Laboratório de Engenharia de Sistemas e Robótica

UFPB

Modelos de Linguagem Larga (LLM)

2. Fundamentos Teóricos

Prof.: Alisson Brito (alisson@ci.ufpb.br)

LASER

Laboratório de Engenharia de Sistemas e Robótica

UFPB

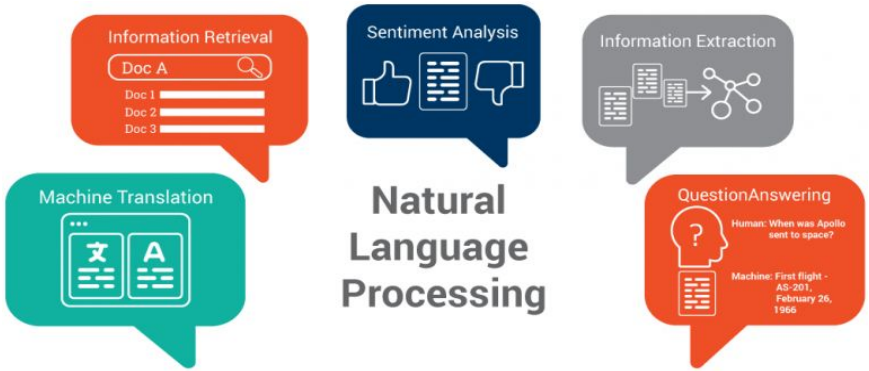
Agenda

1. **Fundamentos de Processamento de Linguagem Natural (PLN)**
   - Introdução ao que é PLN, aplicações, conceitos básicos.
2. **Técnicas de Pré-processamento**
   - Prepara e limpa o texto para que os algoritmos consigam trabalhar melhor com ele
3. **Embeddings (Modelos de Representação de Texto)**
   - Depois que o texto está limpo, explicamos como representá-lo como vetores.
4. **Tarefas Práticas de PLN**
   - Ajudam o computador a entender melhor a estrutura e o significado do textos.

1. Processamento de Linguagem Natural

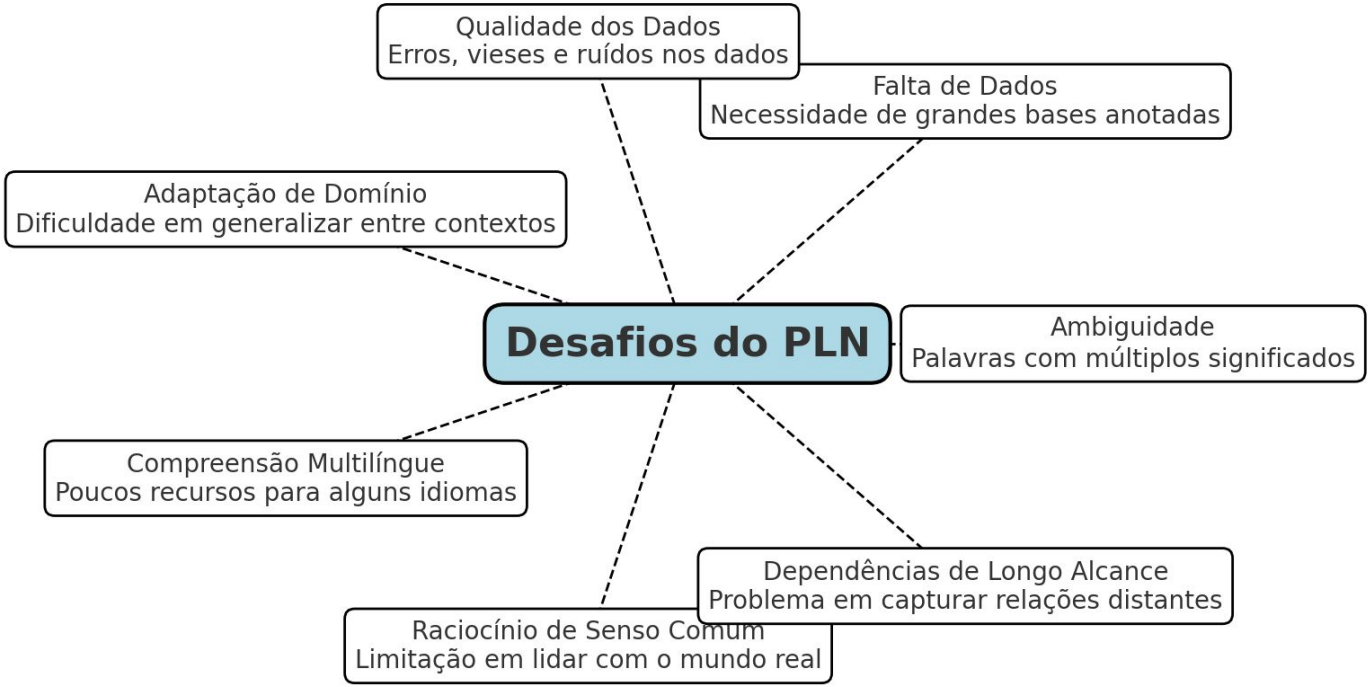
Área da inteligência artificial que se concentra em permitir que computadores **compreendam, interpretem, processem e gerem** linguagem humana

- Assistentes virtuais (Siri, Alexa, Google Assistant);
- Corretores automáticos de texto;
- Tradução automática (Google Translate);
- Análise de sentimentos em redes sociais;
- Chatbots inteligentes, etc.



1. Processamento de Linguagem Natural

Apesar dos avanços recentes, ainda existem muitos **desafios** a serem superados



Ambiguidade: palavras e frases podem ter múltiplos significados, exigindo análise de contexto.

Falta de dados: treinar modelos requer grandes bases anotadas, cuja obtenção é cara e trabalhosa.

Qualidade dos dados: conjuntos disponíveis podem conter erros, vieses ou ruídos.

Adaptação de domínio: modelos treinados em um contexto têm dificuldade em se ajustar a outros.

Compreensão multilíngue: é desafiador criar modelos eficazes em várias línguas, especialmente em idiomas com poucos recursos, como o português brasileiro.

Raciocínio de senso comum: modelos ainda falham em lidar bem com situações do mundo real.

Dependências de longo alcance: captar relações distantes em textos extensos é uma limitação comum.

Desafios do Processamento de Linguagem Natural.
Fonte: (Chhabra et al., 2023)

2. Pré-processamento de texto em PLN

Conjunto de etapas para transformar o texto bruto em algo que o computador consiga entender.

Principais técnicas:

- Tokenização
- Normalização
- Remoção de stopwords
- Stemming / Lematização
- Vetorização / Representação de texto

2.1. Normalização

A geralmente é a primeira etapa do pré-processamento que pode incluir ações como colocar tudo em minúsculas, remover acentos, remover sinais de pontuação.

```
import nltk
…
texto = "Eu gosto de livros."
# Colocar tudo em minúsculas
texto = texto.lower()
# Remover acentos
def remover_acentos(texto):
    return ''.join(c for c in unicodedata.normalize('NFD', texto)
                   if unicodedata.category(c) != 'Mn')
texto = remover_acentos(texto)
...
print("Texto normalizado:", texto)                    # Saída: eu gosto de livros
```

2.2. Stopwords

Elimina palavras que costumam não ter muito significado, diminui o tamanho dos vetores

```
import nltk
…
texto = "Eu gosto de livros e de ler todos os dias."
tokens = word_tokenize(texto)
# Remover stopwords
stop_words = set(stopwords.words('portuguese'))
tokens_sem_stopwords = [t  for t in tokens if t.lower() not in stop_words]

print("Tokens sem stopwords:", tokens_sem_stopwords)


#Saída
Tokens sem stopwords: ['gosto', 'livros', 'ler', 'dias', '.']
```

2.3. Stemming

Corta a palavra de forma “bruta” para reduzir à raiz

```
import nltk
…

tokens = word_tokenize(texto)
# Stemming
stemmer = RSLPStemmer()
tokens_stemmed = [stemmer.stem(t) for t in tokens]

print("Saída:", tokens_stemmed)
```

#Saída: ['Eu', 'gost', 'de', 'livr', 'e', 'de', 'ler', 'tod', 'os', 'di', '.']

2.4. Lematização

Usa dicionário ou regras linguísticas para retornar a forma canônica da palavra

```
import nltk
…
texto = "Eu gosto de livros e de ler todos os dias."
tokens = word_tokenize(texto)

# Lematização com Spacy
doc = nlp(" ".join(tokens))
tokens_lemmatized = [token.lemma_  for token in doc]
```

#Saída: ['Eu', 'gostar', 'de', 'livro', 'e', 'de', 'ler', 'todo', 'o', 'dia', '.']

2.5 Tokenização

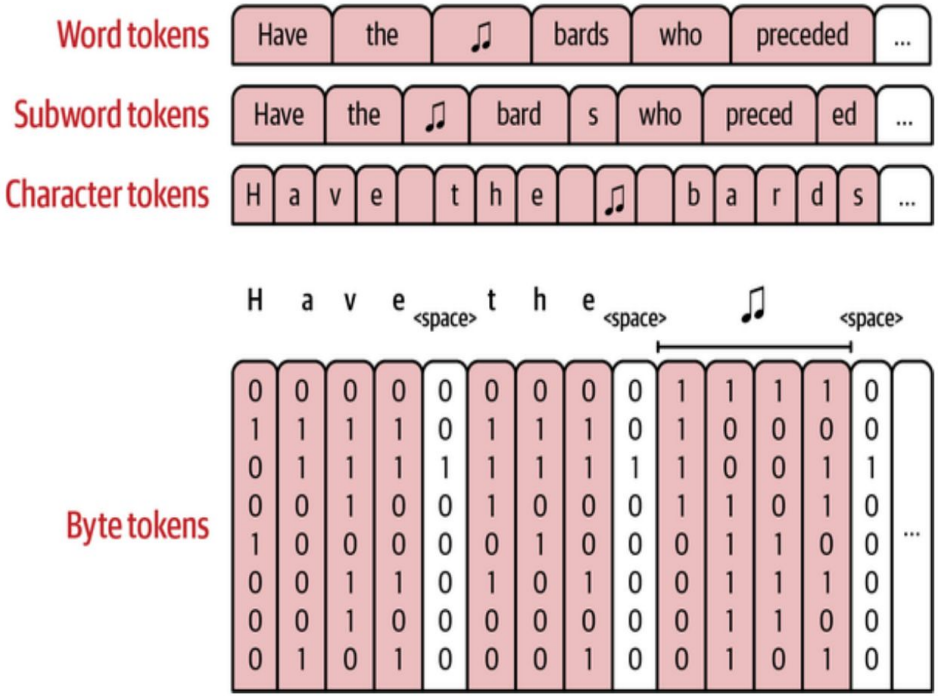
Divide o texto em unidades discretas (tokens)

Dependendo da forma, temos alguns tipos de tokenização:

1. **Word tokens** : "Eu gosto de livros" → ["Eu", "gosto", "de", "livros"]
2. **Subword tokens**: "incrivelmente" → ["in", "crível", "mente"]
    - BPE (Byte Pair Encoding)
3. **Character tokens** : "livro" → ["l", "i", "v", "r", "o"]
4. **Byte tokens** : "Oi" → ["01001111", "01001001"]

2.5 Tokenização

- Word tokens
  - Cada palavra é um token
  - Utilizado no Word2vec e GloVe
- Subword tokens
  - Quebra as palavras em partes menores (prefixos, sufixos, raízes)
  - Lida melhor com palavras novas
  - (BERT, BPE etc)
- Character tokens
  - Cada caractere é um token.
- Byte tokens
  - Divide os caracteres Unicode em bytes individuais



Word Tokens

Cada palavra é um token

```
# biblioteca para PLN
import nltk

nltk.download("punkt")        # Baixa o tokenizador principal
nltk.download("punkt_tab")    # Baixa os arquivos auxiliares exigidos pelo NLTK 3.8+

# Importa a função word_tokenize, responsável por dividir o texto em tokens (palavras/pontuação)
from nltk.tokenize import word_tokenize

texto = "Eu gosto de livros"
tokens = word_tokenize(texto) # Aplica a tokenização
print(tokens)

# Saída esperada:
# ['Eu', 'gosto', 'de', 'livros']
```

Subword Tokens

Quebra a palavra em partes menores que ainda têm sentido.

```
...

# Carrega o tokenizador GPT-2 (que usa BPE)
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

texto = "incrivelmente"

# Tokenização em subwords
tokens = tokenizer.tokenize(texto)
ids = tokenizer.encode(texto)

print("Saída:", tokens)
print("IDs:", ids)
```

Saída: ['in', 'crivel', 'mente']

- **"in"** prefixo comum (infeliz, injusto).
- **"crivel"** vem de “crível” (acreditável) parte da raiz de palavras como “incrível”.
- **"mente"** sufixo típico de advérbios em português (ex.: rapidamente, calmamente).

Character Tokens

```
texto = "livro"
tokens = list(texto)
print(tokens)

# Saída: ['l','i','v','r','o']
```

Cada caractere é um token.

Byte Tokens

```
texto = "Oi"
tokens = [format(b, '08b') for b in
texto.encode('utf-8')]
print(tokens)

# Saída: ['01001111', '01001001']
```

Divide os caracteres Unicode em bytes individuais

Byte Pair Encoding (BPE)

Algoritmo de tokenização baseada em subpalavras, onde a ideia é equilibrar entre caracteres e palavras inteiras, criando um vocabulário eficiente.

Podemos entender o BPE em duas partes:

- **Treinamento** - Nesta etapa ele aprende quais são os melhores tokens para o seu vocabulário.

- **Segmentador** - Ao receber uma entrada, aplica o que aprendeu para quebrar um novo texto em tokens, usando "as regras de separação" aprendidas conforme o vocabulário do dataset de treinamento.

Byte Pair Encoding (BPE)

- **Treinamento -** Estabelece o vocabulário inicial baseado no corpus
  - Exemplo, para: `"incrivelmente incrivel"`
    `["i","n","c","r","i","v","e","l","m","e","n","t", "e", "i","n","c","r","i","v","e","l"]`
- Verifica a frequência de pares adjacentes
  - Encontra `"in"` como par frequente - combinação de `"i" + "n"`
- Mescla o par e atualiza o vocabulário
  - `["in","c","r","i","v","e","l","m","e","n","t", "e", "in","c","r","i","v","e","l"]`

- Processo iterativo
- …e muito mais, como controle da quantidade de repetições etc. (Sennrich et al. 2016 https://arxiv.org/abs/1508.07909)

Byte Pair Encoding (BPE)

- **Segmentador**
- Após o treinamento, pode aplicar o "aprendizado" em textos dados como entrada
  - Exemplo:

    `"incrivelmente"` pode virar → `["in", "crivel", "mente"]`

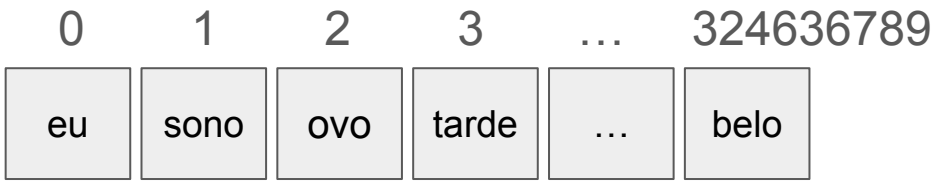
function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

BPE - Pseudocódigo do algoritmo

2.6. Vetorização/ Representação de texto

- Modelos calculam com números!
- Obviamente é necessário transformar texto em números de alguma forma normalizada
- Abordagem intuitiva: Indexar em um vetor gigante com o vocabulário

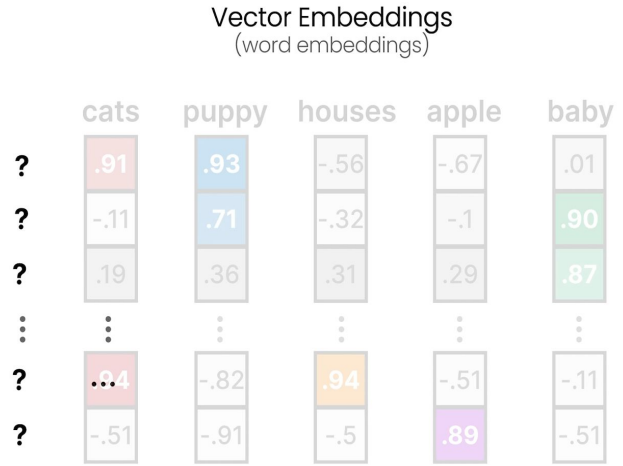


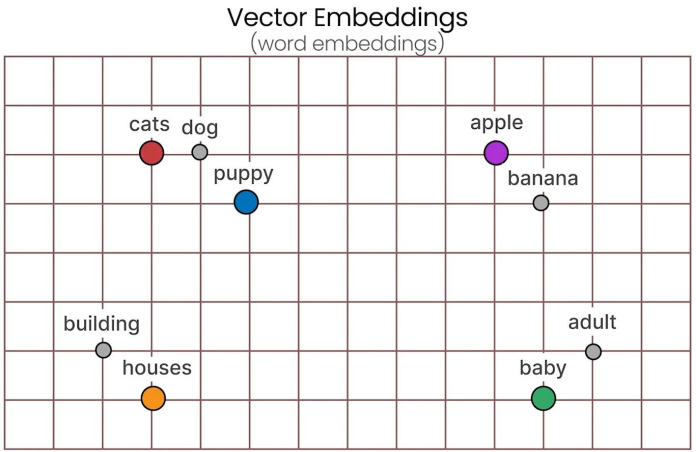
2.6. Vetorização/ Representação de texto

- Principais problemas:
  - Limitado a valores inteiros
  - À medida que vocabulário aumenta, aumenta a propensão a estouro
  - Atribuição de valor indireto, o que deve influenciar também no treinamento
  - Por que a representação numérica para "eu" (0) é menor que "sono" (1), e belo mais ainda?
- Muitos dos problemas são resolvidos pelos métodos mais simples como
  - Bag-of-Words.
  - One-Hot Encoding.
  - TF-IDF.
- A representação vetorial de um token permite que seu significado abstrato seja codificado em múltiplos níveis

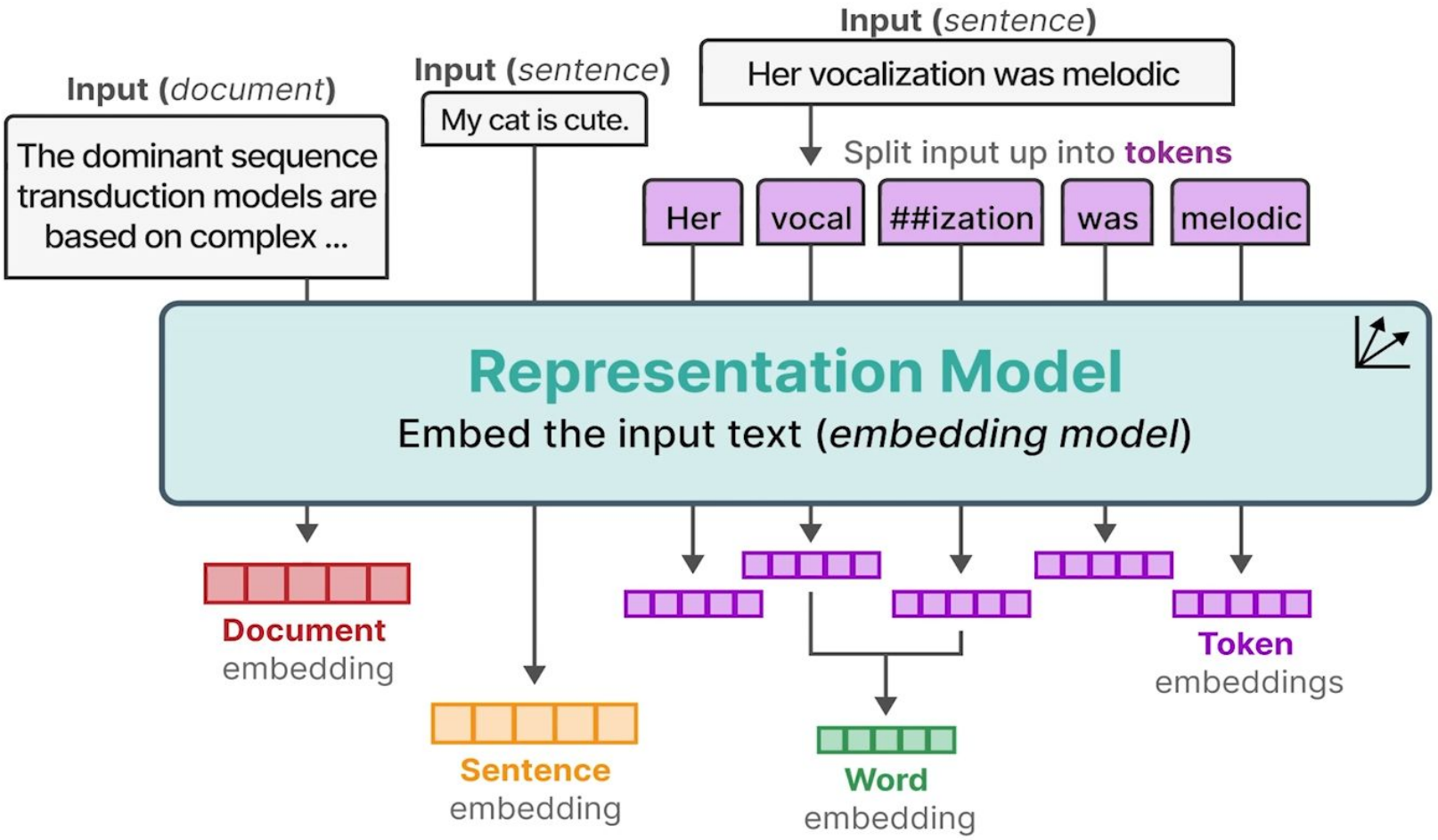
3. Embeddings

- Depois que o texto é **pré-processado** (limpo, normalizado e tokenizado), podemos representá-lo numericamente usando **word embeddings (Word2Vec, GloVe, BERT etc)**
- Representações vetoriais (pontos em hiperplano)
- Em modelos mais complexos não é possível distinguir cada característica representada por cada dimensão
- Pontos com representações próximas tem significado semântico similar





Types of Embeddings



Word Embeddings

Representações vetoriais colocam palavras em um **espaço contínuo de múltiplas dimensões**, permitindo que semântica e similaridade entre palavras sejam capturadas numericamente.

- **Word embeddings estático**
- Cada palavra recebe **um único vetor**, independentemente do contexto.
  - **Word2Vec (2013, Google)**
  - Cria vetores densos para palavras a partir de grandes corpos de texto.
  - Baseado em duas arquiteturas:
    - **CBOW (Continuous Bag of Words)** que prevê uma palavra a partir do contexto
    - **Skip-gram** que prevê o contexto a partir de uma palavra.
  - **GloVe (2014, Stanford)**
  - Combina estatísticas globais de co-ocorrência das palavras com aprendizado vetorial.

vetor("rei") - vetor("homem") + vetor("mulher") ≈ vetor("rainha")

Word Embeddings

- **Word embeddings com contexto**
- Cada token recebe **representações diferentes dependendo do contexto**
  - **BERT:** Produz **vetores dinâmicos**: o mesmo token recebe representações diferentes dependendo do contexto.

```
In [20]: sentence1 = "The bat flew out of the cave at night."
         sentence2 = "He swung the bat and hit a home run."

         word = "bat"

         bert_embedding1 = get_bert_embeddings(sentence1, word).detach().numpy()
         bert_embedding2 = get_bert_embeddings(sentence2, word).detach().numpy()
         word_embedding = word_vectors[word]
```

```
In [21]: print("BERT Embedding for 'bat' in sentence 1:", bert_embedding1[:5])
         print("BERT Embedding for 'bat' in sentence 2:", bert_embedding2[:5])
         print("GloVe Embedding for 'bat':", word_embedding[:5])

         bert_similarity = cosine_similarity([bert_embedding1], [bert_embedding2])[0][0]
         word_embedding_similarity = cosine_similarity([word_embedding], [word_embedding])[0][0]

         print()
         print(f"Cosine Similarity between BERT embeddings in different contexts: {bert_similarity}")
         print(f"Cosine Similarity between GloVe embeddings: {word_embedding_similarity}")
```

```
BERT Embedding for 'bat' in sentence 1: [ 0.4131588  -0.12908243 -0.44865882 -0.4049273  -0.15305819]
BERT Embedding for 'bat' in sentence 2: [ 0.64066947 -0.31121433 -0.44089764 -0.16551133 -0.20056137]
GloVe Embedding for 'bat': [-0.47601  0.81705  0.11151 -0.22687 -0.80672]

Cosine Similarity between BERT embeddings in different contexts: 0.4599575102329254
Cosine Similarity between GloVe embeddings: 1.0000001192092896
```

Observação: Representação vetorial torna possível álgebra com as palavras

```
In [18]: result = word_vectors.most_similar(positive=['king', 'woman'],
                                            negative=['man'], topn=1)

         # Output the result
         print(f"""
             The word closest to 'king' - 'man' + 'woman' is: '{result[0][0]}'
             with a similarity score of {result[0][1]}""")
```

```
The word closest to 'king' - 'man' + 'woman' is: 'queen'
with a similarity score of 0.7698540687561035
```

4. Tarefas do PLN

Tarefas do Processamento de Linguagem Natural (PLN) são **operações ou problemas específicos que os algoritmos de PLN tentam resolver sobre o texto**, **depois que ele foi pré-processado.**

Elas podem ser classificadas em diferentes tipos, dependendo do objetivo:

- POS tagging (Part-of-Speech Tagging)
- Parsing sintático
- Reconhecimento de Entidades Nomeadas (NER)
- Resolução de correferência
- Desambiguação de sentido (WSD – Word Sense Disambiguation)

4. Tarefas do PLN

- **Part-of-speech (POS) tagging:**
  Identifica a classe gramatical de cada token
  Saída: **um rótulo para cada token**, indicando a classe gramatical
  Exemplo: "I/PRON like/VERB books/NOUN"

- **Parser sintático**:
  Analisa a estrutura da frase, mostrando a função sintática de cada token, ou seja, como as palavras se organizam (sujeito, verbo, objeto etc.)
  Saída: **uma estrutura de relações entre palavras**
  Exemplo: sujeito → "I"
  verbo → "like"
  objeto → "books"

4. Tarefas do PLN

- **Reconhecimento de Entidades Nomeadas (NER)**:
  Detecta nomes de pessoas (PERSON), localidades (GPE), datas (DATE) etc
  Exemplo: "New York" → GPE, "J.K. Rowling" → PERSON

- **Correferência**: entende a que pronomes e expressões se referem.
  Exemplo: "Ana … She … Her" → todos se referem a **Ana**

- **Desambiguação:** determina o sentido correto de palavras com múltiplos significados, considerando o contexto
  Exemplo: "bank" → "banco financeiro" ou "margem de rio"

  "I deposited money in the bank" → "bank" = banco financeiro
  "The boat is near the river bank" → "bank" = margem do rio

Bibliotecas para aplicar técnicas de PLN:

1. **NLTK (Natural Language Toolkit)**

- Criado para pesquisa e ensino em PLN.
- Suporta múltiplos idiomas, mas muitos recursos são limitados ao inglês.
- Para outros idiomas, precisa baixar corpora específicos :
  - *cess_esp* para espanhol
  - *floresta* para português
- Usado para POS, NER, tokenização, stemming, lematização, stopwords

```
!pip install nltk

import nltk
nltk.download('punkt')  # baixa o recurso de tokenização
```

Bibliotecas para aplicar técnicas de PLN

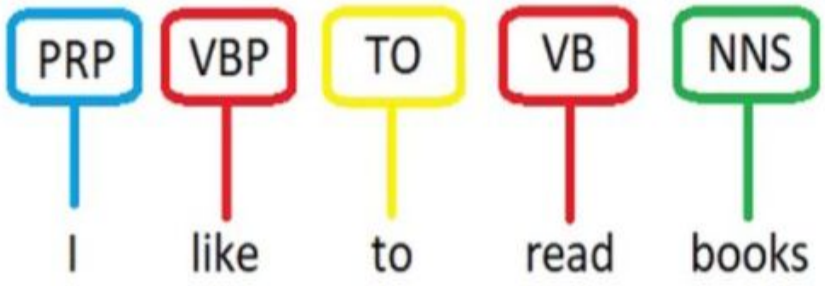
2. SpaCy

- Focado em alto desempenho e aplicações práticas.
- Suporte a vários idiomas, mas recursos dependem do modelo treinado.
- Exemplos de modelos:
  - Inglês: e*n_core_web_sm, en_core_web_md, en_core_web_lg, en_core_web_trf*
  - Espanhol: *es_core_news_sm*
  - Português: *pt_core_news_sm*

```
!pip install spacy
!python -m spacy download en_core_web_sm # Modelo em inglês
!python -m spacy download pt_core_news_sm # Modelo em português
!python -m spacy download es_core_news_sm # Modelo em espanhol
```

4.1. Part-of-speech tagging

- Identifica a classe gramatical específica a cada palavra em um texto (como substantivos, verbos, adjetivos, etc.).
- Converte uma frase em uma lista de palavras com suas marcações



S : *Sentence* (frase completa).
NP : *Noun Phrase* (substantivo ou pronome).
PRP: *Personal Pronoun* (I, you, he, she,...).
VP : *Verb Phrase* (verbo).
VB : *Verb, Base Form* (Verbo base: like, read, go).
NNS: *Noun, Plural* (Substantivo no plural).
TO : Partícula "*to*" que marca infinitivo (ex.: to read).

4.1. Part-of-speech tagging

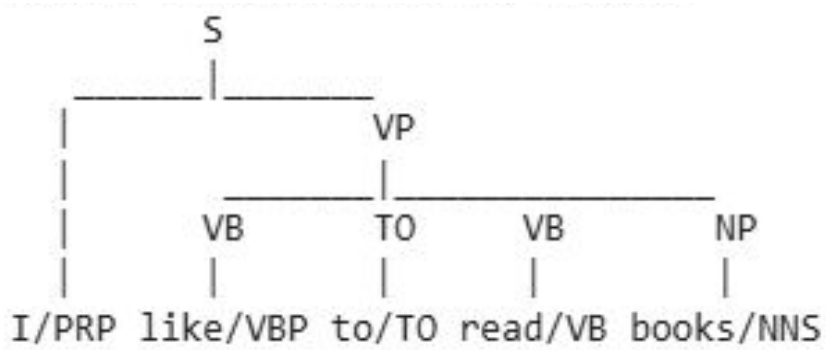
```
import nltk # Modelo linguístico
...
frase = "I like to read books"
tokens = nltk.word_tokenize(frase)
#Etiquetagem (PoS tagging)
tags = nltk.pos_tag(tokens)

print("Token | Classe Gramatical")
for token, tag in tags:
    print(f"{token:6} | {tag}")
```

```
#Saída →
Token | Classe Gramatical
--------------------------
I      | PRP
like   | VBP
to     | TO
read   | VB
books  | NNS
```

4.2. Parser Sintático

Analisa a estrutura da frase, mostrando como as palavras se organizam (sujeito, verbo, objeto etc.)




```
I like to read books
```

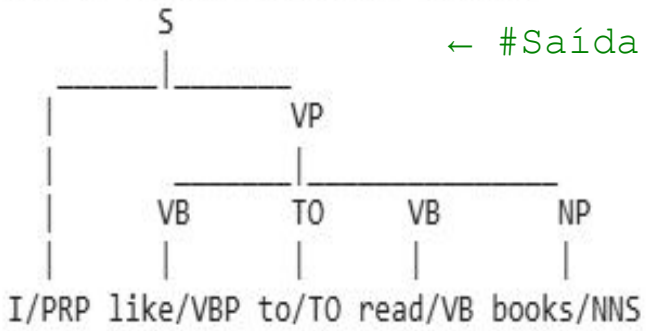
S : *Sentence* (frase completa).
NP : *Noun Phrase* (substantivo ou pronome).
PRP: *Personal Pronoun* (I, you, he, she,...).
VP : *Verb Phrase* (verbo).
VB : *Verb, Base Form* (Verbo base: like, read, go).
NNS: *Noun, Plural* (Substantivo no plural).
TO : Partícula "*to*" que marca infinitivo (ex.: to read).

4.2. Parser Sintático

```
import nltk # Modelo linguístico
...
frase = "I like to read books"
...

grammar = r""" # Definir gramática detalhada
  NP: {<DT|PRP\$>?<JJ>*<NN.*>}# Sintagma nominal
  VB: {<VB.*>}                # Verbo principal ou infinitivo
  TO: {<TO>}                  # Partícula "to"
  VP: {<VB><TO>?<VB><NP>}     # VP detalhado
"""
parser = nltk.RegexpParser(grammar)
tree = parser.parse(tags)
tree.pretty_print()
```

← #Saída



4.3. Reconhecimento de Entidades Nomeadas (NER)

Detecta nomes de pessoas, lugares, organizações, datas e outros elementos específicos

O modelo linguístico identifica entidades:

- `PERSON` → pessoas
- `ORGANIZATION` → organizações
- `GPE` → localidades geopolíticas
- `DATE`
- `TIME`
- `MONEY`
- etc.

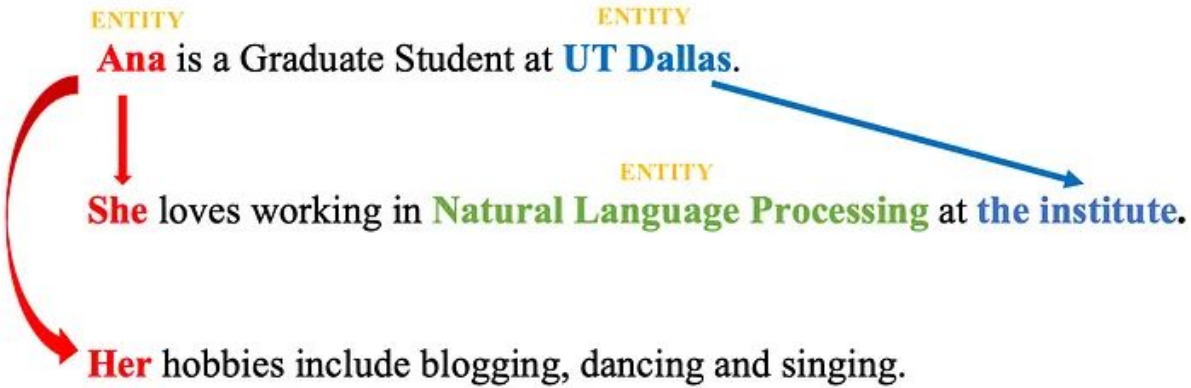
4.3. Reconhecimento de Entidades Nomeadas (NER)

```
import spacy # modelo linguístico
...
nlp = spacy.load("en_core_web_sm")#modelo pré-treinado para inglês
frase = "I like to read books by J.K. Rowling when I travel to New York in
December and sometimes visit the Metropolitan Museum of Art."
...
entidades = [(ent.text, ent.label_) for ent in doc.ents]
print("Entidades extraídas:")
for entidade, tipo in entidades:
    print(f"{entidade} → {tipo}")
```

```
#Saída →  Entidades extraídas:
          J.K. Rowling → ORG
          New York → GPE
          December → DATE
          the Metropolitan Museum of Art → ORG
```

4.4. Resolução de Correferência

Processo de resolução de pronomes para identificar a quais entidades eles se referem As entidades podem ser uma pessoa, um lugar, uma organização ou um evento



“Ana”, “Processamento de Linguagem Natural” e “UT Dallas” são entidades.
“Ela” e “Dela” são referências à entidade “Ana”
“O instituto” é uma referência à entidade “UT Dallas”.

```python
import spacy

# Modelo em inglês
nlp = spacy.load("en_core_web_sm")
texto = ("Ana is a graduate student at UT Dallas. She loves working in Natural Language Processing at the institute.    "
         "Her hobbies include blogging, dancing and singing." )
...

pessoa_ent = next((ent for ent in doc.ents if ent.label_ == "PERSON"), None)
if pessoa_ent:
    # Selecionar pronomes que vêm depois da entidade
    refs = [tok.text  for tok in doc if tok.pos_ == "PRON" and tok.i > pessoa_ent.end]
    clusters.append((pessoa_ent.text, refs))

org_ent = next((ent for ent in doc.ents if ent.label_ == "ORG"), None)
if org_ent: # Selecionar substantivos que correspondem à palavra "institute" depois da entidade
    refs = [tok.text  for tok in doc if tok.pos_ == "NOUN" and tok.text.lower() == "institute" and tok.i > org_ent.end]
    clusters.append((org_ent.text, refs))

for entidade, referencias  in clusters:
    print(f"[{entidade}: {referencias}]")
```

#Saída →

```
[Ana: ['She', 'Her']]
[UT Dallas: ['institute']]
```

4.5. Desambiguação

Processo de **determinar o significado correto de uma palavra ou expressão que possui múltiplos sentidos**, levando em conta o contexto em que ela aparece.

```
# Frase ambígua
texto = "I went to the bank to deposit some money."
...
# Desambiguação da palavra "bank"
sentido = lesk(texto,  "bank")
if sentido:
    print("Palavra:", "bank")
    print("Definição:", sentido.definition())
else:
    print("Não foi possível desambiguar a palavra." )
```

Dúvidas?