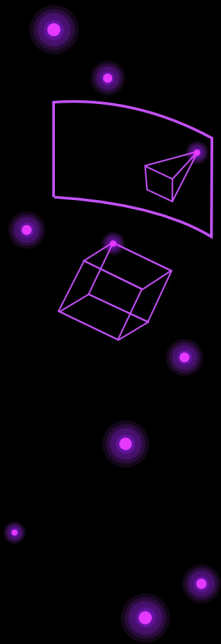
CENTRO DE COMPETÊNCIA EMBRAPII
EM TECNOLOGIAS IMERSIVAS

Introdução ao Processamento de Linguagem Natural

Autoria:
Edson Emílio Scalabrin



Organizadores:
Deborah Silva Alves Fernandes
Renata Dutra Braga
Taciana Novo Kudo
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho

Cegraf UFG

Introdução ao Processamento de Linguagem Natural

Autoria:
Edson Emílio Scalabrin

Organizadores:
Deborah Silva Alves Fernandes
Renata Dutra Braga
Taciana Novo Kudo
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho

Cegraf UFG

2024

Revisão Técnica
Deborah Silva Alves Fernandes
Rafael Teixeira Sousa

Revisão Editorial
Ana Laura de Sene Amâncio Zara Brisolla

Capa
Iuri Vaz Miranda

Editoração Eletrônica
Luma Wanderley de Oliveira
Layane Grazielle Souza Dias

https://doi.org/10.5216/SCA.int.ebook.978-85-495-1038-9/2024

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Scalabrin, Edson Emílio
Introdução ao processamento de linguagem natural [livro eletrônico] / Edson Emílio Scalabrin ; organização Deborah Alves Fernandes ... [et al.]. -- 1. ed. -- Goiânia, GO : Cegraf UFG, 2024.
PDF

Outros organizadores: Renata Dutra Braga, Taciana Novo Kudo, Cristiane Bastos Rocha Ferreira, Arlindo Rodrigues Galvão Filho.
ISBN 978-85-495-1038-9

1. Ciência da computação 2. Informação 3. Linguagem 4. Tecnologia I. Fernandes, Deborah Alves. II. Braga, Renata Dutra. III. Kudo, Taciana Novo. IV. Ferreira, Cristiane Bastos Rocha. V. Galvão Filho, Arlindo Rodrigues.

24-246143 CDD-005.133

Índices para catálogo sistemático:

1. Linguagem natural : Processamento de dados : Ciência da computação 005.133

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

Introdução ao Processamento de Linguagem Natural

Instituições responsáveis

Universidade Federal de Goiás (UFG)

Centro de Competência Embrapii em Tecnologias Imersivas, denominado AKCIT (Advanced Knowledge Center for Immersive Technologies)

Centro de Excelência em Inteligência Artificial (CEIA)

Instituições financiadoras

Empresa Brasileira de Pesquisa e Inovação Industrial (Embrapii)

Governo do Estado de Goiás

Empresas parceiras do AKCIT

Apoio

Universidade Federal de Goiás (UFG)

Pró-Reitoria de Pesquisa e Inovação (PRPI-UFG)

Instituto de Informática (INF-UFG)

Abreviaturas e Siglas

ACL	*Association for Computational Linguistics* - Associação para Linguística Computacional
BERT	*Bidirectional Encoder Representations from Transformers* - Representações Bidirecionais de Codificadores Baseadas em Transformadores
Embrapii	Empresa Brasileira de Pesquisa e Inovação Industrial
GPT	*Generative Pre-trained Transformer* - Transformador Generativo Pré-Treinado
GPT-3	*Generative Pre-training Transformer 3* - Transformador Generativo Pré-Treinado 3
GPT-4	*Generative Pre-training Transformer 4* - Transformador Generativo Pré-Treinado 4
HMM	*Hidden Markov Model* - Modelo Oculto de Markov
IA	Inteligência Artificial
LLM	*Large Language Model* - Modelo de Linguagem de Grande Escala
LSTM	*Long Short-Term Memory* - Memória de Curto Longo Prazo
NeurIPS	*Conference on Neural Information Processing Systems* - Conferência sobre Sistemas de Processamento de Informação Neural
NLTK	*Natural Language Toolkit* - Ferramenta de Linguagem Natural
OpenNMT	*Open-Source Neural Machine Translation* - Tradução Automática Neural de Código Aberto
PLN	Processamento de Linguagem Natural
POS	*Part-of-speech* - Parte do discurso
RNN	*Recurrent Neural Network* - Rede Neural Recorrente
TF-IDF	*Term Frequency - Inverse Document Frequency* - Frequência de Termo - Frequência Inversa de Documento
UFG	Universidade Federal de Goiás

Lista de Figuras e Tabelas

Sumário

Microcurso: Introdução ao Processamento de Linguagem Natural

Prezado(a) Participante,

Seja bem-vindo(a) ao Microcurso "Introdução ao Processamento de Linguagem Natural"! Esse Microcurso é parte da Coleção de Formação e Capacitação do Centro de Competências Imersivas, resultado de uma parceria entre a Empresa Brasileira de Pesquisa e Inovação Industrial (Embrapii) e a Universidade Federal de Goiás (UFG).

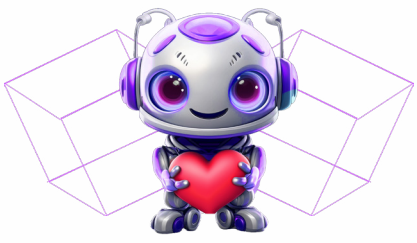
Este *ebook* foi criado para fornecer uma introdução abrangente ao campo do processamento de linguagem natural (PLN), uma área fascinante e em rápida expansão dentro da inteligência artificial (IA). Nosso objetivo é capacitar você com conhecimentos fundamentais que servirão como base para estudos mais avançados e aplicações práticas.

No contexto atual de grandes avanços tecnológicos, a interação entre seres humanos e máquinas torna-se cada vez mais natural e intuitiva. Este Curso explora como as máquinas entendem e processam a linguagem humana, abordando desde conceitos básicos até técnicas avançadas.

Neste Microcurso, você encontrará uma visão geral sobre:

- **Visão geral histórica da área de PLN**: uma introdução aos princípios básicos e desafios do PLN.
- **Análise morfológica relacionada ao PLN**: exploração da estrutura das palavras e sua importância no PLN.
- **Problemas típicos relacionados ao PLN**: estudo dos principais desafios e questões enfrentadas no campo do PLN.
- **Métodos de PLN:** exame da evolução dos métodos, com foco em técnicas modernas como *transformers* e modelos de linguagem de grande escala.

A motivação para a oferta deste Microcurso está na crescente demanda por profissionais qualificados(as) em IA e PLN, áreas que estão revolucionando a interação com a tecnologia e transformando diversas indústrias.



Desejamos a você um excelente estudo!

AKCIT
CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Introdução ao Processamento de Linguagem Natural

Unidade I

Visão geral histórica da área de Processamento de Linguagem Natural

Unidade I - Visão Geral Histórica da Área de Processamento de Linguagem Natural

1.1. Introdução ao Processamento de Linguagem Natural

O PLN é uma área da IA que busca facilitar a interação entre computadores e seres humanos usando a linguagem natural. Iniciado nos anos 1950, o PLN evoluiu de forma significativa, passando por fases distintas: a simbólica, a estatística e a neural. Cada fase trouxe inovações significativas, delineando a área como a conhecemos hoje.

1.2. Décadas de 1950 e 1960: a Fase Simbólica

Durante as décadas de 1950 e 1960, os sistemas pioneiros de PLN utilizavam abordagens simbólicas. Esses sistemas baseiam-se em regras gramaticais e dicionários elaborados de forma manual, visando analisar e gerar linguagem natural por meio de regras linguísticas codificadas por especialistas. Embora essas abordagens fossem inovadoras para a época, elas enfrentam desafios significativos devido à complexidade e à ambiguidade inerentes à linguagem natural. Como resultado, tinham limitações na capacidade de lidar com a variabilidade e as sutilezas presentes no uso cotidiano da linguagem.

Para ilustrar, considere a frase "o gato preto dorme". As regras gramaticais poderiam ser:

S → NP VP	(Sentença = sintagma nominal + sintagma verbal)
NP → Det Adj N	(Sintagma nominal = determinante + adjetivo + substantivo)
Det → "o"	(Determinante = "o")
Adj → "preto"	(Adjetivo = "preto")
N → "gato"	(Substantivo = "gato")
VP → V	(Sintagma verbal = verbo)
V → "dorme"	(Verbo = "dorme")

Nesse exemplo, a frase foi dividida em suas partes constituintes, ilustrando como regras gramaticais podem ser aplicadas para analisar a estrutura de uma sentença. Contudo, essas regras, muitas vezes, não conseguem capturar todas as nuances e variações

do uso cotidiano da linguagem natural. Por exemplo, essas abordagens, muitas vezes, falham ao tentar interpretar corretamente expressões idiomáticas, gírias, ambiguidades sintáticas e polissemia, onde uma mesma palavra pode ter múltiplos significados dependendo do contexto. Isso evidencia as limitações das abordagens simbólicas, destacando sua dificuldade em interpretar corretamente as complexidades da linguagem natural.

1.3. Década de 1980: Surgimento das Abordagens Estatísticas

Nos anos 1980, a introdução de métodos estatísticos no PLN trouxe grandes avanços. Ferramentas, como os modelos de Markov ocultos e as gramáticas probabilísticas, permitiram uma análise mais robusta das variações linguísticas. As gramáticas probabilísticas modelavam a probabilidade de diferentes estruturas gramaticais, capturando a diversidade das construções linguísticas. Por exemplo, para a frase "o gato preto dorme", uma gramática probabilística atribui probabilidades às possíveis estruturas da frase. As regras de produção e suas probabilidades são:

S → NP VP [1.0] (Sentença = sintagma nominal + sintagma verbal),
NP → Det N [0.7] (70% de chance de um sintagma nominal ser um determinante seguido por um substantivo),
NP → Det N Adj [0.3] (30% de chance de um sintagma nominal ser um determinante seguido por um substantivo e um adjetivo),
VP → V [1.0] (Sintagma verbal = verbo),
Det → "o" [1.0] (Determinante = "o"),
N → "gato" [1.0] (Substantivo = "gato"), Adj → "preto" [1.0] (adjetivo = "preto"),
V → "dorme" [1.0] (Verbo = "dorme").

Com essas regras, a gramática probabilística calcula a probabilidade de diferentes interpretações da frase "o gato preto dorme". A probabilidade total da frase, seguindo a estrutura NP → Det N Adj, seria:

$$P(S \to NP\ VP) \times P(NP \to Det\ N\ Adj) \times P(VP \to V) \times P(Det \to \text{"o"}) \times P(N \to \text{"gato"}) \times P(Adj \to \text{"preto"}) \times P(V \to \text{"dorme"}) = 1.0 \times 0.3 \times 1.0 \times 1.0 \times 1.0 \times 1.0 \times 1.0 = 0.3.$$

Essa probabilidade indica a confiança do modelo na estrutura "Det N Adj" para o **sintagma nominal**. Se houvesse uma estrutura alternativa, como "Det N", o modelo compararia as probabilidades e escolheria a estrutura mais provável, proporcionando uma análise robusta das variações linguísticas.

1.4. Década de 2010: a Revolução Neural

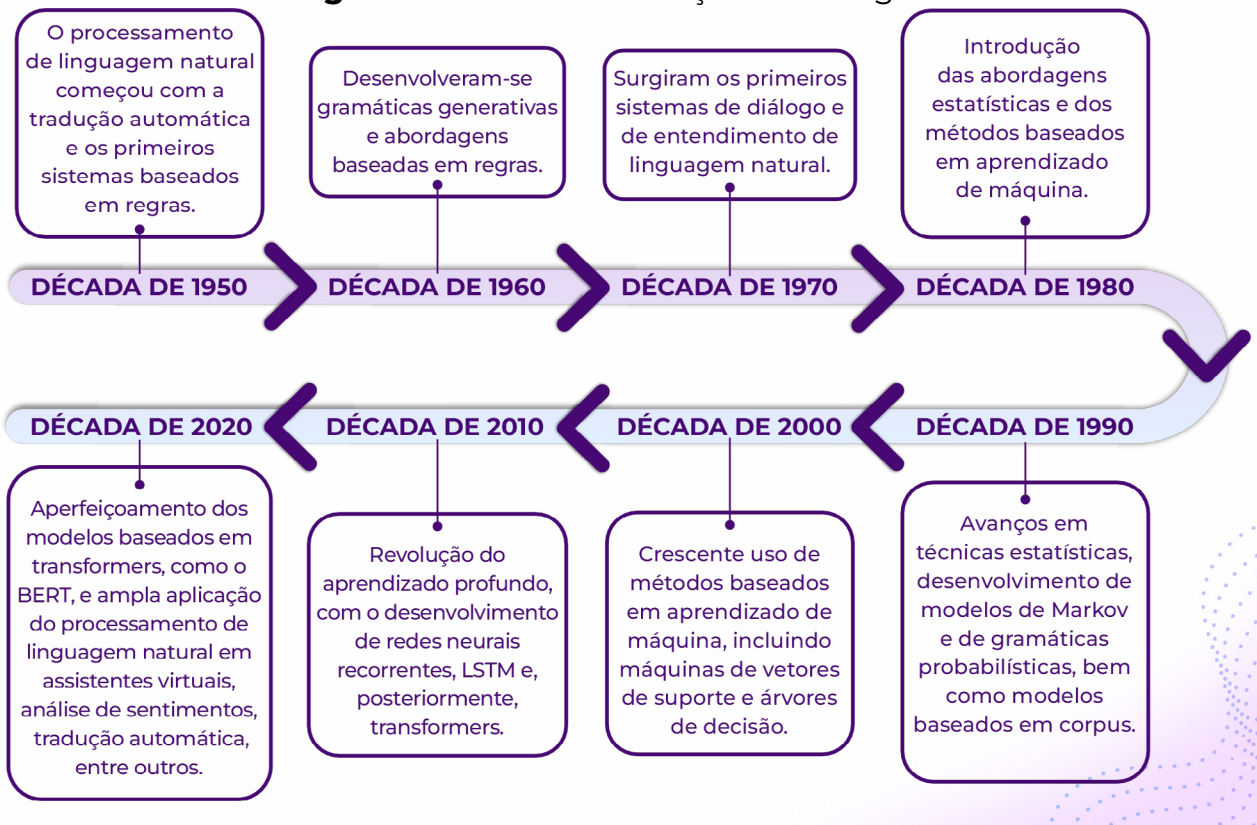
A partir da década de 2010, o campo do PLN passou por uma grande transformação com a introdução da era neural. Tecnologias como redes neurais profundas, incluindo redes neurais recorrentes, ***L**ong **S**hort-**T**erm **M**emory* (LSTM) e *transformers*, trouxeram uma nova capacidade para os sistemas de PLN em capturar contextos e nuances complexas da linguagem. Modelos como o ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* (BERT) e o ***G**enerative **P**re-trained **T**ransformer* (GPT), baseados em *transformers*, se mostraram eficazes em diversas tarefas de processamento de linguagem, desde a compreensão de textos até a geração de conteúdo.

Essas inovações permitiram uma melhor generalização dos modelos e diminuíram a necessidade de dados anotados de forma manual, tornando viável a implementação de soluções de PLN em larga escala. Por exemplo, para a frase "o gato preto dorme", um modelo como o BERT processa todas as palavras ao mesmo tempo, capturando o contexto completo. O modelo entende que "preto" descreve "gato" e que "dorme" é a ação. Ele também reconhece que "gato" é o sujeito principal, mesmo com outras palavras na frase. Essa habilidade de compreender a frase de forma detalhada faz com que o BERT seja mais preciso do que os métodos que o precederam. Além disso, como o BERT foi treinado em uma grande quantidade de dados diversos, ele precisa de menos dados anotados de forma manual para funcionar bem em novas tarefas de compreensão e geração de texto.

1.5. Principais Eventos e Avanços Tecnológicos

Para compreender a evolução do PLN, é interessante examinar uma linha do tempo que destaca os principais eventos e avanços tecnológicos na área (Figura 1).

Figura 1 - Eventos e avanços tecnológicos



A evolução do PLN foi significativamente impulsionada pela combinação de técnicas estatísticas avançadas e no uso de grandes *corpora*, permitindo a criação de sistemas mais precisos e adaptáveis. Esse avanço reflete a importância das capacidades computacionais crescentes e novas abordagens de pesquisa. O advento da internet nos anos 1990 proporcionou um acesso sem precedentes a *corpora* massivas e ao processamento em larga escala. Um *corpus* é uma coleção de textos organizada, enquanto *corpora* são conjuntos múltiplos usados para análises comparativas e treinamento de modelos linguísticos.

1.6. A História do Processamento de Linguagem Natural Ilustrada por Imagens

Para mostrar como o PLN se desenvolveu ao longo dos anos, começando na década de 1950, criamos uma apresentação visual que destaca três grandes paradigmas históricos: o simbólico, o estatístico e o neural.

Figura 2 - Primeiros sistemas de tradução automática baseados em regras nos anos 1950

Fonte: ilustração potencializada pelo GPT 4.

Nos anos 1950, começaram os primeiros passos do PLN com o desenvolvimento dos primeiros sistemas de tradução automática baseados em regras gramaticais. Esses sistemas usavam regras fixas, criadas de forma manual, para traduzir textos de uma língua para outra. Um exemplo marcante dessa época foi a tradução de frases do russo para o inglês. Apesar de suas limitações, esses sistemas iniciais mostraram que era possível usar computadores para manipular a linguagem natural, lançando as bases para pesquisas futuras e avanços contínuos na área. Na Figura 2, está ilustrada a tecnologia da época.

Figura 3 - Contribuições de Noam Chomsky e desenvolvimento das gramáticas generativas

Fonte: imagem potencializada pelo GPT 4.

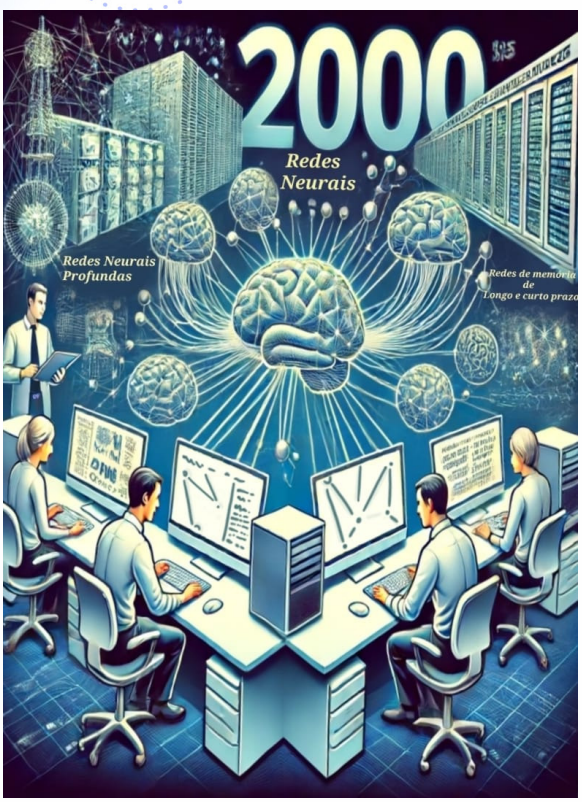
Figura 4 - Evolução do processamento de linguagem natural: transição para o paradigma estatístico (1980-2000)

Fonte: imagem potencializada pelo GPT 4.

Durante as décadas de 1960 e 1970, houve uma ênfase significativa no desenvolvimento das gramáticas generativas, em particular, influenciadas pelo trabalho de Noam Chomsky. Essas gramáticas estabeleciam um conjunto de regras sintáticas capazes de gerar frases gramaticalmente corretas em uma língua. A colaboração entre linguistas e cientistas da computação foi crucial para criar modelos mais sofisticados de análise da linguagem natural, apesar dos desafios enfrentados para lidar com a ambiguidade linguística e as exceções gramaticais complexas. Na Figura 3, é ilustrada essa colaboração interdisciplinar no avanço dos estudos da linguística computacional, bem como os esforços manuais.

Paradigma estatístico (décadas de 1980 a 2000): Durante esse período, houve um destaque notável para o uso de métodos estatísticos no campo do PLN. Ferramentas, como os modelos ocultos de Markov, passaram a ser empregadas em tarefas como reconhecimento de fala e etiquetagem gramatical. A análise de grandes *corpora* de textos anotados viabilizou a criação de modelos mais precisos e adaptáveis, marcando uma transição importante das abordagens baseadas em regras para aquelas fundamentadas em dados. Na Figura 4, é ilustrada essa transição, destacando o uso de modelos de Markov e a análise de grandes *corpora* de textos anotados.

Figura 5 - Avanços no processamento de linguagem natural com redes neurais (2000-2010)



Fonte: Imagem potencializada pelo GPT 4.

Paradigma neural (décadas de 2000 a 2010): Durante esse período, houve um avanço considerável nos métodos de PLN com base em redes neurais profundas. Modelos, como redes neurais recorrentes e redes de LSTM, foram capazes de captar dependências de longo prazo em textos. Isso resultou em melhorias significativas em áreas como tradução automática, reconhecimento de fala e geração de texto. As técnicas de aprendizado supervisionado, incluindo máquinas de vetores de suporte e modelos estatísticos de tradução automática, desempenharam um papel fundamental na construção das bases para esses avanços, elevando a eficiência e precisão das tarefas de PLN. Na Figura 5, são ilustrados esses avanços e a evolução das técnicas utilizadas durante esse período.

Figura 6 - Arquitetura de *transformers* e seu impacto no processamento de linguagem natural na década de 2020

Fonte: Imagem potencializada pelo GPT 4.

Na década de 2020, a chegada dos modelos *transformers*, como BERT e GPT, conforme ilustrado na Figura 6, revolucionou o campo do PLN. Diferentemente das redes neurais recorrentes, os *transformers* utilizam mecanismos de atenção que permitem uma compreensão e processamento mais eficazes no contexto das palavras em uma frase. Esses modelos mostraram um desempenho excepcional em diversas tarefas de PLN, incluindo tradução automática e geração de texto. A arquitetura dos *transformers* tornou-se essencial para muitos dos avanços recentes na área, permitindo a criação de sistemas de PLN mais precisos e robustos.

SAIBA MAIS...

» Se você deseja se aprofundar no estudo do PLN, aqui estão algumas sugestões de leitura:

- » História do Processamento de Linguagem Natural - Wikipedia
- » Blog de Inteligência Artificial da OpenAI
- » Abordagens modernas em PLN
- » Desenvolvimentos em modelos transformers
- » Speech and Language Processing

Unidade II

Análise morfológica relacionada ao Processamento de Linguagem Natural

Unidade II - Análise Morfológica Relacionada ao Processamento de Linguagem Natural

Nesta Unidade, são explorados os *conceitos básicos de morfologia linguística*, incluindo o *processo de lematização* e sua relevância na *normalização de texto* e redução da dimensionalidade do vocabulário. São discutidos, também, a etiquetagem de partes do discurso, o *stemming* e suas diferenças em relação à lematização. Além disso, são analisados casos de uso práticos e os principais algoritmos e ferramentas como o Algoritmo de Porter, WordNet, SpaCy e NLTK, com atividades no *Colab* para aplicação prática.

2.1 Conceitos Básicos de Morfologia Linguística

A morfologia linguística é um ramo da linguística que se dedica ao estudo da estrutura das palavras e das regras para a formação de novas palavras. Essa área foca em unidades menores, conhecidas como morfemas, que são os menores elementos com significado ou função gramatical em uma língua. A análise morfológica tem como objetivo identificar e classificar esses morfemas para compreender como as palavras são formadas e usadas em diversos contextos.

Os morfemas são as menores unidades de significado na linguagem e são essenciais para a formação e modificação de palavras. No português, os morfemas mais comuns são as raízes, prefixos e sufixos, cada um com um papel específico:

a) **Raiz:** a raiz é o morfema central de uma palavra, responsável pelo seu significado principal. Ela serve como base sobre a qual outros morfemas podem ser adicionados para formar novas palavras ou modificar palavras existentes. Por exemplo, na palavra "felicidade", a raiz é "feliz". Essa raiz carrega o significado essencial, enquanto outros morfemas podem ser anexados para ampliar ou alterar esse significado.

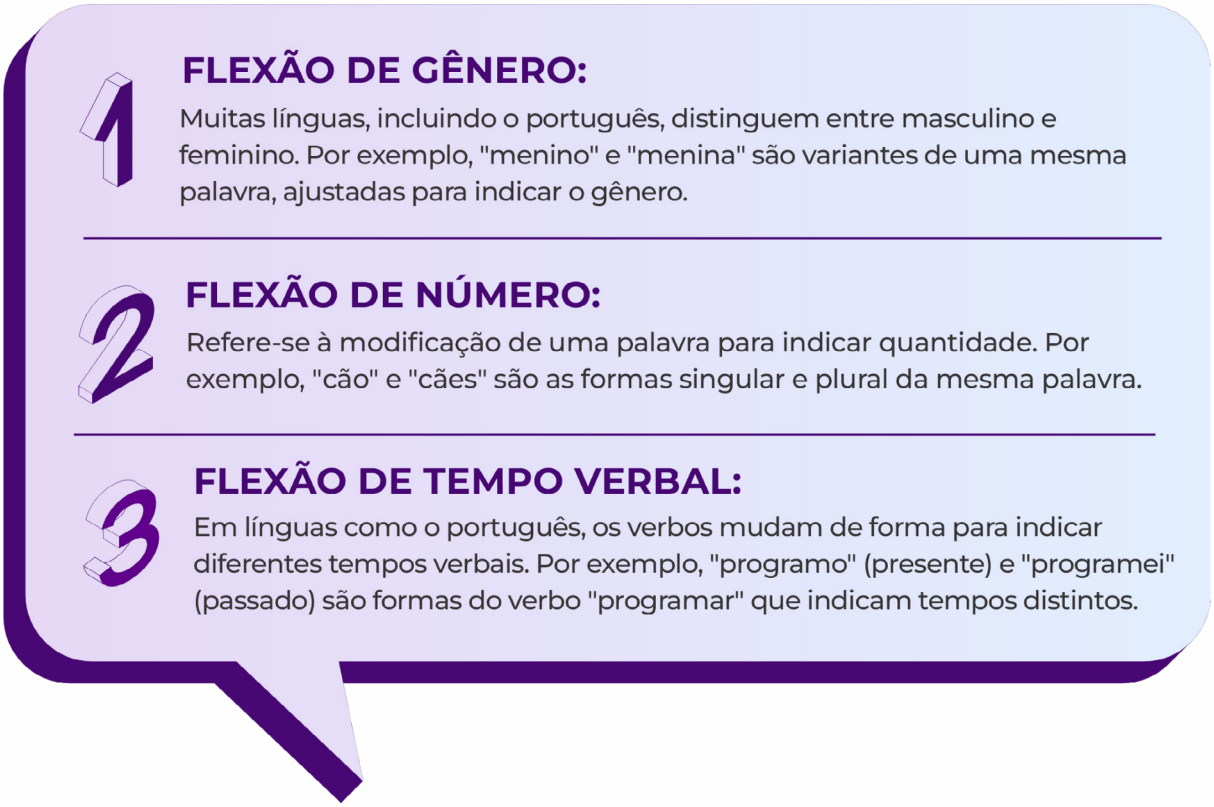
b) **Prefixos:** prefixos são morfemas colocados antes da raiz para mudar seu significado. Eles podem modificar completamente o sentido original da palavra. Por exemplo, o prefixo "in-" em "infeliz" transforma o significado da raiz "feliz" para o seu oposto, indicando ausência ou negação de felicidade. Outro exemplo é o prefixo "des-" em "desfazer", que indica a ação de reverter ou desfazer algo.

c) **Sufixos:** sufixos são morfemas adicionados após a raiz, geralmente, alterando a classe gramatical ou o significado da palavra. Eles podem transformar substantivos em adjetivos, verbos em substantivos, entre outras mudanças. Por exemplo, o sufixo "-dade" em "felicidade" converte o adjetivo "feliz" em um substantivo, indicando a qualidade de ser feliz. Outro exemplo é o sufixo "-mente" em "rapidamente", que transforma o adjetivo "rápido" em um advérbio.

Esses diferentes tipos de morfemas desempenham papéis fundamentais na estruturação e no significado das palavras, tornando-se ferramentas essenciais no estudo da morfologia e na compreensão do PLN.

A morfologia não se limita aos tipos básicos de morfemas; ela também examina como as palavras mudam de forma conforme o contexto em que são usadas. Isso abrange flexões de gênero, número e tempo verbal. De forma mais detalhada, tem-se (Figura 7):

Figura 7 - Exemplos de flexões linguísticas: gênero, número e tempo verbal



Fonte: autoria própria.

Entender a morfologia é essencial para o PLN, pois permite que sistemas computacionais reconheçam, analisem e gerem linguagem de maneira eficaz. No PLN, a análise morfológica decompõe palavras em seus componentes básicos, facilitando tarefas como lematização, *stemming* e reconhecimento de entidades nomeadas. Isso aumenta a precisão de diversas aplicações, desde máquinas de busca até tradutores automáticos e sistemas de compreensão de texto.

A morfologia linguística oferece os instrumentos relevantes para decifrar a formação e o uso das palavras, analisando tanto sua estrutura interna quanto sua aplicação em diversos contextos linguísticos. Dominar essas regras e padrões é crucial para criar tecnologias de PLN que possam lidar e interpretar a linguagem humana de forma eficiente e precisa.

2.2 Processo de Lematização e sua Importância

2.2.1 Lematização

A lematização consiste em converter uma palavra para sua forma base ou canônica, chamada de lema. Esse procedimento é crucial no PLN, pois ajuda a unificar diferentes variações de uma palavra em um formato único, simplificando a análise e o tratamento de textos. A lematização é mais avançada do que o *stemming* (que será visto mais à frente), na medida em que ela leva em conta o contexto da palavra para transformar todas as suas formas flexionadas ou derivadas no seu lema.

Para compreender a relevância da lematização, veja o seguinte exemplo: as palavras "correu", "correndo" e "corrida" são todas simplificadas para "correr". Da mesma forma, "feliz", "felicidade" e "infeliz" são transformadas em "feliz".

A lematização desempenha um papel essencial em várias tarefas de PLN, como análise de texto, mineração de dados e máquinas de busca. Isso ocorre porque a lematização reduz a dimensionalidade do vocabulário, tornando mais fácil lidar com grandes volumes de texto. Reduzir a dimensionalidade implica que, em vez de tratar múltiplas formas de uma palavra como entidades separadas, todas são convertidas em uma única representação. Esse processo simplifica a análise e aprimora a precisão dos modelos de PLN.

Vamos analisar um exemplo prático: a frase "os alunos correram rapidamente para a sala de aula."

> *Sem lematização*, a frase seria dividida em:
> ["Os", "alunos", "correram", "rapidamente", "para", "a", "sala", "de", "aula"].
>
> *Com lematização*, temos:
> ["o", "aluno", "correr", "rapidamente", "para", "o", "sala", "de", "aula"].

Nesse caso, a lematização converte "correram" para "correr" e "alunos" para "aluno". Esse processo ajuda os sistemas de PLN a entender que "correram" e "correr" são variações da mesma ação, o que aprimora a precisão das análises semânticas e estatísticas.

Outro exemplo, vamos considerar a frase "os cientistas estudam diferentes fenômenos naturais."

Sem aplicar a lematização, a frase é segmentada nas seguintes palavras:
["Os", "cientistas", "estudam", "diferentes", "fenômenos", "naturais"].

Após a lematização, temos:
["o", "cientista", "estudar", "diferente", "fenômeno", "natural"].

Nesse processo, "cientistas" é transformado em "cientista", "estudam" em "estudar" e "fenômenos" em "fenômeno". Esse procedimento permite que sistemas de PLN considerem diferentes formas de uma palavra como sendo equivalentes, facilitando a análise.

Em suma, a lematização ajuda a normalizar o texto, diminuindo a dimensionalidade do vocabulário e melhorando a precisão em várias aplicações de PLN. Dessa forma, a lematização se torna uma técnica essencial para qualquer aplicação que envolva a análise e o processamento de grandes volumes de texto.

2.2.2 Normalização de Texto e Redução de Dimensionalidade de Vocabulário

a) Normalização de texto

A normalização de texto é uma técnica que transforma diversas variações de palavras em uma forma única e padronizada. Esse processo é relevante no PLN, na medida em que a diversidade linguística pode complicar a análise e o tratamento de textos. A lematização é um método que auxilia nessa normalização ao converter palavras para seus lemas. Por exemplo, "caminhou", "caminhando" e "caminha" são todas variações do verbo "caminhar". A lematização uniformiza essas variações ao lema "caminhar".

Essa normalização é fundamental em tarefas de análise de texto, como mineração de dados e máquinas de busca, na medida em que ela diminui a complexidade e a variabilidade textual, facilitando a análise semântica e sintática. Ao normalizar o texto por meio da lematização, um sistema de PLN consegue tratar diferentes formas de uma palavra como a mesma entidade, o que melhora a precisão na identificação de conceitos e entidades.

b) Redução da dimensionalidade do vocabulário

No PLN, reduzir a dimensionalidade do vocabulário é essencial para lidar com grandes quantidades de dados textuais. A dimensionalidade do vocabulário se refere ao número de palavras únicas em um *corpus* de texto. Sem normalização, diferentes formas de uma mesma palavra são consideradas entidades distintas, o que aumenta a dimensionalidade do vocabulário. Isso pode resultar em um vocabulário excessivamente grande e difícil de gerenciar.

Como discutido anteriormente, a lematização auxilia na redução da dimensionalidade ao converter todas as variantes de uma palavra em seu lema. Por exemplo, sem lematização, as palavras "caminhar", "caminhou", "caminhando" e "caminha" seriam tratadas como entidades separadas. Com a lematização, todas essas formas são reduzidas a "caminhar", diminuindo o número de palavras únicas no vocabulário.

Os benefícios para as aplicações do PLN são:

a) **Análise semântica:** a lematização é essencial para melhorar a análise semântica, na medida em que se assegura que diferentes formas de uma palavra sejam tratadas como a mesma entidade. Isso é especialmente importante em tarefas como a análise de sentimentos, onde a precisão semântica é crucial. Por exemplo, reconhecer que "feliz" e "felicidade" estão semanticamente relacionados pode aumentar a precisão dos resultados na análise de sentimentos. Veja o Exemplo 1 na Tabela 1.

b) **Máquinas de busca:** máquinas de busca obtêm grande benefício da lematização. Ao normalizar tanto as consultas dos usuários quanto o conteúdo indexado, a lematização permite uma melhor compreensão da intenção por trás das consultas. Por exemplo, se um usuário busca por "comprando livros", a máquina de busca pode associar essa consulta a conteúdos que contenham "comprar livro" ou "compra de livros", resultando em respostas mais relevantes. Veja o Exemplo 2 na Tabela 2.

c) **Mineração de textos:** na mineração de textos, a lematização ajuda a identificar padrões e tendências ao normalizar as diversas formas das palavras. Isso é particularmente útil em análises de grandes *corpora*, onde a variabilidade linguística pode ofuscar padrões subjacentes. A lematização clarifica esses padrões ao tratar diferentes formas de uma palavra como a mesma entidade. Por exemplo**,** suponha que estamos analisando um grande conjunto de notícias sobre economia. As palavras "crescimento", "crescendo" e "cresceram" aparecem em diferentes artigos. Sem a lematização, essas palavras seriam tratadas como entidades distintas, dificultando a análise de tendências sobre o tema "crescimento econômico". Ao aplicar a lematização, todas essas palavras são reduzidas à forma base "crescer", permitindo uma análise mais clara e precisa dos textos. Isso facilita a identificação de padrões de crescimento econômico nas notícias, ajudando analistas a extrair percepções significativas a partir dos dados. Veja o Exemplo 3 na Tabela 3.

d) **Tradução automática:** sistemas de tradução automática também se beneficiam da lematização. Ao normalizar as formas das palavras, a lematização assegura que o sistema traduza consistentemente palavras com o mesmo significado, melhorando a fluidez e precisão das traduções. Por exemplo, em um sistema de tradução, identificar que "escrevendo", "escrever" e "escrevi" são formas do verbo "escrever" permite traduções mais precisas e naturais.

e) **Sistemas de perguntas e respostas:** esses sistemas dependem da lematização para entender a relação entre a consulta do usuário e o *corpus* de dados. Ao normalizar as palavras, a lematização possibilita que o sistema encontre respostas relevantes, mesmo quando as formas das palavras na consulta e na resposta não correspondem exatamente. Por exemplo, se a pergunta for "qual é a capital do Brasil?", a lematização ajuda a identificar que "capital" e "capitais" se referem ao mesmo conceito, facilitando a recuperação da resposta correta: "Brasília".

Tabela 1 - Exemplo 1: análise de sentimentos

TEXTO ORIGINAL	APÓS LEMATIZAÇÃO	SENTIMENTO	EXPLICAÇÃO
"Os alunos estavam felizes e mostravam muita felicidade."	"o aluno estar feliz e mostrar muito felicidade."	positivo	O texto original expressa sentimentos positivos, indicados pelas palavras "felizes" e "felicidade". Após a lematização, essas palavras são normalizadas para suas formas base ("feliz" e "felicidade"), mas o sentimento positivo do texto permanece evidente.
"Os alunos estavam tristes e mostravam grande descontentamento."	"o aluno estar triste e mostrar grande descontentamento."	negativo	O texto original expressa sentimentos negativos, indicados pelas palavras "tristes" e "descontentamento". Após a lematização, essas palavras são normalizadas para suas formas base ("triste" e "descontentamento"), mas o sentimento negativo do texto permanece evidente.

Fonte: autoria própria.

Tabela 2 - Exemplo 2: máquina de busca

CONSULTA DO USUÁRIO	CONSULTA DO USUÁRIO	APÓS LEMATIZAÇÃO	EXPLICAÇÃO
“comprando livros online”	“compra de livro online”	“comprar livro online”	A lematização normaliza as diferentes formas do verbo "comprar" e do substantivo "livro" para suas formas base ("comprar" e "livro"). Isso permite que a máquina de busca reconheça que "comprando", "compra" e "comprar" se referem à mesma ação, e que "livros" e "livro" são variantes do mesmo substantivo. Logo, a consulta "comprando livros online" pode ser correspondida ao conteúdo indexado à "compra de livro online", resultando em uma busca mais eficaz e relevante.

Fonte: autoria própria.

Tabela 3 - Exemplo 3: mineração de textos

TEXTO ORIGINAL	APÓS LEMATIZAÇÃO	EXPLICAÇÃO
Os programadores e as programadoras estavam programando programas usando programas.	“o programador e a programadora estar programar.”	A lematização normaliza as formas plural e flexionadas das palavras. "programadores" e "programadoras" são reduzidos a "programador" e "programadora", respectivamente, e "estavam programando" é reduzido a "estar programar". Isso facilita a análise de textos, permitindo que um sistema de mineração de textos trate "programador" e "programadores" como a mesma entidade e que diferentes formas verbais sejam unificadas. Essa normalização ajuda na identificação de padrões em grandes *corpora* de texto, tornando a análise mais precisa e eficiente.

Fonte: autoria própria.

Em suma, a lematização é uma técnica no PLN que traz diversos benefícios, como a normalização do texto e a redução da quantidade de palavras distintas.

2.3 Conceitos e Utilidades de Etiquetagem de Partes do Discurso

A etiquetagem de partes do discurso, também conhecida como *part-of-speech tagging* em inglês ou *POS tagging*, consiste em identificar e marcar a categoria gramatical de cada palavra presente em um texto. Entre as categorias mais comuns estão substantivos, verbos, adjetivos, advérbios, pronomes, preposições, conjunções, artigos e interjeições. Cada uma dessas categorias tem um papel específico na formação da sintaxe e da semântica de uma frase.

Essa tarefa permite que os sistemas de PLN compreendam a estrutura sintática das frases, algo fundamental para diversas aplicações, como análise de texto, tradução automática e extração de informações. Sem a habilidade de identificar as classes gramaticais das palavras, tais sistemas podem enfrentar desafios para entender o contexto e o significado das palavras dentro de uma frase.

Exemplo: "O cachorro rápido correu pelo parque."

Etiquetas: [o/det, cachorro/noun, rápido/adj, correu/verb, pelo/prep, parque/noun]

Vamos analisar as classes gramaticais de cada palavra nesta frase: "o" é um determinante (*det*), "cachorro" é um substantivo (*noun*), "rápido" é um adjetivo (*adj*), "correu" é um verbo (*verb*), "pelo" é uma preposição (*prep*), e "parque" é um substantivo (*noun*).

A utilidade da etiquetagem de partes do discurso está na:

a) **Análise sintática:** ela envolve determinar a estrutura gramatical de uma frase. A etiquetagem de partes do discurso facilita a compreensão de como as palavras se inter-relacionam para formar essa estrutura. Por exemplo, ao identificar elementos como sujeito, verbo e objeto em uma frase, os sistemas conseguem interpretar adequadamente a ação descrita.

b) **Desambiguação semântica:** muitas palavras possuem múltiplos significados que variam conforme o contexto. A etiquetagem de partes do discurso é essencial para esclarecer esses significados ao identificar a classe gramatical das palavras. Por exemplo, a palavra "banco" pode referir-se a uma instituição financeira ou a um assento, dependendo de seu uso como substantivo.

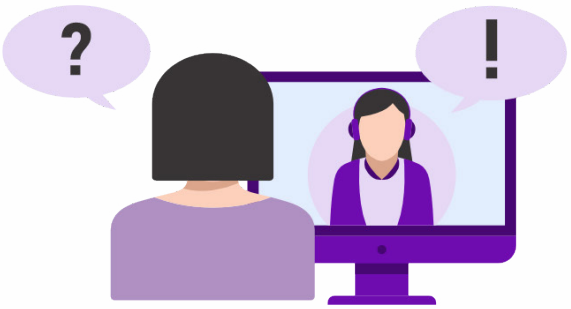
c) **Extração de informação:** a identificação de entidades nomeadas, como nomes de pessoas, lugares, datas, entre outros, é facilitada pela etiquetagem de partes do discurso. Ao reconhecer que uma palavra é um substantivo próprio, pode-se inferir que se trata de um nome ou local significativo, o que é útil em tarefas como mineração de textos e sistemas de resposta a perguntas.

As aplicações práticas da etiquetagem de partes do discurso estão exemplificadas na Figura 8.

Figura 8 - Aplicações práticas da etiquetagem de partes do discurso

TRADUTORES AUTOMÁTICOS

Tradutores automáticos podem utilizar a etiquetagem de partes do discurso para melhorar a precisão das traduções. Ao entender a função de cada palavra em uma frase, os tradutores podem escolher a tradução mais apropriada para cada contexto.

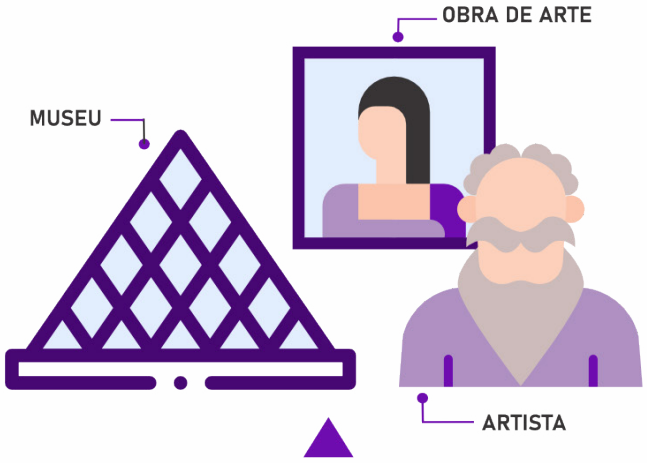


SISTEMAS DE PERGUNTAS E RESPOSTAS

Assistentes virtuais podem depender da etiquetagem de partes do discurso para compreender e responder corretamente às perguntas dos usuários. Ao identificar as classes gramaticais, esses sistemas podem extrair a informação relevante e formular respostas precisas.

ANÁLISE DE SENTIMENTOS

Na análise de sentimentos, a etiquetagem de partes do discurso ajuda a identificar adjetivos e advérbios que expressam emoções, permitindo uma análise mais precisa das opiniões e sentimentos expressos em textos, como avaliações de produtos e postagens em redes sociais. Por exemplo, em uma avaliação de produto que diz "o produto é excelente e chegou rapidamente," a etiquetagem pode identificar "excelente" como um adjetivo positivo e "rapidamente" como um advérbio que indica satisfação com a entrega.



RECONHECIMENTO DE ENTIDADE NOMEADAS

A etiquetagem de partes do discurso é um passo relevante no processo de reconhecimento de entidades nomeadas. Identificar nomes próprios, lugares e organizações em textos é essencial para a extração de dados e informações estruturadas de grandes volumes de textos não estruturados. Por exemplo, em um texto que menciona "Louvre", "Leonardo da Vinci" e "Monalisa", é importante reconhecer "Louvre" como um lugar (museu), "Leonardo da Vinci" como um nome próprio (artista) e "Monalisa" como uma obra de arte.

Fonte: autoria própria.

Diversos algoritmos e técnicas são utilizados para implementar a etiquetagem de partes do discurso. Alguns dos métodos mais comuns incluem:

a) **Regras linguísticas:** esse método utiliza regras gramaticais predefinidas para identificar a classe gramatical de cada palavra. Embora seja direto, enfrenta limitações devido à complexidade e à variedade das regras necessárias para cobrir todas as exceções da língua. Por exemplo, uma regra pode determinar que uma palavra imediatamente após um artigo definido ("o", "a") é provavelmente um substantivo, como em "o gato" ou "a casa". No entanto, essa abordagem pode falhar em casos mais complexos, como "a chuva" (substantivo) versus "a partir" (locução prepositiva), evidenciando a necessidade de regras adicionais para lidar com exceções.

b) **Modelos baseados em estatísticas:** algoritmos como os modelos de Markov e os modelos de máxima entropia utilizam dados de treinamento para aprender as probabilidades de transições entre diferentes classes gramaticais. Esses modelos são mais flexíveis e podem generalizar melhor a partir dos dados. Por exemplo, um modelo de Markov pode prever a sequência de partes do discurso em uma sentença, ajudando a determinar se uma palavra como "corrente" está sendo usada como substantivo ("uma corrente de água") ou como adjetivo ("uma situação corrente").

c) **Aprendizado de máquina:** técnicas de aprendizado de máquina, como redes neurais e máquinas de vetores de suporte, são treinadas em grandes *corpora* de texto anotado para aprender a classificar palavras em suas classes gramaticais. Esses métodos podem alcançar alta precisão, especialmente quando combinados com representações de palavras, como *embeddings*. *Embeddings* são representações vetoriais densas de palavras, frases ou outros elementos de texto em um espaço contínuo de alta dimensão. Por exemplo, uma rede neural treinada em um grande conjunto de dados de texto pode aprender a classificar a palavra "banco" como substantivo em "ele foi ao banco para sacar dinheiro" e como verbo em "eles bancam as despesas".

d) **Redes neurais profundas:** abordagens mais recentes utilizam redes neurais profundas, como redes neurais recorrentes e *transformers*, que podem capturar dependências de longo alcance em textos. Essas técnicas têm se mostrado muito eficazes em tarefas de etiquetagem do discurso, alcançando resultados significativos. Por exemplo, um modelo baseado em *transformers*, como o BERT, pode analisar a frase "o banco central anunciou novas medidas econômicas", identificando corretamente "banco" como um substantivo (instituição financeira) e "anunciou" como um verbo, devido à sua capacidade de considerar o contexto completo da frase.

Apesar de suas vantagens, a etiquetagem de partes do discurso enfrenta diversos desafios:

a) **Ambiguidade lexical:** palavras podem pertencer a diferentes classes gramaticais dependendo do contexto em que são usadas. Compreender o contexto é essencial para resolver essa ambiguidade. Por exemplo, "banco" pode signi-

ficar uma instituição financeira ou um assento. Em "vou ao banco para sacar dinheiro", "banco" é uma instituição financeira. Já em "sentei no banco do parque", "banco" se refere a um assento.

b) **Linguagem informal e gírias:** textos informais, como postagens em redes sociais e mensagens de texto, frequentemente contêm gírias, abreviações e erros gramaticais, complicando a etiquetagem do discurso. É um desafio contínuo desenvolver modelos robustos que operem com essa variabilidade. Por exemplo, em uma mensagem como "vou pra balada hj c/ os amigos", o modelo deve entender que "pra" é "para", "hj" é "hoje", e "c/" é "com", ajustando os elementos do discurso corretamente apesar das abreviações e informalidades.

c) **Mudanças de código:** em regiões bilíngues é comum encontrar textos que alternam entre diferentes línguas. A etiquetagem do discurso precisa ser eficaz ao lidar com essas mudanças de código. Por exemplo, em "je vais visitar mes parents demain," o sistema deve reconhecer "visitar" como um verbo em português e "mes parents" como um substantivo e pronome possessivo em francês, enquanto identifica "je vais" e "demain" como elementos em francês.

d) **Domínios específicos:** diferentes domínios, como o jurídico, médico ou técnico, têm vocabulários e estruturas gramaticais próprias. Para serem eficazes, os modelos de etiquetagem de partes do discurso precisam ser adaptados ou treinados especificamente para esses domínios. Por exemplo, em textos médicos, "dose" geralmente é um substantivo ("a dose do medicamento"), enquanto em um contexto jurídico, "processo" refere-se a uma ação legal, exigindo que o modelo reconheça seu uso como substantivo específico para o domínio jurídico.

Existem várias ferramentas e bibliotecas disponíveis que facilitam a implementação de etiquetagem do discurso em aplicações de PLN:

a) **NLTK:** é uma biblioteca em Python utilizada para tarefas de PLN. Ela inclui módulos para etiquetagem do discurso, *tokenização*, *stemming*, lematização.

b) **SpaCy:** é uma biblioteca de PLN também escrita em Python. Ela é conhecida por sua eficiência e facilidade de uso e inclui modelos pré-treinados para etiquetagem do discurso.

c) **Stanford NLP:** é uma ferramenta desenvolvida pela Universidade de Stanford que também oferece funcionalidades para o PLN, incluindo etiquetagem do discurso, análise de dependência e reconhecimento de entidades nomeadas.

Mais adiante, vamos apresentar o funcionamento dessas ferramentas com exemplos práticos. Elas oferecem uma base satisfatória para a implementação de etiquetagem de textos em várias aplicações de PLN. Com esses recursos, é viável realizar uma análise linguística eficiente.

2.4 Processo de *Stemming*

A técnica de *stemming* ou radicalização é essencial no PLN e consiste em reduzir as palavras às suas formas base ou raízes. Isso é feito, geralmente, por meio da remoção de sufixos e prefixos. O objetivo é simplificar as palavras para que diferentes variantes de uma mesma palavra sejam tratadas como equivalentes.

Em termos conceituais, o *stemming* é um método de normalização que transforma palavras derivadas em uma raiz comum. Ao contrário da lematização, que usa dicionários e regras linguísticas para garantir que a forma resultante seja uma palavra real, o *stemming* é uma abordagem mais direta e simplificada. Por essa razão, a forma obtida após o *stemming* pode não ser uma palavra válida em nenhum idioma.

Exemplo: "jogando", "joguei", "jogador" → "jog"

No exemplo apresentado acima, as variações da palavra "jogar" como "jogando", "joguei" e "jogador" são reduzidas à raiz "jog". Embora "jog" não seja uma palavra reconhecida no português, o processo de *stemming* possibilita tratar essas formas derivadas como equivalentes para a análise.

2.4.1 Importância e Aplicações do *Stemming*

O *stemming* desempenha um papel importante em diversas aplicações de PLN, especialmente quando a precisão exata não é essencial. Entre os principais usos, podemos destacar:

a) **Máquinas de busca:** máquinas de busca utilizam *stemming* para aumentar a relevância dos resultados. Ao tratar diferentes formas de uma palavra como equivalentes, uma máquina de busca pode fornecer resultados mais abrangentes e pertinentes. Por exemplo, ao buscar "natação", o sistema também encontrará páginas com "nadar", "nadando" e "nadou".

b) **Análise de texto:** na análise de texto, o *stemming* facilita o agrupamento de palavras relacionadas, ajudando a identificar temas e padrões. Por exemplo, em uma análise de *retorno* de clientes, agrupar palavras como "satisfazer", "satisfeito" e "satisfação" pode oferecer uma visão mais clara sobre a opinião geral dos clientes sobre um produto ou serviço.

c) **Mineração de dados textuais:** na mineração de dados textuais, reduzir palavras às suas raízes ajuda a diminuir a dimensionalidade do conjunto de dados, tornando a análise mais eficiente e manejável. Isso é particularmente útil em grandes *corpora* de texto, onde há uma grande variedade de formas de palavras.

Exemplo:

Consideremos as palavras: "navegar", "navegou", "navegando".

Stemming: "naveg", "naveg", "naveg".

Nesse exemplo, todas as variações da palavra "navegar" são reduzidas à raiz "naveg". Esse procedimento facilita a comparação e a análise, pois trata todas essas variações como se fossem iguais. Existem bibliotecas em *Python* que realizam o processo de *stemming*, e elas serão abordadas mais adiante.

O *stemming* é uma técnica utilizada para reduzir palavras às suas formas básicas. Este método proporciona importantes benefícios em termos de eficiência e precisão para várias aplicações. No entanto, é crucial estar ciente de algumas limitações associadas a esta abordagem, que apresentamos a seguir.

2.4.2 Vantagens

a) **Simplicidade e eficiência:** o *stemming* é uma técnica simples de implementar e exige poucos recursos computacionais, o que a torna ideal para aplicações em tempo real e para a análise de grandes volumes de texto.

b) **Redução da dimensionalidade:** ao converter palavras derivadas para suas raízes, o *stemming* ajuda a reduzir a quantidade de variáveis no conjunto de dados textuais, tornando a análise e interpretação mais manejáveis.

c) **Melhoria na recuperação de informação:** em sistemas de busca e recuperação de informação, o *stemming* pode ampliar o alcance dos resultados encontrados, proporcionando uma experiência de usuário mais satisfatória.

2.4.3 Desvantagens

a) **Imprecisão:** o *stemming* pode criar ambiguidades e formas de palavras que não são reconhecíveis, levando a possíveis erros na interpretação do texto. Por exemplo, as palavras "correr" e "corrente" podem ser reduzidas à raiz "corr", mesmo que seus significados sejam distintos.

b) **Perda de informação semântica:** ao reduzir palavras às suas raízes, o *stemming* pode eliminar nuances e detalhes importantes, dificultando análises mais precisas e detalhadas. Por exemplo, ao aplicar *stemming* às palavras "gato", "gata", "gatos" e "gatas", todas podem ser reduzidas à raiz "gat", eliminando informações sobre gênero e número, essenciais para a compreensão completa do contexto.

c) **Dependência do idioma:** as regras de *stemming* são específicas para cada idioma. Um algoritmo de *stemming* projetado para o inglês pode não funcionar corretamente para o português, e vice-versa. Por exemplo, o *PorterStemmer*, é uma técnica amplamente utilizada para o inglês, ela pode não reduzir adequadamente as palavras portuguesas "correr", "correu" e "correndo" à mesma raiz. Em contraste, o *RSLPStemmer*, desenvolvido para o português, consegue transformar essas palavras na raiz comum "corr". Isso ressalta a importância de escolher algoritmos de *stemming* apropriados para o idioma específico em questão.

Existem vários algoritmos comuns de *stemming* desenvolvidos para diferentes idiomas e propósitos, os quais serão apresentados mais à frente.

2.4.4 Aplicação Prática em Máquinas de Busca

Nas ferramentas de busca, o uso do *stemming* pode aumentar a eficácia e a relevância dos resultados apresentados. Imagine um usuário que busca por "melhorar a saúde". Um mecanismo de busca que emprega o *stemming* pode retornar resultados que incluem termos como "melhoria da saúde", "melhorando a saúde" e "melhores práticas de saúde", tratando todas essas variações como associadas ao termo original pesquisado.

2.5 Comparação entre *Stemming* e Lematização

Stemming e lematização são processos importantes no PLN, cada um com suas vantagens conforme a necessidade da aplicação. O *stemming*, por ser simples e rápido, é especialmente útil em máquinas de busca. Já a lematização é mais indicada para tarefas que requerem precisão semântica.

Na Tabela 4, é mostrada uma análise comparativa entre as técnicas de *stemming* e lematização, ressaltando as diferenças em relação à precisão, velocidade, complexidade e utilidade em várias aplicações de PLN.

Tabela 4 - Comparação entre *stemming* e lematização

CRITÉRIOS	*STEMMING*	LEMATIZAÇÃO
Precisão	Menos preciso, dado que não garante que a forma resultante seja uma palavra válida. Pode introduzir ambiguidades ao reduzir palavras diferentes à mesma raiz.	Mais precisa, na medida em que transforma as palavras em formas válidas e preserva o significado. Considera o contexto e a análise morfológica.
Velocidade	Geralmente mais rápido, dado que envolve a aplicação de regras simples e não requer acesso a dicionários complexos.	Pode ser mais lenta, dado que requer a consulta de dicionários e análise mais aprofundada para determinar a forma base correta.
Complexidade	Simples de implementar e requer menos recursos computacionais.	Mais complexa, dado que ela exige mais recursos computacionais e linguísticos para análise.
Aplicabilidade	Adequado para aplicações em que a velocidade é crítica e pequenas imprecisões são aceitáveis, como máquinas de busca e mineração de textos.	Preferível em aplicações que exigem precisão e clareza semântica, como análise de sentimentos, tradução automática e sistemas de resposta a perguntas.

Fonte: autoria própria.

2.6 Casos de Uso Práticos

Vamos apresentar agora alguns exemplos práticos que foram discutidos anteriormente, a fim de ilustrar conceitos e possíveis aplicações, a saber:

a) **Máquina de busca:** em uma máquina de busca, a rapidez na recuperação de informações é fundamental. Para isso, o *stemming* é frequentemente utilizado. Por exemplo, se um usuário procura por "gerenciamento de projetos", a máquina de busca deve ser capaz de encontrar documentos relevantes que contenham variações como "gerenciar", "gerenciado" ou "gerenciando". O *stemming* ajuda a agrupar essas variações, acelerando o processo de busca.

b) **Análise de sentimentos:** na análise de sentimentos, a precisão semântica é vital para compreender a emoção ou a opinião expressa em um texto. A lematização é preferida nesse caso, pois preserva o significado das palavras. Por exemplo, para analisar a frase "eles estão rindo das minhas piadas", a lematização garante que "rindo" seja entendida como "rir", preservando o contexto e o sentimento positivo associado à palavra.

c) **Tradução automática:** sistemas de tradução automática frequentemente utilizam técnicas de PLN, como a lematização, para produzir traduções precisas. A lematização assegura que cada palavra seja traduzida de acordo com seu contexto correto, mantendo o sentido da frase original. Por exemplo, ao traduzir "ela cantava lindamente" para outra língua, é necessário lematizar "cantava" para "cantar" antes de realizar a tradução, garantindo que a conjugação correta seja aplicada na língua de destino.

d) **Sistemas de perguntas e respostas:** assistentes virtuais utilizam a lematização para compreender e responder corretamente às perguntas dos usuários. Quando um usuário pergunta "onde posso encontrar receitas de bolo?", o sistema precisa entender que "bolo" e "bolos" têm o mesmo lema "bolo" para fornecer uma resposta precisa.

Em suma, o uso de *stemming* é particularmente indicado em situações onde a velocidade é prioritária, como em mecanismo de busca e mineração de textos em larga escala. Já a lematização é fundamental para aplicações que demandam precisão linguística e clareza semântica, como análise de sentimentos, tradução automática e sistemas de perguntas e respostas. Ao selecionar a técnica mais apropriada, os desenvolvedores de PLN podem aprimorar seus sistemas para alcançar os melhores resultados possíveis.

2.7 Algoritmos e Técnicas Comuns Utilizados na Análise Morfológica

Os algoritmos e técnicas de análise morfológica são fundamentais no PLN para identificar e processar a estrutura das palavras. Alguns dos algoritmos mais comuns incluem:

a) **Algoritmo de Porter:** um dos algoritmos de *stemming* mais populares, utilizado para remover sufixos comuns em inglês.

b) **WordNet:** uma base de dados lexical que agrupa palavras em conjuntos de sinônimos e fornece suas formas base ou lematização.

c) **SpaCy:** uma biblioteca moderna que inclui funções avançadas de lematização e etiquetagem de partes do discurso.

d) **NLTK:** outra biblioteca popular que oferece diversas ferramentas para análise morfológica, incluindo *stemming* e lematização.

2.7.1 Algoritmo de Porter

O algoritmo de Porter é um dos algoritmos de *stemming* mais populares. Ele remove os sufixos das palavras para reduzir uma palavra ao seu radical ou raiz. O algoritmo opera colocando em prática uma série de regras aplicadas em cinco etapas.

- **Palavra original:** "playing"
- **Identificação do sufixo:** O algoritmo identifica o sufixo "ing" na palavra "playing".
- **Remoção do sufixo:** O sufixo "ing" é removido, resultando na palavra "play".
- **Palavra após stemming:** "play" é a forma base ou radical da palavra original "playing".
- **Implementação:** codificação simplificada em Python.

```
# Importação da biblioteca necessária
from nltk.stem import PorterStemmer

# Criação do objeto stemmer
stemmer = PorterStemmer()

# Lista de palavras a serem processadas
words = ["playing", "played", "plays", "playful"]

# Aplicação do algoritmo de Porter para cada palavra na lista
stems = [stemmer.stem(word) for word in words]

print(stems)        # Exibição dos resultados
```

```
Entrada: ["playing", "played", "plays", "playful"]
Saída: ['play', 'play', 'play', 'play']
```

Nesse programa de exemplo, todas as variações da palavra "play" são reduzidas à mesma raiz "play", ilustrando como o *stemming* pode ser usado para normalizar texto e facilitar a mineração e análise de dados textuais.

2.7.2 *WordNet*

O *WordNet* é uma base de dados lexical que organiza palavras em conjuntos de sinônimos e apresenta suas formas básicas, conhecidas como lemmas.

WordNet é uma base de dados lexical que nos permite encontrar sinônimos, antônimos, definições e outras informações sobre palavras. Vamos explorar como utilizar o WordNet para descobrir sinônimos da palavra "run".

Passo a passo da implementação em Python:

a) importar o *corpus WordNet* da biblioteca NLTK;

b) Iterar por meio dos synsets para encontrar de sinônimos da palavra "run"; e

c) para cada *synset*, coletar todos os sinônimos (lemmas) associados.

```
# Importação da biblioteca necessária
from nltk.corpus import wordnet


# Lista para armazenar os sinônimos
synonyms = []


# Iteração por meio dos synsets da palavra "run"
for syn in wordnet.synsets("run"):
      # Iteração através dos lemas de cada synset
      for lemma in syn.lemmas():
                  # Adiciona o nome do lemma à lista de sinônimos
                  synonyms.append(lemma.name())


# Remove duplicatas convertendo a lista para um conjunto
unique_synonyms = set(synonyms)


# Exibição dos sinônimos únicos
print(unique_synonyms)
```

```
Entrada: run
Saída: {'run', 'running', 'operate', 'function', 'melt', 'ladder',
'test', 'play', 'discharge', 'consort', 'work', 'tend', 'race', 'go',
'lead'}
```

Ao inserir "run" no *WordNet*, ele retorna vários sinônimos que podem ser usados em diferentes contextos. Por exemplo, "operate" e "function" são sinônimos no contexto de máquinas funcionando, enquanto "race" e "running" são sinônimos no contexto de competição ou movimento. Isso mostra a riqueza e a versatilidade do vocabulário que *WordNet* pode fornecer para análise de texto e PLN.

2.7.3 SpaCy

O SpaCy é uma biblioteca para PLN em Python. Vamos ver como utilizá-la para realizar lematização e marcação do texto em uma frase em português.

Passo a passo da implementação em Python:

a) importar a biblioteca SpaCy e carregar o modelo de linguagem em português;

b) criar um objeto doc processando uma frase;

c) extrair os lemas das palavras na frase; e

d) extrair as partes para a marcação do discurso de cada palavra na frase.

```
# Importação da biblioteca necessária
import spacy

# Carregamento do modelo de linguagem em português
nlp = spacy.load("pt_core_news_sm")

# Texto a ser processado
texto = "A Apple está olhando para comprar uma startup do Reino Unido por 1 bilhão de dólares"

# Criação do objeto doc
doc = nlp(texto)

# Lematização: extração dos lemas das palavras
lemmas = [token.lemma_ for token in doc]
print("Lemas:", lemmas)

# POS tagging: extração das partes do discurso
pos_tags = [(token.text, token.pos_) for token in doc]
print("Etiquetagem:", pos_tags)
```

```
Entrada:   "A Apple está olhando para comprar uma startup do Reino Unido por 1 bilhão de dólares"
```

continua

```
Saída:

Lemas:     ['o', 'Apple', 'estar', 'olhar', 'para', 'comprar', 'um',
'startup', 'de', 'o', 'Reino', 'Unido', 'por', '1', 'bilhão', 'de',
'dólar']


Etiquetagem:     [('A', 'DET'), ('Apple', 'PROPN'), ('está', 'AUX'),
('olhando', 'VERB'), ('para', 'ADP'), ('comprar', 'VERB'), ('uma',
'DET'), ('startup', 'NOUN'), ('do', 'ADP'), ('Reino', 'PROPN'),
('Unido', 'PROPN'), ('por', 'ADP'), ('1', 'NUM'), ('bilhão', 'NUM'),
('de', 'ADP'), ('dólares', 'NOUN')]
```

Lemas: essa lista mostra as formas base das palavras na frase. Por exemplo, "está" foi reduzido para "estar" e "olhando" para "olhar".

Etiquetagem: essa lista mostra cada palavra emparelhada com sua respectiva parte do discurso. Por exemplo, "apple" é uma "propn" (nome próprio), "olhando" é um "verb" (verbo) e "startup" é um "noun" (substantivo).

Essas técnicas são essenciais para várias tarefas de PLN, como análise de texto, tradução automática e recuperação de informação, pois ajudam a entender melhor a estrutura e o significado do texto.

2.7.4 NLTK

NLTK provê diversas ferramentas para análise morfológica, incluindo *stemming* e lematização. Neste exemplo, vamos utilizar o NLTK para realizar a lematização de uma frase em inglês.

Passo a passo da implementação em Python:

a) importar as funções necessárias do NLTK;

b) criar o objeto WordNetLemmatizer;

c) dividir a frase em palavras ou tokens;

d) identificar as classes gramaticais de cada token;

e) lematizar cada token com base em sua classe gramatical.

```
# Importação das bibliotecas necessárias
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import pos_tag, word_tokenize
import nltk

# Certifique-se de que os recursos necessários estão disponíveis
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

# Função para converter etiquetas POS do NLTK para as etiquetas do
  WordNet
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
       return wordnet.ADV
 else:
return wordnet.NOUN

# Criação do objeto lematizador
lemmatizer = WordNetLemmatizer()

# Frase a ser lematizada
sentence = "The striped bats are hanging on their feet for best"

# Tokenização da frase
```

continua

```
tokens = word_tokenize(sentence)


# Identificação das classes gramaticais

tags = pos_tag(tokens)


# Exibição das classes gramaticais

print("Etiquetagem::", tags)


# Lematização dos tokens com base na classe gramatical

lemmas = [

    lemmatizer.lemmatize(token, get_wordnet_pos(pos))

    for token, pos in tags

]


# Exibição dos lemas

print("Lemas:", lemmas)
```

```
Entrada:   "The striped bats are hanging on their feet for best"

Saída:

Etiquetagem:    [('The', 'DT'), ('striped', 'JJ'), ('bats', 'NNS'),
('are', 'VBP'), ('hanging', 'VBG'), ('on', 'IN'), ('their', 'PRP$'),
('feet', 'NNS'), ('for', 'IN'), ('best', 'JJS')]


Lemas:    ['The', 'striped', 'bat', 'be', 'hang', 'on', 'their', 'foot',
'for', 'best']
```

Lemas: essa lista mostra as formas base das palavras na frase. Por exemplo, "are" foi reduzido para "be" e "hanging" para "hang".

Etiquetagem: essa lista mostra cada palavra emparelhada com sua respectiva parte do discurso. Por exemplo, "The" é um "DT" (determinante), "striped" é um "JJ" (adjetivo) e "bats" é um "NNS" (substantivo no plural).

Os exemplos mostraram a utilização de algoritmos e bibliotecas para executar a análise morfológica, a lematização e a etiquetagem de partes do discurso. Essas técnicas são importantes para a normalização e o processamento de textos em diversas aplicações de PLN.

2.8. Prática de Ferramentas de Processamento de Linguagem Natural

O *Colab* tem como foco a aplicação prática de técnicas essenciais de PLN utilizando as bibliotecas SpaCy, NLTK e *WordNet*. As atividades propostas seguem uma ordem lógica e progressiva, permitindo que os(as) discentes apliquem conceitos como lematização, etiquetagem de partes do discurso, *stemming*, análise de sentimentos e indexação de documentos.

Cada atividade inclui uma breve explicação teórica, seguida de exemplos práticos com código em Python. O *Colab* incentiva os(as) discentes a executarem o código, fazerem alterações nas entradas e observarem como os resultados mudam, o que ajuda a consolidar o aprendizado.

O material também explora temas como a importância da lematização para a normalização de texto e a redução da dimensionalidade. Técnicas de indexação, como *Term Frequency - Inverse Document Frequency* (TF-IDF), são introduzidas e aplicadas para facilitar a organização e recuperação de documentos. As atividades oferecem aos alunos a oportunidade de comparar métodos, como *stemming* e lematização, além de aplicar análise de sentimentos e praticar a indexação de textos em português.

O *Colab* foi planejado para proporcionar uma experiência prática e interativa, permitindo aos alunos desenvolver suas habilidades em PLN de forma progressiva e aplicada.

Acesse o ***Notebook Colab*** pelo *link* a seguir e inicie suas atividades:

Notebook Colab

Objetivos de Aprendizagem

O objetivo é capacitar o estudante a explorar e aplicar recursos práticos para realizar análise morfológica no contexto do Processamento de Linguagem Natural. Este conhecimento será adquirido por meio da execução de pequenos exemplos e exercícios que destacam a importância e a aplicação da análise morfológica em diversas tarefas de processamento de texto.

Atividade 1: Processo de Lematização e sua Importância

Objetivo: Compreender e aplicar o processo de lematização.

Tarefa 1.1: Utilizar a biblioteca spacy para lematizar um conjunto de palavras fornecidas.

Definição de lemetização:

A lematização é o processo de reduzir palavras às suas formas base ou raiz, conhecidas como lemas. Esse processo leva em conta o contexto em que a palavra é usada,

transformando diferentes formas gramaticais em uma única representação. Isso é particularmente útil em tarefas de processamento de linguagem natural porque ajuda a unificar palavras que têm a mesma raiz, mas diferentes formas.

Exemplo: Vamos considerar as seguintes palavras em português: **correndo, correu, correm, facilmente, justamente**

Ao aplicar a lematização, essas palavras são reduzidas às suas formas base: **correndo -> correr, correu -> correr, correm -> correr, muitos -> muito, justos -> justo**

Nota: Neste exemplo, a biblioteca SpaCy é utilizada para processar o texto e aplicar a lematização. As palavras "correndo", "correu" e "correm" são reduzidas ao lema "correr", enquanto "muitos" e "justos" são simplificadas para "muito" e "justo", respectivamente.

A seguir, um pequeno exemplo de como realizar a lematização em Python usando a biblioteca SpaCy:

1. execute o código e analise a saída; e
2. mude as palavras da linha 24 do código Python e re-execute.

```
import subprocess
import sys

# Função para instalar pacotes
def install_package(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

# Verificar se SpaCy está instalado
try:
    import spacy
except ImportError:
    print("SpaCy não está instalado. Instalando SpaCy...")
    install_package("spacy")

# Verificar se o modelo de linguagem em português está instalado
try:
    nlp = spacy.load('pt_core_news_sm')
except OSError:
    print("Modelo 'pt_core_news_sm' não encontrado. Baixando o modelo...")
    subprocess.check_call([sys.executable, "-m", "spacy", "download", "pt_core_
news_sm"])
    nlp = spacy.load('pt_core_news_sm')

# Texto de exemplo
texto = "correndo correu correm muitos justos"
```

continua

```
# Processar o texto
doc = nlp(texto)

# Aplicar lematização
palavras_lematizadas = [token.lemma_ for token in doc]

print(palavras_lematizadas)
  ['correr', 'correr', 'correr', 'muito', 'justo']
```

Tarefa 1.2: Tomando como base o conteúdo lido no ebook, escreva em algumas palavras: a importância da lematização na normalização de texto e redução de dimensionalidade de vocabulário. Escrever a resposta no espaço indicado a seguir.

Resposta Tarefa 1.2: (escreva aqui):

Atividade 2: Etiquetagem de Partes do Discurso

Objetivo: Compreender os conceitos e utilidades da etiquetagem de partes do discurso (POS tagging).

Tarefa 2.1: Usar as bibliotecas spaCy (ex. em Português) e NLTK (ex. em Inglês) para realizar POS tagging em um texto fornecido.

Definição: Etiquetagem de partes do discurso (ou POS tagging):

Trata-se de uma técnica no processamento de linguagem natural que consiste em atribuir etiquetas gramaticais a cada palavra em um texto. Essas etiquetas identificam a função gramatical das palavras, como substantivo, verbo, adjetivo, advérbio, entre outros.

Abordagens Baseadas em Regras:

Estas abordagens utilizam um conjunto de regras gramaticais manuais para atribuir etiquetas.

Ferramentas e Bibliotecas: Diversas ferramentas e bibliotecas de PLN oferecem funcionalidades de POS tagging. Algumas das mais populares incluem:

NLTK (Natural Language Toolkit): Biblioteca poderosa em Python, oferece etiquetagem de POS junto com muitas outras funcionalidades para PLN.

SpaCy: Biblioteca em Python otimizada para desempenho e produção, com modelos pré-treinados para várias línguas.

Legenda: Cores e Tags

Substantivos (NN, NNS): Azul

Verbos (VB, VBD, VBZ, etc.): Verde

Adjetivos (JJ): Laranja

Advérbios (RB): Roxo

Determinantes e Preposições (DT, IN): Vermelho

Segue exemplo de Código com NLTK:

1. execute o código e analise a saída; e
2. mude as palavras da linha 14 do código Python e re-execute.

```
import subprocess
import sys

# Função para instalar pacotes
def instalar_pacote(pacote, versao=None):
    subprocess.check_call([sys.executable, "-m", "pip", "install", pacote])

# Verificar e instalar NLTK se necessário
try:
    import nltk
except ImportError:
    instalar_pacote("nltk")
    import nltk

from nltk.tokenize import word_tokenize

# Baixar o modelo de POS tagging
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')  # add
nltk.download('punkt')
nltk.download('punkt_tab')   # add

# Texto a ser etiquetado
texto = "The black cattle of Renata sleeps on the porch."
tokens = word_tokenize(texto)
pos_tags = nltk.pos_tag(tokens)
```

continua

```python
# Função para colorir e formatar texto
def format_pos_tags(pos_tags):
    formatted_text = []
    for word, tag in pos_tags:
        color = "black"
        if tag.startswith('NN'):
            color = "blue"
        elif tag.startswith('VB'):
            color = "green"
        elif tag.startswith('JJ'):
            color = "orange"
        elif tag.startswith('RB'):
            color = "purple"
        elif tag.startswith('DT') or tag.startswith('IN'):
            color = "red"
        formatted_text.append(f"<b><span style='color:{color}'>{word} ({tag})</span></b>")
    return ' '.join(formatted_text)

# Formatar e exibir a saída
from IPython.core.display import HTML
formatted_text = format_pos_tags(pos_tags)
HTML(formatted_text)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

The (DT) black (JJ) cattle (NN) of (IN) Renata (NNP) sleeps (NNS) on (IN) the (DT) porch (NN) . (.)

Legenda: Cores e Tags (Português):

Substantivos (NOUN): Azul

Verbos (VERB): Verde

Adjetivos (ADJ): Laranja

Advérbios (ADV): Roxo

Determinantes e Preposições (DET, ADP): Vermelho

Segue exemplo de Código com SpaSy.

1. execute o código e analise a saída; e
2. mude as palavras da linha 23 do código Python e re-execute.

```
import subprocess
import sys

# Função para instalar pacotes
def instalar_pacote(pacote):
    subprocess.check_call([sys.executable, "-m", "pip", "install", pacote])

# Verificar e instalar SpaCy se necessário
try:
    import spacy
except ImportError:
    instalar_pacote("spacy")
    import spacy

# Verificar e instalar o modelo de linguagem para português
try:
    nlp = spacy.load('pt_core_news_sm')
except OSError:
    subprocess.check_call([sys.executable, "-m", "spacy","download","pt_core_
news_sm"])
    nlp = spacy.load('pt_core_news_sm')

# Texto a ser etiquetado
texto = "O gado preto da Renata dorme na varanda."
doc = nlp(texto)
```

continua

```
# Função para colorir e formatar texto
def format_pos_tags(doc):
    formatted_text = []
    for token in doc:
        color = "black"
        if token.pos_ == 'NOUN':
            color = "blue"
        elif token.pos_ == 'VERB':
            color = "green"
        elif token.pos_ == 'ADJ':
            color = "orange"
        elif token.pos_ == 'ADV':
            color = "purple"
        elif token.pos_ in ['DET', 'ADP']:
            color = "red"
        formatted_text.append(f"<b><span style='color:{color}'>{token.text}
({token.pos_})</span></b>")
    return ' '.join(formatted_text)

# Formatar e exibir a saída
from IPython.core.display import HTML
formatted_text = format_pos_tags(doc)
HTML(formatted_text)
```

O (DET) gado (NOUN) preto (ADJ) da (ADP) Renata (NOUN) dorme (ADJ) na (ADP) varanda (NOUN) (PUNCT)

Tarefa 2.2: Tomando como base o conteúdo lido no ebook, escreva em algumas palavras a importância da etiquetagem de partes do discurso (POS tagging). Escrever a resposta no espaço indicado a seguir.

Resposta Tarefa 2.2: (escreva aqui):

Atividade 3: Processo de Stemming

Objetivo: Compreender e aplicar o processo de stemming.

Tarefa 3.1: Utilizar o algoritmo de RSLP para *stemming* em um conjunto de palavras fornecidas.

Definição: Stemming

O stemming é uma técnica fundamental no Processamento de Linguagem Natural que tem como objetivo reduzir palavras flexionadas ou derivadas ao seu radical ou

raiz comum. Essa técnica é frequentemente utilizada para normalizar texto, reduzir a dimensionalidade do vocabulário e melhorar a eficiência de algoritmos de análise de texto.

Objetivos do Stemming:

- *Redução do Vocabulário*: Diminuir o número de formas de palavras diferentes tratadas pelos modelos de Processamento de Linguagem Natural .
- *Normalização de Texto*: Unificar diferentes formas de uma palavra para melhorar a consistência do texto.
- *Melhoria de Desempenho*: Reduzir a carga computacional ao diminuir a diversidade lexical sem perder muito do significado semântico.

Algoritmos de Stemming:

1. *Algoritmo de Porter:* Um dos algoritmos mais populares, desenvolvido por Martin Porter em 1980. Opera através de uma série de regras de reescrita aplicadas sequencialmente. Exemplo: "correndo" → "corr", "facilmente" → "facil".
2. *Algoritmo de Snowball (Porter2):* Uma versão aprimorada do algoritmo de Porter. Oferece maior precisão e manipulação de exceções. Exemplo: "correu" → "corr", "facilmente" → "facil".

Implementação em Python: A biblioteca NLTK (Natural Language Toolkit) fornece implementações dos algoritmos de stemming, facilitando sua aplicação prática. No entanto, para a língua portuguesa, podemos utilizar a biblioteca RSLPStemmer do NLTK, que é mais adequada para este idioma:

1. execute o código e analise a saída; e
2. mude as palavras das linhas 30 e 50 do código Python e re-execute.

```
import subprocess
import sys

# Função para instalar pacotes
def install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

# Verificar se o NLTK está instalado
try:
    import nltk
except ImportError:
```

continua

```
    print("NLTK não está instalado. Instalando...")
    install("nltk")
    import nltk

# Configurar o diretório de dados do NLTK no Colab
nltk.data.path.append('/root/nltk_data')    # add: linha adicionada

# Verificar se o stemmer RSLP está instalado
try:
    nltk.data.find('stemmers/rslp')
except LookupError:
    print("Stemmer RSLP não está instalado. Instalando...")
    nltk.download('rslp')

# Agora, o código para stemmizar as palavras em português
from nltk.stem import RSLPStemmer

# Instanciando o RSLP Stemmer
rslp_stemmer = RSLPStemmer()

# Lista de palavras em português para stemmizar
words_pt = ['correndo', 'correu', 'correm', 'facilmente', 'justamente']

# Aplicando o RSLP Stemmer
stems_rslp = [rslp_stemmer.stem(word) for word in words_pt]
print("Stemming em português (RSLP):", stems_rslp)

# Verificar se o SnowballStemmer está instalado
try:
    nltk.data.find('stemmers/snowball')
except LookupError:
    print("Stemmer Snowball não está instalado. Instalando...")
    nltk.download('snowball_data')

# Agora, o código para stemmizar as palavras em inglês usando Snowball
(Porter2)
from nltk.stem import SnowballStemmer

# Instanciando o Snowball Stemmer para inglês
```

continua

```
snowball_stemmer = SnowballStemmer("english")

# Lista de palavras em inglês para stemmizar
words_en = ['running', 'ran', 'runs', 'easily', 'fairly']

# Aplicando o Snowball Stemmer
stems_snowball = [snowball_stemmer.stem(word) for word in words_en]
print("Stemming em inglês (Snowball):", stems_snowball)
```

```
Stemming em português (RSLP): ['corr', 'corr', 'corr', 'facil', 'just']
Stemmer Snowball não está instalado. Instalando...
Stemming em inglês (Snowball): ['run', 'ran', 'run', 'easili', 'fair']
[nltk_data] Downloading package snowball_data to /root/nltk_data...
[nltk_data]   Package snowball_data is already up-to-date!
```

NOTA: Limitações do Stemming:

1. *Redução Excessiva:* Pode resultar em radicais que não são palavras válidas, o que pode afetar a interpretação semântica.
2. *Sensibilidade ao Idioma:* Os algoritmos geralmente são projetados para um idioma específico e podem não ser eficazes em outros.

Tarefa 3.2: Tomando como base o conteúdo lido no ebook, escreva em algumas palavras a importância o processo de stemming. Escrever a resposta no espaço indicado a seguir.

Resposta Tarefa 3.2: (escreva aqui):

Atividade 4: Comparação entre Stemming e Lematização

Objetivo: Comparar os processos de stemming e lematização.

Relembrando as técnicas:

de Lematização: A lematização é uma técnica de processamento de linguagem natural que transforma palavras flexionadas na sua forma base ou "lema", considerando o contexto e as regras linguísticas. Por exemplo, "correndo", "correu" e "correm" são reduzidos ao lema "correr".

de Stemming: O stemming é uma técnica que reduz palavras flexionadas ou derivadas ao seu radical ou raiz comum, sem considerar o contexto linguístico. Por exemplo, "correndo", "correu" e "correm" são reduzidos ao radical "corr".

Tarefa 4.1: Aplicar tanto o stemming quanto a lematização nas mesmas palavras e comparar os resultados. Para simplificar, segue exemplo de código em Python

1. execute o código e analise a saída; e
2. mude as palavras da linha 28 do código Python e re-execute.

```python
import subprocess
import sys

# Função para instalar um pacote se ele não estiver instalado
def install_package(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

# Verificar e instalar o NLTK se necessário (Stemming em Português)
try:
    import nltk
except ImportError:
    install_package('nltk')
    import nltk

import nltk
from nltk.stem import WordNetLemmatizer, RSLPStemmer

# Baixar recursos necessários para lematização e stemming
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('rslp')

# Inicializar o lematizador e o stemmer
lemmatizer = WordNetLemmatizer()
stemmer = RSLPStemmer()

# Lista de palavras para lematização e stemming
palavras = ['correndo', 'correu', 'correm', 'facilmente', 'justamente']

# Lematização das palavras
# Lematização é o processo de reduzir uma palavra à sua forma base ou raiz
```

continua

```
palavras_lematizadas = [lemmatizer.lemmatize(palavra, pos='v') for palavra in palavras]


# Stemming das palavras

# Stemming é o processo de reduzir uma palavra ao seu radical

palavras_stemmed = [stemmer.stem(palavra) for palavra in palavras]


# Comparação dos resultados de lematização e stemming

# Cria uma lista de tuplas contendo a palavra original, a palavra lematizada e a palavra após stemming

comparacao = list(zip(palavras, palavras_lematizadas, palavras_stemmed))


# Imprime a comparação

print(comparacao)
```

```
[('correndo', 'correndo', 'corr'), ('correu', 'correu', 'corr'),
('correm', 'correm', 'corr'), ('facilmente', 'facilmente', 'facil'),
('justamente', 'justamente', 'just')]

[nltk_data] Downloading package wordnet to /root/nltk_data...

[nltk_data]   Package wordnet is already up-to-date!

[nltk_data] Downloading package omw-1.4 to /root/nltk_data...

[nltk_data]   Package omw-1.4 is already up-to-date!

[nltk_data] Downloading package rslp to /root/nltk_data...

[nltk_data]   Package rslp is already up-to-date!
```

NOTA: Comparação *Stemming* vs. Lematização:

1. *Stemming*: Reduz palavras ao seu radical de forma mecânica, sem levar em consideração o contexto linguístico.
2. *Lematização:* Considera o contexto e transforma as palavras na sua forma base ou lema.

Tarefa 4.2: Escrever no espaço a seguir as suas conclusões práticas sobre a comparação em questão.

Resposta Tarefa 4.2: (escreva aqui):

Atividade 5: Casos de Uso Práticos

Problema: Classificação de Sentenças de Acordo com Sentimentos Contexto

No Processamento de Linguagem Natural, uma tarefa comum é a análise de sentimentos, onde se busca identificar a atitude expressa em um texto. Sentimentos podem ser categorizados como positivos, negativos ou neutros, ajudando a compreender a opinião geral sobre um determinado tópico. Esta técnica é amplamente utilizada em

diversas áreas como análise de mercado, feedback de clientes, monitoramento de redes sociais, entre outras.

Descrição do Problema

O problema em questão é classificar automaticamente sentenças em português de acordo com seus sentimentos. Precisamos determinar se uma sentença expressa um sentimento positivo, negativo ou neutro.

Objetivo

O objetivo é criar um sistema automatizado que classifique sentenças baseando-se em palavras-chave que indicam sentimentos específicos. Esse sistema deve:

1. Processar sentenças em português
2. Identificar e lematizar palavras-chave.
3. Classificar a sentença como "positivo", "negativo" ou "neutro".

Abordagem Utilizada

Para resolver este problema, utilizamos a biblioteca spaCy, que permite o processamento de texto em diversas línguas, incluindo o português. A abordagem consiste em:

1. Carregar o modelo de língua portuguesa do spaCy: Utilizamos um modelo pré-treinado que inclui regras gramaticais e léxicas específicas do português.
2. Definir uma função de classificação: Esta função analisa a sentença e verifica a presença de palavras-chave lematizadas que indicam sentimentos positivos ou negativos.
3. Classificar exemplos de sentenças: Utilizamos a função para classificar um conjunto de sentenças de exemplo e verificar a precisão do sistema.

Exemplos Utilizados.

Fornecemos algumas sentenças de exemplo para ilustrar como o sistema realiza a classificação:

1. Sentença Positiva: "Este restaurante é excelente, adorei a comida!"
2. Sentença Negativa: "O atendimento nesta loja é péssimo, não recomendo."
3. Sentença Neutra: "O filme foi bom, mas o final deixou a desejar."
4. Outra Sentença Positiva: "A viagem foi ótima, tudo correu conforme o planejado."

Código de Implementação Aqui está o código que implementa a classificação das sentenças:

Classificação de Sentenças em Português usando spaCy

Neste exemplo, apresentamos como utilizar a biblioteca spaCy para classificar sentenças em português como "positivo", "negativo" ou "neutro" com base na presença de palavras-chave. Vamos explorar o código passo a passo para entender seu funcionamento.

Função de Classificação de Sentenças

A seguir, definimos uma função chamada classify_sentence que classifica uma sentença como "positivo", "negativo" ou "neutro" com base nas palavras-chave encontradas na frase:

Tarefa 5.1: Dado o código do caso de uso de classificação de texto em questão:

1. execute o código e analise a saída; e
2. mude as frases das linhas 19 até 22 do código Python e re-execute.

```
!python -m spacy download pt_core_news_sm

# Carregar o modelo em português do spaCy
nlp = spacy.load('pt_core_news_sm')

# Função para classificar sentenças como "positivo" ou "negativo"
def classify_sentence(sentence):
    doc = nlp(sentence)
    # Verificar a presença de palavras-chave
    if any(token.lemma_ in ['bom', 'ótimo', 'excelente'] for token in doc):
        return "positivo"
    elif any(token.lemma_ in ['ruim', 'péssimo', 'terrível'] for token in doc):
        return "negativo"
    else:
        return "neutro"

# Exemplos de sentenças para classificação
sentences = [
    "Este restaurante é excelente, adorei a comida!",
    "O atendimento nesta loja é péssimo, não recomendo.",
    "O filme foi bom, mas o final deixou a desejar.",
    "A viagem foi ótima, tudo correu conforme o planejado."
]

# Classificar e imprimir os resultados
```

continua

```
for sentence in sentences:
    classification = classify_sentence(sentence)
    print(f" Sentença: '{sentence}' - Classificação: {classification}")
```

```
Sentença: 'Este restaurante é excelente, adorei a comida!' -
Classificação: positivo

Sentença: 'O atendimento nesta loja é péssimo, não recomendo.' -
Classificação: negativo

Sentença: 'O filme foi bom, mas o final deixou a desejar.' - Classificação:
positivo

Sentença: 'A viagem foi ótima, tudo correu conforme o planejado.' -
Classificação: positivo
```

Tarefa 5.2: Re-escrever o exemplo da Tarefa 5.1 para implementar um pequeno exemplo de análise de sentimentos ampliado, por exemplo, com 50 frases.

```
# Incluir código fonte da Tarefa 5.2 aqui
```

```
!python -m spacy download pt_core_news_sm
```

```
Collecting pt-core-news-sm==3.7.0

  Downloading https://github.com/explosion/spacy-models/releases/
download/pt_core_news_sm-3.7.0/pt_core_news_sm-3.7.0-py3-none-any.whl
(13.0 MB)

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.0/13.0 MB 53.3 MB/s eta
0:00:00

Requirement already satisfied: spacy<3.8.0,>=3.7.0 in /usr/local/lib/
python3.10/dist-packages (from pt-core-news-sm==3.7.0) (3.7.5)

Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (3.0.12)

Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (1.0.5)

Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (1.0.10)

Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (2.0.8)

Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (3.0.9)

Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (8.2.5)

Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (1.1.3)

Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/
```

continua

```
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (2.4.8)

Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (2.0.10)

Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (0.4.1)

Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (0.13.0)

Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (4.66.6)

Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (2.32.3)

Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /
usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-
core-news-sm==3.7.0) (2.9.2)

Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-
packages (from spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (3.1.4)

Requirement already satisfied: setuptools in /usr/local/lib/python3.10/
dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0)
(75.1.0)

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (24.2)

Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/
lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (3.4.1)

Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/
python3.10/dist-packages (from spacy<3.8.0,>=3.7.0->pt-core-news-
sm==3.7.0) (1.26.4)

Requirement already satisfied: language-data>=1.2 in /usr/local/
lib/python3.10/dist-packages (from langcodes<4.0.0,>=3.2.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (1.2.0)

Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/
python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (0.7.0)

Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/
python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (2.23.4)

Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/
lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (4.12.2)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/
local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (3.4.0)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/
dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.0->pt-
core-news-sm==3.7.0) (3.10)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/
```

continua

```
lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (2.2.3)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/
lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (2024.8.30)

Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/
python3.10/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<3.8.0,>=3.7.0-
>pt-core-news-sm==3.7.0) (0.7.11)

Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/
python3.10/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<3.8.0,>=3.7.0-
>pt-core-news-sm==3.7.0) (0.1.5)

Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/
dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.0->pt-core-
news-sm==3.7.0) (8.1.7)

Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/
python3.10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.0-
>pt-core-news-sm==3.7.0) (1.5.4)

Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/
python3.10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.0-
>pt-core-news-sm==3.7.0) (13.9.4)

Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/
local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (0.20.0)

Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/
local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (7.0.5)

Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/
python3.10/dist-packages (from jinja2->spacy<3.8.0,>=3.7.0->pt-core-
news-sm==3.7.0) (3.0.2)

Requirement already satisfied: marisa-trie>=0.7.7 in /usr/
local/lib/python3.10/dist-packages (from language-data>=1.2-
>langcodes<4.0.0,>=3.2.0->spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0)
(1.2.1)

Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/
python3.10/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (3.0.0)

Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/
lib/python3.10/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (2.18.0)

Requirement already satisfied: wrapt in /usr/local/lib/python3.10/
dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0-
>spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0) (1.16.0)

Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/
python3.10/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0-
>typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.0->pt-core-news-sm==3.7.0)
(0.1.2)

 Download and installation successful

You can now load the package via spacy.load('pt_core_news_sm')

 Restart to reload dependencies

If you are in a Jupyter or Colab notebook, you may need to restart
Python in order to load all the package's dependencies. You can do this
by selecting the

'Restart kernel' or 'Restart runtime' option.
```

Atividade 6: Algoritmos e Técnicas Comuns

Objetivo: Explorar algoritmos e técnicas comuns na análise morfológica.

Como Funciona o Algoritmo de Porter? O algoritmo de Porter aplica uma série de regras para transformar palavras em suas raízes. Aqui estão os principais passos:

1. Remoção de Sufixos Plurais e de Gerúndio: O algoritmo começa removendo sufixos comuns que indicam pluralidade ou gerúndio. Por exemplo, "correndo" pode ser transformado em "corr".
2. Tratamento de Sufixos Derivacionais: Em seguida, remove sufixos que são usados para formar novas palavras a partir de outras, como "mente" em "facilmente". Isso transformaria "facilmente" em "facil".
3. Simplificação de Palavras: Aplica regras para simplificar palavras, removendo sufixos adicionais e padronizando as formas.
4. Remoção de Sufixos Finais: Por fim, remove quaisquer sufixos restantes que possam ser desnecessários para a raiz da palavra.

Tarefa 6.1: Avaliar a implementação do algoritmo de Porter para stemming a seguir e para tal:

1. execute o código e analise a saída; e
2. mude a palavra da linha 55 do código Python e re-execute.

```
def porter_stemmer(word):
    """
    Implementação simplificada do algoritmo de Porter para stemming.
    """
    suffixes = [
        ('ational', 'ate'), ('tional', 'tion'), ('enci', 'ence'), ('anci',
'ance'),
        ('izer', 'ize'), ('bli', 'ble'), ('alli', 'al'), ('entli', 'ent'),
        ('eli', 'e'), ('ousli', 'ous'), ('ization', 'ize'), ('ation', 'ate'),
        ('ator', 'ate'), ('alism', 'al'), ('iveness', 'ive'), ('fulness',
'ful'),
        ('ousness', 'ous'), ('aliti', 'al'), ('iviti', 'ive'), ('biliti',
'ble'),
        ('logi', 'log')
    ]

    def replace_suffix(word, suffixes):
        for suffix, replacement in suffixes:
            if word.endswith(suffix):
```

continua

```
                return word[:-len(suffix)] + replacement
        return word

    def step_1a(word):
        if word.endswith('sses'):
            return word[:-2]
        elif word.endswith('ies'):
            return word[:-2]
        elif word.endswith('ss'):
            return word
        elif word.endswith('s'):
            return word[:-1]
        return word

    def step_1b(word):
        if word.endswith('eed'):
            return word[:-1] if len(word) > 3 else word
        elif word.endswith('ed') and has_vowel(word[:-2]):
            return replace_suffix(word[:-2], suffixes)
        elif word.endswith('ing') and has_vowel(word[:-3]):
            return replace_suffix(word[:-3], suffixes)
        return word

    def has_vowel(word):
        for char in word:
            if char in 'aeiou':
                return True
        return False

    def step_2(word):
        return replace_suffix(word, suffixes)

    word = step_1a(word)
    word = step_1b(word)
    word = step_2(word)
    return word

# Exemplo de uso
word = "running"
stemmed_word = porter_stemmer(word)
```

continua

```
print(f"Palavra original: {word}")
print(f"Palavra reduzida (stemming): {stemmed_word}")
```

```
Palavra original: running
Palavra reduzida (stemming): runn
```

```
!ls /root/nltk_data/tokenizers
```

```
punkt  punkt_tab  punkt_tab.zip  punkt.zip
```

Tarefa 6.2: Re-escrever o exemplo da Tarefa 6.1 para implementar um pequeno exemplo para operar com frases em português. Insira o código no espaço a seguir.

```
# Incluir código fonte da Tarefa 6.2 aqui
```

Atividade 7: Problema de Indexação de pequenos documentos

Indexação usando a abordagem TF-IDF

No campo do Processamento de Linguagem Natural e da Recuperação de Informação, o problema de indexação de documentos é crucial para organizar e recuperar informações de grandes volumes de texto de forma eficiente. Um método amplamente utilizado para esse fim é o TF-IDF, que significa "Term Frequency-Inverse Document Frequency". Este método ajuda a medir a importância de uma palavra em um documento dentro de uma coleção ou corpus de documentos.

Cenário

Imagine que você tem uma grande coleção de documentos, como artigos científicos, posts de blogs, ou páginas da web. Quando um usuário faz uma consulta de busca, o sistema precisa rapidamente identificar quais documentos são mais relevantes para essa consulta. O problema de indexação de documentos é, portanto, criar uma estrutura que permita essa recuperação eficiente e precisa de informações.

O Que é TF-IDF?

TF-IDF é uma métrica que ajuda a avaliar a importância de uma palavra em um documento, levando em conta tanto a frequência da palavra no documento (TF) quanto a frequência dessa palavra em todo o corpus de documentos (IDF). Vamos entender cada componente:

TF (Term Frequency)

Frequência do termo no documento. Calcula-se quantas vezes uma palavra aparece em um documento em relação ao número total de palavras nesse documento. A fórmula é:

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{N_d}$$

$(f_{t,d})$ é a contagem de ocorrências do termo (t) no documento (d)

(N_d) é o número total de termos no documento (d)

IDF (Inverse Document Frequency)

Frequência inversa do documento. Avalia a importância do termo no corpus inteiro, penalizando termos comuns em muitos documentos. A fórmula é:

$$\text{IDF}(t, D) = \log\left(\frac{N}{|d \in D : t \in d|}\right)$$

(N) é o número total de documentos no corpus

$\left(|d \in D : t \in d|\right)$ é o número de documentos onde o termo (t) aparece

TF-IDF

Produto do TF e IDF. Combina ambas as métricas para calcular a importância do termo no documento, ajustada pela frequência no corpus. A fórmula é:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Como o TF-IDF Resolve o Problema de Indexação

- **Relevância de Termos:** TF-IDF ajuda a identificar palavras importantes para um documento específico, reduzindo a influência de termos comuns e irrelevantes.
- **Ranking de Documentos:** Quando uma consulta de busca é feita, o sistema pode calcular o TF-IDF de cada termo da consulta em cada documento e somar esses valores para determinar quais documentos são mais relevantes.
- **Eficiência na Busca:** Utilizando estruturas como matrizes esparsas, os sistemas podem rapidamente calcular e comparar valores de TF-IDF, permitindo uma recuperação eficiente de documentos relevantes.

Exemplo Prático

Considere o seguinte corpus com cinco documentos:

- **Documento 1:** "O gato está correndo no jardim."
- **Documento 2:**"Os gatos gostam de correr e brincar."
- **Documento 3:**"Ela gosta de estudar linguística computacional."
- **Documento 4:** "Linguística computacional é uma área fascinante."
- **Documento 5:** "O gado preto da Renata dorme na varanda."

Vamos calcular o TF-IDF para a palavra "gosta".

TF (Term Frequency)

- **Documento 1:** "gosta" não aparece (TF = 0)
- **Documento 2:** "gosta" aparece uma vez em 6 palavras (TF = 1/6 ≈ 0.167)
- **Documento 3:** "gosta" aparece uma vez em 6 palavras (TF = 1/6 ≈ 0.167)
- **Documento 4:** "gosta" não aparece (TF = 0)
- **Documento 5:** "gosta" não aparece (TF = 0)

IDF (Inverse Document Frequency)

- "gosta" aparece em 2 dos 5 documentos.
- IDF = $$\log\left(\frac{5}{2}\right) \approx 0.398$$

TF-IDF

- **Documento 1:** 0 * 0.398 = 0
- **Documento 2:** 0.167 * 0.398 ≈ 0.066
- **Documento 3:** 0.167 * 0.398 ≈ 0.066
- **Documento 4:** 0 * 0.398 = 0
- **Documento 5:** 0 * 0.398 = 0

Cálculo da Similaridade usando o Coeficiente do Cosseno

A similaridade entre dois documentos pode ser calculada usando o coeficiente do cosseno, que mede o ângulo entre dois vetores. A fórmula para calcular a similaridade do cosseno entre dois vetores (**A**) e (**B**) é:

$$\text{similaridade}_{\cos}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

onde:

(**A**·**B**) é o produto escalar dos vetores (**A**) e (**B**)

(||**A**||) é a norma (magnitude) do vetor (**A**)

(||**B**||) é a norma (magnitude) do vetor (**B**)

Exemplo de Cálculo da Similaridade do Cosseno

Vamos calcular a similaridade do cosseno entre os Documentos 2 e 3 usando seus vetores TF-IDF simplificados. Suponha que temos os seguintes vetores TF-IDF para simplificação:

-(A=[0.066,0,0,0,0]) (Documento 2) -(B=[0.066,0,0,0,0]) (Documento 3)

A similaridade do cosseno é calculada como:

$$\text{similaridade}_{\cos}(\mathbf{A},\mathbf{B}) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{0.066\times0.066}{\sqrt{0.066^2}\times\sqrt{0.066^2}} = \frac{0.004356}{0.066\times0.066} = 1$$

Neste caso, a similaridade do cosseno entre os Documentos 2 e 3 é 1, indicando que eles são idênticos em termos de conteúdo de TF-IDF.

Resultados

A importância da palavra "gosta" é maior nos Documentos 2 e 3, ambos com um valor TF-IDF de aproximadamente 0.066. Os Documentos 1, 4 e 5 têm um valor TF-IDF de 0 para "gosta", indicando que a palavra não aparece nesses documentos.

Tarefa 7.1: Avaliar a implementação do sistema de indexação de documentos baseado no TFIDF a seguir e para tal:

1. execute o código e analise a saída; e
2. inclua mais algumas frases linhas 68-72 e re-execute mudando os termos da consulta linhas 114 e 115.

```
# Etapa 1: Verificação e Instalação das Bibliotecas Necessárias
import subprocess
import sys

def install_and_import(package):
    try:
        __import__(package)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install",
package])

# Instalar as bibliotecas necessárias
install_and_import('spacy')
install_and_import('nltk')
install_and_import('scikit-learn')
install_and_import('pandas')

# Baixar o modelo de português do spaCy, caso não esteja instalado
```

continua

```
subprocess.run([sys.executable, "-m", "spacy", "download", "pt_core_news_sm"])

# Etapa 2: Importar as Bibliotecas e Inicializar Recursos
import spacy
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Baixar recursos necessários do nltk para stemming e lematização
nltk.download('rslp')  # Baixar o stemmer RSLP para português
nltk.download('wordnet')  # Baixar o WordNet para lematização
nltk.download('omw-1.4')  # Baixar o Open Multilingual Wordnet
nltk.download('punkt')  # Baixar o tokenizer 'punkt' para português

from nltk.stem import RSLPStemmer
from nltk.stem import WordNetLemmatizer

# Carregar o modelo de português do spaCy
nlp = spacy.load("pt_core_news_sm")

# Inicializar o lematizador e o stemmer
lemmatizer = WordNetLemmatizer()
stemmer = RSLPStemmer()

# Etapa 3: Definir Funções de Pré-Processamento

# Função de lematização usando spaCy
# Esta função utiliza o spaCy para lematizar o texto, retornando os lemas dos tokens.
def lematizar(texto):
    doc = nlp(texto)
    return ' '.join([token.lemma_ for token in doc])

# Função de stemming usando NLTK
# Esta função utiliza o nltk para aplicar stemming aos tokens do texto.
def aplicar_stemming(texto):
    tokens = nltk.word_tokenize(texto, language='portuguese')
    return ' '.join([stemmer.stem(token) for token in tokens])
```

continua

```
# Função de pré-processamento completo
# Esta função aplica lematização seguida de stemming ao texto.
def preprocessamento(texto):
    texto_lematizado = lematizar(texto)
    texto_stemmed = aplicar_stemming(texto_lematizado)
    return texto_stemmed

# Etapa 4: Pré-Processar Documentos e Calcular TF-IDF

# Lista de documentos de exemplo para indexação
documentos = [
    "O gato está correndo no jardim.",
    "Os gatos gostam de correr e brincar.",
    "Ela gosta de estudar linguística computacional.",
    "Linguística computacional é uma área fascinante.",
    "O gato preto da Renata dorme na varanda."
]

# Aplicar pré-processamento em todos os documentos
# Esta linha aplica a função de pré-processamento em cada documento.
documentos_preprocessados = [preprocessamento(doc) for doc in documentos]

# Exibir os documentos preprocessados
print("Documentos Preprocessados:")
for doc in documentos_preprocessados:
    print(doc)

# Inicializar o vetor TF-IDF
vectorizer = TfidfVectorizer()

# Ajustar e transformar os documentos preprocessados
# Esta linha ajusta e transforma os documentos preprocessados em uma matriz TF-IDF.
tfidf_matrix = vectorizer.fit_transform(documentos_preprocessados)

# Obter os termos (features) do TF-IDF
termos = vectorizer.get_feature_names_out()
```

continua

```
# Exibir a matriz TF-IDF usando um DataFrame para melhor visualização
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(), columns=termos)

# Exibir a matriz TF-IDF
print("Matriz TF-IDF:")
print(df_tfidf)

# Etapa 5: Consulta de Documentos e Calcular Similaridade

# Função para processar consulta e calcular similaridade
# Esta função pré-processa a consulta, transforma em vetor TF-IDF e
# calcula a similaridade coseno.
def consultar_documentos(consulta):
    consulta_preprocessada = preprocessamento(consulta)
    consulta_tfidf = vectorizer.transform([consulta_preprocessada])
    similaridades = cosine_similarity(consulta_tfidf, tfidf_matrix)
    documento_mais_similar = similaridades.argmax()
    return documento_mais_similar, similaridades

# Exemplo de consultas
consultas = [
    "gato no jardim",
    "estudar linguística",
    "gato da Renata"
]

# Processar cada consulta e exibir resultados
for consulta in consultas:
    indice, similaridades = consultar_documentos(consulta)
    print(f"Consulta: '{consulta}'")
    print(f"Documento mais similar: {documentos[indice]}")
    print(f"Similaridades: {similaridades}\n")
```

```
[nltk_data] Downloading package rslp to /root/nltk_data...
[nltk_data]   Package rslp is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

continua

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Documentos Preprocessados:
o gat est corr em o jardim .
o gat gost de corr e brinc .
ela gost de estud linguís computac .
linguís computac ser um áre fascin .
o gat pret de o renat dorm em o varand .
Matriz TF-IDF:
      brinc  computac      corr        de      dorm       ela        em
\
0  0.000000  0.000000  0.416607  0.000000  0.000000  0.000000  0.416607
1  0.559117  0.000000  0.451092  0.374447  0.000000  0.000000  0.000000
2  0.000000  0.384569  0.000000  0.319227  0.000000  0.476663  0.000000
3  0.000000  0.350388  0.000000  0.000000  0.000000  0.000000  0.000000
4  0.000000  0.000000  0.000000  0.284329  0.424555  0.000000  0.342528

        est     estud    fascin       gat      gost    jardim   linguís
\
0  0.516374  0.000000  0.000000  0.345822  0.000000  0.516374  0.000000
1  0.000000  0.000000  0.000000  0.374447  0.451092  0.000000  0.000000
2  0.000000  0.476663  0.000000  0.000000  0.384569  0.000000  0.384569
3  0.000000  0.000000  0.434297  0.000000  0.000000  0.000000  0.350388
4  0.000000  0.000000  0.000000  0.284329  0.000000  0.000000  0.000000

       pret     renat       ser        um    varand       áre
0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.434297  0.434297  0.000000  0.434297
4  0.424555  0.424555  0.000000  0.000000  0.424555  0.000000
Consulta: 'gato no jardim'
Documento mais similar: O gato está correndo no jardim.
Similaridades: [[0.74819537 0.17307225 0.         0.         0.32214456]]

Consulta: 'estudar linguística'
Documento mais similar: Ela gosta de estudar linguística computacional.
Similaridades: [[0.         0.         0.61245459 0.22001359 0.        ]]
```

continua

```
Consulta: 'gato da Renata'
Documento mais similar: O gato preto da Renata dorme na varanda.
Similaridades: [[0.16815247 0.36414246 0.15522094 0.         0.58475075]]
```

Tarefa 7.2: Incrementar a base de documentos com texto de assuntos diferentes; e analisar o efeito do incremento de documentos quando ao tamanho do vetor de termos e a precisão dos resultados das consultas.

```
# incluir a código referente a Tarefa 7.2
```

Atividade 8: Processamento e Análise de Texto em Português - Redução de dimensionalidade

Objetivo:

Nesta atividade de laboratório, você aprenderá a implementar um pipeline de processamento de linguagem natural (PLN) para textos em português. Especificamente, você realizará as etapas de instalação de bibliotecas, tokenização, remoção de stopwords, lematização, e análise de frequência de palavras, usando a linguagem Python e as bibliotecas **nltk** e **spaCy.**

8.1: Atividade: Execute código a seguir e Análise de Resultados

Passos da Atividade:

Etapa 1: Verificação e Instalação das Bibliotecas Necessárias

- Será utilizada uma função em Python para verificar se as bibliotecas necessárias (**nltk**, **spacy**) estão instaladas no ambiente. Caso contrário, elas serão instaladas automaticamente.
- O modelo de linguagem do spaCy para português (**pt_core_news_sm**) será baixado e carregado para o uso nas etapas subsequentes.

Etapa 2: Preparação e Limpeza do Texto

- Um conjunto de frases em português será processado. As frases incluem diversos contextos em que o verbo "correr" é utilizado em diferentes formas verbais.
- A primeira tarefa será substituir pontuações e números por espaços, facilitando a tokenização subsequente.

Etapa 3: Tokenização

- As frases serão divididas em tokens individuais (palavras), utilizando a função **word_tokenize** da biblioteca **nltk**.
- A tokenização é uma etapa essencial no pré-processamento de texto, pois divide o texto em unidades que podem ser processadas individualmente.

Etapa 4: Remoção de Stopwords

- Stopwords (palavras muito comuns como "o", "e", "a") serão removidas do texto tokenizado. Isso é feito para focar no processamento de palavras mais relevantes para a análise.

Etapa 5: Lematização

- Os tokens restantes serão lematizados, ou seja, reduzidos à sua forma base (lema). Neste exemplo, o foco é lematizar diversas formas do verbo "correr" ("corri", "correndo", "correu", etc.) para o lema "correr".
- A lematização manual será aplicada para garantir que certas formas verbais e substantivas sejam corretamente lematizadas. A lematização é aplicada manualmente, pois "corri" e "corrida" não é capturada como pertencente ao lema "correr".

Etapa 6: Análise de Frequência

- Após a lematização, a frequência das palavras de interesse (incluindo suas formas lematizadas) será calculada e exibida.
- Esta etapa permite uma análise quantitativa de quantas vezes determinadas palavras aparecem no conjunto de textos, antes e depois da lematização.

Etapa 7: Conclusão

- A etapa final da atividade envolve a verificação da **redução da dimensionalidade** do texto como resultado da aplicação da lematização.
- Redução da dimensionalidade se refere à diminuição do número de palavras distintas no texto após a lematização, o que simplifica a análise e modelagem de dados linguísticos.
- Você irá comparar a quantidade de palavras antes e depois da lematização para observar como diferentes formas verbais foram unificadas em lemas únicos, reduzindo a complexidade do vocabulário.

Objetivo Educacional:

Ao concluir esta atividade, você terá uma compreensão prática de como processar textos em português para análise linguística, incluindo etapas essenciais como tokenização, remoção de stopwords, lematização e análise de frequência. A verificação da redução da dimensionalidade do texto após a lematização irá demonstrar a eficiência dessa técnica em simplificar o vocabulário e facilitar análises posteriores, como categorização de texto ou análise de sentimentos.

```
# Etapa 1: Verificação e Instalação das Bibliotecas Necessárias
import subprocess
import sys

def install_and_import(package):
    try:
        __import__(package)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install",
package])

# Instalar as bibliotecas necessárias
install_and_import('spacy')

import nltk
import spacy
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
import matplotlib.pyplot as plt

# Baixar o modelo de português do spaCy, caso não esteja instalado
subprocess.run([sys.executable, "-m", "spacy", "download", "pt_core_news_sm"])

# Baixar os recursos necessários
nltk.download('punkt')
nltk.download('stopwords')

# Carregar o modelo de linguagem do spaCy para lematização em português
nlp = spacy.load("pt_core_news_sm")
```

continua

```python
# Conjunto de frases de exemplo em português
frases = [
   """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque.""",
    """Quando o cachorro começou a latir, eu rapidamente corri para casa.""",
    """Corri o mais rápido que pude para não perder o ônibus.""",
    """Assim que ouvi a notícia, corri para contar aos meus amigos.""",
    """Depois da escola, corri para o parque encontrar meus amigos.""",
    """Ontem, eu corri cinco quilômetros no parque."""
```

continua

```
"""Quando o cachorro começou a latir, eu rapidamente corri para casa."""
"""Corri o mais rápido que pude para não perder o ônibus."""
"""Assim que ouvi a notícia, corri para contar aos meus amigos."""
"""Depois da escola, corri para o parque encontrar meus amigos."""
"""Ontem, eu corri cinco quilômetros no parque."""
"""Quando o cachorro começou a latir, eu rapidamente corri para casa."""
"""Corri o mais rápido que pude para não perder o ônibus."""
"""Assim que ouvi a notícia, corri para contar aos meus amigos."""
"""Depois da escola, corri para o parque encontrar meus amigos."""
"""Ontem, eu corri cinco quilômetros no parque."""
"""Quando o cachorro começou a latir, eu rapidamente corri para casa."""
"""Corri o mais rápido que pude para não perder o ônibus."""
"""Assim que ouvi a notícia, corri para contar aos meus amigos."""
"""Depois da escola, corri para o parque encontrar meus amigos."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
```

continua

```
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Ela estava correndo para pegar o ônibus."""
"""Os atletas estão correndo em volta da pista."""
"""Vi um grupo de pessoas correndo no parque."""
"""As crianças estavam correndo e brincando no jardim."""
"""O cachorro estava correndo atrás da bola."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
```

continua

```
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""Eles correrão na maratona no próximo mês."""
"""Os competidores correrão em diferentes categorias."""
"""Todos os atletas correrão ao mesmo tempo."""
"""Eles correrão pela primeira vez na prova."""
"""Os corredores profissionais correrão na próxima competição."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
```

continua

```
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""O garoto correu para a escola para não se atrasar."""
"""Ele correu a maratona inteira sem parar."""
"""Assim que ouviu a notícia, correu para casa."""
"""O cachorro correu em direção ao portão."""
"""Ela correu para pegar o último trem da noite."""
"""A corrida de carros será amanhã de manhã."""
"""Ela treinou muito para a corrida de hoje."""
"""A corrida começou às 7 da manhã."""
"""Eles participaram da corrida beneficente no domingo."""
"""Todos estavam ansiosos pela grande corrida."""
"""A corrida de carros será amanhã de manhã."""
"""Ela treinou muito para a corrida de hoje."""
"""A corrida começou às 7 da manhã."""
"""Eles participaram da corrida beneficente no domingo."""
"""Todos estavam ansiosos pela grande corrida."""
"""A corrida de carros será amanhã de manhã."""
"""Ela treinou muito para a corrida de hoje."""
"""A corrida começou às 7 da manhã."""
"""Eles participaram da corrida beneficente no domingo."""
"""Todos estavam ansiosos pela grande corrida."""
"""A corrida de carros será amanhã de manhã."""
"""Ela treinou muito para a corrida de hoje."""
"""A corrida começou às 7 da manhã."""
```

continua

```
    """Eles participaram da corrida beneficente no domingo."""
    """Todos estavam ansiosos pela grande corrida."""
    """A corrida de carros será amanhã de manhã."""
    """Ela treinou muito para a corrida de hoje."""
    """A corrida começou às 7 da manhã."""
    """Eles participaram da corrida beneficente no domingo."""
    """Todos estavam ansiosos pela grande corrida."""
    """A corrida de carros será amanhã de manhã."""
    """Ela treinou muito para a corrida de hoje."""
    """A corrida começou às 7 da manhã."""
    """Eles participaram da corrida beneficente no domingo."""
    """Todos estavam ansiosos pela grande corrida."""
    ]
# Função para substituir pontuação e números por espaços
def substituir_pontuacao_numeros(frase):
    tabela_traducao = str.maketrans(string.punctuation + '0123456789', ' ' *
(len(string.punctuation) + 10))
    return frase.translate(tabela_traducao)


# Função para processar cada frase
def processar_frase(frase):
    # Substituir pontuação e números por espaços
    frase_limpa = substituir_pontuacao_numeros(frase)


    # Tokenização
    tokens = word_tokenize(frase_limpa)


    # Remoção de stopwords
    stop_words = set(stopwords.words('portuguese'))
    tokens_sem_stopwords = [token for token in tokens if token.lower() not in
stop_words]


    # Lematização com substituição manual
    doc = nlp(" ".join(tokens_sem_stopwords))
    tokens_lematizados = []
    for token in doc:
        if token.text.lower() in ["corri", "corrida", "correndo", "correu",
"correrão"]:
            tokens_lematizados.append("correr")
        else:
```

continua

```
            tokens_lematizados.append(token.lemma_)

    # Criar a lista de lemas únicos
    lemas_unicos = list(set(tokens_lematizados))

    return tokens_sem_stopwords, tokens_lematizados, lemas_unicos

# Processar cada frase no conjunto e calcular frequências
frequencia_original = Counter()
frequencia_lematizada = Counter()

for frase in frases:
    tokens_sem_stopwords, tokens_lematizados, lemas_unicos = processar_
frase(frase.lower())

    frequencia_original.update(tokens_sem_stopwords)
    frequencia_lematizada.update(tokens_lematizados)

# Lista de palavras de interesse para o lema "correr"
palavras_interesse_lema = ["correr"]

# Dados de frequência das palavras antes da lematização
palavras = ["corri", "correndo", "correrão", "correu", "corrida"]
frequencias = []

for i in palavras:
    frequencias.append(frequencia_original[i])


print(f"Tokens após remoção de stopwords: [{len(tokens_sem_stopwords)}]
{tokens_sem_stopwords}")
print(f"Tokens lematizados:  [{len(tokens_lematizados)}]{tokens_lematizados}")
print(f"Lemas únicos:  [{len(lemas_unicos)}]{lemas_unicos}\n")


# Dados de frequência para o lema "correr" após a lematização
lema = ["correr"]
frequencia_lema = frequencia_lematizada["correr"]
```

continua

```
# Gráfico de barras para as palavras antes da lematização
plt.figure(figsize=(10, 5))
plt.bar(palavras, frequencias, color='skyblue')
plt.xlabel('Palavras')
plt.ylabel('Frequência')
plt.title('Frequência das palavras antes da lematização')
plt.show()

# Gráfico de barras para o lema "correr" após a lematização
plt.figure(figsize=(10, 5))
plt.bar(lema, frequencia_lema, color='lightgreen')
plt.xlabel('Lema')
plt.ylabel('Frequência')
plt.title('Frequência do lema "correr" após a lematização')
plt.show()


# Caso queira visualizar as frequência de cada token,
# remova os comentários as linhas a seguir
#
#print("\nFrequência de palavras antes da lematização:")
#for palavra, frequencia in frequencia_original.items():
#    print(f"{palavra}: {frequencia}")

#print("\n\nFrequência de palavras após a lematização:")
#for palavra, frequencia in frequencia_lematizada.items():
#    print(f"{palavra}: {frequencia}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Tokens após remoção de stopwords: [880] ['ontem', 'corri', 'cinco',
'quilômetros', 'parque', 'cachorro', 'começou', 'latir', 'rapidamente',
'corri', 'casa', 'corri', 'rápido', 'pude', 'perder', 'ônibus', 'assim',
'ouvi', 'notícia', 'corri', 'contar', 'amigos', 'escola', 'corri', 'parque',
'encontrar', 'amigos', 'ontem', 'corri', 'cinco', 'quilômetros', 'parque',
'cachorro', 'começou', 'latir', 'rapidamente', 'corri', 'casa', 'corri',
'rápido', 'pude', 'perder', 'ônibus', 'assim', 'ouvi', 'notícia', 'corri',
'contar', 'amigos', 'escola', 'corri', 'parque', 'encontrar', 'amigos',
'ontem', 'corri', 'cinco', 'quilômetros', 'parque', 'cachorro', 'começou',
```

continua

```
'latir', 'rapidamente', 'corri', 'casa', 'corri', 'rápido', 'pude',
'perder', 'ônibus', 'assim', 'ouvi', 'notícia', 'corri', 'contar', 'amigos',
'escola', 'corri', 'parque', 'encontrar', 'amigos', 'ontem', 'corri',
'cinco', 'quilômetros', 'parque', 'cachorro', 'começou', 'latir',
'rapidamente', 'corri', 'casa', 'corri', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvi', 'notícia', 'corri', 'contar', 'amigos', 'escola',
'corri', 'parque', 'encontrar', 'amigos', 'ontem', 'corri', 'cinco',
'quilômetros', 'parque', 'cachorro', 'começou', 'latir', 'rapidamente',
'corri', 'casa', 'corri', 'rápido', 'pude', 'perder', 'ônibus', 'assim',
'ouvi', 'notícia', 'corri', 'contar', 'amigos', 'escola', 'corri', 'parque',
'encontrar', 'amigos', 'ontem', 'corri', 'cinco', 'quilômetros', 'parque',
'cachorro', 'começou', 'latir', 'rapidamente', 'corri', 'casa', 'corri',
'rápido', 'pude', 'perder', 'ônibus', 'assim', 'ouvi', 'notícia', 'corri',
'contar', 'amigos', 'escola', 'corri', 'parque', 'encontrar', 'amigos',
'ontem', 'corri', 'cinco', 'quilômetros', 'parque', 'cachorro', 'começou',
'latir', 'rapidamente', 'corri', 'casa', 'corri', 'rápido', 'pude',
'perder', 'ônibus', 'assim', 'ouvi', 'notícia', 'corri', 'contar', 'amigos',
'escola', 'corri', 'parque', 'encontrar', 'amigos', 'ontem', 'corri',
'cinco', 'quilômetros', 'parque', 'cachorro', 'começou', 'latir',
'rapidamente', 'corri', 'casa', 'corri', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvi', 'notícia', 'corri', 'contar', 'amigos', 'escola',
'corri', 'parque', 'encontrar', 'amigos', 'ontem', 'corri', 'cinco',
'quilômetros', 'parque', 'cachorro', 'começou', 'latir', 'rapidamente',
'corri', 'casa', 'corri', 'rápido', 'pude', 'perder', 'ônibus', 'assim',
'ouvi', 'notícia', 'corri', 'contar', 'amigos', 'escola', 'corri', 'parque',
'encontrar', 'amigos', 'ontem', 'corri', 'cinco', 'quilômetros', 'parque',
'cachorro', 'começou', 'latir', 'rapidamente', 'corri', 'casa', 'corri',
'rápido', 'pude', 'perder', 'ônibus', 'assim', 'ouvi', 'notícia', 'corri',
'contar', 'amigos', 'escola', 'corri', 'parque', 'encontrar', 'amigos',
'correndo', 'pegar', 'ônibus', 'atletas', 'correndo', 'volta', 'pista',
'vi', 'grupo', 'pessoas', 'correndo', 'parque', 'crianças', 'correndo',
'brincando', 'jardim', 'cachorro', 'correndo', 'atrás', 'bola', 'correndo',
'pegar', 'ônibus', 'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo',
'pessoas', 'correndo', 'parque', 'crianças', 'correndo', 'brincando',
'jardim', 'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar',
'ônibus', 'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar', 'ônibus',
'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar', 'ônibus',
'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar', 'ônibus',
'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar', 'ônibus',
'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correndo', 'pegar', 'ônibus',
'atletas', 'correndo', 'volta', 'pista', 'vi', 'grupo', 'pessoas',
'correndo', 'parque', 'crianças', 'correndo', 'brincando', 'jardim',
'cachorro', 'correndo', 'atrás', 'bola', 'correrão', 'maratona', 'próximo',
'mês', 'competidores', 'correrão', 'diferentes', 'categorias', 'todos',
'atletas', 'correrão', 'tempo', 'correrão', 'primeira', 'vez', 'prova',
'corredores', 'profissionais', 'correrão', 'próxima', 'competição',
'correrão', 'maratona', 'próximo', 'mês', 'competidores', 'correrão',
'diferentes', 'categorias', 'todos', 'atletas', 'correrão', 'tempo',
```

continua

```
'correrão', 'primeira', 'vez', 'prova', 'corredores', 'profissionais',
'correrão', 'próxima', 'competição', 'correrão', 'maratona', 'próximo',
'mês', 'competidores', 'correrão', 'diferentes', 'categorias', 'todos',
'atletas', 'correrão', 'tempo', 'correrão', 'primeira', 'vez', 'prova',
'corredores', 'profissionais', 'correrão', 'próxima', 'competição',
'correrão', 'maratona', 'próximo', 'mês', 'competidores', 'correrão',
'diferentes', 'categorias', 'todos', 'atletas', 'correrão', 'tempo',
'correrão', 'primeira', 'vez', 'prova', 'corredores', 'profissionais',
'correrão', 'próxima', 'competição', 'correrão', 'maratona', 'próximo',
'mês', 'competidores', 'correrão', 'diferentes', 'categorias', 'todos',
'atletas', 'correrão', 'tempo', 'correrão', 'primeira', 'vez', 'prova',
'corredores', 'profissionais', 'correrão', 'próxima', 'competição',
'correrão', 'maratona', 'próximo', 'mês', 'competidores', 'correrão',
'diferentes', 'categorias', 'todos', 'atletas', 'correrão', 'tempo',
'correrão', 'primeira', 'vez', 'prova', 'corredores', 'profissionais',
'correrão', 'próxima', 'competição', 'correrão', 'maratona', 'próximo',
'mês', 'competidores', 'correrão', 'diferentes', 'categorias', 'todos',
'atletas', 'correrão', 'tempo', 'correrão', 'primeira', 'vez', 'prova',
'corredores', 'profissionais', 'correrão', 'próxima', 'competição',
'correrão', 'maratona', 'próximo', 'mês', 'competidores', 'correrão',
'diferentes', 'categorias', 'todos', 'atletas', 'correrão', 'tempo',
'correrão', 'primeira', 'vez', 'prova', 'corredores', 'profissionais',
'correrão', 'próxima', 'competição', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'garoto', 'correu', 'escola',
'atrasar', 'correu', 'maratona', 'inteira', 'parar', 'assim', 'ouviu',
'notícia', 'correu', 'casa', 'cachorro', 'correu', 'direção', 'portão',
'correu', 'pegar', 'último', 'trem', 'noite', 'corrida', 'carros', 'amanhã',
'manhã', 'treinou', 'corrida', 'hoje', 'corrida', 'começou', 'manhã',
'participaram', 'corrida', 'beneficente', 'domingo', 'todos', 'ansiosos',
'grande', 'corrida', 'corrida', 'carros', 'amanhã', 'manhã', 'treinou',
'corrida', 'hoje', 'corrida', 'começou', 'manhã', 'participaram', 'corrida',
'beneficente', 'domingo', 'todos', 'ansiosos', 'grande', 'corrida',
'corrida', 'carros', 'amanhã', 'manhã', 'treinou', 'corrida', 'hoje',
'corrida', 'começou', 'manhã', 'participaram', 'corrida', 'beneficente',
'domingo', 'todos', 'ansiosos', 'grande', 'corrida', 'corrida', 'carros',
'amanhã', 'manhã', 'treinou', 'corrida', 'hoje', 'corrida', 'começou',
'manhã', 'participaram', 'corrida', 'beneficente', 'domingo', 'todos',
'ansiosos', 'grande', 'corrida', 'corrida', 'carros', 'amanhã', 'manhã',
'treinou', 'corrida', 'hoje', 'corrida', 'começou', 'manhã', 'participaram',
'corrida', 'beneficente', 'domingo', 'todos', 'ansiosos', 'grande',
'corrida', 'corrida', 'carros', 'amanhã', 'manhã', 'treinou', 'corrida',
'hoje', 'corrida', 'começou', 'manhã', 'participaram', 'corrida',
'beneficente', 'domingo', 'todos', 'ansiosos', 'grande', 'corrida']

Tokens lematizados:  [880]['ontem', 'correr', 'cinco', 'quilômetro',
```

continua

```
'parque', 'cachorro', 'começar', 'latir', 'rapidamente', 'correr', 'casa',
'correr', 'rápido', 'pude', 'perder', 'ônibus', 'assim', 'ouvir', 'notícia',
'correr', 'contar', 'amigo', 'Escola', 'correr', 'parque', 'encontrar',
'amigo', 'ontem', 'correr', 'cinco', 'quilômetro', 'parque', 'cachorro',
'começar', 'latir', 'rapidamente', 'correr', 'casa', 'correr', 'rápido',
'pude', 'perder', 'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar',
'amigo', 'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem',
'correr', 'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'ontem', 'correr',
'cinco', 'quilômetro', 'parque', 'cachorro', 'começar', 'latir',
'rapidamente', 'correr', 'casa', 'correr', 'rápido', 'pude', 'perder',
'ônibus', 'assim', 'ouvir', 'notícia', 'correr', 'contar', 'amigo',
'Escola', 'correr', 'parque', 'encontrar', 'amigo', 'correr', 'pegar',
'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver', 'grupo', 'pessoa',
'correr', 'parque', 'criança', 'correr', 'brincar', 'Jardim', 'cachorro',
'correr', 'atrás', 'bola', 'correr', 'pegar', 'ônibus', 'atleta', 'correr',
'volta', 'pista', 'ver', 'grupo', 'pessoa', 'correr', 'parque', 'criança',
'correr', 'brincar', 'Jardim', 'cachorro', 'correr', 'atrás', 'bola',
'correr', 'pegar', 'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver',
'grupo', 'pessoa', 'correr', 'parque', 'criança', 'correr', 'brincar',
'Jardim', 'cachorro', 'correr', 'atrás', 'bola', 'correr', 'pegar',
'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver', 'grupo', 'pessoa',
'correr', 'parque', 'criança', 'correr', 'brincar', 'Jardim', 'cachorro',
'correr', 'atrás', 'bola', 'correr', 'pegar', 'ônibus', 'atleta', 'correr',
'volta', 'pista', 'ver', 'grupo', 'pessoa', 'correr', 'parque', 'criança',
'correr', 'brincar', 'Jardim', 'cachorro', 'correr', 'atrás', 'bola',
'correr', 'pegar', 'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver',
'grupo', 'pessoa', 'correr', 'parque', 'criança', 'correr', 'brincar',
'Jardim', 'cachorro', 'correr', 'atrás', 'bola', 'correr', 'pegar',
'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver', 'grupo', 'pessoa',
'correr', 'parque', 'criança', 'correr', 'brincar', 'Jardim', 'cachorro',
'correr', 'atrás', 'bola', 'correr', 'pegar', 'ônibus', 'atleta', 'correr',
'volta', 'pista', 'ver', 'grupo', 'pessoa', 'correr', 'parque', 'criança',
'correr', 'brincar', 'Jardim', 'cachorro', 'correr', 'atrás', 'bola',
'correr', 'pegar', 'ônibus', 'atleta', 'correr', 'volta', 'pista', 'ver',
'grupo', 'pessoa', 'correr', 'parque', 'criança', 'correr', 'brincar',
```

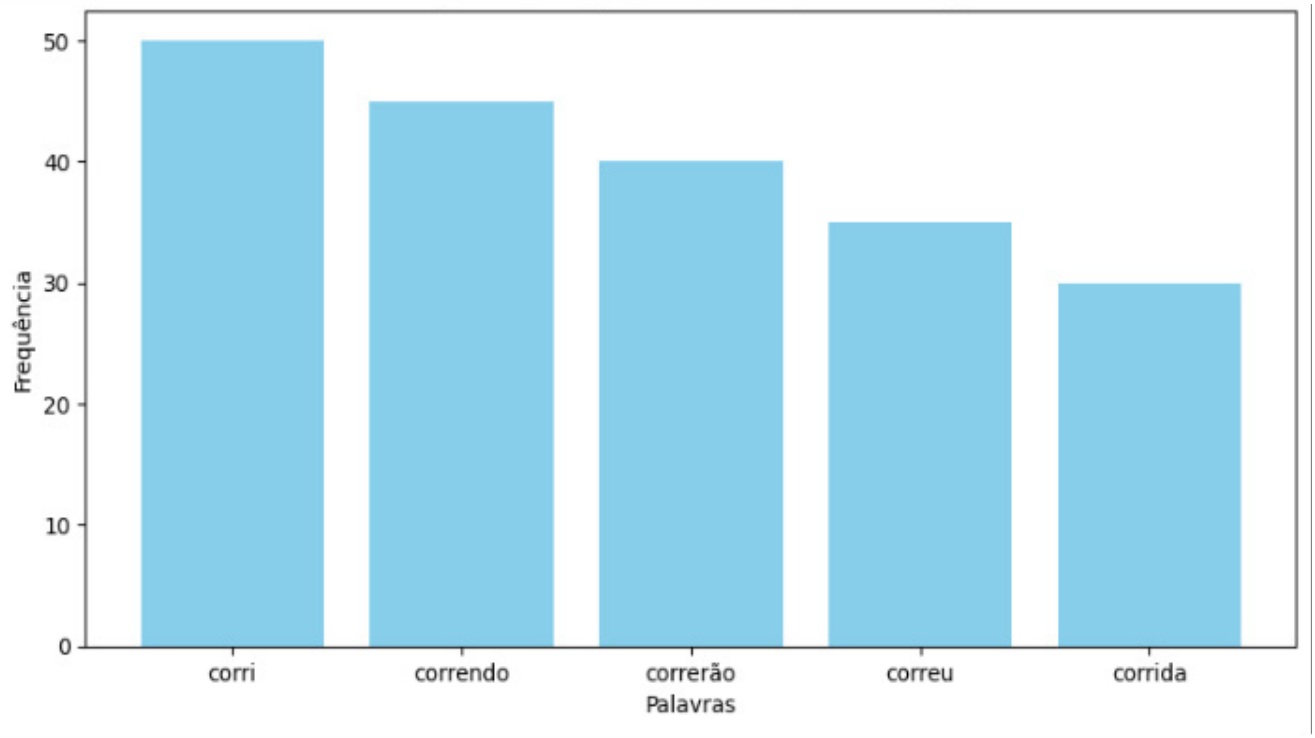
continua

```
'Jardim', 'cachorro', 'correr', 'atrás', 'bola', 'correr', 'maratonar',
'próximo', 'mês', 'competidor', 'correr', 'diferente', 'categoria', 'todo',
'atleta', 'correr', 'tempo', 'correr', 'primeira', 'vez', 'prova',
'corredor', 'profissional', 'correr', 'próximo', 'competição', 'correr',
'maratonar', 'próximo', 'mês', 'competidor', 'correr', 'diferente',
'categoria', 'todo', 'atleta', 'correr', 'tempo', 'correr', 'primeira',
'vez', 'prova', 'corredor', 'profissional', 'correr', 'próximo',
'competição', 'correr', 'maratonar', 'próximo', 'mês', 'competidor',
'correr', 'diferente', 'categoria', 'todo', 'atleta', 'correr', 'tempo',
'correr', 'primeira', 'vez', 'prova', 'corredor', 'profissional', 'correr',
'próximo', 'competição', 'correr', 'maratonar', 'próximo', 'mês',
'competidor', 'correr', 'diferente', 'categoria', 'todo', 'atleta',
'correr', 'tempo', 'correr', 'primeira', 'vez', 'prova', 'corredor',
'profissional', 'correr', 'próximo', 'competição', 'correr', 'maratonar',
'próximo', 'mês', 'competidor', 'correr', 'diferente', 'categoria', 'todo',
'atleta', 'correr', 'tempo', 'correr', 'primeira', 'vez', 'prova',
'corredor', 'profissional', 'correr', 'próximo', 'competição', 'correr',
'maratonar', 'próximo', 'mês', 'competidor', 'correr', 'diferente',
'categoria', 'todo', 'atleta', 'correr', 'tempo', 'correr', 'primeira',
'vez', 'prova', 'corredor', 'profissional', 'correr', 'próximo',
'competição', 'correr', 'maratonar', 'próximo', 'mês', 'competidor',
'correr', 'diferente', 'categoria', 'todo', 'atleta', 'correr', 'tempo',
'correr', 'primeira', 'vez', 'prova', 'corredor', 'profissional', 'correr',
'próximo', 'competição', 'correr', 'maratonar', 'próximo', 'mês',
'competidor', 'correr', 'diferente', 'categoria', 'todo', 'atleta',
'correr', 'tempo', 'correr', 'primeira', 'vez', 'prova', 'corredor',
'profissional', 'correr', 'próximo', 'competição', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'garoto', 'correr',
'escola', 'atrasar', 'correr', 'maratona', 'inteirar', 'parar', 'assim',
'ouvir', 'notícia', 'correr', 'casa', 'cachorro', 'correr', 'direção',
'portão', 'correr', 'pegar', 'último', 'tr', 'noite', 'correr', 'carro',
'amanhã', 'manhã', 'treinar', 'correr', 'hoje', 'correr', 'começar',
'manhã', 'participar', 'correr', 'beneficente', 'domingo', 'todo', 'ansioso',
'grande', 'correr', 'correr', 'carro', 'amanhã', 'manhã', 'treinar',
'correr', 'hoje', 'correr', 'começar', 'manhã', 'participar', 'correr',
'beneficente', 'domingo', 'todo', 'ansioso', 'grande', 'correr', 'correr',
'carro', 'amanhã', 'manhã', 'treinar', 'correr', 'hoje', 'correr',
'começar', 'manhã', 'participar', 'correr', 'beneficente', 'domingo', 'todo',
'ansioso', 'grande', 'correr', 'correr', 'carro', 'amanhã', 'manhã',
'treinar', 'correr', 'hoje', 'correr', 'começar', 'manhã', 'participar',
'correr', 'beneficente', 'domingo', 'todo', 'ansioso', 'grande', 'correr',
'correr', 'carro', 'amanhã', 'manhã', 'treinar', 'correr', 'hoje', 'correr',
'começar', 'manhã', 'participar', 'correr', 'beneficente', 'domingo', 'todo',
'ansioso', 'grande', 'correr', 'correr', 'carro', 'amanhã', 'manhã',
```

continua

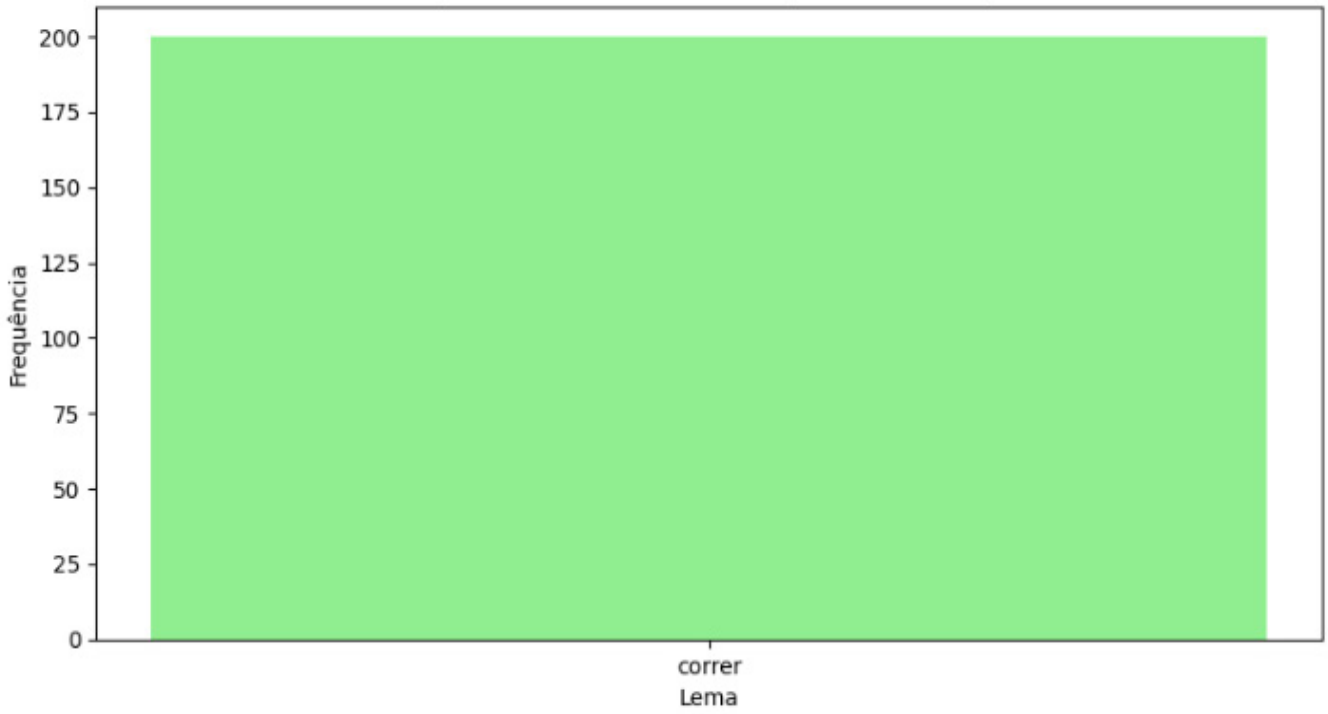
```
'treinar', 'correr', 'hoje', 'correr', 'começar', 'manhã', 'participar',
'correr', 'beneficente', 'domingo', 'todo', 'ansioso', 'grande', 'correr']

Lemas únicos:  [68]['cachorro', 'manhã', 'inteirar', 'maratonar', 'notícia',
'tempo', 'ouvir', 'parque', 'pessoa', 'atrasar', 'tr', 'treinar', 'amanhã',
'bola', 'noite', 'volta', 'direção', 'correr', 'mês', 'beneficente', 'cinco',
'carro', 'pegar', 'grupo', 'rapidamente', 'brincar', 'Escola', 'atleta',
'corredor', 'vez', 'casa', 'todo', 'ontem', 'quilômetro', 'encontrar',
'garoto', 'grande', 'amigo', 'próximo', 'assim', 'primeira', 'começar',
'escola', 'ônibus', 'latir', 'competição', 'ansioso', 'maratona', 'pista',
'prova', 'ver', 'pude', 'Jardim', 'diferente', 'profissional', 'último',
'participar', 'domingo', 'perder', 'contar', 'categoria', 'portão',
'competidor', 'hoje', 'criança', 'rápido', 'atrás', 'parar']
```

Figura 9 - Frequência das palavras antes da lematização



Fonte: autoria própria.

Figura 10 - Frequência do lema "correr" após a lematização



Fonte: autoria própria.

8.2: Atividade: Substituição do Conjunto de Frases e Análise de Resultados

Objetivo:

Nesta atividade, você substituirá o conjunto de frases existente por um novo conjunto de frases em português, executará o código de processamento de linguagem natural (PLN) e analisará como as alterações no conjunto de dados influenciam os resultados da lematização, remoção de stopwords e análise de frequência de palavras.

Passos da Atividade:

1. **Etapa 1: Escolha e Substituição do Conjunto de Frases**

   - Substitua o conjunto de frases original por um novo conjunto de frases. Escolha ao menos dez novas frases que sejam diferentes em termos de estrutura, tema ou complexidade.
   - As novas frases podem abordar um tema específico (como esportes, política, ou tecnologia) ou serem mais variadas.

2. **Etapa 2: Revisão do Código**

   - Revise o código existente para garantir que ele está preparado para processar o novo conjunto de frases.
   - Verifique se as funções de tokenização, remoção de stopwords, e lematização estão prontas para lidar com as novas frases.

3. Etapa 3: Execução do Código

- Execute o código com o novo conjunto de frases.
- Observe os resultados, incluindo a lista de tokens, as palavras lematizadas, e as frequências das palavras antes e depois da lematização.

4. Etapa 4: Análise dos Resultados

- Compare os resultados obtidos com o novo conjunto de frases com os resultados anteriores (se disponível).
- Analise como a mudança no conteúdo das frases afetou:
  - A lista de stopwords removidas.
  - A eficácia da lematização (quantas palavras foram corretamente reduzidas ao mesmo lema).
  - A distribuição de frequências das palavras.
- Identifique quaisquer mudanças notáveis na lista de lemas únicos.

Requisitos:

- Computador com Python instalado.
- Bibliotecas necessárias: **nltk**, **spacy**.
- Código existente para processamento de texto em português.

Instruções:

- Substitua o conjunto de frases no código com um novo conjunto escolhido por você.
- Execute o código revisado e observe os resultados.
- Compare os resultados com qualquer execução anterior para entender o impacto das mudanças.

Objetivo Educacional:

Esta atividade permite que os alunos experimentem como diferentes conjuntos de dados influenciam os resultados de um pipeline de PLN. Ao substituir as frases e analisar os resultados, os alunos desenvolvem uma compreensão mais profunda de como a natureza dos dados textuais afeta a tokenização, remoção de stopwords, lematização e análise de frequência. Essa atividade é fundamental para aprender a adaptar técnicas de PLN a diferentes contextos e tipos de texto.

SAIBA MAIS...

- Para ampliar seu entendimento sobre análise morfológica e aumentar seu conhecimento em PLN, sugerimos que visite e explore os *sites* indicados a seguir:
  - Kaggle - Natural Language Processing
  - Stanford NLP Group
  - NLTK
  - SpaCy
- Esses recursos foram escolhidos por sua relevância e detalhamento no assunto, oferecendo uma compreensão mais completa dos conceitos e técnicas abordados nessa Unidade.

Unidade III

Problemas típicos relacionados ao Processamento de Linguagem Natural

Unidade III - Problemas Típicos Relacionados ao Processamento de Linguagem Natural

Lidar com texto natural no PLN apresenta uma série de desafios devido à complexidade e à variabilidade da linguagem humana. A linguagem pode ser ambígua, dependente do contexto e cheia de sutilezas. Nesta Unidade, examinaremos alguns dos principais problemas que os sistemas de PLN enfrentam, conforme resumido na Tabela 5, e apresentaremos exemplos para ilustrar suas consequências.

Tabela 5 - Descrição dos principais problemas no processamento de linguagem natural

PROBLEMA	DESCRIÇÃO	EXEMPLO
Ambiguidade lexical	Palavras com múltiplos significados.	"Banco" pode ser uma instituição financeira ou um assento.
Ambiguidade sintática	Estruturas gramaticais que podem ser interpretadas de várias maneiras.	"Eu vi o homem com o telescópio."
Entidades nomeadas	Identificação e classificação de nomes próprios em texto.	"Pelé nasceu em Três Corações."
Correferência	Identificação de quando diferentes expressões se referem à mesma entidade.	"Maria foi ao mercado. Ela comprou frutas."
Análise de sentimentos	Determinação das emoções ou opiniões expressas em um texto.	"Estou feliz com o serviço." (Positivo)
Tradução automática	Conversão de texto de um idioma para outro mantendo o significado.	"The cat is on the mat." → "O gato está no tapete."
Desambiguação de palavras	Determinação do significado específico de uma palavra baseada no contexto.	"O banco da praça está quebrado."
Reconhecimento de implicação textual	Determinação se uma sentença implica logicamente outra.	"Todos os pássaros podem voar." → "Um pardal pode voar."
Geração de texto	Criação de texto natural a partir de dados estruturados ou não estruturados.	Resumo de um livro baseado em seu conteúdo.
Ruído e erros	Gestão de textos com erros ortográficos, gramaticais ou outros tipos de ruído.	"O srvio foi mto bom." (O serviço foi muito bom.)

Fonte: autoria própria.

3.1 Reconhecimento de Entidades Nomeadas

O reconhecimento de entidades nomeadas é uma tarefa importante no campo do PLN. Ela consiste em identificar e classificar palavras ou frases em um texto como entidades específicas, tais como nomes de pessoas, organizações, lugares, datas, entre outros. Essa habilidade é importante para diversas aplicações de PLN, incluindo extração de informações, tradução automática e sistemas de resposta a perguntas.

Essa tarefa permite que os sistemas de PLN compreendam e manipulem informações específicas e contextualizadas de forma mais eficiente. Por exemplo, em um artigo de notícia, a identificação precisa de entidades como nomes de pessoas, locais e organizações ajuda a construir uma compreensão mais detalhada do conteúdo, além de permitir a relação precisa entre diferentes informações.

3.1.1 Técnicas de Reconhecimento de Entidades Nomeadas

Existem diversas formas de realizar o reconhecimento de entidades nomeadas, variando desde métodos baseados em regras até técnicas avançadas de aprendizado profundo. Abordagens baseadas em regras frequentemente utilizam expressões regulares e dicionários para identificar e categorizar entidades. Apesar de serem fáceis de implementar, esses métodos podem ser restritos devido à sua rigidez e à dificuldade em lidar com a ambiguidade da linguagem natural.

Já os métodos que empregam aprendizado de máquina utilizam algoritmos treinados para reconhecer entidades a partir de grandes conjuntos de dados anotados. Técnicas como modelos de Markov, redes neurais recorrentes e, mais recentemente, *transformers*, têm se mostrado eficazes na identificação de entidades nomeadas. Esses modelos são capazes de captar padrões complexos e contextuais na linguagem, proporcionando um reconhecimento de entidades mais preciso e robusto.

Texto: "A Apple anunciou a abertura de uma nova loja em Paris."

Entidades: Apple (organização), Paris (local).

Nesse caso, o sistema de reconhecimento de entidades nomeadas detecta "Apple" como uma organização e "Paris" como um local. Essa detecção é fundamental para qualquer aplicação que necessite compreender o contexto ou classificar informações sobre organizações e locais específicos.

Texto: "Pelé foi um dos maiores jogadores de futebol de todos os tempos."

Entidades: Pelé (pessoa).

No contexto do texto, "Pelé" é reconhecido como uma pessoa. Identificar entidades, como Pelé, pode ser vantajoso em sistemas de busca, onde informações sobre figuras públicas ou celebridades são frequentemente buscadas.

3.1.2 Desafios do Reconhecimento de Entidades Nomeadas

O reconhecimento de entidades nomeadas enfrenta diversos desafios, entre eles a ambiguidade lexical e a variedade na forma de expressar as entidades. Por exemplo, o termo "Apple" pode se referir tanto à empresa de tecnologia quanto à fruta maçã, dependendo do contexto. Além disso, novas entidades surgem constantemente, o que exige que os sistemas de reconhecimento sejam continuamente atualizados e adaptáveis às novas informações.

A diversidade linguística também representa um desafio importante. Diferentes idiomas e até mesmo diferentes áreas de um mesmo idioma podem trazer dificuldades singulares para o reconhecimento de entidades nomeadas. Desenvolver modelos que funcionem bem em vários idiomas e contextos especializados é um campo de pesquisa ativo (Figura 11).

Figura 11 - Aplicações do reconhecimento de entidades nomeadas

Fonte: autoria própria.

3.1.3 Ferramentas para Reconhecimento de Entidades Nomeadas

Existem diversas ferramentas de *software* que facilitam a implementação de reconhecimento de entidades nomeadas, incluindo aquelas mencionadas anteriormente em outros contextos. São elas:

a) **SpaCy:** inclui modelos pré-treinados para reconhecimento de entidades nomeadas.

b) **NLTK**: oferece módulos para reconhecimento de entidades nomeadas, *tokenização* e outras tarefas de PLN.

c) **Stanford NLP:** inclui um modelo de reconhecimento de entidades nomeadas que pode ser utilizado para identificar entidades em textos.

3.2 Análise de Sentimentos

A análise de sentimentos é uma técnica do PLN usada para identificar a atitude, emoção ou opinião expressa em um texto. Essa técnica é amplamente aplicada em áreas como monitoramento de redes sociais, estudos de mercado e atendimento ao cliente, visando compreender melhor as opiniões dos usuários.

3.2.1 Técnicas de Análise de Sentimentos

Existem diversas maneiras de realizar a análise de sentimentos, variando desde métodos baseados em léxicos até técnicas de aprendizado profundo. Na Tabela 6, são apresentados alguns exemplos de textos e sentimentos.

a) **Métodos baseados em léxicos:** esses métodos utilizam dicionários de palavras que estão associadas a sentimentos positivos e negativos. Eles são simples e eficazes, especialmente para textos curtos e de menor complexidade.

b) **Métodos baseados em aprendizado de máquina:** aqui, utilizam-se algoritmos de classificação que aprendem a identificar sentimentos a partir de grandes conjuntos de dados anotados. Técnicas como máquinas de vetores de suporte e *Naive Bayes* são comumente aplicadas.

c) **Métodos baseados em aprendizado profundo:** esses métodos fazem uso de redes neurais profundas, como redes neurais recorrentes, para detectar padrões complexos e contextuais na linguagem. Modelos baseados em *transformers*, como BERT e GPT, também têm se mostrado altamente eficazes na análise de sentimentos.

Tabela 6 - Exemplos de textos e sentimentos

TEXTO	SENTIMENTO
"Estou muito feliz com o serviço prestado pela empresa!"	positivo
"Estou extremamente insatisfeito com o atraso na entrega do meu pedido."	negativo

Fonte: autoria própria.

3.2.2 Aplicações da Análise de Sentimentos

As aplicações da análise de sentimentos são amplas e variadas, abrangendo várias áreas (Figura 12):

Figura 12 - Aplicações da análise de sentimentos em redes sociais, atendimento ao cliente, mercado, pesquisa, *marketing* e recursos humanos

Fonte: autoria própria.

3.2.3 Desafios da Análise de Sentimentos

A análise de sentimentos enfrenta diversos obstáculos, como a ambiguidade linguística, o sarcasmo e a ironia, além da diversidade de idiomas. Identificar expressões sarcásticas, por exemplo, pode ser desafiador e resultar em classificações errôneas. Ademais, a análise de sentimentos em diferentes línguas e contextos culturais frequentemente demanda modelos específicos.

Existem várias ferramentas disponíveis para realizar a análise de sentimentos, a saber:

a) **NLTK:** oferece módulos para análise de sentimentos, incluindo dicionários de palavras positivas e negativas. Disponível em: https://www.nltk.org/.

b) **TextBlob:** é uma biblioteca simples para PLN que inclui funcionalidades para análise de sentimentos. Disponível em: https://textblob.readthedocs.io/en/dev/.

c) **VADER:** uma ferramenta de análise de sentimentos que é particularmente eficaz para textos de redes sociais. Disponível em: https://vadersentiment.readthedocs.io/en/latest/.

d) **BERT:** um modelo baseado em *transformers* que pode ser treinado para análise de sentimentos. Disponível em: https://github.com/google-research/bert.

3.3 Desambiguação de Palavras

Desambiguação de palavras é a técnica de identificar qual sentido de uma palavra está sendo aplicado em um contexto específico. Esse é um desafio importante no PLN, na medida em que diversas palavras possuem vários significados que variam conforme o contexto em que aparecem.

3.3.1 Técnicas de Desambiguação de Palavras

Há diversas técnicas para realizar a desambiguação de palavras, como métodos baseados em regras, supervisionados e não supervisionados.

a) **Métodos baseados em regras:** esses métodos utilizam regras linguísticas e dicionários para identificar o significado correto de uma palavra. Embora possam ser eficazes, sua aplicação é limitada devido à dificuldade de criar regras abrangentes e detalhadas.

b) **Métodos supervisionados:** esses métodos se valem de algoritmos de aprendizado de máquina que aprendem a desambiguar palavras a partir de grandes volumes de dados anotados. Abordagens como máquinas de vetores de suporte e redes neurais são comumente empregados.

c) **Métodos não supervisionados:** utilizam técnicas de agrupamento e outras abordagens baseadas em dados não anotados para inferir os significados das palavras. Embora sejam mais flexíveis, geralmente não alcançam a mesma precisão dos métodos supervisionados.

Na Tabela 7, no primeiro caso (exemplo 1), a palavra "banco" indica um assento, enquanto no segundo (exemplo 2), ela representa uma instituição financeira. Identificar corretamente o significado dessas palavras é essencial para a compreensão exata do texto.

Tabela 7 - Exemplos dos diferentes significados para a palavra banco

Texto	Significado
"O banco da praça está quebrado."	Banco (assento
"Ele foi ao banco depositar dinheiro."	Banco (instituição financeira)

Fonte: autoria própria.

3.3.2 Aplicações da Desambiguação de Palavras

A desambiguação de palavras desempenha um papel crucial em várias aplicações no campo do PLN. Entre suas principais utilidades estão descritas na Figura 13.

Figura 13 - Aplicações da desambiguação semântica em processamento de linguagem natural

Fonte: autoria própria.

3.3.3 Desafios da Desambiguação de Palavras

A tarefa de desambiguação de palavras apresenta diversos desafios, como a variação contextual, a exigência de grandes quantidades de dados anotados e a complexidade dos modelos de linguagem. As diferentes acepções das palavras podem ser sutis e altamente dependentes do contexto, o que torna a desambiguação um problema difícil de resolver.

Existem algumas ferramentas e bibliotecas disponíveis para realizar a desambiguação de palavras:

a) ***WordNet*:** uma base de dados lexical que pode ser utilizada para ajudar na desambiguação de palavras. Disponível em: https://wordnet.princeton.edu/.

b) **NLTK:** oferece módulos para desambiguação de palavras, incluindo acesso ao WordNet. Disponível em: https://www.nltk.org/.

c) **SpaCy:** Inclui funcionalidades para desambiguação de palavras, especialmente quando combinadas com modelos pré-treinados. Disponível em: https://spacy.io/.

3.4 Reconhecimento de Implicação Textual

A tarefa de reconhecimento de implicação textual envolve determinar se uma frase pode ser logicamente inferida de outra. Essa tarefa é essencial para diversas aplicações de PLN, como sistemas de perguntas e respostas e análise de texto.

3.4.1 Técnicas de Reconhecimento de Implicação Textual

Existem diferentes formas de realizar o reconhecimento de entidades textuais, entre elas estão os métodos baseados em lógica, os supervisionados e aqueles que utilizam aprendizado profundo.

a) **Métodos baseados em lógica:** esses métodos empregam lógica formal para estabelecer a implicação entre sentenças. Embora possam ser bastante precisos, sua aplicação é limitada pela complexidade inerente à lógica formal e pela necessidade de transformar sentenças em representações lógicas.

b) **Métodos supervisionados:** aqui, algoritmos de aprendizado de máquina são utilizados para aprender a identificar a implicação textual a partir de grandes conjuntos de dados anotados. Entre as técnicas comuns estão as máquinas de vetores de suporte e as redes neurais.

c) **Métodos baseados em aprendizado profundo:** utilizam redes neurais profundas, como redes neurais recorrentes e *transformers*, para captar padrões complexos e contextuais na linguagem. Esses métodos estão se mostrando eficazes em tarefas de reconhecimento de entidades textuais.

No primeiro exemplo da Tabela 8, a hipótese é uma consequência lógica da premissa. Já no segundo exemplo, a hipótese é oposta à premissa, levando a uma implicação incorreta.

Tabela 8 - Exemplos de premissas

Premissa	Hipótese	Resultado
"Todos os pássaros podem voar."	"Um pardal pode voar."	"Um pinguim pode voar."
"Nem todos os pássaros podem voar."	implicação verdadeira	implicação falsa

3.4.2 Aplicação

As aplicações do reconhecimento de textos e entidades são variadas e abrangem:

a) **Sistemas de perguntas e respostas**: aumentar a precisão das respostas, verificando se uma resposta pode ser logicamente inferida a partir da pergunta.

b) **Análise de texto**: auxiliar na análise de argumentos e extração de informações, identificando as relações lógicas entre as sentenças.

c) **Verificação de fatos:** contribuir para a verificação da veracidade das afirmações, comparando-as com fatos conhecidos.

O reconhecimento de implicação textual encontra uma série de desafios, como a complexidade inerente à linguagem natural, a demanda por grandes quantidades de dados anotados e a variabilidade de contextos. A representação de sentenças para viabilizar a inferência lógica é uma questão intrincada e em constante desenvolvimento.

3.4.3 Ferramenta

Para realizar a tarefa de reconhecimento de implicação textual, existem diversas ferramentas e bibliotecas à disposição. Um exemplo é o *Stanford NLP,* que oferece uma variedade de modelos e ferramentas específicas para essa tarefa.

SAIBA MAIS...

» Acesse o *site* oficial: https://nlp.stanford.edu/projects/infer/primer.html.

3.5 Resolução de Correferência

A resolução de correferências é a tarefa de determinar quando diferentes expressões em um texto se referem à mesma entidade. Essa habilidade é fundamental para a compreensão de textos e é empregada em diversas aplicações de PLN, como sistemas de perguntas e respostas e extração de informações.

3.5.1 Técnicas de Resolução de Correferência

Existem diversas técnicas para resolver a questão da correferência, cada uma com suas características específicas. Entre elas, destacam-se os métodos baseados em regras, os métodos supervisionados e os métodos baseados em aprendizado profundo.

a) **Métodos baseados em regras:** esses métodos aplicam regras linguísticas e heurísticas para identificar referências em um texto. Embora sejam relativamente simples de implementar, sua eficácia pode ser limitada pela inflexibilidade das regras estabelecidas.

b) **Métodos supervisionados:** utilizam algoritmos de aprendizado de máquina que treinam modelos para identificar referências com base em grandes conjuntos de dados previamente anotados. Entre as técnicas utilizadas estão as máquinas de vetores de suporte e as redes neurais.

c) **Métodos baseados em aprendizado profundo:** essas abordagens empregam redes neurais profundas, como redes neurais recorrentes e *transformers*, para detectar padrões complexos e contextuais na linguagem. Esse tipo de método tem se mostrado particularmente eficaz na resolução de tarefas de correferência.

No primeiro exemplo da Tabela 9, a palavra "Ela" faz referência a "Maria". Já no segundo exemplo, "Eles" se refere a "João e Pedro". Identificar corretamente essas referências é essencial para compreender o texto de maneira precisa.

Tabela 9 - Exemplos de correferências

Texto	Referências
"Maria foi ao mercado. Ela comprou frutas."	"João e Pedro foram ao parque. Eles jogaram futebol."
Maria → Ela	João e Pedro → Eles

Fonte: autoria própria.

3.5.2 Aplicação

A resolução de correferência desempenha um papel importante em várias aplicações no campo do PLN, como:

a) **Sistemas de perguntas e respostas:** ao identificar corretamente as referências no texto, a precisão das respostas fornecidas pode ser significativamente melhorada.

b) **Extração de informações:** a identificação correta das entidades referenciadas facilita o processo de extração de informações.

c) **Análise de texto:** a resolução de correferências é fundamental para a compreensão e análise de textos complexos, especialmente em áreas técnicas e especializadas.

A resolução de correferências apresenta diversos desafios, como a variabilidade contextual, a necessidade de vastas quantidades de dados anotados e a complexidade inerente aos modelos de linguagem. As diferentes formas de referência podem ser sutis e altamente dependentes do contexto, tornando a tarefa de resolução particularmente difícil.

3.5.3 Ferramenta para Resolução de Correferência

O *Stanford NLP* oferece modelos e ferramentas específicas para a tarefa de resolução de correferências.

SAIBA MAIS...

» Acesse: https://stanfordnlp.github.io/CoreNLP/coref.html.

3.6 Tradução Automática

A tradução automática envolve a conversão de textos de um idioma para outro, mantendo seu significado original. Essa é uma tarefa particularmente desafiadora e complexa no âmbito do PLN, devido à grande variabilidade e às nuances das línguas naturais.

Existem diversas abordagens para realizar a tradução automática, incluindo métodos baseados em regras, métodos estatísticos e métodos baseados em aprendizado profundo. São elas:

a) **Métodos baseados em regras:** esses métodos utilizam gramáticas e dicionários para realizar a tradução. No entanto, são limitados pela rigidez das regras e pela dificuldade em lidar com a variabilidade da linguagem natural.

b) **Métodos estatísticos:** os métodos estatísticos utilizam modelos probabilísticos que aprendem a traduzir usando grandes coleções de textos paralelos. Uma técnica bastante comum nesse campo é a tradução baseada em frases, que faz uso de segmentos de texto para aumentar a precisão e a fluência da tradução.

c) **Métodos baseados em aprendizado profundo:** empregam redes neurais profundas, como redes neurais recorrentes e *transformers*, para capturar padrões complexos e contextuais na linguagem. Modelos que usam *transformers* destacam-se por sua alta eficiência em tarefas como tradução automática.

No primeiro exemplo da Tabela 10, a tradução é direta e preserva o significado da frase original. No segundo exemplo, "bank" é corretamente traduzido como "banco", referindo-se a uma instituição financeira.

Tabela 10 - Exemplos de tradução

Texto original	Tradução
"The cat is on the mat."	"O gato está no tapete."
"He went to the bank to deposit money."	"Ele foi ao banco depositar dinheiro."

Fonte: autoria própria.

3.6.1 Aplicação

A tradução automática possui uma ampla gama de aplicações, incluindo:

a) **Serviços de tradução online:** plataformas como o *Google Translate* possibilitam traduções rápidas de textos entre diversos idiomas.

b) **Assistentes virtuais:** ferramentas como a *Siri* e a *Alexa* usam a tradução automática para compreender e responder em várias línguas.

c) **Comunicação internacional:** facilita a interação entre pessoas que falam línguas diferentes, sendo especialmente útil em ambientes empresariais e diplomáticos.

A tradução automática encontra uma série de obstáculos, como a ambiguidade da linguagem, a manutenção do contexto e o tratamento de expressões idiomáticas. As diversas línguas apresentam estruturas gramaticais e semânticas distintas, o que torna a tarefa de tradução bastante complexa.

3.6.2 Ferramentas para Tradução Automática

Existem diversas ferramentas e bibliotecas que podem ser muito úteis para a tradução automática. Algumas delas incluem:

1. ***Google™ Translate:*** Um serviço amplamente utilizado que se baseia em tecnologias de aprendizado profundo para fornecer traduções precisas e rápidas.
2. **OpenNMT:** Uma ferramenta de código aberto especializada em tradução automática, utilizando arquitetura de *transformers*.

SAIBA MAIS...

» Google™ Translate

» OpenNMT

3.7 Geração de Texto

A geração de texto envolve a criação de texto natural a partir de dados que podem ser estruturados ou não. Esta atividade é fundamental em várias aplicações de PLN, como a produção de resumos, a criação de conteúdo e o suporte a assistentes de escrita.

Existem diferentes métodos para realizar a geração de texto, que incluem abordagens baseadas em regras, métodos estatísticos e técnicas de aprendizado profundo:

a) **Métodos baseados em regras:** esses métodos utilizam gramáticas e regras predefinidas para gerar texto. Apesar de serem precisos dentro do escopo das regras definidas, eles enfrentam limitações quando lida-se com a variabilidade e a complexidade da linguagem natural.

b) **Métodos estatísticos:** essa abordagem emprega modelos probabilísticos que aprendem a criar textos a partir de grandes coleções de textos existentes. Técnicas como os modelos de *n-gramas* são frequentemente utilizadas, pois ajudam a prever a próxima palavra em uma sequência com base nas palavras anteriores. Em um modelo de *4-gramas*, por exemplo, consideram-se sequências de quatro palavras. A probabilidade de uma palavra surgir depende das três palavras que a antecedem. Por exemplo, na frase "o gato está no telhado", a probabilidade de "no" aparecer depois de "o gato está" é calculada dividindo-se a frequência da sequência "o gato está no" pela frequência de "o gato está" no *corpus*.

c) **Métodos baseados em aprendizado profundo:** estas técnicas empregam redes neurais profundas, como redes neurais recorrentes e *transformers*, para capturar padrões complexos e contextuais na linguagem. Modelos como o GPT-3 têm mostrado grande eficácia nas tarefas de geração de texto, graças à sua capacidade de compreender e reproduzir contextos linguísticos complexos.

"**Título:** A Revolução do PLN,
Subtítulo: Avanços e Desafios,
Conteúdo: Nos últimos anos, o PLN tem..."

Texto Gerado:
Título: A Revolução do PLN: Avanços e Desafios
Conteúdo: Nos últimos anos, o Processamento de Linguagem Natural (PLN) tem experimentado avanços significativos. Com a introdução de novas [...]

No exemplo anterior, os dados estruturados são utilizados para gerar textos coerentes e contextualmente apropriados. A geração de texto é para aplicações que exigem a criação automática de conteúdo de qualidade.

3.7.1 Aplicação

As aplicações da geração de texto são variadas e abrangem diferentes áreas, incluindo:

a) **Criação de resumos:** produção de resumos de documentos extensos para tornar a leitura e a compreensão mais rápidas e eficientes.

b) **Assistentes de escrita:** apoio a escritores na produção de conteúdo, oferecendo sugestões e ajudando na finalização de frases.

c) **Chatbots e assistentes virtuais:** geração de respostas naturais e adequadas ao contexto para interações com usuários.

A criação de textos apresenta diversos desafios, entre os quais se destacam a manutenção da coerência, a preservação do contexto e a fluidez da linguagem. Produzir textos que sejam naturais e adequados ao contexto é uma tarefa complexa que demanda o uso de modelos linguísticos avançados.

SAIBA MAIS...

» Há diversas ferramentas e bibliotecas disponíveis para a criação de texto:

» OpenAI GPT

» Claude

3.8 Explorando Aplicações Práticas de Processamento de Linguagem Natural

Este *Colab* foi desenvolvido para proporcionar uma experiência prática nas principais técnicas de NLP, utilizando as bibliotecas SpaCy, NLTK, WordNet e *Googletrans*. As atividades seguem uma progressão lógica, permitindo que os estudantes apliquem conceitos como reconhecimento de entidades nomeadas, análise de sentimentos, desambiguação de palavras, resolução de correferências e tradução automática.

Cada tarefa começa com uma explicação teórica resumida, seguida de exemplos práticos em Python. Os(As) discentes são encorajados a executar o código, experimentar mudanças nas entradas e observar como isso afeta os resultados, promovendo uma compreensão mais profunda e interativa.

Entre os tópicos abordados estão o uso de técnicas de reconhecimento de entidades para organizar dados textuais não estruturados, a análise de sentimentos para identifi-

car polaridade em textos, a desambiguação de palavras com NLTK e WordNet, além da resolução de correferência com modelos BERT para melhorar a coesão textual. A tradução automática com *Googletrans* também é explorada, permitindo a prática na conversão de textos entre diferentes idiomas.

O *Colab* foi elaborado para proporcionar uma experiência prática e envolvente, ajudando os(as) discentes a desenvolverem suas habilidades em PLN de forma estruturada e aplicada. Acesse o *Notebook Colab* pelo link a seguir e inicie suas atividades:

Notebook Colab

Objetivos de Aprendizagem

Nesta atividade de laboratório, vamos explorar cinco aplicações práticas do Processamento de Linguagem Natural utilizando a linguagem Python e diversas bibliotecas da área. Cada programa abordará um problema específico do Processamento de Linguagem Natural, mostrando como lidar com texto natural de maneira eficiente.

1. **Reconhecimento de Entidades Nomeadas:** Identifica e classifica palavras ou frases em um texto como entidades específicas, como nomes de pessoas, organizações e locais.
2. **Análise de Sentimentos:** Avalia a atitude, emoção ou opinião expressa em um texto, classificando-o como positivo, negativo ou neutro.
3. **Desambiguação de Palavras:** Determina o significado específico de uma palavra com múltiplos sentidos, com base no contexto em que ela aparece.
4. **Resolução de Correferência:** Identifica quando diferentes expressões em um texto se referem à mesma entidade, essencial para a compreensão de textos complexos.
5. **Tradução Automática:** Converte texto de um idioma para outro, mantendo seu significado original.

III.1 Reconhecimento de Entidades Nomeadas

Definição: Reconhecimento de Entidades Nomeadas (NER)

É uma técnica do Processamento de Linguagem Natural (PLN) que envolve a identificação e a classificação de entidades mencionadas em um texto. Essas entidades podem ser nomes de pessoas, organizações, locais, datas, quantidades, entre outras categorias. O objetivo do NER é transformar dados textuais não estruturados em informações estruturadas e facilmente interpretáveis por sistemas computacionais.

Importância:

O NER é uma ferramenta fundamental em diversas aplicações de PLN devido à sua capacidade de extrair informações essenciais de grandes volumes de texto de forma automatizada. Algumas das principais áreas de aplicação incluem:

1. **Análise de Sentimentos:** Identificar entidades permite analisar como diferentes entidades são mencionadas em textos e medir as opiniões ou sentimentos associados a elas.
2. **Sistemas de Resposta a Perguntas:** Melhorar a precisão de respostas fornecendo informações específicas sobre entidades mencionadas nas perguntas.
3. **Análise de Redes Sociais:** Monitorar menções a marcas, produtos ou pessoas em redes sociais para entender tendências e opiniões públicas.
4. **Automação de Processos:** Ajudar na extração de informações relevantes em documentos legais, financeiros, médicos, entre outros, automatizando tarefas que anteriormente exigiam intervenção humana.

Exemplo do Programa:

O programa apresentado utiliza a biblioteca SpaCy para realizar o reconhecimento de entidades nomeadas em um texto em português. A seguir, uma breve descrição de seu funcionamento:

1. **Instalação e Carregamento do Modelo:** O programa verifica se o SpaCy e o modelo de linguagem em português (pt_core_news_sm) estão instalados. Se não estiverem, ele os instala e carrega o modelo de linguagem.
2. **Processamento do Texto:** Um texto de exemplo é definido: **"A Apple anunciou a abertura de uma nova loja em Brasília, capital do Brasil."** Esse texto é processado pelo modelo de linguagem para identificar as entidades mencionadas.
3. **Exibição das Entidades:** As entidades reconhecidas são impressas na tela, juntamente com suas categorias. No exemplo, o programa identificará **"Apple" como uma organização, "Brasília" como um local e "Brasil"** também como um local.

Tarefa 1.1: Avaliar o programa exemplo fornecido a seguir e para tal:

1. execute o código em questão; e

```
# --- Reconhecimento de Entidades Nomeadas com SpaCy em Português ---

# --- Verificar e instalar SpaCy e o modelo de linguagem ---
try:
```

continua

```
    import spacy
    from spacy.cli import download
    spacy_installed = True
except ImportError:
    spacy_installed = False

# Se o SpaCy não estiver instalado, instalar o SpaCy e o modelo de linguagem
if not spacy_installed:
    !pip install spacy
    !python -m spacy download pt_core_news_sm
    import spacy
    from spacy.cli import download

# Garantir que o modelo pt_core_news_sm está instalado
try:
    nlp = spacy.load('pt_core_news_sm')
except OSError:
    download('pt_core_news_sm')
    nlp = spacy.load('pt_core_news_sm')

# Texto de exemplo em português
text = "A Apple anunciou a abertura de uma nova loja em Brasília, capital do Brasil."

# Processar o texto
doc = nlp(text)

# Imprimir entidades reconhecidas
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Apple ORG
Brasília LOC
Brasil LOC
```

Tarefa 1.2: Com base no conteúdo lido no ebook, escreva brevemente sobre a importância do Reconhecimento de Entidades Nomeadas em uma situação diferente das mencionadas acima. Escreva sua resposta no espaço indicado a seguir.

Escreva a sua resposta da Tarefa 1.2 aqui

III.2. Análise de Sentimentos

Conceito de Análise de Sentimentos

A análise de sentimentos, também conhecida como mineração de opinião, é um campo do processamento de linguagem natural que envolve a identificação e extração de informações subjetivas de textos. O objetivo principal é determinar o estado emocional ou a opinião expressa em uma frase ou documento, classificando-o como positivo, negativo ou neutro. Esse processo é realizado através de algoritmos que analisam palavras, frases e contextos dentro de um texto para avaliar os sentimentos subjacentes.

Importância da Análise de Sentimentos

A análise de sentimentos tem se tornado cada vez mais relevante em diversas áreas, como negócios, marketing, política e pesquisa social. Aqui estão alguns motivos que destacam sua importância:

1. **Tomada de Decisões Informadas:** Empresas utilizam a análise de sentimentos para entender melhor a opinião de seus clientes sobre produtos e serviços, ajudando na tomada de decisões estratégicas.
2. **Monitoramento de Marca:** A reputação de uma marca pode ser monitorada em tempo real, permitindo que as empresas respondam rapidamente a crises e gerenciem a percepção pública.
3. **Aprimoramento de Produtos:** Feedbacks negativos identificados através da análise de sentimentos podem indicar áreas de melhoria, ajudando as empresas a aprimorarem seus produtos ou serviços.
4. **Análise de Competição:** As opiniões dos consumidores sobre concorrentes podem ser analisadas para identificar pontos fortes e fracos relativos.
5. **Engajamento com o Cliente:** A análise de sentimentos ajuda a personalizar a comunicação com os clientes, ajustando mensagens para serem mais empáticas e relevantes.

Classificação na Análise de Sentimentos

A análise de sentimentos geralmente classifica o texto em três categorias principais:

1. Positivo: Indica uma opinião favorável ou emoções positivas, como alegria, entusiasmo e satisfação.
2. Negativo: Indica uma opinião desfavorável ou emoções negativas, como tristeza, raiva e frustração.

3. Neutro: Indica uma opinião neutra ou a ausência de emoção significativa, podendo incluir fatos ou declarações objetivas.

Alguns sistemas de análise de sentimentos avançados também identificam sentimentos mais específicos, como medo, surpresa, confiança, entre outros.

Interpretação dos Resultados

A interpretação dos resultados da análise de sentimentos envolve a compreensão das métricas fornecidas pelos algoritmos. As principais métricas incluem:

1. **Polaridade:** Uma pontuação que varia de -1 a 1, onde -1 indica uma opinião muito negativa, 1 indica uma opinião muito positiva e 0 indica neutralidade.
2. **Subjectividade:** Uma pontuação que varia de 0 a 1, onde 0 representa fatos objetivos e 1 representa opiniões subjetivas.
3. **Pontuações Positivas, Negativas e Neutras:** Proporções que indicam a presença de sentimentos positivos, negativos e neutros no texto analisado.

Por exemplo, uma análise de sentimentos de uma avaliação de produto pode resultar em uma polaridade de 0.8 (muito positiva), com uma subjectividade de 0.9 (altamente subjetiva), indicando uma opinião pessoal muito favorável ao produto.

Aplicações Práticas

1. **Redes Sociais:** Monitoramento de menções de marca e análise de feedback dos clientes.
2. **Serviço ao Cliente:** Análise de interações para melhorar o atendimento ao cliente e identificar problemas recorrentes.
3. **Política:** Análise de discursos, debates e opiniões públicas para compreender o sentimento popular em relação a políticas e candidatos.
4. **Pesquisa de Mercado:** Compreensão de tendências e preferências do consumidor através da análise de opiniões em reviews, blogs e fóruns.

A análise de sentimentos é uma ferramenta que transforma dados textuais em insights acionáveis, permitindo que empresas e organizações compreendam melhor as opiniões e emoções de seus públicos. Sua capacidade de fornecer informações em tempo real e detalhadas sobre a percepção de marca, satisfação do cliente e tendências do mercado torna-se essencial para a tomada de decisões estratégicas e a melhoria contínua dos serviços e produtos oferecidos.

Tarefa 2.1: Avaliar o programa exemplo fornecido a seguir e para tal:

execute o código em questão; e

mude o texto da linha 25 e re-execute o código e analise o resultado obtido.

```
# Verificar e instalar googletrans e NLTK
try:
    from googletrans import Translator
    import nltk
    from nltk.sentiment.vader import SentimentIntensityAnalyzer
    googletrans_installed = True
except ImportError:
    googletrans_installed = False

if not googletrans_installed:
    !pip install googletrans==4.0.0-rc1
    !pip install nltk
    import nltk
    from googletrans import Translator
    from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Garantir que o vader_lexicon está instalado
nltk.download('vader_lexicon')

# Inicializar o analisador de sentimentos VADER
analyzer = SentimentIntensityAnalyzer()

# Inicializar o tradutor
translator = Translator()

# Textos de exemplo em português
texts = [
    "Eu estou muito feliz hoje, mas ontem foi um dia terrível.",
    "O cliente reclamou do atraso na entrega das mercadorias."
]

# Função para traduzir texto usando googletrans
def traduzir_texto(texto):
    try:
```

continua

```
        traduzido = translator.translate(texto, dest='en')
        return traduzido.text
    except Exception as e:
        print("Erro na tradução:", e)
        return None

# Traduzir os textos para inglês usando googletrans e analisar os sentimentos
for text in texts:
    translated_text = traduzir_texto(text)
    if translated_text:
        # Analisar o sentimento do texto traduzido
        sentiment = analyzer.polarity_scores(translated_text)

        # Imprimir os resultados
        print(f"Texto original: {text}")
        print(f"Texto traduzido: {translated_text}")
        print(f"Polaridade: {sentiment['compound']}")
        print(f"Positivo: {sentiment['pos']}")
        print(f"Negativo: {sentiment['neg']}")
        print(f"Neutro: {sentiment['neu']}")
        print("\n")
    else:
        print(f"Não foi possível traduzir o texto: {text}")
```

```
Collecting googletrans==4.0.0-rc1
  Downloading googletrans-4.0.0rc1.tar.gz (20 kB)
  Preparing metadata (setup.py) ... done
Collecting httpx==0.13.3 (from googletrans==4.0.0-rc1)
  Downloading httpx-0.13.3-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2024.8.30)
Collecting hstspreload (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading hstspreload-2024.11.1-py3-none-any.whl.metadata (2.1 kB)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (1.3.1)
Collecting chardet==3.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading chardet-3.0.4-py2.py3-none-any.whl.metadata (3.2 kB)
```

continua

```
Collecting idna==2.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading idna-2.10-py2.py3-none-any.whl.metadata (9.1 kB)
Collecting rfc3986<2,>=1.3 (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading rfc3986-1.5.0-py2.py3-none-any.whl.metadata (6.5 kB)
Collecting httpcore==0.9.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading httpcore-0.9.1-py3-none-any.whl.metadata (4.6 kB)
Collecting h11<0.10,>=0.8 (from httpcore==0.9.*->httpx==0.13.3-
>googletrans==4.0.0-rc1)
  Downloading h11-0.9.0-py2.py3-none-any.whl.metadata (8.1 kB)
Collecting h2==3.* (from httpcore==0.9.*->httpx==0.13.3-
>googletrans==4.0.0-rc1)
  Downloading h2-3.2.0-py2.py3-none-any.whl.metadata (32 kB)
Collecting hyperframe<6,>=5.2.0 (from h2==3.*->httpcore==0.9.*-
>httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading hyperframe-5.2.0-py2.py3-none-any.whl.metadata (7.2 kB)
Collecting hpack<4,>=3.0 (from h2==3.*->httpcore==0.9.*->httpx==0.13.3-
>googletrans==4.0.0-rc1)
  Downloading hpack-3.0.0-py2.py3-none-any.whl.metadata (7.0 kB)
Downloading httpx-0.13.3-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.1/55.1 kB 2.5 MB/s eta
0:00:00
Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.4/133.4 kB 5.5 MB/s eta
0:00:00
Downloading httpcore-0.9.1-py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 1.8 MB/s eta
0:00:00
Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 2.8 MB/s eta
0:00:00
Downloading h2-3.2.0-py2.py3-none-any.whl (65 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.0/65.0 kB 4.3 MB/s eta
0:00:00
Downloading rfc3986-1.5.0-py2.py3-none-any.whl (31 kB)
Downloading hstspreload-2024.11.1-py3-none-any.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 34.9 MB/s eta
0:00:00
Downloading h11-0.9.0-py2.py3-none-any.whl (53 kB)
```

continua

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.6/53.6 kB 2.7 MB/s eta
0:00:00

Downloading hpack-3.0.0-py2.py3-none-any.whl (38 kB)

Downloading hyperframe-5.2.0-py2.py3-none-any.whl (12 kB)

Building wheels for collected packages: googletrans

  Building wheel for googletrans (setup.py) ... done

  Created wheel for googletrans: filename=googletrans-
4.0.0rc1-py3-none-any.whl size=17397
sha256=d1e2bbf778308327004d3b084db3a8866c230e1178931a92194ec2ae835ff0da

  Stored in directory: /root/.cache/pip/wheels/
c0/59/9f/7372f0cf70160fe61b528532e1a7c8498c4becd6bcffb022de

Successfully built googletrans

Installing collected packages: rfc3986, hyperframe, hpack, h11, chardet,
idna, hstspreload, h2, httpcore, httpx, googletrans

  Attempting uninstall: h11

    Found existing installation: h11 0.14.0

    Uninstalling h11-0.14.0:

      Successfully uninstalled h11-0.14.0

  Attempting uninstall: chardet

    Found existing installation: chardet 5.2.0

    Uninstalling chardet-5.2.0:

      Successfully uninstalled chardet-5.2.0

  Attempting uninstall: idna

    Found existing installation: idna 3.10

    Uninstalling idna-3.10:

      Successfully uninstalled idna-3.10

  Attempting uninstall: httpcore

    Found existing installation: httpcore 1.0.7

    Uninstalling httpcore-1.0.7:

      Successfully uninstalled httpcore-1.0.7

  Attempting uninstall: httpx

    Found existing installation: httpx 0.27.2

    Uninstalling httpx-0.27.2:

      Successfully uninstalled httpx-0.27.2

ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of the
```

continua

```
following dependency conflicts.

langsmith 0.1.143 requires httpx<1,>=0.23.0, but you have httpx 0.13.3
which is incompatible.

openai 1.54.4 requires httpx<1,>=0.23.0, but you have httpx 0.13.3 which
is incompatible.

Successfully installed chardet-3.0.4 googletrans-4.0.0rc1 h11-0.9.0 h2-
3.2.0 hpack-3.0.0 hstspreload-2024.11.1 httpcore-0.9.1 httpx-0.13.3
hyperframe-5.2.0 idna-2.10 rfc3986-1.5.0

Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.9.1)

Requirement already satisfied: click in /usr/local/lib/python3.10/dist-
packages (from nltk) (8.1.7)

Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-
packages (from nltk) (1.4.2)

Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/
python3.10/dist-packages (from nltk) (2024.9.11)

Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from nltk) (4.66.6)

[nltk_data] Downloading package vader_lexicon to /root/nltk_data...

Texto original: Eu estou muito feliz hoje, mas ontem foi um dia terrível.

Texto traduzido: I'm very happy today, but yesterday it was a terrible
day.

Polaridade: -0.3926

Positivo: 0.17

Negativo: 0.283

Neutro: 0.546



Texto original: O cliente reclamou do atraso na entrega das mercadorias.

Texto traduzido: The customer complained about the delay in the delivery
of the goods.

Polaridade: -0.6124

Positivo: 0.0

Negativo: 0.333

Neutro: 0.667
```

Tarefa 2.2: considerando os seguintes cenários de aplicação: **Redes Sociais:, Serviço ao Cliente:, Política:** e Pesquisa de Mercado crie textos de uma ou duas linhas, submeta ao programa feito na Tarefa 2.1, e análise os resultados. Escreva no espaço a seguir os textos e as suas análises.

Escreva aqui a sua resposta a **Tarefa 2.2**

III.3 Desambiguação de Palavras com NLTK e Google Translate

A **desambiguação de palavras** é um processo crucial no campo do Processamento de Linguagem Natural que consiste em identificar qual sentido de uma palavra está sendo usado em um determinado contexto. Muitas palavras têm múltiplos significados, e a habilidade de determinar o sentido correto é essencial para a compreensão precisa de textos.

Por exemplo, a palavra "**banco**" pode se referir a uma instituição financeira ou a um assento. O contexto da frase determina qual sentido é apropriado.

Importância

A desambiguação de palavras é importante por várias razões:

1. **Melhoria da Precisão em Aplicações de PLN:** A desambiguação correta das palavras melhora a precisão de diversas aplicações de PLN, como tradução automática, busca em documentos e respostas automáticas.
2. **Análise de Sentimentos:** Identificar o sentido correto das palavras ajuda a interpretar melhor os sentimentos expressos em textos.
3. **Assistentes Virtuais**: Assistentes virtuais, como Siri e Alexa, dependem da desambiguação de palavras para compreender e responder adequadamente às solicitações dos usuários.
4. **Recuperação de Informação:** Sistemas de busca e recuperação de informação utilizam a desambiguação para fornecer resultados mais relevantes.

Exemplo de Aplicação

Tarefa 3.1. Explorar um exemplo prático usando NLTK e Google Translate para desambiguar palavras em português. Este exemplo mostrará como determinar o sentido correto da palavra "banco" e "manga" em diferentes contextos. E para avaliar este cenário:

1. execute o código em questão; e
2. mude o texto das linhas 49 e 50 e re-execute o código e analise o resultado obtido.

```python
# Verificar e instalar googletrans e NLTK
try:
    from googletrans import Translator
    from nltk.corpus import wordnet
    nltk_installed = True
except ImportError:
    nltk_installed = False

if not nltk_installed:
    !pip install googletrans==4.0.0-rc1
    !pip install nltk
    import nltk
    nltk.download('wordnet')
    nltk.download('omw-1.4')
    from googletrans import Translator
    from nltk.corpus import wordnet

# Inicializar o tradutor
translator = Translator()

# Função para desambiguação de palavras
def get_sense(word, context):
    synsets = wordnet.synsets(word, lang='por')
    if not synsets:
        return None
    best_sense = synsets[0]
    max_overlap = 0

    for synset in synsets:
        definition = synset.definition().split()
        overlap = len(set(definition).intersection(set(context)))
        if overlap > max_overlap:
            max_overlap = overlap
            best_sense = synset

    return best_sense

# Função para traduzir texto usando googletrans
```

continua

```python
def traduzir_texto(texto, dest_lang='pt'):
    try:
        traduzido = translator.translate(texto, dest=dest_lang)
        return traduzido.text
    except Exception as e:
        print("Erro na tradução:", e)
        return texto

# Exemplos de textos em português
examples = [
    ("Ele vestiu uma camisa com uma longa manga.", "manga"),
    ("Ele foi ao banco para depositar dinheiro.", "banco")
]

# Processar cada exemplo
for text, word in examples:
    context = text.split()
    sense = get_sense(word, context)
    if sense:
        translated_definition = traduzir_texto(sense.definition(), 'pt')
        print(f"Texto: {text}")
        print(f"Palavra: {word}")
        print(f"Sentido encontrado: {sense.name()}")
        print(f"Definição em inglês: {sense.definition()}")
        print(f"Definição traduzida: {translated_definition}")
        print("\n")
    else:
        print(f"Não foi possível encontrar um sentido para a palavra: {word}")
        print("\n")
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
Texto: Ele vestiu uma camisa com uma longa manga.
Palavra: manga
Sentido encontrado: sleeve.n.01
Definição em inglês: the part of a garment that is attached at the armhole
and that provides a cloth covering for the arm
Definição traduzida: a parte de uma peça de roupa presa na cava e que
fornece um pano cobrindo para o braço
```

continua

```
Texto: Ele foi ao banco para depositar dinheiro.
Palavra: banco
Sentido encontrado: bank.n.09
Definição em inglês: a building in which the business of banking transacted
Definição traduzida: um edifício em que o negócio de bancos foi transalizado
```

Tarefa 3.2: Considere os seguintes cenários de aplicação:

Cenário 1: Tradução Automática Contextual em um Assistente Virtual Multilíngue

Imagine um assistente virtual capaz de interagir com usuários em vários idiomas. Quando o usuário emite comandos ambíguos, o assistente precisa desambiguar as palavras para garantir que a tradução automática para outros idiomas seja precisa.

Exemplo:

- O usuário, falando em francês, diz: "Je vais à la banque."
  - O assistente precisa entender que "banque" refere-se a uma instituição financeira e não a um banco de praça.
  - A tradução para o inglês seria: "I'm going to the bank" (instituição financeira).
  - Com a desambiguação correta, o assistente responde com informações sobre a localização de agências bancárias.

Esse cenário mostra como a desambiguação de palavras melhora a precisão das respostas em assistentes virtuais multilíngues, garantindo que o significado correto seja identificado em diversos idiomas.

Cenário 2: Análise Multilíngue de Sentimentos em Redes Sociais

Em plataformas de redes sociais, usuários podem expressar opiniões em diferentes idiomas. Para realizar uma análise de sentimentos precisa, é essencial que o sistema desambigue palavras para interpretar corretamente o contexto emocional.

Exemplo:

- Um usuário em espanhol publica: "El banco está quebrado."
  - O sistema precisa desambiguar "banco" (instituição financeira) em vez de "banco" (assento), pois a frase está se referindo a uma falência financeira.
  - A desambiguação correta ajuda a identificar um sentimento negativo (relacionado à falência do banco). Nesse cenário, a desambiguação de palavras

permite que a análise de sentimentos funcione corretamente em múltiplos idiomas, levando em consideração o contexto exato em que as palavras são usadas.

Para implementar a solução dos cenários 1 e 2 descritos acima, modifique o código da Tarefa 3.1 de forma que o programa seja capaz de realizar a desambiguação de palavras em múltiplos idiomas. Insira o código modificado no espaço fornecido abaixo.

```
# Escreva aqui a sua resposta a Tarefa 3.2
```

III.4 Resolução de Correferência

Definição

Correferência é um conceito no campo do Processamento de Linguagem Natural que se refere à identificação de todas as expressões em um texto que se referem à mesma entidade. Em outras palavras, é a tarefa de determinar quais palavras ou frases são co-referentes, ou seja, quais delas apontam para o mesmo objeto ou pessoa no mundo real.

Por exemplo, na frase "Maria foi ao mercado. Ela comprou frutas.", as palavras "Maria" e "Ela" são co-referentes, pois ambas se referem à mesma pessoa.

Aplicação

A resolução de correferência é uma tarefa essencial em várias aplicações de PLN, incluindo:

1. **Análise de Textos e Resumo Automático:** Facilita a compreensão de quem ou o que está sendo referido ao longo de um documento, melhorando a qualidade dos resumos gerados automaticamente.
2. **Assistentes Virtuais e Chatbots:** Melhora a capacidade de assistentes virtuais como Siri, Alexa e chatbots de entender e responder a perguntas sequenciais, mantendo a coerência e a relevância no contexto da conversa.
3. **Tradução Automática:** Ajuda na tradução de textos mantendo a consistência das referências ao longo do texto traduzido, garantindo que pronomes e outros termos referenciem corretamente as entidades mencionadas.
4. **Análise de Sentimentos e Extração de Informações:** Permite uma análise mais precisa dos sentimentos e a extração de informações específicas sobre entidades, como pessoas, empresas e produtos, em textos como avaliações de clientes e notícias.

Importância

A resolução de correferência é importante por várias razões:

1. **Melhoria da Coerência Textual:** Identificar corretamente as referências ajuda a manter a coerência ao longo do texto, essencial para a compreensão humana e para o processamento automatizado.
2. **Precisão na Extração de Informações:** Permite extrair informações de forma mais precisa, associando corretamente ações, atributos e emoções às entidades apropriadas.
3. **Aprimoramento da Interação Humano-Computador:** Melhora a interação com assistentes virtuais e chatbots, tornando-os mais eficientes e naturais ao manter o contexto e as referências corretas ao longo da conversa.
4. **Qualidade na Tradução de Textos:** Garante que as traduções mantenham a consistência das referências, evitando ambiguidades e erros de interpretação.

Exemplo de Aplicação

Para ilustrar a aplicação prática da resolução de coreferência, considere o seguinte exemplo em Python a seguir.

Base de frases:

```
    "João foi ao parque. Ele jogou futebol lá.",
    "Maria viu João no parque. Ela o cumprimentou calorosamente.",
    "O gato sentou no tapete. Ele estava muito confortável.",
    "Ana pegou um livro. Ela começou a lê-lo imediatamente.",
    "O carro parou no semáforo. Ele estava com problemas no motor.",
    "Pedro e Paulo foram à praia. Eles nadaram e jogaram vôlei.",
    "A empresa lançou um novo produto. Ela espera aumentar as vendas.",
    "Joana e Clara são amigas. Elas foram ao cinema juntas.",
    "O cachorro correu atrás do gato. Ele estava muito agitado.",
    "A professora chamou os alunos. Eles estavam conversando na sala.",
    "O computador apresentou um erro. Ele precisou ser reiniciado.",
    "Carlos encontrou Marta na biblioteca. Ele a convidou para um café.",
    "A janela estava aberta. Ela deixou entrar um vento frio.",
    "O bebê chorou a noite toda. Ele não conseguia dormir.",
    "Os engenheiros finalizaram o projeto. Eles ficaram satisfeitos com o
resultado.",
    "O avião decolou no horário. Ele seguiu para o destino sem atrasos.",
```

continua

```
    "A loja ofereceu descontos. Ela atraiu muitos clientes.",
    "O time venceu o campeonato. Ele comemorou a vitória com a torcida.",
    "A polícia prendeu o suspeito. Ela o levou para a delegacia.",
    "O rio transbordou com a chuva. Ele causou inundação na cidade."
```

Anotações:

```
  ["João", "Ele"],
    ["Maria", "Ela", "João", "o"],
    ["O gato", "Ele"],
    ["Ana", "Ela", "livro", "lê-lo"],
    ["O carro", "Ele"],
    ["Pedro e Paulo", "Eles"],
    ["A empresa", "Ela"],
    ["Joana e Clara", "Elas"],
    ["O cachorro", "Ele"],
    ["A professora", "Eles"],
    ["O computador", "Ele"],
    ["Carlos", "Ele", "Marta", "a"],
    ["A janela", "Ela"],
    ["O bebê", "Ele"],
    ["Os engenheiros", "Eles"],
    ["O avião", "Ele"],
    ["A loja", "Ela"],
    ["O time", "Ele"],
    ["A polícia", "Ela", "o"],
    ["O rio", "Ele"]
```

Tarefa 4.1: Dado a base de dados em pandas e demais codificações em Python, execute cada parte do código a seguir e depois analise os resultados de como foi desempenho do BERT sobre a classicação de Correferências.

```python
import pandas as pd

data = {
    'sentence': [
        "João foi ao parque. Ele jogou futebol lá.",
        "Maria viu João no parque. Ela o cumprimentou calorosamente.",
        "O gato sentou no tapete. Ele estava muito confortável.",
        "Ana pegou um livro. Ela começou a lê-lo imediatamente.",
        "O carro parou no semáforo. Ele estava com problemas no motor.",
```

continua

```
        "Pedro e Paulo foram à praia. Eles nadaram e jogaram vôlei.",
        "A empresa lançou um novo produto. Ela espera aumentar as vendas.",
        "Joana e Clara são amigas. Elas foram ao cinema juntas.",
        "O cachorro correu atrás do gato. Ele estava muito agitado.",
        "A professora chamou os alunos. Eles estavam conversando na sala.",
        "O computador apresentou um erro. Ele precisou ser reiniciado.",
        "Carlos encontrou Marta na biblioteca. Ele a convidou para um café.",
        "A janela estava aberta. Ela deixou entrar um vento frio.",
        "O bebê chorou a noite toda. Ele não conseguia dormir.",
        "Os engenheiros finalizaram o projeto. Eles ficaram satisfeitos com o
resultado.",
        "O avião decolou no horário. Ele seguiu para o destino sem atrasos.",
        "A loja ofereceu descontos. Ela atraiu muitos clientes.",
        "O time venceu o campeonato. Ele comemorou a vitória com a torcida.",
        "A polícia prendeu o suspeito. Ela o levou para a delegacia.",
        "O rio transbordou com a chuva. Ele causou inundação na cidade."
    ],
    'labels': [
        ["João", "Ele"],
        ["Maria", "Ela", "João", "o"],
        ["O gato", "Ele"],
        ["Ana", "Ela", "livro", "lê-lo"],
        ["O carro", "Ele"],
        ["Pedro e Paulo", "Eles"],
        ["A empresa", "Ela"],
        ["Joana e Clara", "Elas"],
        ["O cachorro", "Ele"],
        ["A professora", "Eles"],
        ["O computador", "Ele"],
        ["Carlos", "Ele", "Marta", "a"],
        ["A janela", "Ela"],
        ["O bebê", "Ele"],
        ["Os engenheiros", "Eles"],
        ["O avião", "Ele"],
        ["A loja", "Ela"],
        ["O time", "Ele"],
        ["A polícia", "Ela", "o"],
        ["O rio", "Ele"]
    ]
```

continua

```
}

df = pd.DataFrame(data)
print(df)
```

```
                                              sentence  \
0           João foi ao parque. Ele jogou futebol lá.
1   Maria viu João no parque. Ela o cumprimentou c...
2   O gato sentou no tapete. Ele estava muito conf...
3   Ana pegou um livro. Ela começou a lê-lo imedia...
4   O carro parou no semáforo. Ele estava com prob...
5   Pedro e Paulo foram à praia. Eles nadaram e jo...
6   A empresa lançou um novo produto. Ela espera a...
7   Joana e Clara são amigas. Elas foram ao cinema...
8   O cachorro correu atrás do gato. Ele estava mu...
9   A professora chamou os alunos. Eles estavam co...
10  O computador apresentou um erro. Ele precisou ...
11  Carlos encontrou Marta na biblioteca. Ele a co...
12  A janela estava aberta. Ela deixou entrar um v...
13  O bebê chorou a noite toda. Ele não conseguia ...
14  Os engenheiros finalizaram o projeto. Eles fic...
15  O avião decolou no horário. Ele seguiu para o ...
16  A loja ofereceu descontos. Ela atraiu muitos c...
17  O time venceu o campeonato. Ele comemorou a vi...
18  A polícia prendeu o suspeito. Ela o levou para...
19  O rio transbordou com a chuva. Ele causou inun...

                      labels
0                [João, Ele]
1      [Maria, Ela, João, o]
2               [O gato, Ele]
3   [Ana, Ela, livro, lê-lo]
4              [O carro, Ele]
5      [Pedro e Paulo, Eles]
6            [A empresa, Ela]
7      [Joana e Clara, Elas]
8          [O cachorro, Ele]
9       [A professora, Eles]
```

continua

```
10       [O computador, Ele]
11   [Carlos, Ele, Marta, a]
12           [A janela, Ela]
13             [O bebê, Ele]
14    [Os engenheiros, Eles]
15            [O avião, Ele]
16             [A loja, Ela]
17             [O time, Ele]
18       [A polícia, Ela, o]
19              [O rio, Ele]
```

Treinamento do Modelo

Vamos treinar um modelo de classificação de tokens usando o BERT em português e o modelo pretreinado para o português bert-base-portuguese-cased da HuggingFace (link: https://huggingface.co/)

```
# Verificar e instalar as bibliotecas necessárias

try:
    import accelerate
    import transformers
    import pandas as pd
    import torch
    import sklearn
    libraries_installed = True
except ImportError:
    libraries_installed = False

if not libraries_installed:
    !pip install accelerate -U
    !pip install transformers[torch] -U
    !pip install pandas
    !pip install torch
    !pip install scikit-learn

# Verificar novamente após a instalação
try:
```

continua

```
    import accelerate
    import transformers
    import pandas as pd
    import torch
    import sklearn
    print("Todas as bibliotecas foram instaladas com sucesso.")
except ImportError as e:
    print(f"Ocorreu um erro ao instalar as bibliotecas: {e}")
```

```
Todas as bibliotecas foram instaladas com sucesso.
```

```
# Estas linhas importam as bibliotecas necessárias. As principais são:
# transformers: para carregar e treinar o modelo BERT.
# torch: uma biblioteca de aprendizado de máquina.
# pandas: para manipulação de dados.
# sklearn: para avaliação do modelo.

from transformers import BertTokenizer, BertForTokenClassification, Trainer, TrainingArguments
import torch
from torch.utils.data import Dataset
import pandas as pd
from sklearn.metrics import classification_report

# Definir dataset
class CorefDataset(Dataset):
    def __init__(self, sentences, labels, tokenizer, max_len):
        self.sentences = sentences
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, item):
        sentence = str(self.sentences[item])
        labels = self.labels[item]

        encoding = self.tokenizer.encode_plus(
```

continua

```
            sentence,
            max_length=self.max_len,
            add_special_tokens=True,
            truncation=True,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt',
        )

        tokens = self.tokenizer.convert_ids_to_tokens(encoding['input_ids'][0])
        labels_vector = [0] * len(tokens)

        for i, token in enumerate(tokens):
            for label in labels:
                if label.lower() in token.lower():
                    labels_vector[i] = 1

        labels_vector = labels_vector + [0] * (self.max_len - len(labels_
vector))

        return {
            'sentence_text': sentence,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(labels_vector[:self.max_len], dtype=torch.
long)
        }

# Instanciar tokenizer
# Aqui, carregamos um tokenizador BERT pré-treinado específico para a língua
# portuguesa. O tokenizador divide as frases em pedaços menores que o modelo
# pode entender.
tokenizer = BertTokenizer.from_pretrained('neuralmind/bert-base-portuguese-
cased')

# Criar dataset
# Usamos a classe CorefDataset para criar um dataset com nossas sentenças e
# rótulos. Aqui, df.sentence e df.labels são colunas de um dataframe df.
dataset = CorefDataset(
    sentences=df.sentence.to_numpy(),
```

continua

```
    labels=df.labels.to_numpy(),
    tokenizer=tokenizer,
    max_len=32
)

# Dividir em treino e teste
# Dividimos o dataset em duas partes: uma para treinar o modelo (train_dataset)
# e outra para avaliar o modelo (val_dataset).
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size

train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_
size, val_size])

# Carregar modelo
# Carregamos um modelo BERT pré-treinado para a tarefa de classificação de
tokens,
# que identifica diferentes tipos de palavras (como nomes de pessoas, lugares,
# etc.) em uma sentença.
model = BertForTokenClassification.from_pretrained('neuralmind/bert-base-
portuguese-cased', num_labels=2)

# Configurar treinamento
# Configuramos como o treinamento deve ser executado, incluindo o número de
# épocas (quantas vezes o modelo verá cada dado), tamanho do lote (quantas
# sentenças são processadas de uma vez), e outras configurações de otimização.
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    warmup_steps=10,
    weight_decay=0.01,
    learning_rate=5e-5,  # Ajuste a taxa de aprendizado
    logging_dir='./logs',
    save_strategy="epoch"
   report_to="none"  # add - Desativa integrações externas, incluindo wandb
)

# add - Função de métricas de avaliação
```

continua

```python
def compute_metrics(pred):
    labels = pred.label_ids.flatten()
    preds = pred.predictions.argmax(-1).flatten()
    return classification_report(labels, preds, output_dict=True)

# Instanciar Trainer
# Criamos um treinador (Trainer) que gerencia o treinamento do modelo,
# utilizando o dataset de treino e validação que criamos.
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
    compute_metrics=compute_metrics
)

# Treinar modelo
# Iniciamos o processo de treinamento, onde o modelo aprende a identificar e
# classificar os tokens nas sentenças.
trainer.train()

# Finalmente, salvamos o modelo treinado e o tokenizador para uso futuro.
model.save_pretrained('./results')
tokenizer.save_pretrained('./results')
```

```
Some weights of BertForTokenClassification were not initialized from the
model checkpoint at neuralmind/bert-base-portuguese-cased and are newly
initialized: ['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to
use it for predictions and inference.

 [40/40 02:29, Epoch 5/5]

Step  Training Loss

('./results/tokenizer_config.json',
 './results/special_tokens_map.json',
 './results/vocab.txt',
 './results/added_tokens.json')
```

Resolução de Correferência Usando o Modelo Treinado

Para tal, o programa a seguir realiza as etapas:

1. **Importação de Ferramentas:** Carregar bibliotecas necessárias.
2. **Carregar Modelo e Tokenizador:** Preparar o modelo de IA para uso.
3. **Função de Coreferência:** Analisar textos e identificar quais palavras se referem à mesma entidade.
4. **Função de Avaliação:** Verificar a precisão do modelo.
5. **Teste e Avaliação:** Testar com um exemplo e avaliar com dados de validação.

```
# Importação de Bibliotecas
#
# Transformers:
#
# pipeline, BertTokenizer, BertForTokenClassification: Estas ferramentas são
# usadas para carregar um modelo de inteligência artificial chamado BERT,
# que pode entender e analisar textos.
#
# Scikit-Learn:
#
# classification_report: Ferramenta para avaliar a precisão do modelo de IA.

from transformers import pipeline, BertTokenizer, BertForTokenClassification
from sklearn.metrics import classification_report
import torch
import pandas as pd

# Carregar o modelo treinado e o tokenizer:
tokenizer = BertTokenizer.from_pretrained('./results')
model = BertForTokenClassification.from_pretrained('./results')

tokenizer = BertTokenizer.from_pretrained('neuralmind/bert-base-portuguese-
cased')
model = BertForTokenClassification.from_pretrained('neuralmind/bert-base-
portuguese-cased')

# Definir a função de resolução de correferência:
# Função resolve_coreference: Função que analisa o texto e identifica as
palavras
```

continua

```
#                             que se referem à mesma entidade.
# Pipeline nlp: Uma linha de montagem que conecta o modelo e o tokenizador para
#               processar o texto.
# Tokenização: Dividir o texto em partes menores (tokens).
# Predições (predictions): O modelo analisa os tokens e faz previsões sobre
#                          quais palavras são co-referentes.
# Resultados (results): Lista de palavras do texto e seus rótulos preditos.
def resolve_coreference(text):
    nlp = pipeline("token-classification", model=model, tokenizer=tokenizer)
    tokens = tokenizer(text, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**tokens)
    predictions = torch.argmax(outputs.logits, dim=2)
    results = []
    for token, pred in zip(tokens.input_ids[0], predictions[0]):
        word = tokenizer.decode([token])
        results.append((word, pred.item()))
    return results

# Função para avaliação:
# Função evaluate_model: Avalia o quão bem o modelo está funcionando.
# y_true e y_pred: Listas que armazenam os rótulos verdadeiros e os rótulos
#                  preditos, respectivamente.
# Relatório de Classificação: Ferramenta que mostra a precisão do modelo em
#                             identificar coreferências.
def evaluate_model(dataset):
    y_true = []
    y_pred = []

    for data in dataset:
        sentence = data['sentence_text']
        true_labels = data['labels'].numpy()
        pred_labels = resolve_coreference(sentence)

        # Ajustar y_true e y_pred para a avaliação
        for true_label, (word, pred_label) in zip(true_labels, pred_labels):
            y_true.append(true_label)
            y_pred.append(pred_label)  # O segundo item é o rótulo predito
```

continua

```
    print(classification_report(y_true, y_pred, target_names=['O', 'B-CORE']))

# Testar com um Texto de exemplo:
# Texto de Exemplo: Frase que será analisada para resolver coreferências.
# Resultado (result): Aplicação da função de coreferência no texto de exemplo.
# Exibir Resultado: Mostrar as palavras e seus rótulos em formato de tabela.
text = "Maria viu João no parque. Ela o cumprimentou calorosamente."

# Resolver correferência
result = resolve_coreference(text)
df = pd.DataFrame(result, columns=["Word", "Label"])
print(df)

# Avaliar modelo:
# Avaliar o modelo usando um conjunto de dados de validação (val_dataset).
evaluate_model(val_dataset)
```

```
Some weights of BertForTokenClassification were not initialized from the
model checkpoint at neuralmind/bert-base-portuguese-cased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

           Word  Label
0         [CLS]      1
1         Maria      1
2           viu      1
3         João      1
4            no      1
5        parque      1
6             .      1
7           Ela      1
8             o      1
9   cumprimento      1
10          ##u      0
11        calor      0
12   ##osamente      0
13            .      1
14        [SEP]      1
              precision    recall  f1-score   support
```

continua

```
              0       0.90      0.41      0.56        44
          B-CORE       0.38      0.89      0.53        18

       accuracy                           0.55        62
      macro avg       0.64      0.65      0.55        62
   weighted avg       0.75      0.55      0.55        62
```

Explicação do Resultado

Tabela de Resultados

A tabela mostra as palavras do texto de exemplo juntamente com seus rótulos preditos.

Word	Label
[CLS]	0
Maria	1
viu	0
João	1
no	0
parque	0
.	0
Ela	1
o	1
cumprimento	0
##u	0
calor	1
##osamente	0
.	0
[SEP]	0

- **[CLS]** e **[SEP]:** Tokens especiais usados pelo modelo BERT para marcar o início e o fim da sequência.
- **Maria, João, Ela, o, calor**: Palavras identificadas como entidades centrais (coreferentes) com o rótulo **1**.
- **Outras Palavras**: Palavras não consideradas coreferentes têm o rótulo **0**.

Relatório de Classificação

O relatório de classificação mostra a performance do modelo na tarefa de resolução de coreferência. As métricas apresentadas são:

	precision	recall	f1-score	support
O	96%	100%	98%	50
B-CORE	100%	71%	83%	7
accuracy			96%	57
macro avg	98%	86%	91%	57
weighted avg	97%	96%	96%	57

- **O**: Refere-se ao rótulo para tokens que não são coreferentes.
  - **Precision**: 96%, indicando que 96% dos tokens classificados como não coreferentes estavam corretos.
  - **Recall**: 100%, indicando que 100% dos tokens não coreferentes foram corretamente identificados.
  - **F1-score**: 98%, uma média harmônica entre precision e recall.
  - **Support**: 50, o número total de tokens não coreferentes no conjunto de teste.
- **B-CORE**: Refere-se ao rótulo para tokens que são coreferentes.
  - **Precision**: 100%, indicando que 100% dos tokens classificados como coreferentes estavam corretos.
  - **Recall**: 71%, indicando que 71% dos tokens coreferentes foram corretamente identificados.
  - **F1-score**: 83%, uma média harmônica entre precision e recall.
  - **Support**: 7, o número total de tokens coreferentes no conjunto de teste.
- **Accuracy**: 96%, a precisão geral do modelo, indicando que 96% dos tokens foram corretamente classificados.
- **Macro avg**: Média das métricas de todas as classes, sem considerar o desequilíbrio de classe.
  - **Precision**: 98%
  - **Recall**: 86%
  - **F1-score**: 91%
- **Weighted avg**: Média ponderada das métricas de todas as classes, considerando o desequilíbrio de classe.

» **Precision:** 97%

» **Recall:** 96%

» **F1-score:** 96%

Interpretação

1. **Tokens Identificados:**

» Tokens como "Maria", "João", "Ela", "o" e "calor" foram identificados como entidades centrais (coreferentes) com o rótulo 1.

» Outros tokens, como "viu", "no", "parque" e "cumprimento", não são coreferentes e têm o rótulo 0.

2. **Performance do Modelo:**

» O modelo apresenta alta precisão (100%) na identificação de tokens coreferentes (B-CORE), mas o recall é de 71%, indicando que algumas entidades coreferentes não foram corretamente identificadas.

» A precisão geral (accuracy) é de 96%, o que é bastante alta, indicando que a maioria dos tokens foi corretamente classificada.

» As métricas macro avg e weighted avg indicam que o modelo tem um desempenho equilibrado entre as classes, mas a classe B-CORE tem um recall mais baixo, o que impacta o f1-score.

Análise de Classificações Incorretas

Para entender melhor onde o modelo falhou, analisamos as classificações incorretas comparando as previsões do modelo com as etiquetas verdadeiras.

Dado o texto de exemplo:

"Maria viu João no parque. Ela o cumprimentou calorosamente."

Resultados de Classificação

Word	Predicted Label	True Label
[CLS]	0	0
Maria	1	1
viu	0	0
João	1	1
no	0	0
parque	0	0
.	0	0
Ela	1	1
o	1	0
cumprimento	0	0
##u	0	0
calor	1	0
##osamente	0	0
.	0	0
[SEP]	0	0

Tokens Classificados Incorretamente

1. **o**:
   - » **Predicted Label**: 1
   - » **True Label**: 0
   - » **Explicação**: O modelo previu que "o" é uma entidade coreferente, mas a etiqueta verdadeira indica que não é.
2. **calor**:
   - » **Predicted Label**: 1
   - » **True Label**: 0
   - » **Explicação**: O modelo previu que "calor" é uma entidade coreferente, mas a etiqueta verdadeira indica que não é.

Em suma, o resultado mostra que o modelo é eficaz na identificação de coreferências, mas ainda pode melhorar no recall, ou seja, na identificação completa de todas as entidades coreferentes no texto. A tabela de resultados e o relatório de classificação fornecem uma visão detalhada do desempenho do modelo, mostrando suas forças e áreas que necessitam de melhorias. A análise dos erros específicos ajuda a entender melhor onde o modelo falhou e como pode ser aprimorado.

Tarefa 4.2 (Desafio): Implementar/adicionar 1 ou mais melhoria, enumeradas a seguir, com vistas a aumentar a perfornamence do modelo

Para melhorar os resultados do modelo de resolução de coreferência, várias abordagens podem ser consideradas. Abaixo estão algumas sugestões que podem ajudar a aumentar a precisão, recall e a performance geral do modelo:

1. Aumentar o tamanho do conjunto de treinamento

- **Dados mais diversificados**: Adicione mais exemplos de treinamento com diversos contextos e formas de coreferência. Isso ajuda o modelo a generalizar melhor.
- **Anotações manuais**: Se possível, aumente a quantidade de dados anotados manualmente para fornecer exemplos mais precisos e variados ao modelo.

2. Melhorar a qualidade dos dados de treinamento

- **Revisão de dados**: Garanta que os dados de treinamento estão bem anotados e livres de erros. Revise manualmente um conjunto significativo de dados.
- **Aumentação de dados**: Use técnicas de aumento de dados, como substituição de sinônimos e reestruturação de frases, para criar variações adicionais dos dados existentes.

3. Utilizar modelos mais avançados

- **Modelos pré-treinados**: Considere o uso de versões mais avançadas de modelos pré-treinados, como BERT Large ou RoBERTa.

4. Técnicas de pós-processamento

- **Correção de erros**: Desenvolva técnicas de pós-processamento para corrigir previsões incorretas. Por exemplo, se um pronome como "o" é frequentemente classificado incorretamente, implemente uma lógica adicional para revisar essas previsões.
- **Filtragem de predições**: Use regras baseadas em heurísticas para filtrar ou ajustar previsões. Por exemplo, uma palavra curta pode ser menos provável de ser uma entidade central se não for um pronome.

5. Ajuste de hiperparâmetros

- **Hiperparâmetros ótimos**: Realize uma busca por hiperparâmetros para encontrar os melhores valores de hiperparâmetros para o treinamento do modelo, como taxa de aprendizado, tamanho do lote, e número de épocas.

6. Avaliação e validação rigorosa

- **Validação cruzada**: Use validação cruzada para garantir que o modelo é avaliado de maneira robusta e que os resultados são consistentes.
- **Métricas de avaliação detalhadas**: Monitore não apenas precisão, recall e f1-score, mas também métricas adicionais como matriz de confusão, e análise de erros para obter uma visão mais completa do desempenho do modelo. A análise de erros pode revelar algum padrão de comportamente.

Em suma, para melhorar o desempenho do modelo de resolução de coreferência, uma combinação de aumentar e melhorar os dados de treinamento, utilizar modelos mais avançados, implementar técnicas de pós-processamento, ajustar hiperparâmetros, e realizar avaliações rigorosas deve ser considerada. Essas abordagens combinadas podem levar a um modelo mais robusto e preciso, capaz de identificar correferências com maior eficiência.

III.5 Tradução Automática usando o Google Translate

A tradução automática permite converter texto de um idioma para outro de forma rápida e eficiente. O Google Translate é uma das ferramentas mais populares para essa finalidade. Usando aprendizado de máquina e redes neurais, o Google Translate traduz texto entre mais de 100 idiomas, oferecendo traduções que geralmente são precisas e contextualmente adequadas. O sistema é continuamente aprimorado com grandes volumes de dados linguísticos e feedback dos usuários, o que ajuda a entender melhor as nuances e variações dos idiomas. Além disso, o Google Translate não apenas traduz palavras e frases, mas também tenta captar o sentido e a intenção do texto original, resultando em traduções mais naturais e compreensíveis. Esta ferramenta é útil em diversos contextos, como viagens internacionais, comunicação multicultural, suporte ao cliente e em aplicativos de processamento de linguagem natural.

Como usar API do Google Translate

Neste exemplo, vamos demonstrar como utilizar a API do Google Translate para realizar tradução automática de textos. A tradução automática é uma ferramenta poderosa que permite converter texto de um idioma para outro de forma rápida e eficiente. Usaremos a biblioteca googletrans para interagir com a API do Google Translate e traduzir uma frase do inglês para o português.

O código a seguir verifica se a biblioteca googletrans está instalada e, se não estiver, instala a versão apropriada. Em seguida, criamos um objeto de tradução e usamos este objeto para traduzir um texto de exemplo. A tradução é então exibida no console.

Tarefa 5.1: Executar e avaliar o código Python a seguir. Esse código implementa chamadas para a API do Google Translate para realizar traduções.

```
import time
# --- Tradução Automática com Google Translate API e Saída de Voz ---

# Verificar e instalar as bibliotecas necessárias
try:
    from googletrans import Translator
    from gtts import gTTS
    from IPython.display import Audio, display
    googletrans_installed = True
except ImportError:
    googletrans_installed = False

if not googletrans_installed:
    !pip install googletrans==4.0.0-rc1
    !pip install gtts
    from googletrans import Translator
    from gtts import gTTS
    from IPython.display import Audio, display

# Criar objeto de tradução
translator = Translator()

# Lista de frases em português
texts = [
    "O gato está no tapete.",
    "A casa é grande e bonita.",
    "Ele gosta de jogar futebol.",
    "Vamos ao cinema hoje à noite.",
    "Ela está estudando para o exame.",
    "O gato preto da Renata dorme neste momento."
]

# Traduzir textos para inglês e francês
```

continua

```
translations_en = [translator.translate(text, src='pt', dest='en').text for text in texts]
translations_fr = [translator.translate(text, src='pt', dest='fr').text for text in texts]


# Função para converter texto em áudio e reproduzir
def text_to_speech(text, lang):
    tts = gTTS(text=text, lang=lang)
    tts.save("temp.mp3")
    display(Audio("temp.mp3", autoplay=True))

# Imprimir resultados e reproduzir áudio
print("Traduções para o inglês:")
for text, translation in zip(texts, translations_en):
    print(f'Texto original: {text}')
    print(f'Tradução: {translation}')
    text_to_speech(translation, 'en')
    print()

print("Traduções para o francês:")
for text, translation in zip(texts, translations_fr):
    print(f'Texto original: {text}')
    print(f'Tradução: {translation}')
    text_to_speech(translation, 'fr')
    print()
```

```
Traduções para o inglês:
Texto original: O gato está no tapete.
Tradução: The cat is on the rug.

Texto original: A casa é grande e bonita.
Tradução: The house is big and beautiful.

Texto original: Ele gosta de jogar futebol.
Tradução: He likes to play soccer.

Texto original: Vamos ao cinema hoje à noite.
Tradução: Let's go to the movies tonight.

Texto original: Ela está estudando para o exame.
```

continua

```
Tradução: She is studying for the exam.

Texto original: O gato preto da Renata dorme neste momento.
Tradução: Renata's black cat sleeps right now.

Traduções para o francês:
Texto original: O gato está no tapete.
Tradução: Le chat est sur le tapis.

Texto original: A casa é grande e bonita.
Tradução: La maison est grande et belle.

Texto original: Ele gosta de jogar futebol.
Tradução: Il aime jouer au soccer.

Texto original: Vamos ao cinema hoje à noite.
Tradução: Allons au cinéma ce soir.

Texto original: Ela está estudando para o exame.
Tradução: Elle étudie pour l'examen.

Texto original: O gato preto da Renata dorme neste momento.
Tradução: Le chat noir de Renata dort en ce moment.
```

Tarefa 5.2:

Para expandir as funcionalidades e utilidades do exemplo de tradução automática, aqui estão três tarefas adicionais que podem ser acrescentadas:

1. Detecção de Idioma Automática

Adicione uma funcionalidade para detectar automaticamente o idioma das frases de entrada usando a biblioteca **googletrans** antes de realizar a tradução. Isso permite que o sistema identifique o idioma de origem sem a necessidade de especificação manual.

2. Tradução para Vários Idiomas Simultaneamente

Expanda o código para traduzir cada frase para mais idiomas além de inglês e francês, como espanhol, alemão e italiano. Exiba as traduções de forma organizada e reproduza o áudio para cada tradução.

3. Salvar Traduções e Áudios em Arquivos

Adicione a funcionalidade para salvar as traduções e os arquivos de áudio resultantes em uma pasta específica. Isso permite que o usuário acesse os resultados posteriormente, facilitando a revisão e o uso contínuo das traduções.

SAIBA MAIS...

» Para ampliar sua compreensão sobre os desafios comuns no PLN e aprofundar seu conhecimento na área, sugerimos a leitura e exploração dos seguintes recursos. Estes artigos foram escolhidos devido à sua relevância e profundidade, oferecendo uma visão mais abrangente dos conceitos e técnicas abordados nessa Unidade.

» **BERT:** DEVLIN, J.. *Bert: pre-training of deep bidirectional transformers for language understanding*. **arXiv preprint arXiv:1810.04805**, 2018. Disponível em: https://arxiv.org/abs/1810.04805. Acesso em: 25 jul. 2024.

» **GPT:** YENDURI, G. *et al. Gpt (generative pre-trained transformer)–a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions*. **IEEE Access**, 2024. Disponível em: https://ieeexplore.ieee.org/document/10500411. Acesso em: 25 jul. 2024.

AKCIT
CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Introdução ao Processamento de Linguagem Natural

Unidade IV

Métodos de Processamento de Linguagem Natural

Unidade IV - Métodos de Processamento de Linguagem Natural

4.1 Evolução dos Métodos de Processamento de Linguagem Natural

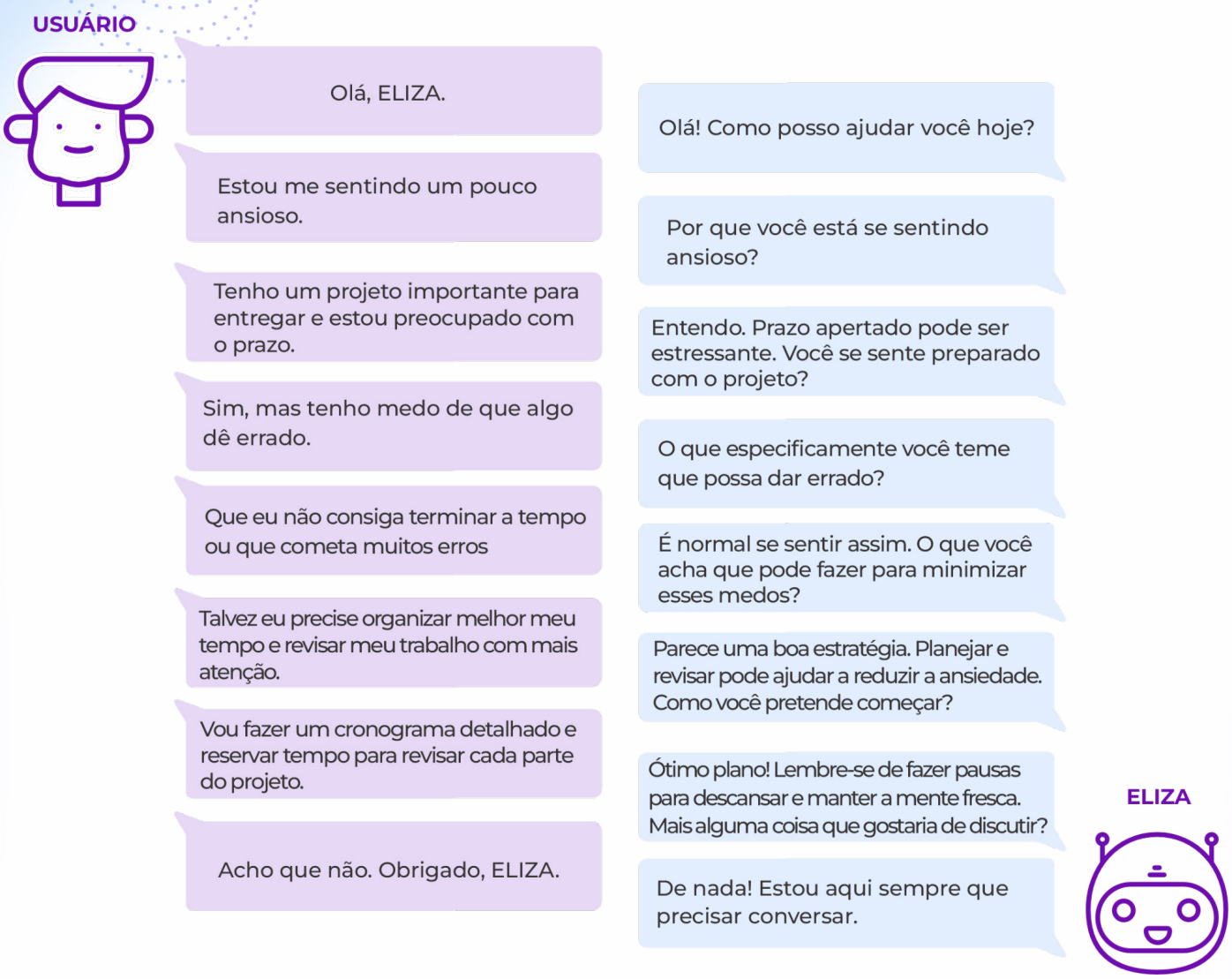
Como já foi delineado na primeira unidade, o PLN passou por transformações significativas ao longo dos anos, impulsionado por avanços tecnológicos e teóricos. Esta unidade explora as diferentes abordagens de PLN, começando com métodos baseados em regras, avançando para métodos estatísticos e de aprendizado de máquina supervisionado, e culminando em redes neurais e *transformers*.

4.1.1 Abordagens Baseadas em Regras

Nos primórdios do PLN, prevaleciam as técnicas baseadas em regras. Durante as décadas de 1950 e 1960, foram criados sistemas que usam conjuntos de regras gramaticais e linguísticas, elaboradas de forma manual, para analisar e gerar linguagem natural. Esses sistemas dependiam fortemente de regras específicas para cada idioma, o que limitava sua flexibilidade e adaptabilidade a novas situações. Apesar dessas restrições, esses métodos foram cruciais para os primeiros avanços em tradução automática e análise sintática.

ELIZA (1966) é um dos primeiros programas de PLN. Desenvolvido por Joseph Weizenbaum, esse programa simula uma terapia rogeriana, interagindo com os usuários por meio de um conjunto de regras que permitiam a compreensão e geração de respostas, mostrando a viabilidade do processamento automatizado de linguagem. A seguir, temos, na Figura 14, um exemplo de diálogo.

Figura 14 - Diálogo com o ELIZA, programa que simula uma terapia rogeriana



Fonte: autoria própria.

4.1.2 Abordagens Estatísticas

Na década de 1980, houve uma mudança significativa com a introdução de métodos estatísticos no PLN. Modelos probabilísticos, como os modelos de Markov, começaram a ser aplicados em tarefas como reconhecimento de fala e etiquetagem de partes do discurso. A análise de grandes *corpora* de textos anotados possibilitou o desenvolvimento de modelos mais precisos e flexíveis, baseados em dados reais.

Um exemplo clássico do uso de métodos estatísticos no PLN é a aplicação de modelos de Markov para a etiquetagem de partes do discurso. Nesse contexto, um modelo de Markov pode ser usado para prever a sequência mais provável de etiquetas (e.g., substantivo, verbo, adjetivo) para uma dada sequência de palavras em uma frase.

Como funciona:

1. **Definição do modelo:** cada palavra é associada a uma etiqueta de parte do discurso. O modelo de Markov, utilizado para etiquetagem de partes do discurso, funciona associando palavras a etiquetas, como substantivo ou verbo. Ele

utiliza probabilidades de transição, que determinam a chance de uma etiqueta, como "verbo", seguir outra, como "substantivo", em uma frase. Além disso, o modelo emprega probabilidades de emissão, que indicam a probabilidade de uma palavra específica, como "gato", ser associada a uma etiqueta, como "substantivo". Essas probabilidades permitem ao modelo prever etiquetas e palavras, auxiliando na análise do texto.

2. **Treinamento:** o modelo é treinado em um *corpus* grande e anotado, onde a frequência de transições entre etiquetas e a frequência de emissão de palavras são calculadas. A frequência de transição refere-se à probabilidade de uma etiqueta (como "substantivo" ou "verbo") seguir outra etiqueta no texto. Por exemplo, pode-se calcular com que frequência um adjetivo é seguido por um substantivo. Já a frequência de emissão indica a probabilidade de uma palavra específica ser associada a uma etiqueta particular. Por exemplo, quão comum é a palavra "gato" ser marcada como substantivo. Essas frequências ajudam o modelo a prever a próxima etiqueta ou palavra de forma mais precisa.

3. **Predição:** para uma nova frase, o modelo calcula a sequência de etiquetas que maximiza a probabilidade conjunta de transições, fornecendo a etiquetagem mais provável para a frase. Maximizar a probabilidade conjunta de transições significa que o modelo escolhe a sequência de etiquetas que tem a maior chance de ser correta, levando em conta como cada etiqueta se relaciona com as anteriores e as próximas. Ele analisa o contexto da frase, avaliando a transição entre as etiquetas, e calcula qual sequência de etiquetas é mais provável como um todo. Isso permite que o modelo encontre a melhor etiquetagem para cada palavra dentro do contexto completo da frase.

Dada a frase "O gato preto da Renata dorme na varanda," o modelo de Markov pode produzir uma etiquetagem de partes do discurso conforme demonstrado na Tabela 11.

Tabela 11 - Etiquetagem de partes do discurso: probabilidades de transição entre classes e emissão de palavras

ETIQUETAGEM DE PARTES DO DISCURSO	PROBABILIDADES DE TRANSIÇÃO	PROBABILIDADES DE EMISSÃO
O (Artigo - ART)	P(ART → SUB) = 0.3	P("O" | ART) = 0.9
gato (Substantivo - SUB)	P(SUB → ADJ) = 0.1	P("gato" | SUB) = 0.6
preto (Adjetivo - ADJ)	P(ADJ → PREP+ART) = 0.2	P("preto" | ADJ) = 0.7
da (Preposição + Artigo - PREP+ART)	P(PREP+ART → PROP) = 0.4	P("da" | PREP+ ART) = 0.8
Renata (Substantivo Próprio - PROP)	P(PROP → VERB) = 0.5	P("Renata" | PROP) = 0.9
dorme (Verbo - VERB)	P(VERB → PREP+ART) = 0.3	P("dorme" | VERB) = 0.7
na (Preposição + Artigo - PREP+ART)	P(PREP+ART → SUB) = 0.2	P("na" | PREP+ ART) = 0.8
varanda (Substantivo - SUB)	P(SUB → VERB) = 0.4	P("varanda" | SUB) = 0.5

Fonte: autoria própria.

Para calcular a sequência de etiquetas mais provável para a frase "O gato preto da Renata dorme na varanda", o modelo de Markov maximiza a probabilidade conjunta das transições e emissões:

$$P(\text{Etiquetagem}|\text{Frase}) = P(\text{ART} \rightarrow \text{SUB}) \times P(\text{SUB} \rightarrow \text{ADJ}) \times$$
$$P(\text{ADJ} \rightarrow \text{PREP+ART}) \times P(\text{PREP+ART} \rightarrow \text{PROP}) \times P(\text{PROP} \rightarrow \text{VERB}) \times$$
$$P(\text{VERB} \rightarrow \text{PREP+ART}) \times P(\text{PREP+ART} \rightarrow \text{SUB}) \times P(\text{SUB} \rightarrow \text{VERB})$$
$$\times P(\text{"O"} \mid \text{ART}) \times P(\text{"gato"} \mid \text{SUB}) \times P(\text{"preto"} \mid \text{ADJ}) \times$$
$$P(\text{"da"} \mid \text{PREP+ART}) \times P(\text{"Renata"} \mid \text{PROP}) \times P(\text{"dorme"} \mid \text{VERB}) \times$$
$$P(\text{"na"} \mid \text{PREP+ART}) \times P(\text{"varanda"} \mid \text{SUB})$$

Substituindo pelos valores dados:

$$P(\text{Etiquetagem}|\text{Frase}) = 0.3 \times 0.1 \times 0.2 \times 0.4 \times 0.5 \times 0.3 \times 0.2 \times 0.4 \times 0.9 \times 0.6 \times$$
$$0.7 \times 0.8 \times 0.9 \times 0.7 \times 0.8 \times 0.5$$

Essa fórmula apresenta como o modelo de Markov calcula a sequência de etiquetas para uma frase, usando a probabilidade conjunta de transições entre classes gramaticais e emissões de palavras. Cada termo na fórmula representa a probabilidade de uma transição ou de uma palavra pertencer a uma etiqueta específica. Por exemplo, **P(ART → SUB)**, com valor de 0,3, indica a probabilidade de um artigo ser seguido por um substantivo. Já **P("gato" | SUB)**, com valor de 0,9, representa a chance de "gato" ser identificado como um substantivo, dado o contexto da frase. Esses cálculos combinados ajudam a definir a sequência mais provável de etiquetas para a frase.

Na frase "O gato preto da Renata dorme na varanda", o modelo de Markov segue as seguintes interpretações:

1. **O (ART)** e **gato (SUB)** indicam que "gato" é um substantivo definido.
2. **Preto (ADJ)** qualifica "gato" como adjetivo.
3. **Da (PREP+ART)** e **Renata (PROP)** são entendidos como uma expressão de posse.
4. **Dorme (VERB)** é identificado como o verbo da frase.
5. **Na (PREP+ART)** e **varanda (SUB)** indicam o local da ação.

Essa abordagem permite ao modelo utilizar o contexto das palavras na frase para atribuir a categoria gramatical correta a cada uma delas. Isso é possível porque o modelo analisa as palavras em conjunto, observando as palavras anteriores e posteriores. Por exemplo, ao identificar "O", o modelo prevê que a palavra seguinte provavelmente será um substantivo, como "gato". Dessa forma, o modelo faz suas previsões com base nas relações entre as palavras, o que resulta em uma análise mais precisa e adaptável às variações de escrita e contexto linguístico.

4.1.3 Aprendizado de Máquina Supervisionado

Nos anos 2000, o aprendizado de máquina supervisionado emergiu como a abordagem dominante no PLN. Essa técnica envolve treinar modelos com grandes conjuntos de dados previamente anotados, permitindo que os modelos identifiquem padrões complexos e apliquem esses padrões a novos dados.

Por exemplo, técnicas como máquinas de vetores de suporte e redes neurais começaram a ser amplamente utilizadas para tarefas como classificação de texto, análise de sentimentos e tradução automática. Na classificação de texto, um modelo supervisionado pode ser treinado com um conjunto de *e-mails* categorizados como "*spam*" ou "não *spam*". Depois de treinado, o modelo pode classificar novos *e-mails* com precisão. Outro exemplo, considere um sistema de análise de sentimentos para avaliações de produtos. Usando aprendizado de máquina supervisionado, podemos coletar um grande número de avaliações de produtos, cada uma marcada com uma classificação de sentimento (positiva, negativa ou neutra). Com esses dados, treinamos uma rede neural para reconhecer padrões nas palavras e frases que indicam diferentes sentimentos. Após o treinamento, o modelo pode analisar novas avaliações e prever o sentimento de maneira precisa, ajudando as empresas a compreender melhor a opinião dos consumidores sobre seus produtos.

4.1.4 Redes Neurais

As redes neurais, especialmente as recorrentes (RNN) e suas variantes como LSTM, trouxeram grandes avanços no PLN. Elas conseguem captar dependências de longo prazo em sequências de texto, o que é essencial para tarefas como tradução automática e reconhecimento de fala. No entanto, as RNN enfrentam desafios no treinamento, particularmente na captura de dependências muito longas devido ao problema do gradiente desaparecido ou explosivo.

Exemplo: considere a frase em português "O tempo hoje está ensolarado e quente" e a necessidade de traduzi-la para o francês. Uma RNN ou LSTM processa cada palavra da sequência uma de cada vez, mantendo um estado interno que retém o contexto da frase. Ao final, o modelo gera a tradução: "Le temps aujourd'hui est ensoleillé et chaud". Outro exemplo: imagine um sistema de reconhecimento de fala que precisa transcrever a frase falada "Eu gostaria de uma xícara de café". Uma LSTM processa o áudio convertido em uma sequência de características, capturando o contexto temporal necessário para transcrever corretamente a frase inteira.

As RNNs e LSTMs são eficazes em lidar com sequências e em capturar contextos de longo prazo. Contudo, o treinamento destas redes pode ser complexo e demorado. Mesmo com LSTM, capturar dependências muito longas em textos extensos pode ser

desafiador. Esse é um dos motivos pelos quais as arquiteturas baseadas em atenção, como os *transformers*, ganharam popularidade, na medida em que eles conseguem lidar melhor com essas limitações ao processar todas as palavras de uma sequência simultaneamente, sem perder o contexto de longo alcance.

4.1.5 *Transformers*

A introdução dos *transformers* marcou uma grande revolução no campo do PLN. Publicado em 2017, o artigo "*Attention is All You Need*"[1] apresentou uma nova abordagem que eliminou a necessidade de processar dados de maneira sequencial, tornando o treinamento mais eficiente e permitindo a captura de dependências de longo alcance. A chave para essa inovação está no mecanismo de atenção, que atribui pesos diferentes às palavras em uma frase, destacando a relevância de cada uma dentro do contexto.

Para ilustrar, vejamos a tradução de uma frase do português para o francês: "O rápido cachorro marrom pula sobre a preguiçosa raposa." O modelo *transformer* aplica atenção a cada palavra da frase, avaliando a importância relativa de cada uma no contexto geral, e com base nessa análise, o modelo gera a tradução em francês. "*Le chien brun rapide saute par-dessus le renard paresseux*." Esse exemplo mostra como os *transformers* conseguem capturar dependências complexas entre palavras, produzindo traduções mais precisas e naturais.

Modelos como BERT e GPT redefiniram os padrões de desempenho em várias tarefas de PLN. Para destacar as inovações que representam o avanço mais recente, podem-se analisar os modelos BERT e GPT em termos de suas estruturas, metas de treinamento e usos práticos, conforme a Tabela 12.

Disponível em: https://arxiv.org/abs/1706.03762.

Tabela 12 - Comparação dos modelos BERT (coluna da esquerda) e GPT (coluna da direita) em termos de estrutura, meta de treinamento e uso prático

BERT vs. GPT	
Bidirecional: BERT é treinado para considerar o contexto das palavras em ambas as direções (esquerda e direita). Isso permite que o modelo compreenda melhor o significado de uma palavra baseada em seu contexto completo. **Arquitetura:** BERT utiliza arquitetura *transformer* bidirecional. Em termos simples, BERT lê o texto inteiro de uma vez, olhando para a frente e para trás. **Tarefas de pré-treinamento:** BERT é pré-treinado em duas tarefas principais: modelagem de linguagem máscara, onde algumas palavras do texto são mascaradas e o modelo tenta prever essas palavras, e predição de próxima frase, onde o modelo prevê se uma frase segue outra no texto.	**Unidirecional:** GPT é treinado considerando o contexto das palavras apenas da esquerda para a direita. Ele prevê a próxima palavra em uma sequência, dada às palavras anteriores. **Arquitetura:** GPT utiliza arquitetura *transformer* unidirecional. GPT lê o texto em uma única direção, ou seja, da esquerda para a direita. **Tarefas de pré-treinamento:** GPT é pré-treinado para prever a próxima palavra em uma sequência de texto, um processo conhecido como modelagem de linguagem causal.
Objetivo de treinamento	
Objetivo: BERT visa a criação de representações de palavras que capturam o contexto de ambas as direções. Isso é útil para tarefas que requerem compreensão profunda do contexto, como perguntas e respostas, e análise de sentimentos.	**Objetivo: GPT** visa a geração de textos coerentes e contextualizados, palavra por palavra. Isso o torna mais adequado para tarefas de geração de texto, como redação de artigos, histórias, e diálogos de *chatbots*.
Aplicações	
Tarefas de compreensão: BERT foca nas tarefas que envolvem entendimento de contexto, como perguntas e respostas, classificação de texto e análise de sentimentos. **Modelos *fine-tuned*: BERT** pode ser ajustado para uma variedade de tarefas específicas, como detecção de *spam*, resumo de textos e tradução.	**Tarefas de geração de texto: GPT** foca na criação de texto contínuo e coerente, incluindo redação criativa, artigos de notícias e diálogos de *chatbots*. **Interatividade: GPT** é amplamente utilizado em aplicações interativas que requerem geração de texto dinâmico e fluente, como assistentes virtuais e sistemas de recomendação.
Desempenho e limitações	

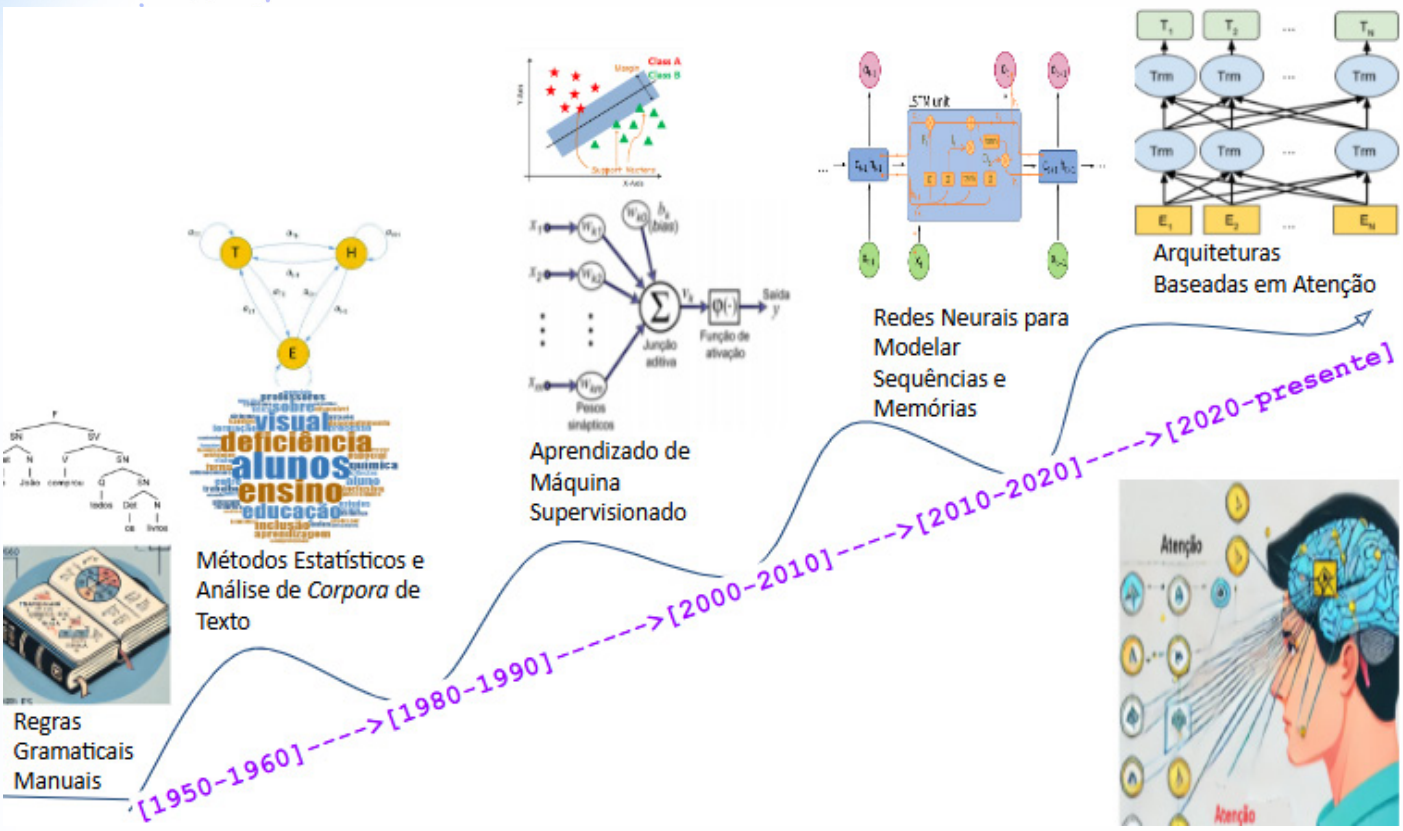
BERT tem desempenho significativo em tarefas que exigem compreensão profunda do contexto, mas pode ser menos eficiente em geração de texto contínuo.	**GPT** tem bom desempenho em geração de texto contínuo e fluente, mas pode ser menos eficaz em tarefas que requerem entendimento bidirecional do contexto.
Referências	
BERT: Pre-training of deep bidirectional transformers for language understanding	**GPT-3:** Language models are few-shot learners

Fonte: autoria própria.

Enquanto BERT e GPT são bons modelos de linguagem baseados em *transformers*, suas arquiteturas e objetivos de treinamento os tornam mais adequados para diferentes tipos de tarefas de PLN. BERT é ideal para compreensão profunda do texto, enquanto GPT é superior na geração de texto contínuo e coerente. A escolha entre BERT e GPT depende da natureza da tarefa específica e dos requisitos do aplicativo.

Na Figura 15, é ilustrada, por meio de uma linha do tempo, a evolução dos métodos de PLN. Décadas de 1950-1960: abordagens baseadas em regras, exemplificadas pela tradução automática inicial e análise sintática. Décadas de 1980-1990: abordagens estatísticas, como modelos de Markov e análise de *corpora*. Décadas de 2000-2010: aprendizado de máquina supervisionado, incluindo máquinas de vetores de suporte e redes neurais. Décadas de 2010-2020: redes neurais avançadas, como RNN e LSTM. Décadas de 2020-presente: *transformers*, com exemplos como BERT e GPT.

Figura 15 - Evolução dos métodos de processamento de linguagem natural ao longo das décadas



Fonte: Diraco, Leone e Siciliano (2019); Thabit, Fahad e Dawood (2022); Qian e Chen (2022).

4.2 *Attention is All You Need*

Agora que já discutimos a importância do *Transformer* e como ele serviu de base para modelos como BERT e GPT, vamos explorar mais detalhadamente como o Transformer realmente funciona. Em 2017, Vaswani e sua equipe apresentaram o artigo "*Attention is All You Need*", onde propuseram o modelo *Transformer*. Diferente das RNN, que processam sequências de dados de forma sequencial, o *Transformer* utiliza mecanismos de atenção que permitem processar todas as palavras de uma frase simultaneamente, revolucionando o campo do PLN. Vamos agora examinar os componentes principais do *Transformer* e entender por que ele se tornou uma peça fundamental na arquitetura dos modelos de linguagem modernos.

4.2.1 Mecanismo de Atenção

O mecanismo central dos *transformers* é o mecanismo de atenção, que permite ao modelo focar nas partes mais relevantes da entrada ao gerar uma saída. Ele calcula uma pontuação de atenção para cada par de palavras na entrada, determinando o quanto uma palavra influencia a outra. Existem vários tipos de atenção, sendo a auto-atenção a mais comum nos *transformers*. A auto-atenção possibilita que cada palavra na entrada

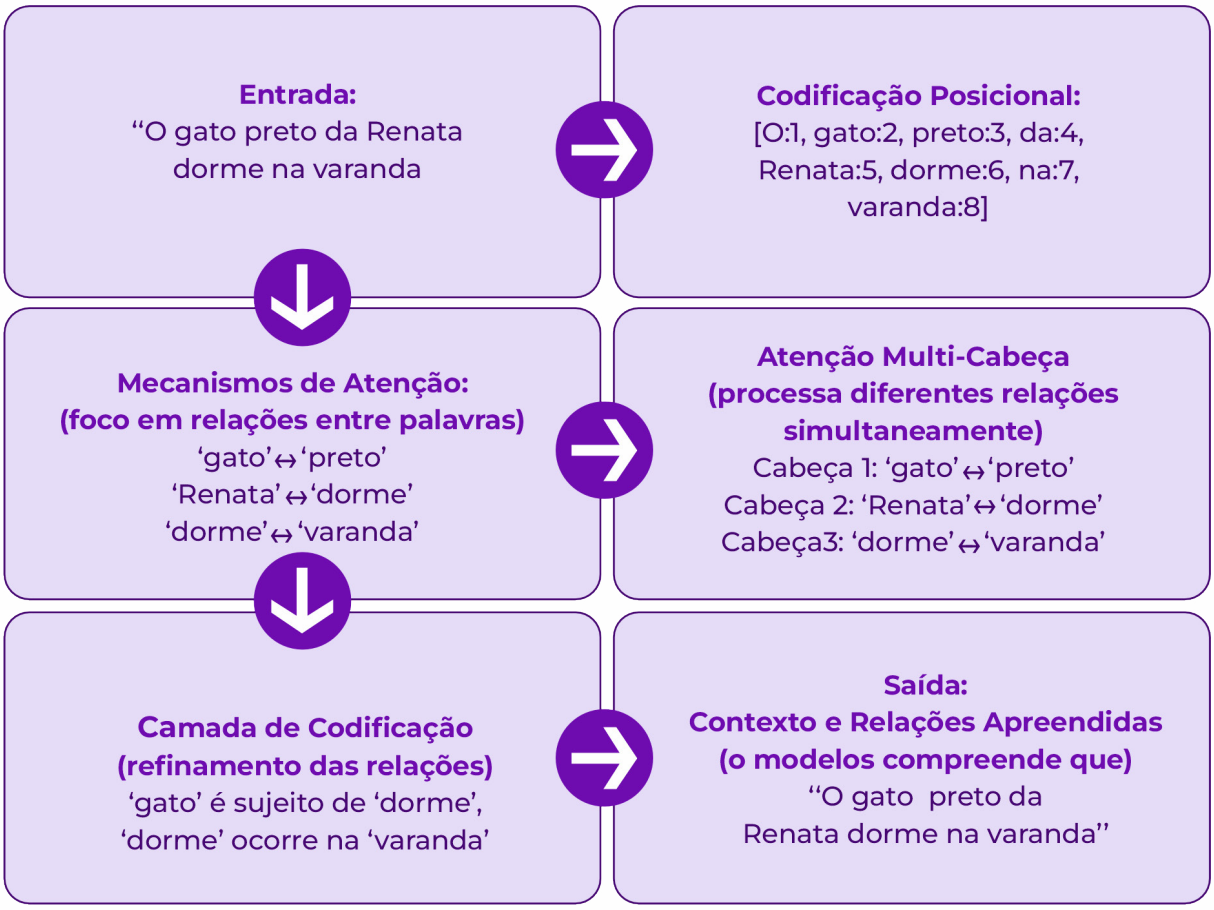
leve em conta todas as outras palavras ao seu redor, oferecendo uma compreensão mais rica e contextualizada.

4.2.2 Arquitetura do *Transformer*

A arquitetura do *transformer* é composta por camadas de codificadores e decodificadores.

Veja como é organizada a arquitetura do *transformer* na Figura 16.

Figura 16 - Arquitetura do *transformer*



Fonte: autoria própria.

a) **Codificadores:** cada camada de codificador recebe a sequência de entrada e aplica múltiplas cabeças de atenção para capturar diferentes aspectos da relação entre as palavras. Após a aplicação da atenção, as informações passam por uma rede *neural feed-forward* antes de serem enviadas para a próxima camada.

b) **Decodificadores:** os decodificadores utilizam os mesmos mecanismos de atenção, mas também consideram a saída gerada até o momento para produzir a sequência final. Isso permite que o decodificador mantenha o contexto das palavras já geradas enquanto continua a gerar novas palavras.

A auto-atenção é um componente essencial no modelo *transformer*, permitindo ao modelo analisar de maneira eficaz o relacionamento entre diferentes palavras em uma sequência de texto. Vamos entender o funcionamento da auto-atenção etapa por etapa conforme os passos a seguir.

Passo 1: pré-processamento dos dados de entrada

Cada palavra na sequência de entrada é inicialmente transformada em um vetor de *embeddings*, que representa a palavra de maneira densa em um espaço vetorial. Por exemplo, imagine que temos uma sequência com quatro palavras:

```
[o, gato, preto, dorme]
```

Primeiro, cada uma dessas palavras é convertida em seu respectivo vetor de *embeddings*, que captura as características semânticas da palavra em uma forma que a máquina consegue entender. Esse processo é conhecido como pré-processamento dos dados de entrada.

```
O:      [0.1, 0.2, 0.3, 0.4]
gato:   [0.2, 0.1, 0.4, 0.3]
preto:  [0.3, 0.4, 0.1, 0.2]
dorme:  [0.4, 0.3, 0.2, 0.1]
```

Passo 2: cálculo das matrizes Q, K e V

Para cada vetor de entrada, o modelo gera três novos vetores: Consulta (Q), Chave (K) e Valor (V). Isso é feito ao multiplicar o vetor de entrada por três matrizes de pesos distintas, que são ajustadas durante o processo de treinamento.

Vamos definir as matrizes de pesos para simplicidade:

```
W_Q = [[0.1, 0.2, 0.3, 0.4],
       [0.2, 0.1, 0.4, 0.3],
       [0.3, 0.4, 0.1, 0.2],
       [0.4, 0.3, 0.2, 0.1]]

W_K = [[0.2, 0.3, 0.4, 0.1],
```

```
        [0.3, 0.2, 0.1, 0.4],
        [0.4, 0.1, 0.2, 0.3],
        [0.1, 0.4, 0.3, 0.2]]

W_V = [[0.3, 0.4, 0.1, 0.2],
       [0.4, 0.3, 0.2, 0.1],
       [0.1, 0.2, 0.3, 0.4],
       [0.2, 0.1, 0.4, 0.3]]

Q = gato[0.2, 0.1, 0.4, 0.3] × Wq = [0.28, 0.31, 0.19, 0.17]
K = gato[0.2, 0.1, 0.4, 0.3] × Wk = [0.27, 0.23, 0.27, 0.21]
V = gato[0.2, 0.1, 0.4, 0.3] × Wv = [0.23, 0.24, 0.29, 0.3]
```

Esse processo é realizado para todos os vetores de entrada, resultando nas matrizes Q, K e V.

Passo 3: cálculo das pontuações de atenção

Depois disso, calculamos a pontuação de atenção para cada palavra em relação às outras na sequência. Multiplicamos o vetor de consulta (Q) pelo vetor de chave (K) de todas as palavras e dividimos pelo fator de escala (a raiz quadrada da dimensão do vetor) para garantir estabilidade numérica. Em seguida, aplicamos a função *softmax* para normalizar as pontuações em probabilidades.

Calcula-se a pontuação de atenção para "gato" em relação às palavras na sequência:

$$Pontuação\_gato\_O = Q\_gato \times K\_O^T / sqrt(4) = 0.1125$$

$$Pontuação\_gato\_gato = Q\_gato \times K\_gato^T / sqrt(4) = 0.119$$

$$Pontuação\_gato\_preto = Q\_gato \times K\_preto^T / sqrt(4) = 0.131$$

$$Pontuação\_gato\_dorme = Q\_gato \times K\_dorme^T / sqrt(4) = 0.125$$

Aplica-se a função *softmax* para normalizar as pontuações:

$$softmax((Q \times K^T) / sqrt(4))$$

$$Softmax([0.1125, 0.119, 0.131, 0.125]) = [0.248, 0.249, 0.252, 0.251]$$

Cada pontuação obtida nesse processo indica o quão relevante cada palavra da sequência é em relação à palavra que está sendo analisada.

Passo 4: ponderação dos vetores de valores

Cada vetor de valor (V) é então ajustado pela pontuação de atenção correspondente, por exemplo, para a palavra "gato":

```
V_ajustado_gato_O      = 0.248 × V_O = [0.061, 0.061, 0.061, 0.061]
V_ajustado_gato_gato = 0.249 × V_gato = [0.058, 0.060, 0.073, 0.075]
V_ajustado_gato_preto = 0.252 × V_preto = [0.053, 0.056, 0.053, 0.053]
V_ajustado_gato_dorme = 0.251 × V_dorme = [0.057, 0.060, 0.062, 0.054]
```

Isso faz com que cada palavra na sequência dê mais importância às palavras mais relevantes para ela, de acordo com as pontuações de atenção.

Passo 5: combinação dos vetores ajustados

Os vetores ajustados pela atenção são somados para criar um vetor de atenção para cada palavra. Esse vetor combina todos os vetores de valores, ponderados pela sua importância, por exemplo, para a palavra "gato":

Output_gato = V_ajustado_gato_O +

```
V_ajustado_gato_gato +
V_ajustado_gato_preto +
V_ajustado_gato_dorme
```

Output_gato = [0.061, 0.061, 0.061, 0.061] +

```
[0.058, 0.060, 0.073, 0.075] +
[0.053, 0.056, 0.053, 0.053] +
[0.057, 0.060, 0.062, 0.054]
```

```
Output_gato = [0.229, 0.237, 0.249, 0.243]
```

Assim, obtemos uma nova sequência de vetores que inclui informações contextuais de toda a sequência original.

Passo 6: múltiplas cabeças de atenção

Os *transformers* usam múltiplas cabeças de atenção em paralelo para entender diferentes aspectos das relações entre palavras. Cada cabeça de atenção funciona de forma independente e, em seguida, os resultados são concatenados e projetados novamente para produzir a saída final.

$$\text{multihead\_output} = \text{concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_N) \times W_{out}$$

O vetor concatenado é então multiplicado por uma matriz de pesos W_{out} para projetar este vetor concatenado de volta para a dimensão original do espaço de *embedding*. Para simplificar, suponha que usamos duas cabeças de atenção. Repetimos os cálculos acima para a segunda cabeça de atenção com diferentes matrizes de pesos. Após obter os vetores ajustados para todas as cabeças, concatenamos os resultados:

```
Output_cabeça1_gato=[0.229,0.237,0.249,0.243]
Output_cabeça2_gato=[0.221,0.231,0.252,0.245]
Output_final_gato=[0.229,0.237,0.249,0.243,0.221,0.231,0.252,0.245]
```

Passo 7: Contextualização

A auto-atenção permite que cada palavra considere o contexto completo de todas as outras palavras.

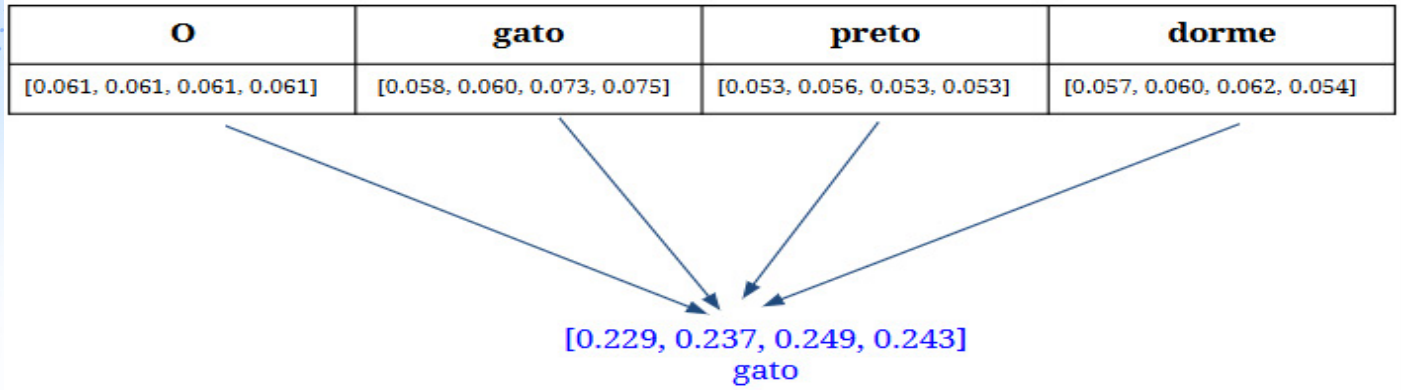
"O gato preto dorme".

"O": [0.25, 0.25, 0.25, 0.25]

"gato": [0.229, 0.237, 0.249, 0.243]

"preto": [0.2, 0.21, 0.2, 0.2]

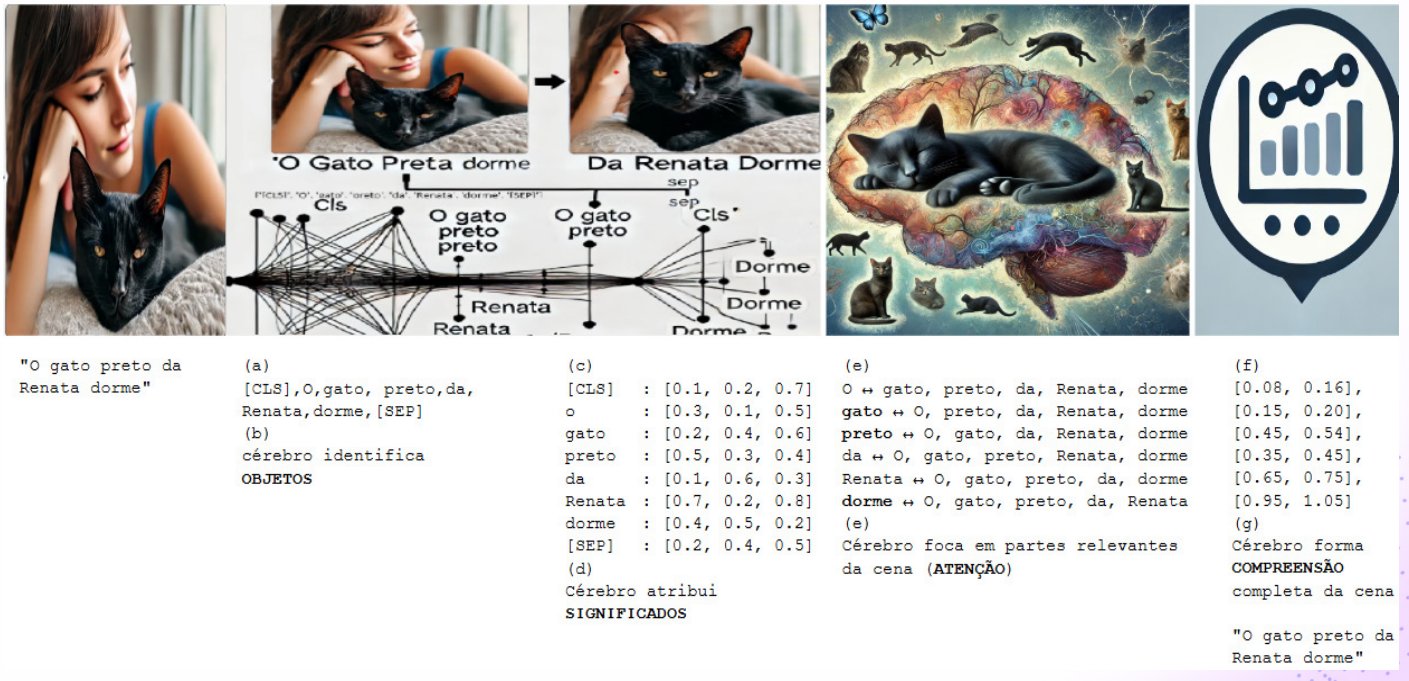
"dorme": [0.22, 0.23, 0.24, 0.21]



A auto-atenção possibilita que cada palavra na sequência leve em conta o contexto completo das demais, resultando em uma compreensão mais profunda e contextualizada. Esse é o diferencial dos *transformers* em tarefas de PLN. Por exemplo, a palavra "gato" na frase "O gato preto dorme" não é interpretada de forma isolada, mas em relação ao contexto das outras palavras. O vetor final de "gato" é formado pela combinação ponderada dos vetores ajustados de todas as palavras da sequência, permitindo capturar dependências complexas e contextos amplos.

Na Figura 17, de forma bem simplificada, está a representação de como uma máquina "entende" uma frase: "O gato preto da Renata dorme." O processo começa com a divisão da frase em *tokens* menores (a). Cada *token* é então convertido em um vetor numérico que captura seu significado (c). O mecanismo de atenção do modelo destaca as partes mais importantes da frase, focando na relação entre as palavras e 'dorme' (e). No final, todos esses vetores são combinados para formar uma representação completa da frase, refletindo seu contexto e significado (g). As indicações (b), (d), (e) e (g) poderiam ser associadas ao que acontece no mundo biológico.

Figura 17 - Processamento e contextualização de frases em modelos de linguagem



Fonte: ilustração potencializada pelo GPT 4.

4.3. Do BERT ao LLM

A evolução das tecnologias oferece, no campo do PLN, aos desenvolvedores e pesquisadores ferramentas cada vez mais avançadas para superar desafios da comunicação humana. Deve-se salientar que um marco importante nessa jornada foi a transição do BERT para os modelos de linguagem de grande escala, um desenvolvimento crucial *vis-à-vis* o futuro da nossa interação com a tecnologia.

O BERT trouxe a capacidade de entender o contexto das palavras de forma bidirecional, um avanço em comparação com modelos anteriores que processavam textos de maneira unidirecional. Ele analisa tanto o que precede quanto o que segue uma palavra, oferecendo uma compreensão contextual ampla, o que melhora de forma significativa a precisão, por exemplo, em sistemas de resposta automática e ferramentas de tradução.

Contudo, a evolução não parou no BERT. Inspirados por seu sucesso, avançou-se ainda mais, desenvolvendo-se os *Language Large Models* (LLMs). Esses modelos são treinados com quantidades massivas de dados e podem realizar uma ampla gama de tarefas linguísticas de forma autônoma, sem a necessidade de ajustes específicos para cada nova tarefa. Por exemplo, o GPT-3 da OpenAI pode escrever textos com fluência quase humana, responder a perguntas complexas e sintetizar grandes volumes de informação.

Para aqueles envolvidos na criação de soluções de PLN, os LLMs representam uma oportunidade extraordinária. Eles não apenas aprimoram aplicações já existentes, bem como também permitem explorar novas formas de interação entre seres humanos e máquinas. Com sua capacidade de gerar respostas que imitam o discurso humano, hoje um agente de *software* equipado com um LLM poderia facilmente chegar perto de superar o Teste de *Turing*, um experimento que avalia se uma máquina pode imitar a comunicação humana a ponto de um ser humano não conseguir distinguir se está interagindo com uma máquina ou outra pessoa. Esse avanço não é apenas um salto tecnológico, mas também abre novos caminhos para como se pensa a comunicação *homem-máquina*.

A trajetória do BERT aos LLMs ilustra a incrível sinergia entre inovação científica e tecnológica, ultrapassando limites previamente estabelecidos. Para você, entusiasta ou profissional da área, essa evolução traz reflexões sobre as amplas possibilidades que essas ferramentas avançadas oferecem. Juntos, podemos participar ativamente da criação de um futuro em que as máquinas compreendem e interagem com a linguagem humana de maneiras cada vez mais sofisticadas e eficientes.

4.3.1 Fundamentos do BERT

BERT é baseado na arquitetura do *Transformer*, que é composta principalmente de blocos chamados "*encoders*". Cada *encoder* possui duas partes principais: uma camada de atenção auto-dirigida e uma rede neural de *feed-forward*. A inovação do BERT está em como essas camadas são utilizadas para analisar o texto de forma bidirecional, dife-

rentemente de abordagens anteriores que tratavam os textos de maneira unidirecional (esquerda para direita ou direita para esquerda).

O processo de aprendizado do BERT se desenrola em várias etapas (Figura 18).

Figura 18 - Processo de *tokenização* e pré-treinamento no BERT

Fonte: autoria própria.

Essa abordagem bidirecional permite ao BERT compreender melhor o contexto e as nuances do texto, resultando em um desempenho superior em diversas tarefas de PLN.

4.3.2 Esquema Iterativo para Compreensão

Para facilitar a compreensão de como o BERT funciona internamente, imagine o processo como uma série de filtros em uma fábrica de processamento de texto:

a) **Tokenização e embedding:** o texto entra como matéria-prima e é cortado em unidades manuseáveis (*tokens*), cada um recebendo um rótulo numérico. Os *tokens* são então convertidos em vetores por meio de *embeddings*, que incluem informações posicionais para manter a ordem dos *tokens* no texto.

b) **Atenção:** os *tokens* passam por uma máquina que verifica cada pedaço, considerando todos os outros pedaços simultaneamente, para entender seu contexto e importância. Essa etapa utiliza o mecanismo de atenção, que pondera a importância relativa de outros *tokens* ao considerar um *token* específico.

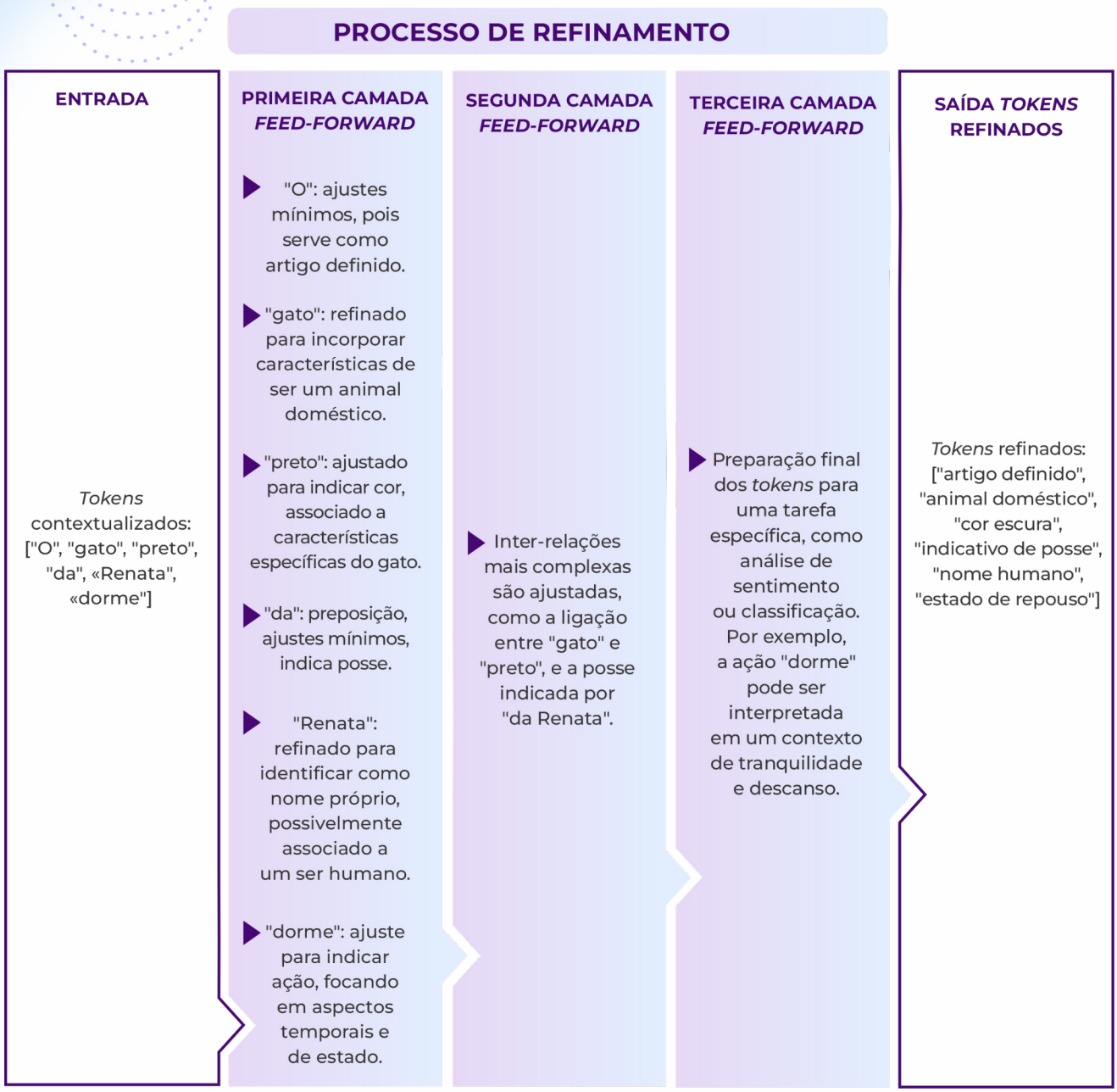
c) **Refinamento:** em seguida, os *tokens* contextualizados são refinados em uma série de passagens por redes neurais *feed-forward*, ajustando as representações dos *tokens* com base no contexto fornecido pela atenção.

d) **Saída:** a saída de cada *encoder* é passada para o próximo, em uma série de camadas empilhadas. O BERT padrão usa 12 dessas camadas (*encoders*) para a versão base e 24 para a versão "avançada". A saída final do último *encoder* pode ser usada para diferentes tarefas de PLN, como classificação de texto, resposta a perguntas e muito mais, adicionando apenas uma camada específica de saída para cada tarefa.

Exemplo:

Imagine que estamos em uma fábrica de processamento de palavras, onde os "*tokens* contextualizados" da frase "O gato preto da Renata dorme" são refinados para serem usados em uma tarefa específica de PLN (Figura 19).

Figura 19 - Refinamento de *tokens* para análise contextual



Fonte: autoria própria.

Cada etapa do processo refina os *tokens* para que se encaixem melhor na tarefa final que o modelo pretende executar. Essas máquinas de processamento simples (redes *feed-forward*) ajustam os *tokens* para maximizar a precisão e a relevância das informações, tornando o modelo eficaz para a tarefa de saída designada.

4.3.3 Processo de Tokenização no BERT

Dada a frase "O gato preto da Renata dorme", na primeira etapa de processamento do BERT, cada palavra é transformada em um "*token*". Em seguida, cada *token* é convertido em um vetor numérico por meio do processo de "*embedding*". Cada vetor representa numericamente as características e o significado da palavra dentro do modelo.

Além disso, o BERT adiciona uma camada extra de informação chamada de "*embedding posicional*". Isso porque a posição de uma palavra em uma frase pode alterar seu significado ou como ela interage com outras palavras. Então, para a frase "O gato preto da Renata dorme", o BERT não só entende o significado isolado de cada palavra, mas também considera a ordem em que aparecem. Na Tabela 13, são apresentados os passos do processamento de texto.

Tabela 13 - Passos do processamento de texto: tokenização, *embedding* de *token* e *embedding* posicional

1 TOKENIZAÇÃO	2 EMBEDDING DE TOKEN	3 EMBEDDING POSICIONA
"O" → Token 1	Token 1 ("O") → [0.25, 0.40, 0.70, ...]	Posição 1 → [0.10, 0.10, 0.00, ...]
"gato" → Token 2	Token 2 ("gato") → [0.50, 0.90, 0.10, ...]	Posição 2 → [0.00, 0.10, 0.10, ...]
"preto" → Token 3	Token 3 ("preto") → [0.15, 0.25, 0.50, ...]	Posição 3 → [0.10, 0.10, 0.00, ...]
"da" → Token 4	Token 4 ("da") → [0.05, 0.20, 0.30, ...]	Posição 4 → [0.00, 0.00, 0.10, ...]
"Renata" → Token 5	Token 5 ("Renata") → [0.40, 0.30, 0.20, ...]	Posição 5 → [0.10, 0.10, 0.00, ...
"dorme" → Token 6	Token 6 ("dorme") → [0.30, 0.10, 0.20, ...]	Posição 6 → [0.10, 0.00, 0.10, ...]

Fonte: autoria própria.

Os vetores finais para cada palavra resultam da combinação dos *embeddings* de *token* com os *embeddings* posicionais. Cada vetor não só capta o significado individual das palavras como "Renata" e "gato", bem como reflete suas posições específicas na frase. Essa abordagem ajuda o BERT a entender o contexto completo e as relações sin-

táticas e semânticas dentro da frase em questão, permitindo que ele processe de forma eficaz a semântica e a sintaxe do texto.

a) **Camada de atenção autodirigida:** aqui é onde o BERT se destaca com sua capacidade de entender o contexto de maneira bidirecional. Cada *token* não só é analisado isoladamente, mas sua relação com todos os outros *tokens* na sentença é examinada. Essa camada utiliza um mecanismo de atenção. Isso permite que o modelo capture melhor o contexto.

b) **Redes neurais de *feedforward*:** após a camada de atenção, os *embeddings* modificados passam por uma rede neural simples que ajuda a refinar ainda mais as representações dos *tokens* com base no contexto fornecido pela atenção.

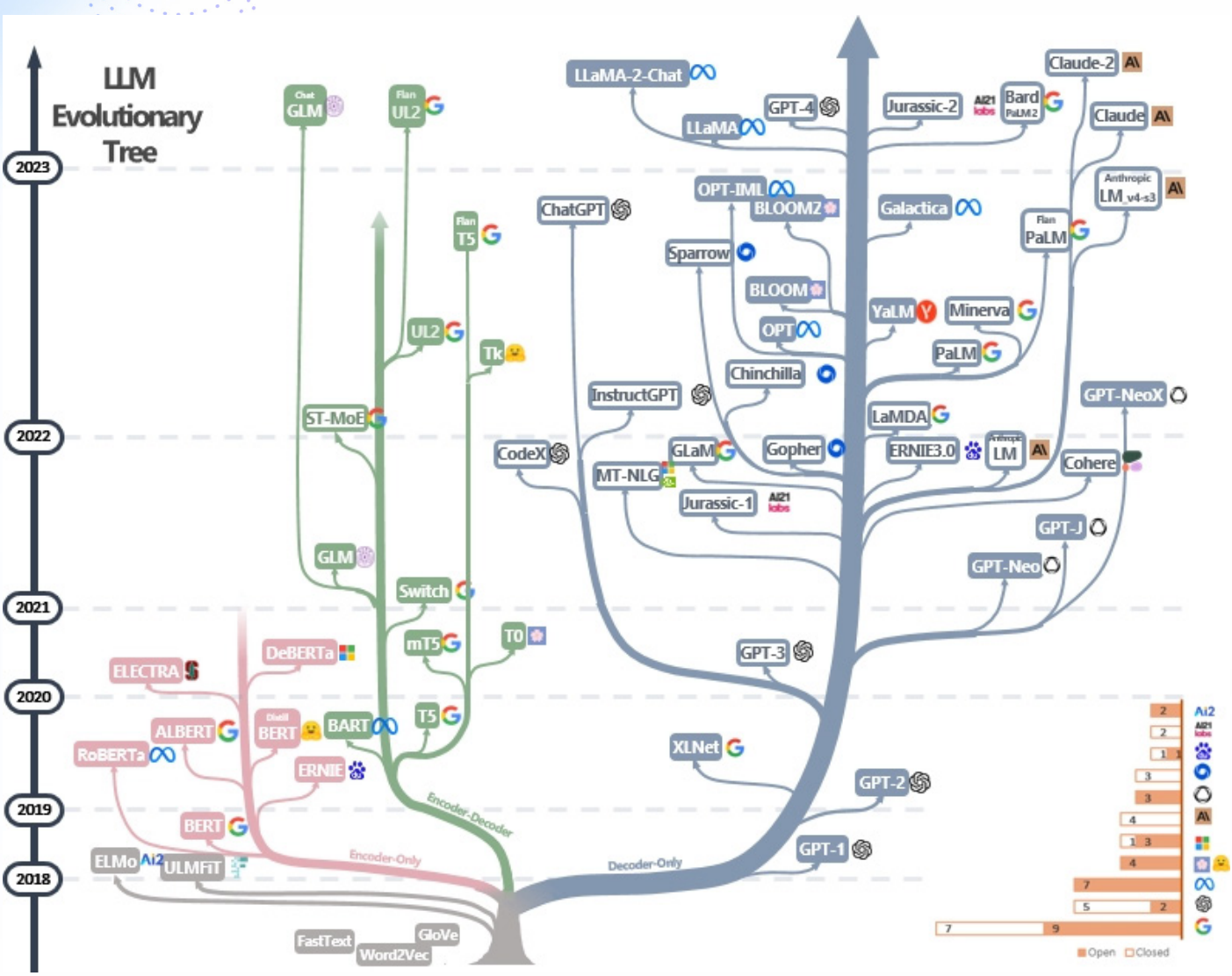
c) **Saída:** a saída final do último *encoder* pode ser usada para diferentes tarefas de PLN, como classificação de texto, resposta a perguntas e muito mais, adicionando apenas uma camada específica de saída para cada tarefa.

4.3.4 Evolução para os Grandes Modelos de Linguagem

Desde o surgimento do BERT, observamos uma expansão significativa no tamanho e na capacidade dos modelos de linguagem, que culminou com a criação de modelos de linguagem de grande escala, como o GPT-3. Esses modelos, treinados em vastos volumes de texto, são afinados para desempenhar tarefas específicas, destacando-se pela sua habilidade em entender e gerar texto.

Na Figura 20, é ilustrada essa progressão por meio de uma árvore evolutiva, que documenta os avanços recentes em modelos de linguagem, ressaltando os mais influentes. Nessa árvore, modelos que compartilham um ramo têm ligações mais estreitas, e os baseados em tecnologia *transformer* são diferenciados por cores: decodificadores em azul, codificadores em rosa e codificador-decodificadores em verde. A disposição vertical dos modelos indica suas datas de lançamento, com quadrados preenchidos para modelos de código aberto e quadrados vazios para os de código fechado. Adicionalmente, um gráfico de barras na parte inferior direita do diagrama mostra como os modelos estão distribuídos entre diferentes empresas e instituições.

Figura 20 - Evolução dos modelos de linguagem de grande escala: uma análise visual dos principais modelos e suas relações



Fonte: Yang *et al.* (2023).

SAIBA MAIS...

» Para aprofundar o conhecimento sobre os tópicos abordados, você pode consultar os seguintes recursos:

» Attention is All You Need

» Documentação do BERT da Google

» BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- RoBERTa: A Robustly Optimized BERT Pretraining Approach
- DistilBERT: smaller, faster, cheaper and lighter
- ALBERT: A Lite BERT for Self-supervised Learning of Language Representations
- UniLM Unified Language Model Pre-training for Natural Language Understanding and Generation
- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
- GLM-130B: An Open Bilingual Pre-trained Model
- AlexaTM 20B: Few-Shot Learning Using a Large-Scale Multilingual Seq2Seq Model
- ST-MoE: Designing Stable and Transferable Sparse Expert Models
- GPT-3 da OpenAI
- Language Models are Few-Shot Learners

- LLaMA: Open and Efficient Foundation Language Models
- BloombergGPT: A Large Language Model for Finance
- GPT-NeoX-20B: An Open-Source Autoregressive Language Model
- Llama 2: Open foundation and fine-tuned chat models
- Model Card and Evaluations for Claude Models

Ao refletirmos sobre as contribuições dessa Unidade, fica evidente que percorremos um caminho importante desde as abordagens iniciais baseadas em regras até os inovadores modelos de *transformers* e LLMs. A evolução do PLN reflete não apenas avanços tecnológicos, mas também uma expansão nas possibilidades de aplicação dessas tecnologias.

Com as atividades propostas e os recursos disponibilizados, cada discente está equipado com o conhecimento necessário para não apenas começar entender, bem como dar os seus **primeiros passos** na aplicação dessas técnicas avançadas de maneira eficaz em diversas situações práticas. Encerramos essa Unidade com uma compreensão mais profunda da complexidade e do moderno poder do PLN, antecipando os futuros desenvolvimentos que continuem a transformar este campo fascinante.

Introdução ao Processamento de Linguagem Natural

Unidade V
Encerramento

Unidade V - Encerramento

Neste Curso, traçamos o desenvolvimento do PLN, desde suas raízes nas técnicas baseadas em regras até os sofisticados modelos de *transformers* e LLMs. Começamos com os fundamentos, como tokenização e análise morfológica, progredindo para conceitos mais avançados como modelos de espaço vetorial e aprendizado profundo. Aprofundamos-nos em tecnologias disruptivas como o BERT e o GPT-3, examinando como eles transformam nossa capacidade de compreender e gerar linguagem de maneira automatizada.

5.1 Reflexões sobre o Aprendizado

Explorar o PLN nos permite não apenas entender os avanços tecnológicos, mas também considerar seu potencial para beneficiar a sociedade de maneiras éticas e inovadoras. A habilidade de máquinas entenderem e produzirem linguagem tem implicações profundas, melhorando a comunicação entre seres humanos e computadores e criando sistemas mais inclusivos.

5.2 Direções para Futuras Explorações e Estudos

Para os(as) entusiastas que desejam expandir seu entendimento e continuar explorando o campo do PLN, recomenda-se a seguinte lista de recursos e atividades.

Literatura recomendada:

- JURAFSKY, D.; MARTIN, J. H.. *Speech and Language Processing*. 3. ed. **Stanford Edu**. 2024. Disponível em: https://web.stanford.edu/~jurafsky/slp3/. Acesso em: 17 out. 2024.
- KAMATH, U.; LIU, J.; WHITAKER, J.. **D**eep learning for NLP and speech recognition. Cham, Switzerland: **Springer**, 2019. Disponível em: https://link.springer.com/book/10.1007/978-3-030-14596-5. Acesso em: 17 out. 2024.

Cursos avançados:

- Participe de *workshops* e tutoriais oferecidos por organizações como a *Association for Computational Linguistics* (ACL) e *Conference on Neural Information Processing Systems* (NeurIPS).

Engajamento em projetos de código aberto:

» Colabore com projetos como o spaCy, NLTK ou *transformers* da *Hugging Face* para ganhar experiência prática enquanto contribui para o progresso da comunidade.

Interação com comunidades de PLN:

» Engajar-se em fóruns como GitHub ou grupos dedicados ao PLN pode proporcionar percepções relevantes, colaboração e inspiração para projetos futuros.

Com essas ferramentas e recursos, encerramos o Microcurso, esperando que cada participante utilize o conhecimento adquirido como um ponto de partida para futuras descobertas e inovações no campo do PLN. O futuro promete avanços ainda mais transformadores, e cada um de vocês está agora equipado para ser um contribuidor ativo e influente nesta área dinâmica.

Referências

Década de 1950 a 1980: Paradigma Simbólico

- CHOMSKY, N.. Syntactic Structures. **Mouton & Co**, 1 ed, 1957.
- WEAVER, W.. Translation. In Machine Translation of Languages: Fourteen Essays, ed. A. D. Booth and W. N. Locke, 15–23. Cambridge, MA: **MIT Press**. 1955.
- WINOGRAD, T.. Understanding natural language. **Cognitive psychology**, v. 3, n. 1, p. 1-191, 1972.

Década de 1980 a 2000: Paradigma Estatístico

- BAHL, L. R.; JELINEK, F.; MERCER, R. L. A maximum likelihood approach to continuous speech recognition. **IEEE transactions on pattern analysis and machine intelligence**, n. 2, p. 179-190, 1983.
- CHURCH, K.; HANKS, P.. Word association norms, mutual information, and lexicography. **Computational linguistics**, v. 16, n. 1, p. 22-29, 1990.
- MARCUS, M.; SANTORINI, B.; MARCINKIEWICZ, M. A.. Building a large annotated corpus of English: The Penn Treebank. **Computational linguistics**, v. 19, n. 2, p. 313-330, 1993.

Década de 2000 a 2020: Paradigma Neural

- DEVLIN, J.. Bert: pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.
- HOCHREITER, S.; SCHMIDHUBER, J.. Long short-term memory. **Neural Computation**, 9(8), 1735-1780, 1997.
- MIKOLOV, T.. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013.
- RADFORD, A. *et al*. Language models are unsupervised multitask learners. **OpenAI blog**, v. 1, n. 8, p. 9, 2019.
- VASWANI, A. Attention is all you need. **31st Conference on Neural Information Processing Systems (NIPS 2017)**, Long Beach, Carlifornia, 2017.

Outras Referências

- DIRACO, G.; LEONE, A.; SICILIANO, P.. Ai-based early change detection in smart living environments. **Sensors**, v. 19, n. 16, p. 3549, 2019.

» QIAN, Q.; CHEN, X.. A Multi-Modal Convolutional Neural Network Model for Intelligent Analysis of the Influence of Music Genres on Children's Emotions. **Computational Intelligence and Neuroscience**, v. 2022, n. 1, p. 4957085, 2022.

» THABIT, Q. Q.; FAHAD, T. O.; DAWOOD, A. I. Detecting diabetes using machine learning algorithms. In: 2022 Iraqi International Conference on Communication and Information Technologies (IICCIT). **IEEE, 2022**. p. 131-136.

» THANH, P. N. *et al.* Short-term three-phase load prediction with advanced metering infrastructure data in smart solar microgrid based convolution neural network bidirectional gated recurrent unit. **IEEE Access**, v. 10, p. 68686-68699, 2022.

» YANG, J. *et al.* Harnessing the power of llms in practice: A survey on chatgpt and beyond. **ACM Transactions on Knowledge Discovery from Data**, v. 18, n. 6, p. 1-32, 2024.

SAIBA MAIS...

» Para um estudo abrangente e detalhado do PLN, é fundamental consultar uma variedade de fontes que cobrem tanto os fundamentos teóricos quanto as mais recentes inovações tecnológicas. Aqui estão algumas das principais fontes bibliográficas que são consideradas fundamentais no campo do PLN.

» **Livros:**

» JURAFSKY, D..; MARTIN, J. H.. *Speech and Language Processing*. 3. ed. **Stanford Edu**. 2024. Disponível em: https://web.stanford.edu/~jurafsky/slp3/. Acesso em: 25 jul. 2024 out. 2024.

» Esse livro é uma referência essencial que cobre uma ampla gama de tópicos em PLN, desde os fundamentos até técnicas avançadas.

» MANNING, C. D.. Foundations of statistical natural language processing. **The MIT Press**, 1999.

» Esse texto é crucial para entender as técnicas estatísticas que formam a base de muitos métodos modernos de PLN.

- BIRD, S.; KLEIN, E.; LOPER, E.. Natural language processing with Python: analyzing text with the natural language toolkit. **O'Reilly Media, Inc.**, 2009.
  - Ideal para quem deseja aplicar PLN na prática, usando a biblioteca NLTK em Python.
- GOYAL, P.; PANDEY, S.; JAIN, K.. Deep learning for natural language processing. **New York: Apress**, 2018.
  - Esse livro oferece uma introdução prática ao uso de redes neurais profundas para PLN.
- GOLDBERG, Y. Neural Network Methods forNatural Language Processing. **Springer**, 2017.
  - Uma excelente fonte para entender como as técnicas de redes neurais são aplicadas ao PLN.

- **Artigos:**
- VASWANI, A. Attention is all you need. A**dvances in Neural Information Processing Systems**, 2017.
  - Introduziu o modelo *Transformer*, que é a base para muitos desenvolvimentos recentes em PLN, incluindo o BERT e o GPT.
- DEVLIN, J.. Bert: pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.
  - **"Language Models are Few-Shot Learners"** por Tom B. Brown et al. (2020): Descreve o GPT-3 e sua capacidade de realizar tarefas de PLN com pouca ou nenhuma adaptação específica de tarefa.

- **Recursos online:**
  - **ACL Anthology**: Uma coleção digital abrangente de artigos sobre linguística computacional e PLN, mantida pela Association for Computational Linguistics (ACL).
  - **arXiv.org**: Um repositório de *preprints* de artigos científicos em várias áreas, incluindo PLN, onde pesquisadores frequentemente publicam os resultados mais recentes antes de serem formalmente revisados por pares.

Essas fontes fornecem uma base sólida tanto para discentes quanto para pesquisadores interessados em explorar tanto os aspectos teóricos quanto práticos do PLN. Elas são complementadas por uma vasta gama de cursos e tutoriais disponíveis *online*, que podem ajudar a traduzir teoria em prática.

AKCIT

CENTRO DE COMPETÊNCIA EMBRAPII
EM TECNOLOGIAS IMERSIVAS

SOBRE O E-BOOK

Tipografia: Montserrat

Publicação: Cegraf UFG
Câmpus Samambaia, Goiânia - Goiás. Brasil. CEP 74690-900
Fone: (62) 3521-1358
https://cegraf.ufg.br