', 'um', 'uma', 'uns', 'vendo', 'ver', 'vez',
'vindo', 'vir', 'vos', 'vós']
```

Em diversas situações as pontuações também podem ser removidas do corpus antes do processamento, a biblioteca **string** do Python possui um texto com as pontuações utilizadas na língua inglesa

```
from string import punctuation
```

PLN – Bag of words

Bag of words (saco de palavras) são listas de palavras que representam sentenças ou documentos. Estas listas consideram as palavras repetidas, mas ignora o contexto gramatical e a ordem em que as palavras são dispostas.

```
bagofwords = ['rato', 'roeu', 'a', 'roupa', 'do', 'rei',
'de', 'Roma', 'O', 'rato', 'roeu', 'a', 'roupa', 'do', 'rei',
'da', 'Rússia' ...]
```

PLN – Bag of words (Binary)

Binary Bag of Words é um método de transformar texto em um vetor binário, onde cada palavra única é representada por um bit.

Exemplo: Imagine que existam duas frases: "cachorro late" e "cachorro morde".

As existentes neste contexto são: **cachorro**, **late**, **morde**, que podem ser representadas:

```
"cachorro late": [1, 1, 0]
"cachorro morde": [1, 0, 1]
```

PLN – Bag of words (Binary)

Vantagens:

- **Simplicidade:** Método direto e fácil de implementar.
- **Eficiência:** Reduz a dimensionalidade em comparação com técnicas mais complexas.
- **Clareza:** Facilita a interpretação dos dados.

Limitações:

- **Perda de Informação:** Não considera a frequência ou a ordem das palavras.
- **Espaço de Memória:** Pode levar a vetores esparsos e grandes.
- **Ambiguidade:** Palavras com múltiplos significados não são diferenciadas.

PLN – Bag of words (Binary)

Implementação usando Scikit Learn.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Dados de exemplo
textos = ["cachorro late", "cachorro morde"]

# Codificação Binary Bag of Words
vectorizer = CountVectorizer(binary=True)
binary_bow = vectorizer.fit_transform(textos).toarray()

print(binary_bow)
```

PLN – Bag of words (Frequency)

Frequency Bag of Words é uma técnica de representação de texto em que a frequência de cada palavra é representada por valores numéricos, representando sua frequência no texto

Exemplo: Usando o exemplo anterior com duas frases **"cachorro late"** e **"cachorro morde".** As palavras únicas são: **cachorro**, **late**, **morde**, que podem ser representadas como:

```
"cachorro late": [2, 1, 0]
"cachorro morde": [2, 0, 1]
```

PLN – Bag of words (Frequency)

A contagem de palavras pode ser guardada também como uma matriz ou dicionário

Exemplo de Bag of words representado como dicionário para uma sentença especifica

```
bagofwords = {'rato': 2, 'roeu':2, 'a':2, 'roupa':2, 'do':2,
'rei':2, 'de':1, 'Roma':1, 'O':1, 'da':1, 'Rússia':1}
```

PLN – Bag of words (Frequency)

Vantagens:

- **Riqueza de Informação:** Considera a frequência das palavras, oferecendo mais contexto.
- **Simplicidade:** Método direto e fácil de implementar.
- **Clareza:** Facilita a interpretação dos dados.

Limitações:

- **Perda de Informação:** Não considera a ordem das palavras.
- **Espaço de Memória:** Pode levar a vetores esparsos e grandes.
- **Ambiguidade:** Palavras com múltiplos significados não são diferenciadas.

PLN – Bag of words (Frequency)

Dividindo o texto em duas sentenças

```
texto = """rato roeu a roupa do rei de Roma, O rato roeu a roupa do rei da Rússia, O rato
roeu a roupa do Rodovalho...
O rato a roer roía
E a rosa Rita Ramalho do rato a roer se ria. O rato roeu a roupa do rei de roma a rainha
com raiva roeu o resto."""
```

Exemplo de Bag of words representado como matriz para duas sentenças

Palavra/ sentença	rato	roeu	a	roupa	do	rei	de	roma	o	da	Rússia	Rodovalho	roer	roia	E	rosa	Rita	Ramalho	se	ria	Rainha	com	raiva	resto
S01	4	3	4	3	3	2	1	1	3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
S02	2	2	4	1	2	1	1	1	2	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1
Total	6	5	8	4	5	3	2	2	5	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1

PLN – Bag of words (Frequency)

Implementação usando Scikit Learn.

```
from sklearn.feature_extraction.text import CountVectorizer

# Dados de exemplo
textos = ["cachorro late", "cachorro morde"]

# Codificação Frequency Bag of Words
vectorizer = CountVectorizer()
freq_bow = vectorizer.fit_transform(textos).toarray()

print(freq_bow)
```

PLN – Matrizes Esparsas (Sparse Matrix)

Uma matriz esparsa é uma matriz na qual a maioria dos elementos são zero.

Exemplo matriz densa

```
matriz = [
    [0, 0, 3, 0],
    [0, 0, 0, 4],
    [0, 0, 0, 5],
    [0, 0, 1, 0]
]
```

Exemplo matriz sparsa

```
[
 [0, 2] => 3
 [1, 3] => 4
 [2, 3] => 5
 [3, 2] => 1
]
```

PLN – Matrizes Esparsas (Sparse Matrix)

A conversão da matriz pode ser feita com o uso da biblioteca **scipy**

```
from scipy.sparse import csr_matrix

matriz = [
    [0, 0, 3, 0],
    [0, 0, 0, 4],
    [0, 0, 0, 5],
    [0, 0, 1, 0]
]

sparse_matrix = csr_matrix(matriz)
print("Matriz Esparsa:\n", sparse_matrix)
print("Array Denso:\n", sparse_matrix.toarray())
```

PLN – Term Frequency (TF)

Term Frequency (Frequencia do Termo) consiste numa divisão entre o número de ocorrências da palavra (ou "termo") pelo número total (contando as repetições) de palavras no texto ou "documento".

Com base no Bag of words anterior

```
terms_count = 6 + 5 + 8 + 4 + 5 + 3 + 2 + 2 + 5 + (3 * 1) + 2 + (11 * 1)

term_frequency = {'rato': 6/terms_count, 'roeu': 5/terms_count, 'a':
8/terms_count, 'roupa': 4/terms_count, 'do': 5/terms_count, 'rei':
3/terms_count, 'de': 2/terms_count, 'Roma': 2/terms_count, 'O':
5/terms_count, 'da': 1/terms_count, 'Rússia': 1/terms_count}
```

PLN – Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF). Por um lado, palavras frequentes em um único texto podem indicar a importância que elas têm no contexto da obra.

Por outro lado**,** se uma palavra tende a se repetir, não somente em um, mas em outros textos, a ideia de relevância é justamente contrária.

O potencial informativo de uma palavra é inversamente proporcional à sua frequência num grande conjunto de documentos diferentes.

PLN – Inverse Document Frequency (IDF)

Document Frequency simbolizado por df_t consistem na frequência em documentos sendo que ***f*** é a frequência absoluta (isto é, a contagem) de documentos ***d*** que contém o termo (ou palavra) ***t***

```
bagofwords = {'rato': 2, 'roeu':2, 'a':2, 'roupa':2, 'do':2,
'rei':2, 'de':1, 'Roma':1, 'O':1, 'da':1, 'Rússia':1}
```

$df_{rato} = 2$

PLN – Inverse Document Frequency (IDF)

Essa é a **Frequência Inversa em Documentos**

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

Sendo **N** o número de documentos total

```
N = 2

df_rato = 2
idf_rato = N / math.log((2 / 2), 10) # N / log_10(2 / 2) == 0
```

Se o corpus 1000 documentos e um determinado termo aparecer em 10 deles, o quociente será $\frac{1000}{10} = 100$, teremos então $\log_{10}(100) = 2$.

Se o termo aparecer em um único documento, indicando ser muito raro, teremos um resultado de IDF maior $\frac{1000}{1} = 1000$, teremos então $\log_{10}(1000) = 3$.

Se o termo aparecer em todos os documentos indicará sua trivialidade, resultando em $\log_{10}(1) = 0$.

PLN – Lemmatization e Stemming

Consistem em reduzir as palavras às suas formas básicas (canônicas/dicionarias), objetivando a redução do vocabulário e simplificação do processamento de texto.

Para entender melhor esse conceito e distinguir entre os dois processos, vamos começar introduzindo termos linguísticos específicos definidos na linguística computacional.

Inflection => (**flexão**) é o processo de formação de palavras para expressar informações gramaticais

Steam => (**Tronco**) é uma parte não flexionada de uma palavra, a parte da palavra que nunca muda, mesmo morfologicamente, quando flexionado

PLN – Lemmatization e Stemming

Affix => (**Afixo**) é um morfema ligado a uma haste de palavra para formar uma nova palavra ou forma de palavra. Os afixos adicionados no início são chamados de prefixos. No meio — infixos, e no final sufixos (às vezes os prefixos e sufixos são ambos nomeados **adfixes**, em contraste com **infixes**)

Lexeme => (**Lexema**) é uma única palavra que pode ter várias formas flexionadas. Essas formas ainda são consideradas variações da mesma palavra subjacente e, juntas, transmitem uma unidade de significado lexical. As formas flexionais conectam palavras relacionadas a essa unidade abstrata básica de significado.

Lemma => (**Lema**) é a forma canônica/dicionário ou citação de um conjunto de formas de palavras

PLN – Lemmatization

Lematização consiste em transformar as palavras em outras palavras, normalmente não flexionadas, também existentes no dicionário, de forma a reduzir o vocabulário necessário.

Exemplo de Lematização

```
sentido => sentir
irmandade => irmão
faturamento => fatura
```

PLN – Lemmatization

```
import spacy
nlp = spacy.load("pt_core_news_sm")
texto = """rato roeu a roupa do rei de Roma,
O rato roeu a roupa do rei da Rússia,
O rato roeu a roupa do RodovaIho...
"""

doc = nlp(texto)
texto__lemmatizado = []
for token in doc:
    lemma = token.lemma_.lower()
    texto__lemmatizado.append(lemma)

print("Texto lemmatizado: ", " ".join(texto__lemmatizado))
```

Para execuçtar a lematização usando Spacy é preciso instalar as bibliotecas a seguir:

```
python -m pip install spacy
python -m spacy download pt_core_news_sm
```

PLN – Stemming

Stemming é uma técnica é eliminar as pequenas diferenças de significado de pluralização ou terminações possessivas de palavras, ou mesmo várias formas verbais. Essa normalização, identificando um tronco comum entre várias formas de uma palavra, é chamada de *stemming*

Exemplo de Stemming

```
copiar => copi
paisagem => pais
```

PLN – Stemming

Algoritmos de *Stemming*

Há diferentes algoritmos de *Stemming*, pois cada linguagem existe técnicas diferentes para a extração do tronco comum entre palavras.

RSLPStemmer para a língua Portuguesa

Cistem para a língua Alemã

ISRIStemmer para o Árabe

LancasterStemmer, **RegexpStemmer**, **Porter Stemmer**, **Snowballstemmer** são voltados para a língua inglesa

PLN – Stemming

A execução do Stemming pode ser feito implementando um algoritmo manualmente, como exemplo, o uso do Regex para remover sufixos.

Exemplo de Stemming com Regex

```
def stem_regex( textos ):
    padrao = r'(?:lhos|lhas|lho|lha|de|os|as|o|a|eu|s|ar|er|ir|or|ur|ia)\b'
    palavras = textos.lower().split()
    nova_lista = []
    for palavra in palavras:
        resultado = re.sub(padrao, '', palavra)
        nova_lista.append(resultado)
    return " ".join(nova_lista)
```

PLN – Stemming

Ou utilizando algoritmo já pronto em bibliotecas. Como exemplo a implementação do RLSPStemmer na biblioteca NLTK

Exemplo de Stemming com RLSPStemmer

```
def stem_rslp( textos ):
    stemmer = RSLPStemmer()
    nova_lista = []
    for palavra in texto.split(" "):
        nova_lista.append(stemmer.stem(palavra))
    return " ".join(nova_lista)
```

PLN – Stemming

É possível comparar o resultado dos algoritmos, resultado do *Stemming* usando **Regex** com o resultado do **RSLPStemmer** da biblioteca **NLTK**

```
stem_regex(texto)
'rat ro  roup d rei  rom,  rat ro  roup d rei d rúss,  rat ro  roup d
rodovaih...  rat  ro roí e  ros rit rama d rat  ro se r.  rat ro  roup d rei
rom  rainh com raiv ro  rest.'

stem_rslp(texto)
'rat roeu a roup do rei de roma,\n rat roeu a roup do rei da rússia,\n rat
roeu a roup do rodovaiho...\n rat a ro roía\n a ros rit ramalh do rat a ro se
ria.\n rat roeu a roup do rei de roma\n rainh com raiv roeu o resto.'
```

Bibliografia

BARBOSA, J. *et. al.;* Introdução ao Processamento de Linguagem Natural usando Python, III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos, v. 1, n. 1, p. 336-360, junho 2017, capitulo 5

FERREIRA, M.; LOPES M.; Para Conhecer Linguística Computacional, São Paulo, ed. Contexto , 2021

HOBSON, L., COLE, H., HANNES, H. Natural Language Processing in Action: Understanding, analyzing, and generating text with Python. Manning, 2019.

INDURKHYA, N. AND DAMERAU, F. J. Handbook of Natural Language Processing. Chapman & Hall/CRC, 2nd edition, 2010

KURPIEL, R. R.; Mastering Stemming and Lemmatization. Hyperskill Blog, site https://hyperskill.org/blog/post/mastering-stemming-and-lemmatization , 2023