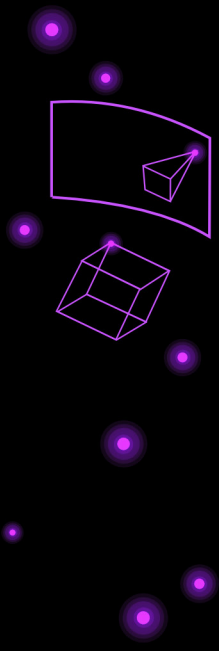
AKCIT
CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Métodos Clássicos de Processamento de Linguagem Natural

Autoria:
Nádia Félix Felipe da Silva



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

Cegraf UFG

Métodos Clássicos de Processamento de Linguagem Natural

Autoria:
Nádia Félix Felipe da Silva

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

Cegraf UFG

2024

Revisão Técnica
Deborah Silva Alves Fernandes

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/SIL.met.ebook.978-85-495-1070-9/2024

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

Silva, Nádia Félix Felipe da
Métodos clássicos de processamento de linguagem natural [livro eletrônico] / Nádia Félix Felipe da Silva ; organização Taciana Novo Kudo..[et al.].
-- Goiânia, GO : Cegraf UFG, 2024.
PDF

Outros organizadores: Deborah Silva Alves Fernandes, Renata Dutra Braga, Cristiane Bastos Rocha Ferreira, Arlindo Rodrigues Galvão Filho.
Bibliografia.
ISBN 978-85-495-1070-9

1. Aprendizagem de máquina 2. Ciência da computação 3. Linguística 4. Morfologia
I. Kudo, Taciana Novo. II. Fernandes, Deborah Silva Alves. III. Braga, Renata Dutra. IV. Ferreira, Cristiane Bastos Rocha. V. Galvão Filho, Arlindo Rodrigues. VI. Título.

25-253945 CDD-005.133

Índices para catálogo sistemático:

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

Eliane de Freitas Leite - Bibliotecária - CRB 8/8415

Métodos Clássicos de 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)

Lista de Abreviaturas e Siglas

AS	AS - Análise de Sentimentos
BERT	*Bidirectional Encoder Representations for Transformers* - Representações Bidirecionais Codificadoras para Transformadores
BoW	*Bag-of-Words* - Saco de Palavras
CBOW	*Continuous Bag of Words* - Modelo Contínuo de Saco de Palavras
Embrapii	Empresa Brasileira de Pesquisa e Inovação Industrial
IA	Inteligência Artificial
IDF	*Inverse Document Frequency* - Frequência Inversa do Documento
LC	Linguística Computacional
LSTM	*Long Short-Term Memory* - Memória de Curto e Longo Prazo
NER	*Named Entity Recognition* - Reconhecimento de Entidades Nomeadas
NP	Sintagma Nominal
PLN	Processamento de Linguagem Natural
SVM	*Support Vector Machines* - Máquinas de Vetores de Suporte
TF	*Term Frequency* - Frequência dos Termos

TF-IDF	*Term Frequency-Inverse Document Frequency* - Frequência dos Termos-Frequência Inversa do Documento
UFG	Universidade Federal de Goiás
URL	*Uniform Resource Locator* - Localizador Uniforme de Recursos
VADER	*Valence Aware Dictionary and sEntiment Reasoner* - Dicionário de Valência e Raciocinador de sEntimentos
VP	Sintagma Verbal

Listas de Figuras e Tabelas

Sumário

Apresentação

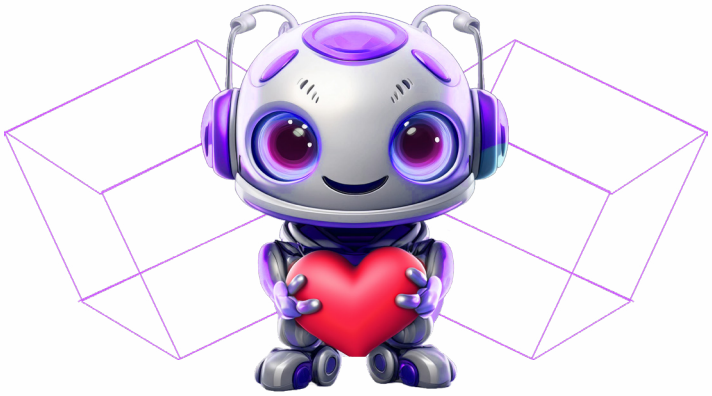
Prezado(a) Participante,

Seja bem-vindo(a) ao Microcurso Métodos Clássicos de Processamento de Linguagem Natural! Este Microcurso faz parte da **Coleção Formação e Capacitação do Centro de Competências Imersivas, uma parceria entre a Embrapii e a Universidade Federal de Goiás (UFG**).

Textos em **linguagem natural** ocorrem em diversos domínios e em uma fração substancial, por exemplo, em páginas da *web*, artigos de notícias, revistas, *blogs*, postagens em *microblogs* como as publicações feitas no X® (antigo Twitter®), postagens no Facebook®, mensagens de bate-papo etc. Esses tipos de dados, puramente textuais, compreendem informações úteis que seriam muito mais valiosas se fossem interpretáveis, para posterior processamento automatizado de computador (por exemplo, por conversão à informação estruturada em um banco de dados). Desbloquear o valor dessa informação por meio do desenvolvimento de técnicas para interpretar a linguagem humana, usando algoritmos de computador, é o que chamamos de Processamento de Linguagem Natural (PLN). O PLN é um campo que combina **ciência da computação**, **inteligência artificial (IA)** e **linguística,** e visa permitir que os computadores resolvam tarefas que envolvem **compreensão e/ou geração de linguagem natural**. Essas tarefas que têm que lidar com a linguagem humana são onipresentes em nossas vidas diárias e vão desde a pesquisa básica (em motores de busca na *web*) até a resposta automática a perguntas ou tradução automática.

Neste Microcurso, nos concentramos no processamento da linguagem na forma escrita e, portanto, não trataremos, por exemplo, de processamento de fala. Mais especificamente, este Microcurso serve como uma introdução a: (1) PLN Clássica, também conhecida como PLN Estatístico, uma vez que depende fortemente de estatística e aprendizado de máquina, bem como (2) PLN Neural, ou seja, métodos baseados em redes neurais. No PLN Estatístico, os computadores não são programados diretamente para processar linguagem, mas normalmente aprendem como processar (ou gerar) linguagem com base nas estatísticas de um conjunto de dados textuais ou

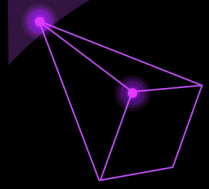
corpus (muitas vezes, vasto) da linguagem natural. Modelos mais recentes baseados em redes neurais são normalmente treinados de ponta a ponta nos dados textuais, normalmente evitando engenharia de recursos específicos (algumas vezes chamada de engenharia de *features* ou engenharia de atributos) e/ou definição explícita de subtarefas na PLN Clássica (por exemplo, *Part-of-Speech (PoS) tagging*, análise de dependência) como etapas em uma solução orientada a um fluxograma de atividades (*pipeline*).

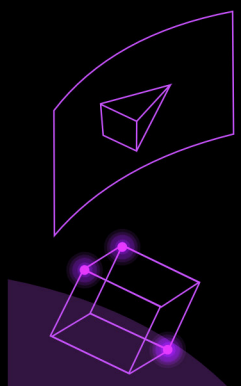


Desejamos um excelente estudo!!

Unidade I

Métodos Clássicos de Processamento de Linguagem Natural





Unidade I: Métodos Clássicos de Processamento de Linguagem Natural

1.1 Introdução aos Métodos Clássicos de Processamento de Linguagem Natural

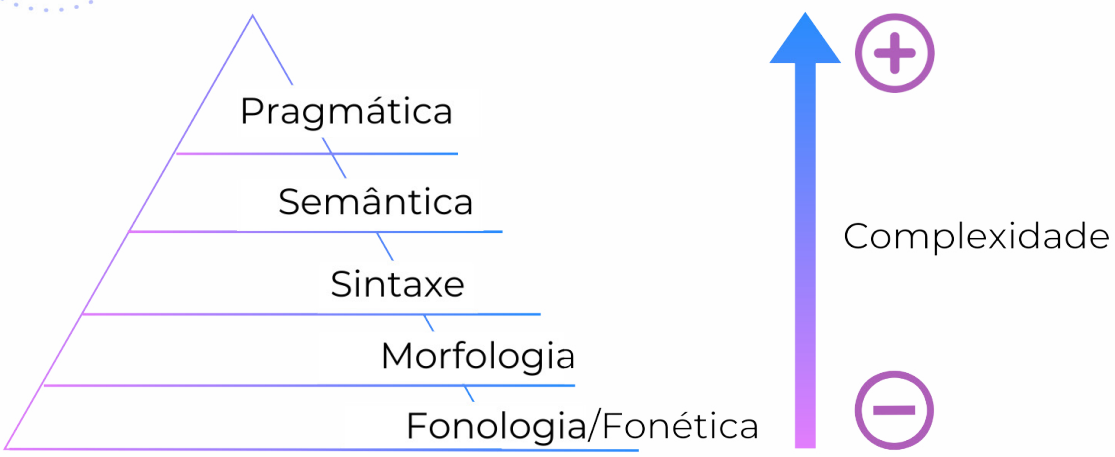
O PLN é um campo da ciência e da engenharia que se dedica a investigar, propor e desenvolver formalismos, modelos, técnicas, métodos e sistemas computacionais que têm a **língua natural como objeto primário.** Mais recentemente, o PLN tem focado no desenvolvimento e estudo de sistemas automáticos que compreendem e geram **linguagens naturais** (ou seja, humanas).

As linguagens humanas são **dispositivos** de comunicação que permitem o compartilhamento e armazenamento eficiente de ideias, fatos e intenções. Como argumenta (Mikolov; Yih; Zweig, 2023), a complexidade da comunicação possibilitada pela linguagem é uma inteligência exclusivamente humana entre as espécies. Neste Microcurso estamos interessados na criação e no estudo de Sistemas Inteligentes, em que a linguagem humana é, para nós, tanto um objeto de estudo – afinal, ela evoluiu para ser tanto aprendida e útil – quanto um grande facilitador para interagir com humanos, mesmo em contextos onde outras modalidades (por exemplo, visão) também são de interesse.

Essa caracterização abrangente dificulta, muitas vezes, distinguir essa área de outras correlatas, como a Linguística Computacional (LC), a Linguística Aplicada, a Linguística de Corpus (na grande área de Linguística), e mesmo com outras da IA, como *Text Mining* (Mineração de Textos). No exterior, grandes conferências denominadas de Linguística Computacional abrangem de fato os estudos de PLN. Por outro lado, não é raro haver distinção entre LC e PLN com base apenas nas diferentes perspectivas sobre um mesmo objeto: se linguística ou computacional, respectivamente.

Tradicionalmente, a NLP pode ser dividida em tarefas ou níveis de conhecimento, representadas por uma pirâmide (Figura 1). No topo, estão as tarefas de maior dificuldade para ensinar a máquina e, na base, as de menor dificuldade.

Figura 1 - Tarefas ou níveis de conhecimento em Processamento em Linguagem Natural



Fonte: adaptada de Jurafsky e Martin (2024).

1.1.1 A Fonologia ou Fonética em Processamento de Linguagem Natural

A fonologia ou fonética é o ramo da linguística que estuda os sons da fala produzidos pelos seres humanos. Ela se preocupa com a produção, transmissão e percepção dos sons da fala, examinando características físicas dos sons, como sua frequência, intensidade, duração e qualidade. Na fonética, os sons da fala são representados por meio de símbolos fonéticos, que descrevem os sons de forma precisa e sistemática. Esses símbolos são usados em transcrições fonéticas para registrar a pronúncia real das palavras. No PLN, a fonética desempenha um papel importante em várias tarefas, como reconhecimento de fala, síntese de fala, alinhamento automático de texto com áudio e identificação de falantes. Compreender a fonética permite que os sistemas de PLN processem e gerem fala de forma mais precisa e natural. Como a fonética não é objeto de estudo deste Microcurso, não a trataremos a fundo.

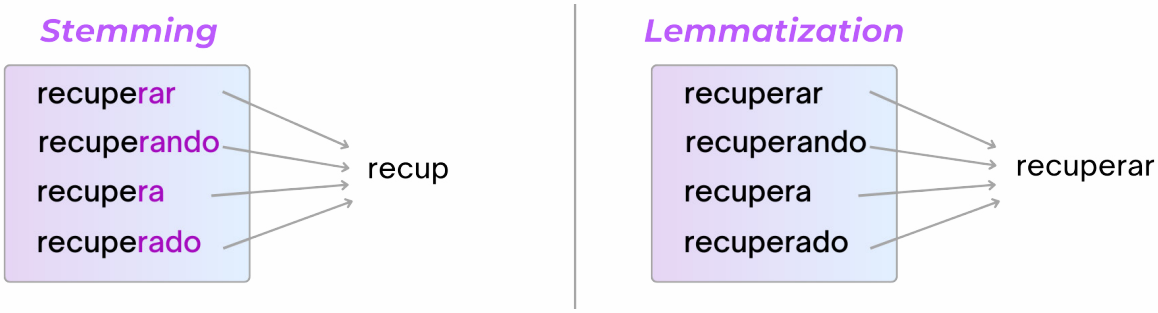
1.1.2 A Morfologia

A morfologia refere-se ao estudo da estrutura e formação das palavras em uma língua. Isso envolve analisar como as palavras são construídas a partir de unidades menores chamadas morfemas, que são as unidades mínimas de significado. A morfologia trata de questões como flexão (alterações na forma de uma palavra para indicar diferentes gramáticas, como número, tempo, gênero, etc.), derivação (a formação de novas palavras por meio de prefixos, sufixos, infixos etc.) e composição (a forma-

ção de palavras complexas combinando morfemas). No PLN, a análise morfológica é útil em tarefas como a lematização/radicalização (reduzir uma palavra flexionada à sua forma base) e em tarefas mais sofisticas, como a análise de sentimentos em textos, onde a forma das palavras pode influenciar seu significado ou polaridade.

A lematização é o processo de reduzir uma palavra flexionada à sua forma base, conhecida como "lema". Por exemplo, as palavras "correr", "correu", "correndo" e "corrida" têm o mesmo lema, que é "correr". Na lematização, a morfologia é usada para identificar e remover afixos (prefixos, sufixos, infixos, etc.) de uma palavra, a fim de encontrar seu lema. Isso envolve o conhecimento das regras morfológicas da língua, como as regras de conjugação verbal, flexão nominal, entre outras. Por exemplo, um algoritmo de lematização pode analisar a palavra "correndo" e, com base nas regras morfológicas da língua, identificar que o sufixo "-ndo" indica uma forma verbal no gerúndio. Ao remover esse sufixo, a forma base "correr" é obtida. A lematização é útil em muitas aplicações de PLN, como na indexação de documentos, na recuperação de informações, na tradução automática e na análise de sentimentos, onde entender o significado básico das palavras é essencial para uma interpretação precisa. Outra tarefa semelhante à da lematização é o *stemming*, que reduz palavras à sua raiz, removendo sufixos e prefixos, sem considerar o contexto ou a análise gramatical. O processo de *stemming* funciona removendo as últimas letras de uma palavra, buscando identificar a raiz morfológica comum a diferentes variantes. Por exemplo, as palavras "correr", "corria", "corrido" e "correria" seriam todas reduzidas à forma raiz "corr". Veja um exemplo de lematização e *stemming* na Figura 2.

Figura 2 - Exemplos de *lematização* e *stemming*



Fonte: Jurafsky e Martin (2024).

1.1.3 A Sintaxe

A sintaxe se refere ao estudo da estrutura gramatical das sentenças/frases em uma língua. Isso inclui a análise das relações entre as palavras em uma sentença e a ordem em que são organizadas para formar uma estrutura coerente. A sintaxe lida

com questões como a identificação de elementos gramaticais, como sujeito, objeto, verbo, adjetivo etc., e as regras que governam sua combinação. Além disso, a sintaxe também aborda fenômenos como concordância de número e gênero, regência verbal, ordem das palavras e estruturas gramaticais complexas, como cláusulas subordinadas e frases coordenadas (orações independentes que se conectam para formar uma frase mais longa, por exemplo: Fui ao mercado **e** comprei frutas.). No PLN, a análise sintática é fundamental para tarefas como análise gramatical, *parsing* sintático (análise da estrutura das frases), geração de linguagem natural e tradução automática, pois permite entender e manipular a estrutura das sentenças de forma precisa.

Exemplos de análise sintática (Mikolov *et al.,* 2013):

1. **Análise de dependência** (*dependency parsing*): identifica a relação entre as palavras.
   - » Sentença: "O gato preto dorme no sofá."
   - » Análise:

     O (det) -> gato (nsubj) -> dorme (root) ->

     no (case) -> sofá (obl) preto (amod) -> gato

   "Gato" é o **sujeito** ("nsubj") do **verbo** "dorme" (raiz ou "root") e "preto" é um **modificador adjetival** ("amod") de "gato".

2. **Análise de constituintes** (*constituency parsing*): divide a sentença em suas partes componentes, como **Sintagma Nominal (NP)** e o **Sintagma Verbal (VP)**.
   - » Sentença: "O gato preto dorme no sofá."
   - » Árvore de constituinte:

     (S

      (NP (DT O) (JJ preto) (NN gato))

      (VP (VB dorme)

       (PP (IN no)

        (NP (NN sofá)))))

   O **NP** é uma estrutura gramatical formada por um substantivo ou pronome que funciona como núcleo, podendo ser acompanhado por determinantes, adjetivos, numerais, pronomes ou complementos que especificam ou qualificam o substantivo. O NP pode desempenhar várias funções dentro de uma oração, como sujeito, objeto direto, objeto indireto, com-

plemento nominal, entre outras. Por exemplo: “Aquela casa amarela.” é um NP, com **“casa”** (núcleo) - substantivo, **“Aquela”** (determinante) – pronome demonstrativo, **amarela** (adjuntos adnominais) – adjetivo. Nesse exemplo, o NP “Aquela casa amarela” pode atuar como sujeito de uma oração, como em “Aquela casa amarela é minha”. O **VP** é uma estrutura gramatical que tem como núcleo um verbo. Ele pode ser acompanhado por outros elementos, como complementos, objetos, advérbios e locuções adverbiais, que ampliam ou especificam o sentido do verbo. O VP desempenha a função de predicado na oração, ou seja, é a parte da oração que faz uma afirmação sobre o sujeito etc..

3. **Reconhecimento de Entidades Nomeadas** (*Named Entity Recognition - NER*): envolve a identificação e a classificação de entidades mencionadas em um texto, como nomes de pessoas, organizações, locais, datas, valores monetários, entre outros.

   » Sentença: “O João comprou um carro em São Paulo.”

   » Análise:

   [O João] (PESSOA) comprou um carro em [São Paulo] (LOCAL).

   Aqui, entidades como **nomes de pessoas** e **lugares** são identificadas e classificadas.

1.1.4 Semântica

A semântica se refere ao estudo do significado das palavras, frases e textos em uma língua. Envolve a compreensão do significado das palavras individualmente (semântica lexical) e das relações de significado entre elas quando combinadas em frases e textos (semântica composicional). A semântica no PLN lida com questões como polissemia (uma palavra com múltiplos significados), sinonímia (palavras com significados semelhantes), antonímia (palavras com significados opostos), hiperonímia e hiponímia (relações de generalização e especialização entre palavras), entre outros fenômenos linguísticos.

Compreender a semântica é essencial em uma variedade de tarefas de PLN, incluindo análise de sentimentos, recuperação de informação, resumo automático de texto, resposta a perguntas, entre outras, pois permite que os sistemas entendam e interpretem corretamente o significado das expressões linguísticas.

1.1.5 Pragmática

A pragmática se refere ao estudo do uso da linguagem em contexto, levando em consideração não apenas o significado literal das palavras, mas também o contexto situacional, as intenções do falante e as inferências que os ouvintes fazem para interpretar o significado pretendido.

A pragmática aborda questões como implicatura (inferências feitas a partir do que é dito, mas não explicitamente), pressuposições (informações que são presumidas como conhecidas pelo interlocutor), *deixis* (expressões que dependem do contexto, como pronomes pessoais e advérbios de lugar) e conversação implicada (o modo como a informação é inferida a partir do contexto da conversa).

No PLN, a pragmática é importante para compreender corretamente o significado das declarações em situações reais, pois as expressões linguísticas, muitas vezes, dependem do contexto para serem interpretadas adequadamente. Isso é especialmente relevante em sistemas de diálogo, onde a compreensão do contexto e das intenções do usuário é crucial para fornecer respostas relevantes e precisas. *Deixis* é um conceito importante na pragmática que se refere ao uso de palavras ou expressões cujo significado depende do contexto situacional em que são utilizadas. Essas palavras ou expressões são chamadas de “deícticas” ou “expressões deixis”. Elas incluem pronomes pessoais, advérbios de lugar, demonstrativos, entre outros.

O significado preciso de uma expressão *deixis* só pode ser compreendido quando se considera o contexto em que é utilizada. Por exemplo, o pronome pessoal “eu” tem um significado diferente dependendo de quem está falando. Da mesma forma, expressões *deixis* de lugar, como “aqui” e “ali”, têm significados relativos ao local onde a comunicação está ocorrendo. Em resumo, *deixis* na pragmática refere-se à capacidade das palavras ou expressões de adquirirem seu significado completo apenas em relação ao contexto em que são utilizadas, tornando-se uma parte essencial da interpretação adequada das mensagens linguísticas.

1.2 Representando Palavras

As crianças humanas interagem com um rico mundo multimodal, com diversas formas de comunicação, seja por meio de vídeo, voz, estímulos dos pais e do mundo ao seu redor e adquirem e aprendem com excepcional eficiência de amostragem (não observando somente a linguagem) e eficiência computacional – cérebros são

máquinas de computação eficientes. E com todos os (impressionantes!) avanços em PLN nas últimas décadas, ainda não alcançamos a capacidade de desenvolver máquinas de aprendizagem como as crianças.

Um problema fundamental na construção de máquinas que aprendem línguas/linguagem é a questão da representação; como deveríamos representar a linguagem em um computador de forma que o computador possa processá-la e/ou gerá-la de maneira robusta? É aqui que este Microcurso se concentra, nas ferramentas fornecidas pela PLN Clássica e também pela aprendizagem profunda, um kit de ferramentas para representar a grande variedade da linguagem natural e algumas das regras e estruturas às quais ela, às vezes, adere. Grande parte deste Microcurso será dedicada a esta questão da representação e o restante deste abordará uma subquestão básica: como representamos palavras?

Considere a sentença:

Ana prepara o café para sua tia.

A palavra *Ana* é um sinal, um símbolo que representa uma entidade em algum lugar do mundo (real ou imaginário). A palavra café é também um símbolo que faz referência a algo. Se, em vez disso, disséssemos que **Ana gosta de fazer café para sua tia**, observe que o símbolo **Ana** ainda se refere a **Ana**, mas agora o **café se refere a uma classe mais ampla** - café em geral, não a uma porção específica de água quente deliciosa. Considere as duas frases a seguir:

Ana prepara o chá para sua tia.

Ana prepara a bebida para sua tia.

Qual sentença é “mais parecida” com a sentença sobre o café? A bebida pode ser café (ou pode ser bem diferente!) e o chá definitivamente não é café, mas é parecido, não? E Ana é parecida com a tia porque ambos descrevem pessoas? E é semelhante ao dela porque ambas escolhem instâncias específicas de uma classe?

O **significado das palavras** é infinitamente complexo, derivando dos objetivos dos humanos de se comunicarem entre si e de alcançarem objetivos no mundo. As pessoas utilizam meios contínuos – fala, sinais – mas produzem sinais numa estrutura simbólica discreta – linguagem – para expressar significados complexos. Expressar e processar as nuances e a natureza selvagem da linguagem – ao mesmo tempo que se consegue a forte transferência de informação que a linguagem pretende alcançar – torna a representação de palavras um problema infinitamente fascinante.

Mas, então, o que é uma palavra? Eu não posso definir uma palavra para você, mas posso dar vários exemplos em Português: chá, café, coragem, retrato, etc. Entendo que se eu usar um símbolo para comunicar com outras pessoas e esse fizer sentido para essa "troca", essa é uma palavra.

Notebook Colab

Objetivos de Aprendizagem

» Esse notebook é um tutorial breve sobre pré-processamento de textos. Será usado um corpus (banco de dados) sobre análise de sentimento em Português, que está disponível em https://www.kaggle.com/datasets/fredericods/ptbr-sentiment-analysis-datasets.Especificamente, vamos utilizar apenas o arquivo olist.csv. Baixe o arquivo olist.csv e memorize o caminho.

A Olist é uma loja de departamentos brasileiros e esta lançou um desafio na plataforma Kaggle e disponibilizou o banco de dados Brazilian E-Commerce Public Dataset, com aproximadamente 100.000 pedidos realizados de 2016 a 2018 em vários marketplaces no Brasil.

» Execute este notebook pausadamente buscando compreender a entrada e saída de cada comando.

A. Porque é importante pré-processar textos?

O texto original pode conter caracteres, palavras ou frases irrelevantes para a análise; Um termo pode estar presente em um documento sob diferentes formas.

A seguir importaremos algumas bibliotecas necessárias para execução deste exemplo.

```
[ ] import pandas as pd
    import numpy as np
```

No nosso caso, nossos conjuntos de dados estão armazenados no próprio drive/núvem google drive. Por isso estamos apontaando para este lugar na linha abaixo. Se o seu arquivo estiver na sua máquina ou outro lugar você precisa explicitar o caminho para a leitura correta.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] df_original = pd.read_csv('/content/drive/MyDrive/nlp_m6/olist.
    csv') #caminho da pasta no drive onde está o olist.csv
```

Na linha abaixo, estamos chamando a variável df_original, uma estrutura Dataframe a qual recebeu o arquivo olist.csv. Um arquivo que possui 41.744 linhas e 8 colunas. Observe que temos 8 colunas, mas neste momento observe principalemente a coluna **review_text** e polarity que são comentários sobre os produtos e a polaridade (respectivamente). Observe que se a polaridade é 1.0 o comentário é positivo e se a polaridade é 0.0 o comentário é negativo.

```
[ ] df_original
```

	original_index	review_text	review_text_processed	review_text_tokenized	polarity	rating	kfold_polarity	kfold_rating
0	97262	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	['perfeito', 'chegou', 'antes', 'do', 'prazo']	1.0	5	1	1
1	72931	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	['foi', 'uma', 'otima', 'compra', 'chegou', 'a...	1.0	5	1	1
2	19659	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	['recebi', 'muito', 'rapido', 'um', 'otimo', '...	1.0	5	1	1
3	43054	Recomendo	recomendo	['recomendo']	1.0	5	1	1
4	59202	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	['so', 'veio', 'uma', 'capa', 'comprei', 'ai',...	0.0	1	1	1
...	...	...	...	...	...	...	...	...
41739	49725	SUCESSO :D	sucesso :d	['sucesso']	1.0	5	10	10
41740	76794	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	['tudo', 'ok', 'produto', 'entregue', 'no', 'p...	1.0	5	10	10
41741	72847	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	['ultimamente', 'estao', 'atrasando', 'muito',...	0.0	1	10	10
41742	50905	recomendo	recomendo	['recomendo']	1.0	5	10	10
41743	6468	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	['mascara', 'veio', 'com', 'um', 'pequeno', 'd...	0.0	2	10	10

41744 rows × 8 columns

Observe o comendo necessário para acessar a coluna review_text, linha 1 da tabela df_original

```
[ ] df_original.iloc[1]['review_text']
```

```
'Foi uma ótima compra! Chegou antes mesmo do prazo e o produto é
igual o que estava no anúncio. Correspondeu as expectativas!'
```

Podemos não estar interessados em outras colunas do conjunto de dados. No comando a seguir definimos quais as colunas queremos manter.

```
[ ] df = df_original[['review_text', 'review_text_processed', 'polarity']]
    df
```

	review_text	review_text_processed	polarity
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0
3	Recomendo	recomendo	1.0
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0
...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0
41742	recomendo	recomendo	1.0
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0

41744 rows × 3 columns

```
[ ] df.iloc[1]['review_text_processed']
```

```
'foi uma otima compra! chegou antes mesmo do prazo e o produto e
igual o que estava no anuncio. correspondeu as expectativas!'
```

```
[ ] exemplo = 'EXCELENTE!! Valeu demais passar um tempo pesquisando
    preços, pois encontrei esse ótimo carregador de celular, nota 10.'
```

B. Relembrando alguns conceitos

- Token: termo de uma sentença, uma ocorrência particular de uma palavra específica em um texto
- Corpus: uma coleção de documentos/sentenças/frases
- Corpora: coletivo de corpus
- Corpus: Conjunto de textos
- Vocabulário: todas as palavras únicas em uma coleção de documentos de texto.

C. Normalizar capitalização

No exemplo a seguir vamos normalizar o texto, colocando todos os caracteres em minúsculo.

```
[ ] print ("Antes de normalizar: \n")
    print(exemplo)
    exemplo = exemplo.lower()
    print ("\n\n")
    print ("Depois de de normalizar: \n")
    print(exemplo)
```

```
Antes de normalizar:
EXCELENTE!! Valeu demais passar um tempo pesquisando preços,
pois encontrei esse ótimo carregador de celular, nota 10.


Depois de de normalizar:
excelente!! valeu demais passar um tempo pesquisando preços,
pois encontrei esse ótimo carregador de celular, nota 10.
```

Observe a coluna review_text do dataframe df. Vamos normalizá-la?

```
[ ] df['review_text']
```

	review_text
0	Perfeito....chegou antes do prazo.....
1	Foi uma ótima compra! Chegou antes mesmo do pr...
2	Recebi muito rapido e um otimo custo beneficio
3	Recomendo
4	Só veio uma capa comprei 3 aí paguei. Mais de ...
...	...
41739	SUCESSO :D
41740	Tudo OK. Produto entregue no prazo correto.
41741	Ultimamente estão atrasando muito as entregas
41742	recomendo
41743	A mascara veio com um pequeno defeito na estam...

41744 rows × 1 columns

Vamos normalizar a coluna toda? A função apply recebe uma outra função e a aplica a cada elemento de uma coluna ou linha de um DataFrame. Neste caso, vamos aplicar a função lower a cada item x de df['review_text'].

```
[ ] df['review_preproc'] = df['review_text'].apply(lambda x: x.lower())
    df
```

```
<ipython-input-23-92f642d50829>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['review_preproc'] = df['review_text'].apply(lambda x: x.lower())
```

	review_text	review_text_processed	polarity	review_preproc
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0	perfeito....chegou antes do prazo.....
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0	foi uma ótima compra! chegou antes mesmo do pr...
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0	recebi muito rapido e um otimo custo beneficio
3	Recomendo	recomendo	1.0	recomendo
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0	só veio uma capa comprei 3 aí paguei. mais de ...
...	...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0	sucesso :d
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0	tudo ok. produto entregue no prazo correto.
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0	ultimamente estão atrasando muito as entregas
41742	recomendo	recomendo	1.0	recomendo
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0	a mascara veio com um pequeno defeito na estam...

41744 rows × 4 columns

D. Retirar Pontuação:

Um pré-processamento necessário em algumas aplicações é o de remover pontuações.

» Opção 1 para remoção de caracteres: Analisar caractere a caractere e mandar na string sem pontuação somente os caracteres desejáveis.

```
[ ] exemplo_sem_punct = "".join(u for u in exemplo if
    u not in ("?", ".", ";", ":", "!", ","))
    exemplo_sem_punct
```

```
'excelente valeu demais passar um tempo pesquisando preços
pois' encontrei esse ótimo carregador de celular nota 10
```

Opção 2: Usar o RegexTokenizer da lib NLTK https://www.nltk.org/

```
[ ] from nltk.tokenize import RegexpTokenizer

    # com '\w+' selecionamos apenas tokens de palavras ou dígitos
    tokenizer = RegexpTokenizer(r'\w+')
```

```
[ ] exemplo = ' '.join(tokenizer.tokenize(exemplo))
    exemplo
```

```
'excelente valeu demais passar um tempo pesquisando preços
pois encontrei esse ótimo carregador de celular nota 10'
```

```
[ ] df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join(tokenizer.
    tokenize(x)))
    df
```

```
<ipython-input-29-2634f419f22d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join(tokenizer.
tokenize(x)))
```

	review_text	review_text_processed	polarity	review_preproc
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0	perfeito chegou antes do prazo
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0	foi uma ótima compra chegou antes mesmo do pra...
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0	recebi muito rapido e um otimo custo beneficio
3	Recomendo	recomendo	1.0	recomendo
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0	só veio uma capa comprei 3 aí paguei mais de 1...
...	...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0	sucesso d
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0	tudo ok produto entregue no prazo correto
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0	ultimamente estão atrasando muito as entregas
41742	recomendo	recomendo	1.0	recomendo
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0	a mascara veio com um pequeno defeito na estam...

41744 rows × 4 columns

E. Retirar acentos

- Processar caracteres em ascii é uma boa prática, por isso retiramos acentos.
- Opção 1: Criar um mapeamento de caracteres acentuados para caracteres sem acento

```
[ ] repl = str.maketrans(
        "áéúíó",
        "aeuio"
    )
    palavra = 'ótimo'
    palavra.translate(repl)
```

```
'otimo'
```

» Opção 2: Usar lib unidecode, https://github.com/avian2/unidecode.

```
[ ] !pip install unidecode
    import unidecode
```

```
Collecting unidecode
  Downloading Unidecode-1.3.8-py3-none-any.whl.metadata (13 kB)
Downloading Unidecode-1.3.8-py3-none-any.whl (235 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.5/235.5 kB 5.2 MB/s eta
0:00:00
Installing collected packages: unidecode
Successfully installed unidecode-1.3.8
```

```
[ ] exemplo.split()
```

```
['excelente',
 'valeu',
 'demais',
 'passar',
 'um',
 'tempo',
 'pesquisando',
 'preços',
 'pois',
 'encontrei',
 'esse',
 'ótimo',
 'carregador',
 'de',
 'celular',
```

continua

```
 'nota',
 '10']
```

```
[ ] exemplo = ' '.join([unidecode.unidecode(termo) for termo in exemplo.split()])
    exemplo
```

```
'excelente valeu demais passar um tempo pesquisando precos
pois encontrei esse otimo carregador de celular nota 10'
```

```
[ ] df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join([unidecode.
    unidecode(termo) for termo in x.split()]))
    df
```

```
<ipython-input-33-c21515af0c3e>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join([unidecode.
unidecode(termo) for termo in x.split()]))
```

	review_text	review_text_processed	polarity	review_preproc
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0	perfeito chegou antes do prazo
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0	foi uma otima compra chegou antes mesmo do pra...
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0	recebi muito rapido e um otimo custo beneficio
3	Recomendo	recomendo	1.0	recomendo
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0	so veio uma capa comprei 3 ai paguei mais de 1...
...	...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0	sucesso d
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0	tudo ok produto entregue no prazo correto
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0	ultimamente estao atrasando muito as entregas
41742	recomendo	recomendo	1.0	recomendo
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0	a mascara veio com um pequeno defeito na estam...

41744 rows × 4 columns

F. Retirar stopwords

» Dentro de um vocabulário, podem existir termos que se repetem muito e agregam pouca informação. Isso costuma acontecer com preposições e artigos, por exemplo. As palavras que ocorrem nesse sentido são separadas em uma lista, e então removidas. *Referência complementar: https://www.nltk.org/howto/portuguese_en.html*

» Opção 1: Podemos manualmente setar termos para retirada em uma lista

```
[ ] stopwords = ['de', 'para', 'uma', 'o', 'e'] # aqui listamos as palavras que mais se repetem, de acordo com o domínio de aplicação
```

» Opção 2: Podemos usar a lista de stopwords compilada pela lib NLTK

```
[ ] import nltk
    nltk.download('stopwords')
    stopwords = nltk.corpus.stopwords.words('portuguese') # este é um conjunto de palavras elencado por especialistas, os quais via estudo, identificaram que se repetem com frequencia.
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Veja quais são estas palavras:

```
[ ] stopwords
```

```
['a',
 'à',
 'ao',
 'aos',
 'aquela',
 'aquelas',
 'aquele',
 'aqueles',
 'aquilo',
 'as',
 'às',
 'até',
 'com',
 'como',
 'da',
 'das',
 'de',
 'dela',
 'delas',
 'dele',
 'deles',
```

continua

```
    'depois',
    'do',
    'dos',
    'e',
    'é',
    'ela',
    'elas',
    'ele',
    'eles',
    'em',
    'entre',
    'era',
    'eram',
    'éramos',
    'essa',
    'essas',
    'esse',
    'esses',
    'esta',
    'está',
    'estamos',
    'estão',
    'estar',
    'estas',
    'estava',
    'estavam',
    'estávamos',
    'este',
    'esteja',
    'estejam',
    'estejamos',
    'estes',
    'esteve',
    'estive',
    'estivemos',
    'estiver',
    'estivera',
    'estiveram',
    'estivéramos',
    'estiverem',
    'estivermos',
```

continua

```
    'estivesse',
    'estivessem',
    'estivéssemos',
    'estou',
    'eu',
    'foi',
    'fomos',
    'for',
    'fora',
    'foram',
    'fôramos',
    'forem',
    'formos',
    'fosse',
    'fossem',
    'fôssemos',
    'fui',
    'há',
    'haja',
    'hajam',
    'hajamos',
    'hão',
    'havemos',
    'haver',
    'hei',
    'houve',
    'houvemos',
    'houver',
    'houvera',
    'houverá',
    'houveram',
    'houvéramos',
    'houverão',
    'houverei',
    'houverem',
    'houveremos',
    'houveria',
    'houveriam',
    'houveríamos',
    'houvermos',
    'houvesse',
```

continua

```
    'houvessem',
    'houvéssemos',
    'isso',
    'isto',
    'já',
    'lhe',
    'lhes',
    'mais',
    'mas',
    'me',
    'mesmo',
    'meu',
    'meus',
    'minha',
    'minhas',
    'muito',
    'na',
    'não',
    'nas',
    'nem',
    'no',
    'nos',
    'nós',
    'nossa',
    'nossas',
    'nosso',
    'nossos',
    'num',
    'numa',
    'o',
    'os',
    'ou',
    'para',
    'pela',
    'pelas',
    'pelo',
    'pelos',
    'por',
    'qual',
    'quando',
    'que',
```

continua

```
    'quem',
    'são',
    'se',
    'seja',
    'sejam',
    'sejamos',
    'sem',
    'ser',
    'será',
    'serão',
    'serei',
    'seremos',
    'seria',
    'seriam',
    'seríamos',
    'seu',
    'seus',
    'só',
    'somos',
    'sou',
    'sua',
    'suas',
    'também',
    'te',
    'tem',
    'tém',
    'temos',
    'tenha',
    'tenham',
    'tenhamos',
    'tenho',
    'terá',
    'terão',
    'terei',
    'teremos',
    'teria',
    'teriam',
    'teríamos',
    'teu',
    'teus',
    'teve',
```

continua

```
 'tinha',
 'tinham',
 'tínhamos',
 'tive',
 'tivemos',
 'tiver',
 'tivera',
 'tiveram',
 'tivéramos',
 'tiverem',
 'tivermos',
 'tivesse',
 'tivessem',
 'tivéssemos',
 'tu',
 'tua',
 'tuas',
 'um',
 'uma',
 'você',
 'vocês',
 'vos']
```

Vamos remover as stopwords do nosso exemplo?:

```
[ ] exemplo = ' '.join([termo for termo in exemplo.split() if termo not in stopwords])
    exemplo
```

```
'excelente valeu demais passar tempo pesquisando precos
pois encontrei otimo carregador celular nota 10'
```

Vamos aplicar a remoção de stopwords em toda a coluna 'review_preproc'?

```
[ ] df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join([termo for
    termo in x.split() if termo not in stopwords]))
    df
```

```
<ipython-input-26-ac93617d81aa>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

continua

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join([termo for termo in x.split() if termo not in stopwords]))
```

	review_text	review_text_processed	polarity	review_preproc
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0	perfeito chegou antes prazo
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0	otima compra chegou antes prazo produto igual ...
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0	recebi rapido otimo custo beneficio
3	Recomendo	recomendo	1.0	recomendo
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0	so veio capa comprei 3 ai paguei 100 reais capa
...	...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0	sucesso d
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0	tudo ok produto entregue prazo correto
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0	ultimamente estao atrasando entregas
41742	recomendo	recomendo	1.0	recomendo
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0	mascara veio pequeno defeito estampa derretido...

41744 rows × 4 columns

G. Stemmização

» O objetivo do processo de stemmização é remover a terminação ou os últimos caracteres de uma palavra, de forma a deixar apenas o radical da mesma. Podem ser removidos, por exemplo, sufixos e vogais temáticas verbais.

» Exemplo:

1. Termos: Correr, correndo, correrei, ... Termo derivado da transformação após aplicação do stemmer: corr ou corre, a depender da implementação do algoritmo. Referência complementar: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

2. O algoritmo de stemmização 'rslp' usado via nltk foi proposto no artigo "A Stemming Algorithm for the Portuguese Language" by Viviane Moreira Orengo and Christian Huyck.

```
[ ] nltk.download('rslp')
    stemmer = nltk.stem.RSLPStemmer()
```

```
[nltk_data] Downloading package rslp to /root/nltk_data...
[nltk_data]   Unzipping stemmers/rslp.zip.
```

```
[ ] exemplo = ' '.join([stemmer.stem(termo) for termo in exemplo.split()])
    exemplo
```

```
'excel val demal pass temp pesquis prec poi encontr otim carreg celul
```

continua

```
[ ] df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join(stemmer.
    stem(termo) for termo in x.split()))
    df
```

```
<ipython-input-41-fc48b2967be1>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['review_preproc'] = df['review_preproc'].apply(lambda x: ' '.join(stemmer.
stem(termo) for termo in x.split()))
```

	review_text	review_text_processed	polarity	review_preproc
0	Perfeito....chegou antes do prazo.....	perfeito....chegou antes do prazo.....	1.0	perfeit cheg ant do praz
1	Foi uma ótima compra! Chegou antes mesmo do pr...	foi uma otima compra! chegou antes mesmo do pr...	1.0	foi uma otim compr cheg ant mesm do praz e o p...
2	Recebi muito rapido e um otimo custo beneficio	recebi muito rapido e um otimo custo beneficio	1.0	receb muit rap e um otim cust benefici
3	Recomendo	recomendo	1.0	recom
4	Só veio uma capa comprei 3 aí paguei. Mais de ...	so veio uma capa comprei 3 ai paguei. mais de ...	0.0	so vei uma cap compr 3 ai pag mais de 100 real...
...	...	...	...	...
41739	SUCESSO :D	sucesso :d	1.0	sucess d
41740	Tudo OK. Produto entregue no prazo correto.	tudo ok. produto entregue no prazo correto.	1.0	tud ok produt entreg no praz corret
41741	Ultimamente estão atrasando muito as entregas	ultimamente estao atrasando muito as entregas	0.0	ultim esta atras muit as entreg
41742	recomendo	recomendo	1.0	recom
41743	A mascara veio com um pequeno defeito na estam...	a mascara veio com um pequeno defeito na estam...	0.0	a masc vei com um pequen defeit na estamp com ...

41744 rows × 4 columns

H. Preciso sempre usar todas essas técnicas?:

Não. O importante é analisar o tipo de texto que se está lidando, o objetivo da tarefa e o algoritmo que será usado após isso. A análise cuidadosa do domínio e problema ajudam a alcançar o objetivo pretendido.

I. Desafios

1. **Implementação de Stopwords Personalizadas:**

» Pergunta: Como você pode adicionar sua própria lista de stopwords ao processo de pré-processamento? Sugestão: analise os textos do corpus Olist e identifique palavras específicas de domínio que se repetem com muita frequência.

» Tarefa: Implemente um exemplo que remova stopwords personalizadas de um texto.

2. **Stemming texto em itálico vs Lematização:**

» Pergunta: Qual é a diferença entre stemming e lematização?

» Tarefa: Implemente um exemplo de stemming usando NLTK e compare os resultados com a lematização do spaCy (https://www.educative.io/answers/how-to-remove-stop-words-using-spacy-in-python).

SAIBA MAIS...

Para o(a) leitor(a) mais interessado em aprofundar nos temas, sugerimos os Capítulos 1 e 2 do Livro **Speech and Language Processing**, dos autores Dan Jurafsky e James H. Martin (2024).

CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Métodos Clássicos de Processamento de Linguagem Natural

Unidade II

Formas clássicas de representação de palavras

Unidade II: Formas Clássicas de Representação de Palavras

Neste Capítulo, trataremos de algo que parece simples, mas não é: identificar a unidade mínima quando tratamos, computacionalmente, a língua. Essa delimitação não é consensual entre pesquisadores e profissionais de PLN. E, mesmo em linguística, há sempre controvérsias e necessidade de pontos de referência para se definir, por exemplo, o que seja uma palavra ou mesmo uma frase.

As subáreas especializadas dos estudos linguísticos entendem como unidade mínima de processamento diferentes elementos conforme seus focos e pontos de vista. A fonologia, por exemplo, considera o fonema como a menor unidade sonora e distintiva de uma língua. Se tomarmos o exemplo do que diferencia as palavras “sábia” (mulher inteligente), “sabiá” (pássaro) e “sabia” (verbo “saber”), percebemos que a sílaba tônica é o diferencial, especialmente quando pensamos em sons e fala e não em escrita.

Já a morfologia considera o morfema como a menor unidade dotada de significado na língua. Nessa perspectiva, temos os “pedacinhos” de palavras e seus valores, como seria o caso da marca de diminutivo “-inho”, que assinala o masculino e o singular em “menininho”, ou o segmento “-ei” no verbo “comprei”, que marca um modo-tempo (pretérito perfeito do modo indicativo) e também um número-pessoa (primeira pessoa do singular).

Assim, conforme o ponto de vista de quem analisa, uma palavra pode ser feita de sons e de sílabas tônicas e/ou composta de vários segmentos gráficos menores. Podemos, ainda, considerar segmentos ou pedaços mais abrangentes, conforme o critério que utilizamos. Um exemplo nessa linha seria a palavra “guarda-pó”, que pode ser considerada como uma palavra só ou a junção de duas palavras. Outro caso ilustrativo é “escova de dente”, que, para alguns, é a união de três palavras, e, para outros, é uma palavra só, mesmo que não tenha hífen. Além dessas questões, também é controverso tratar das abreviaturas, siglas, interjeições, dos modos de escrita diferenciados nas redes sociais com “internetês”, *hashtags*, *emojis*, símbolos e outras peculiaridades.

Fazendo um paralelo, podemos entender que, de modo geral, os modelos de PLN trabalham as palavras como unidade primária de processamento. Vejamos, por exemplo, o caso da frase no Exemplo 1.

Exemplo 1: Jacinta Maria comprou uma cadeira em São Paulo ontem e pagou 25 reais por ela.

Na frase do Exemplo 1, são 15 palavras se considerarmos que é palavra toda a sequência de caracteres separada por um espaço em branco. Mas se pode pensar que Jacinta+Maria e São+Paulo são palavras compostas e que, talvez, o número 25 não seja bem uma palavra, não? A resposta será: depende do critério que você usar e da finalidade que tem ou busca com essa referência de unidade e/ou partes.

Ao fazer o processamento computacional de textos escritos, a definição de que tipo de unidade de processamento se quer buscar/estudar parece estar atrelada às necessidades da tarefa ou do trabalho pretendidos. Geralmente, considera-se que uma palavra é, simplesmente, uma unidade grafológica delimitada, nas línguas europeias, entre espaços em branco, na representação gráfica, ou entre um espaço em branco e um sinal de pontuação. Essa é uma definição bastante concreta e bastante prática. No entanto, ao pensarmos em nossos modelos computacionais e suas aplicações no mundo, é importante nos aprofundarmos um pouco mais na conceituação do que é uma palavra e nas possibilidades de processamento e implicações das decisões tomadas no pré-processamento dos *corpora*.

Segundo Cabré (1999), as palavras são as unidades de referência da realidade empregadas pelos falantes. De acordo com essa definição, as palavras compõem a dimensão linguística mais estreitamente ligada ao mundo real. Ainda segundo a mesma autora, o léxico consiste no conjunto das palavras de uma língua e dos padrões que possibilitam a criatividade do falante. As palavras e, principalmente, as associações infinitas e imprevisíveis que os seres humanos são capazes de traçar entre elas, constituem a manifestação mais concreta e mais produtiva da língua. Assim, é importante que, ao processar dados textuais para gerar modelos computacionais, nos recordemos sempre de que não estamos simplesmente organizando um conjunto de caracteres ou ordenando uma representação ortográfica formal, mas que estamos trabalhando com recursos linguísticos que representam a experiência humana.

Em seguida, devemos considerar também o conceito de palavra computacional, que se refere a uma unidade linguística que foi adaptada ou criada especificamente para facilitar seu processamento por máquinas. Isso pode envolver a manipulação de palavras, frases ou até mesmo caracteres de maneira que seja mais conveniente para algoritmos e sistemas de PLN lidarem com elas.

A necessidade dessas palavras computacionais surge devido às complexidades do PLN por máquinas. A linguagem humana é rica e ambígua, cheia de nuances e variações que podem ser difíceis de interpretar e analisar automaticamente. Portanto, ao transformar palavras em formas mais padronizadas ou simplificadas, os sistemas

de PLN podem executar tarefas como análise gramatical, extração de informações e tradução com maior eficácia. Dependendo do objetivo da tarefa ou da aplicação de PLN, é possível definir quais rotinas de pré-processamento são mais produtivas para criar as palavras computacionais, ou seja, pode-se remover ou não acentos ortográficos, espaços em branco em itens como “fim de semana”, ou hífens como em “guarda-chuva”.

2.1 Formas Clássicas de Representação de Palavras

A maneira mais simples de representar palavras é como entidades independentes e não relacionadas. Você pode pensar nisso como um conjunto, tal como o exemplo:

{..., chá, café, coragem, retrato, ...} .

Um tipo (palavra) é um elemento de um vocabulário; uma palavra em abstrato. Um **token** (palavra) é uma instância de um tipo no contexto.

Aqui vamos apresentar um pouco de terminologia. Iremos nos referir a um **tipo** de palavra como um elemento de um **vocabulário finito (léxico)**, independentemente da observação real da palavra no contexto. Então, acabamos de escrever um conjunto de tipos (conforme exemplo acima). Um *token* é uma instância do tipo, por exemplo, observado em algum contexto. Nossas representações de palavras agora fornecem uma representação única para cada tipo de palavra, e podemos usar essa mesma representação para qualquer ocorrência da palavra ***token*** no contexto. Frequentemente, trabalharemos com vetores neste Microcurso; a representação vetorial convencional de *tokens* é o conjunto de vetores ***1-hot***, ou de base padrão. Assim:

$$v_{chá} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad v_{café} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{1}$$

onde $v_{chá} = e_3$, é o terceiro elemento do vetor base e $v_{café} = e_j$ é o j^{th} elemento do vetor base.

Por que representamos palavras como vetores? Para melhor computar com eles, ou seja, realizar operações matemáticas com as palavras. E ao calcular com vetores *1-hot*, alcançamos o fato crucial de que palavras diferentes são diferentes, mas, infelizmente, não codificamos nenhuma noção significativa de similaridade ou outro relacionamento. Isso ocorre porque, por exemplo, se tomarmos o produto escalar como uma noção de similaridade (ou a distância L2, ou a distância L1, ou...) calcularmos:

$$v_{chá}^{\mathsf{T}} v_{café} = v_{chá}^{\mathsf{T}} v_{coragem} = 0, \tag{2}$$

todas as palavras são igualmente diferentes umas das outras. Observe, também, que, no exemplo anterior, as palavras não estão ordenadas, por exemplo, em ordem alfabética - isso é uma nota importante; não há informações (explícitas) em nível de caractere nesses *tokens*, além da noção estrita de identidade (essa palavra tem a mesma sequência de caracteres/bytes que essa outra palavra. Se sim, elas têm o mesmo vetor; senão, elas têm vetores independentes).

Como não é verdade que todas as palavras sejam igualmente diferentes umas das outras, passaremos por algumas alternativas para representação vetorial das palavras.

2.1.1 Vetores a Partir de Propriedades Discretas

Para qualquer palavra, há uma riqueza de informações que podemos anotar sobre essa palavra. Há informações gramaticais, como pluralidade, há informações derivacionais, como por exemplo a palavra “corredores” é derivada do verbo correr mais uma noção de “fazedor” ou agente (pense em alguém que corre). Há também informações semânticas, como os corredores podem ser um **hipônimo** de humanos, animais ou entidades - um hipônimo é membro de um relacionamento *é-um*; por exemplo, um corredor é um humano.

Existem recursos substanciais em inglês e em alguns outros idiomas para vários tipos de informações anotadas sobre palavras. WordNet (Miller, 1995), por exemplo, anota sinônimos, hipônimos e outras relações semânticas; UniMorph (Nielsen, 2011) anota informações de morfologia (estrutura de subpalavras) em vários idiomas. Com esses recursos, seria possível construir vetores de palavras parecidos com:

$$v_{chá} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \begin{matrix} \text{(plural)} \\ \text{(verbo na terceira pessoa)} \\ \text{Hipônimo de bebida} \\ \\ \text{sinônimo de chai} \end{matrix}$$

A principal vantagem da representação das palavras por meio de vetores *1-hot* é que ele é uma abordagem fácil de entender e implementar. Essa representação também é útil quando os dados não têm uma relação de ordem ou hierarquia, pois todos os vetores estão igualmente distantes uns dos outros.

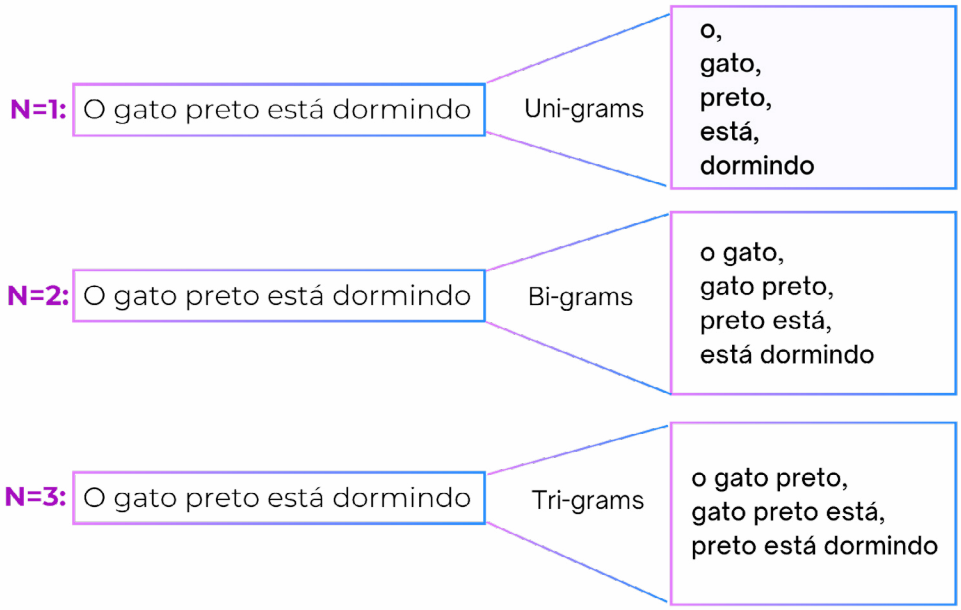
No entanto, o *1-hot* tem várias desvantagens. A principal é que ele pode ser muito ineficiente, especialmente para conjuntos de dados com um grande vocabulário. Além disso, ele não captura nenhuma informação sobre as relações semânticas entre as palavras. Por exemplo, não há maneira de dizer que "chá" e "café" são ambos bebidas e estão mais relacionados entre si do que com "casa". Para superar essas limitações, os pesquisadores desenvolveram métodos mais avançados de representação de palavras, como os *embeddings* de palavras ou *word embeddings* (por exemplo, Word2Vec - técnica abordada na Unidade IV deste *e-book*), que pode capturar a semântica e as relações entre as palavras e sentenças de uma maneira muito mais eficiente e rica.

2.1.2 *Bag-of-Words*

Bag-of-Words (BoW) é uma forma de representar o texto (sentença ou documento) de acordo com a ocorrência das palavras nele. Traduzindo para o português, o "saco de palavras" recebe esse nome porque não leva em conta a ordem ou a estrutura das palavras no texto, apenas se a palavra aparece ou a frequência com que aparece nele. Na abordagem BoW, o texto é dividido em palavras individuais, e essas palavras são tratadas como as características ou "*tokens*" do texto. Quando temos *tokens* de tamanho 1 dá-se o nome de *unigramas,* quando tamanho 2, *bigramas,* de tamanho n, *n-gramas*. N-gramas podem ser úteis para capturar relações entre palavras que aparecem frequentemente juntas, como "inteligência artificial" ou "redes neurais". Por exemplo, na frase "o gato preto está dormindo", trigramas (n=3) seriam ["o gato preto", "gato preto está", "preto está dormindo"]. O uso de n-grams maiores, como trigramas ou até quadrigramas, permite capturar padrões e contextos mais complexos, mas também aumenta significativamente o espaço de características, o que pode levar à necessidade de técnicas de redução de dimensionalidade.

Para gerar a BoW, em primeiro lugar, é construído um **vocabulário** de todas as palavras únicas em um conjunto de documentos. Cada documento é então representado por um **vetor**, onde cada posição no vetor corresponde a uma palavra do vocabulário. Os **valores no vetor** podem ser **binários** (1 se a palavra está presente no documento e 0 se não está) ou a **frequências** com número de vezes que a palavra aparece no documento. Assim, é necessário seguir um *pipeline (metodologia),* como o exemplificado na Figura 3, a seguir.

Figura 3 - Exemplo de *bag-of-words* (BoW)



Fonte: autoria própria.

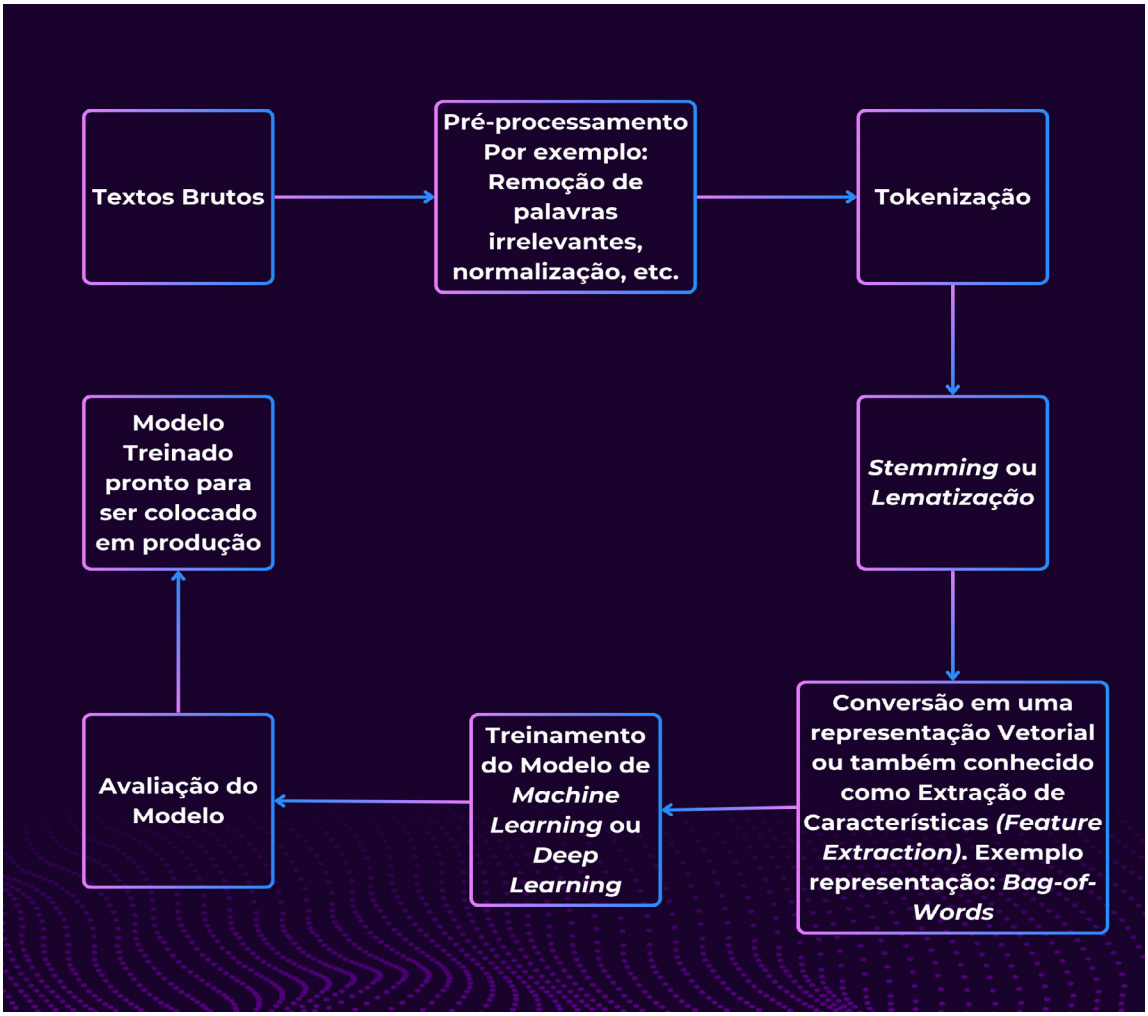
Inicialmente, coletamos os textos que serão usados tanto para o treinamento do modelo quanto para os testes e avaliação. Essa etapa é muito importante, pois os resultados refletirão os dados de entrada (se entrar dados ruidosos ou inapropriados o resultado refletirá o mesmo padrão. Na sequência, os dados de entrada são pré-processados e operações como:

1. **Tokenização**: a divisão de um texto em unidades menores, como palavras ou frases.
2. **Remoção de *stopwords***: *stopwords* são palavras comuns (como “e”, “de”, “o”) que geralmente não adicionam muito valor semântico e são removidas para reduzir o ruído nos dados.
3. **Lematização e stemmatização**: redução de palavras às suas formas básicas ou raízes.
4. **Normalização de texto**: inclui a conversão de todo o texto para letras minúsculas e a remoção de pontuação para uniformizar os dados.

5. **Remoção de números**: em alguns casos, os números podem ser irrelevantes e são removidos para simplificar a análise.
6. **Substituição de Localizador Uniforme de Recursos (URLs), emojis e outras entidades**: URLs, emojis e menções a usuários (em redes sociais) são substituídos ou removidos, dependendo do contexto.
7. **Correção ortográfica**: em textos que podem conter erros ortográficos, essa etapa é utilizada para corrigir essas falhas.

Na etapa seguinte, o texto processado é então convertido em uma representação vetorial, a qual será a entrada de um modelo de aprendizado de máquina (*machine learning*) ou um algoritmo baseado em redes profundas (*deep learning*) (Figura 4).

Figura 4 - *Pipeline*



Fonte: autoria própria.

Exemplo: considere os documentos D1, D2, D3 e a *bag-of-words* obtida na Tabela 1, a seguir.

- D1: “João é mais rápido que Maria”
- D2: “Maria é mais rápida que João”
- D3: “Rubinho perdeu!”

Uma BoW possível para o *corpus* anterior pode ser visualizado na Tabela 1. Observe que a frequência utilizada no exemplo foi a frequência binária.

Tabela 1 - Exemplo de *bag-of-words* obtida a partir do conjunto de textos do exemplo anterior

	joão	mais	rápido	maria	rubinho	perdeu
Documento D1	1	1	1	1	0	0
Documento D2	1	1	1	1	0	0
Documento D3	0	0	0	0	1	1

Fonte: autoria própria.

Para exemplificar todo o conteúdo visto nessa Unidade, continue seus estudos com o *Notebook*, a seguir.

Notebook Colab

Objetivos de Aprendizagem

- Esse notebook é um tutorial breve sobre formas clássicas de representação de palavras. Para exemplicar, iremos implementar uma função containment, uma função que irá comparar dois textos e analisar a similaridade dos mesmos com relação aos seus n-gramas de interseção. Primeiramente iremos entender os conceitos de vocabulário, n-gramas para posteriormente implementar a função.

A. Contar N-gram

Primeiramente temos que contar as ocorrências de n-gramas dos nossos textos. Usaremos o CountVectorizer para converter nosso corpus em uma matriz.

No código abaixo, podemos variar o valor de n e utilizar o CountVectorizer para contar as ocorrências de n gramas. Podemos notar que na célula abaixo estamos criando um vocabulário através da utilização do CountVectorizer e, posteriormente, iremos analisar a matriz.

```
[ ] import numpy as np
    import sklearn
```

B. Unigrama

A execução do exemplo imprime o vocabulário. Podemos ver que existem 8 palavras(tokens) no vocabulário e, portanto, vetores codificados também possuem um comprimento de 8.

```
[ ] from sklearn.feature_extraction.text import CountVectorizer


    texto_a_ser_comparado = "Suponha que esse seja o texto que desejo comparar"
    texto_fonte = "Suponha que essa seja o texto principal"
    texto_fonte2 = "Texto nada a ver com o que eu quero"

    # Número de n_gramas
    n = 1

    # Instancia o contador de n-gramas
    counts = CountVectorizer(analyzer='word', ngram_range=(n,n))

    # Cria um dicionário de n-gramas
    vocab2int = counts.fit([texto_a_ser_comparado, texto_fonte, texto_fonte2]).
    vocabulary_

    # Imprime dicionário de palavras:index
    print(vocab2int)

    {'suponha': 11, 'que': 8, 'esse': 4, 'seja': 10, 'texto': 12, 'desejo': 2, 'comparar':
    1, 'essa': 3, 'principal': 7, 'nada': 6, 'ver': 13, 'com': 0, 'eu': 5, 'quero': 9}
```

C. Bigrama

O mesmo vale para o caso de bigramas. Temos 8 bigramas no vocabulário e, portanto, os vetores codificados com comprimento 8.

```
[ ] # Número de n_gramas
    n = 2

    # Instancia o contador de n-gramas
    counts = CountVectorizer(analyzer='word', ngram_range=(n,n))

    # Cria um dicionário de n-gramas
    vocab2int = counts.fit([texto_a_ser_comparado, texto_fonte, texto_fonte2]).
    vocabulary_

    # Imprime dicionário de palavras:index
    print(vocab2int)
```

```
{'suponha que': 11, 'que esse': 8, 'esse seja': 3, 'seja texto': 10, 'texto que':
14, 'que desejo': 6, 'desejo comparar': 1, 'que essa': 7, 'essa seja': 2, 'texto
principal': 13, 'texto nada': 12, 'nada ver': 5, 'ver com': 15, 'com que': 0, 'que
eu': 9, 'eu quero': 4}
```

D. Trigrama

```
[ ] # Número de n_gramas
    n = 3

    # Instancia o contador de n-gramas
    counts = CountVectorizer(analyzer='word', ngram_range=(n,n))

    # Cria um dicionário de n-gramas
    vocab2int = counts.fit([texto_a_ser_comparado, texto_fonte, texto_fonte2]).
    vocabulary_

    # Imprime dicionário de palavraś:index
    print(vocab2int)
```

```
{'suponha que esse': 11, 'que esse seja': 6, 'esse seja texto': 2, 'seja texto
que': 9, 'texto que desejo': 13, 'que desejo comparar': 4, 'suponha que essa':
10, 'que essa seja': 5, 'essa seja texto': 1, 'seja texto principal': 8, 'texto
nada ver': 12, 'nada ver com': 3, 'ver com que': 14, 'com que eu': 0, 'que eu
quero': 7}
```

E. As palavras do vocabulário

Note que o artigo "o" das frases texto_a_ser_comparado e texto_fonte não aparece no vocabulário. Note ainda que todas as frases encontram-se em minúsculo. Isso ocorre devido ao fato de que quando passamos o parâmetro analyser = 'word', estamos considerando em nossa análise palavras com dois ou mais caracteres e consequentemente ignorando as palavras com apenas um caracter. Excluir esses caracteres (artigos) é um comportamento padrão e desejado em muitas análises de texto devido a sua irrelevância, em grande parte das análises textuais.

Caso você precise desconsiderar o padrão *default* do CountVectorizer e adicionar palavras com caracteres únicos em sua análise, você pode adicionar o argumento token_pattern = r"(?u)\b\w+\b". Essa expressão regular (REGEX) define palavra como tendo uma ou mais caracteres.

F. Array de n-gramas

Vamos usar o CountVectorizer para criar um array com as contagens de n-gramas. Além disso, vamos criar duas sentenças que desejamos analizar, e transformar cada texto em um vetor numérico representando a ocorrência de cada palavra.

Notar que cada linha representa um texto e cada coluna ou index representa os termos do vocabulário. Iremos ver isso claramente no mapeamento abaixo.

- texto_a_ser_comparado = "Suponha que essa seja o texto que desejo comparar"
- texto_fonte = "Suponha que essa seja o texto principal"

```
[ ] # N-gramas
    n = 1

    # Instancia o contador de n-gramas
    counts = CountVectorizer(analyzer='word', ngram_range=(n,n))

    # cria uma matriz de contagem de n-grama para os dois textos
    n_grams = counts.fit_transform([texto_a_ser_comparado, texto_fonte, texto_
    fonte2])

    # Cria um dicionário de n-gramas
    vocab2int = counts.fit([texto_a_ser_comparado, texto_fonte, texto_fonte2]).
    vocabulary_

    n_grams_array = n_grams.toarray()
```

continua

```
print('Vetor de n-gramas:\n\n', n_grams_array)
print()
print('Dicionário de n-gramas (unigrama):\n\n', vocab2int)
```

```
Vetor de n-gramas:

 [[0 1 1 0 1 0 0 0 2 0 1 1 1 0]
 [0 0 0 1 0 0 0 1 1 0 1 1 1 0]
 [1 0 0 0 0 1 1 0 1 1 0 0 1 1]]

Dicionário de n-gramas (unigrama):

 {'suponha': 11, 'que': 8, 'esse': 4, 'seja': 10, 'texto': 12, 'desejo': 2,
'comparar': 1, 'essa': 3, 'principal': 7, 'nada': 6, 'ver': 13, 'com': 0, 'eu': 5,
'quero': 9}
```

```
[ ] texto_a_ser_comparado = "Suponha que essa seja o texto que desejo comparar"
    texto_fonte = "Suponha que essa seja o texto principal"
```

Acima temos os vetores que codificam cada texto. Na linha superior temos os n-gramas do text_a_ser_comparado e na linha inferior temos a codificação do text_fonte. Podemos analisar se os textos possuem n_gramas em comum através de suas colunas. Por exemplo, ambos possuem a palavra texto (índice 7 - ultima coluna coluna). O mesmo vale para os unigramas [essa], [seja], [que] e [suponha]. Notar que o unigrama [que] ocorre duas vezes no segundo texto.

```
[[1 1 1 0 2 1 1 1]     =    comparar  desejo [essa] _________ [que] [seja]
[suponha] [texto]
 [0 0 1 1 1 1 1 1]]    =    ________  ______ [essa] principal [que] [seja]
[suponha] [texto]
```

G. Valores de *containment*

O Containment nada mais é do que uma medida de similaridade entre textos. É basicamente uma normalização da interseção da contagem de n-gramas entre os textos.

Primeiro, precisamos extrair as palavras dos dois documentos de texto para formar um corpus. Em seguida, contamos a interseção de n-gramas (agrupamentos sequenciais de palavras de n palavras) entre os textos. Para o caso de unigramas, podemos considerar como uma contagem dos número de palavras que ambos os

textos têm em comum.

Em seguida, dividimos o valor pelo total de n-gramas do texto a ser comparado (subíndice A - o qual quer ser comparado com o texto fonte) para normalizar o valor.

Cálculo de *Containment*:

1. Calcular a interseção n-grama entre o texto e o texto fonte.
2. Adicionar o número de termos comuns.
3. Normalizar o valor na etapa 2 pelo número de n gramas no texto A.

Abaixo podemos ver a equação de Containment:

$$\frac{\sum count(\text{ngram}_A) \cap count(\text{ngram}_F)}{\sum count(\text{ngram}_A)}$$

H. Vamos criar uma função que recebe um array n-gramas

```
[ ] def containment(n_gram_array):
        ''' Calcula o containment entre dois textos. Normaliza a interseção dos
    contadores de n-gramas
        entre os textos.
        ARG:
        n_gra_array(array): Um array com as contagens de n-gramas dos dois textos
    a serem comparados

        RETURNS:
        O valor de containment normalizado '''


         # Cria uma lista que contém o valor mínimo encontrado nas colunas
         # 0 se não houver correspondências e 1+ para as palavras correspondentes

        intersection_list = np.amin(n_gram_array, axis = 0)

        # Soma número de interseção
        intersection_count = np.sum(intersection_list)

        # Conta número de n-gramas no texto 1
```

continua

```
        A_idx = 0
        A_count = np.sum(n_gram_array[A_idx])


        # Normaliza e calcula valor final
        containment_val = intersection_count / A_count

        return containment_val
```

Para o n_gram calculado anteriormente e n = 1

```
[ ] containment_val = containment(n_grams.toarray())

    print('Containment: ', containment_val)
```

```
Containment:  0.25
```

para n = 2

```
[ ] counts_2grams = CountVectorizer(analyzer='word', ngram_range=(2,2))
    bigram_counts = counts_2grams.fit_transform([texto_a_
    ser_comparado, texto_fonte])

    # Calcula containment
    containment_val = containment(bigram_counts.toarray())

    print('Containment for n=2 : ', containment_val)
```

```
Containment for n=2 :  0.5714285714285714
```

I. Exercício:

Teste a função com diferentes textos , n-gramas e tente imaginar aplicações desse conceito. Por exemplo, podemos usar essa técnica como uma métrica de análise de similaridade para detectar plagiarismo.

J. Similaridade Cosseno

Outra métrica usada para cálculo de similaridade entre textos é o cosseno. Quem já cursou álgebra linear deve lembrar que o produto escalar de dois vetores é igual ao módulo de cada vetor vezes o cosseno do ângulo formado entre eles. E é justamente o cosseno que utilizaremos para determinar o quanto similar são os textos. Lembrando que a função cosseno vai do intervalo fechado -1 à 1, sendo que, quando:

- cosseno = 1 -> Os vetores são paralelos e com mesmo sentido,
- cosseno = 0 -> Os vetores são perpendiculares,
- cosseno = -1 -> Os vetores são paralelos e com sentido oposto.

De maneira simples, quanto maior for o valor do cosseno mais similar são as sentenças. Vamos testar?

```
[ ] from sklearn.metrics.pairwise import cosine_similarity

    texto_a_ser_comparado = "Suponha que esse seja o texto que desejo comparar"
    texto_fonte = "Suponha que essa seja o texto principal"
    texto_fonte2 = "Texto nada a ver com o que eu quero"
    texto_fonte3 = "Suponha que esse seja o texto que desejo comparar

    # N-gramas
    n = 1

    # Instancia o contador de n-gramas
    counts = CountVectorizer(analyzer='word', ngram_range=(n,n))

    # cria uma matriz de contagem de n-grama para os dois textos
    x = counts.fit_transform([texto_a_ser_comparado, texto_fonte, texto_fonte2,
    texto_fonte3])

    cosine_similarity(x[0],x[3])

array([[1.]])
```

K. Desafios

1. Vamos implementar um rankeador de documentos? O desafio é, dada uma **string** de busca denominada “query”, rankear quais os documentos são mais similares em ordem de similaridade, usando a similaridade cosseno. Realize os seguintes Pré-processamentos de Texto: Tokenização, Remova as **stopwords**, Lematize as palavras.

```
[ ] documents = [
        "Natural Language Processing (NLP) enables computers to understand human language.",
        "Machine learning provides systems the ability to automatically learn and improve from experience.",
        "Deep learning is a subset of machine learning involving neural networks with three or more layers.",
        "Natural Language Processing includes tasks like text generation and sentiment analysis.",
        "Supervised learning involves learning from a training dataset with labeled data."
    ]
```

Perguntas para Reflexão:

1. Qual é a importância de pré-processar os textos antes de calcular a similaridade?

O pré-processamento de textos é importante para garantir que os dados estejam limpos, consistentes e prontos para serem analisados/ usados em outros algoritmos, por exemplo, análise de sentimentos, classificação, análise de tópicos. O pré-processamento de textos envolve etapas como remoção de stop words, stemming, lematização, normalização de maiúsculas/minúsculas e remoção de caracteres especiais.

Mas já parou para pensar que ao remover artigos ou preprosições podemos perder a semântica de expressões importantes para o contexto?

2. Como a vetorização ajuda a capturar a importância das palavras nos documentos?

A vetorização converte textos em representações numéricas, geralmente vetores de palavras ou tokens. Técnicas como TF-IDF (Term Frequency-Inverse Document Frequency) ajudam a atribuir pesos às palavras com base em sua frequência no documento e no corpus, destacando termos mais representativos. Isso evita que palavras muito comuns (como “de”, “e”, “é”) tenham peso excessivo, focando nos

termos que ajudam a diferenciar um documento de outro. Vetores gerados por técnicas como embeddings de palavras (Word2Vec, BERT) também capturam relações semânticas entre palavras.

3. Por que a similaridade cosseno é uma métrica apropriada para medir a similaridade entre documentos de texto?

A similaridade cosseno mede o ângulo entre dois vetores, independentemente de sua magnitude, sendo especialmente útil quando os documentos têm comprimentos variáveis. Essa métrica se concentra na direção dos vetores (ou seja, nos padrões de coocorrência de palavras) em vez de suas magnitudes absolutas, o que ajuda a captar a semelhança no conteúdo, ignorando diferenças triviais no tamanho dos documentos. Como o valor resultante varia entre -1 e 1, onde 1 significa alta similaridade e 0 significa ausência de similaridade, é uma forma eficiente de comparar documentos de texto.

SAIBA MAIS...

Para o(a) leitor(a) mais interessado em aprofundar nos temas, sugerimos a leitura do Capítulo 3 do livro a seguir:

- MARTIN, J. H.; JURAFSKY, D.. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River, N.J.: Pearson/Prentice Hall, 2009.

Métodos Clássicos de Processamento de Linguagem Natural

Unidade III

tf-idf

Unidade III: *Term Frequency-Inverse Document Frequency* (TF-IDF)

3.1 Introdução

TF-IDF (*Term Frequency-Inverse Document Frequency*) (Baccianella *et al.*, 2010) é uma técnica muito utilizada em PLN para avaliar a importância de uma palavra em um documento dentro de um *corpus* (conjunto de documentos). É uma medida estatística que reflete a importância de uma palavra em relação à frequência de ocorrência dela no documento e no *corpus* em geral.

A técnica TF-IDF combina duas métricas:

1. ***Term Frequency* (TF)**: mede a frequência com que um termo aparece em um documento. A ideia básica é que quanto mais um termo aparece em um documento, mais importante ele é para aquele documento. A fórmula para o TF de um termo t em um documento d é dada por:

$$\text{TF}(t, d) = \frac{\text{Número de vezes que o termo } t \text{ aparece no documento } d}{\text{Número total de termos no documento } d}$$

2. ***Inverse Document Frequency* (IDF)**: mede a importância de um termo em todo o *corpus*. A intuição é que termos que aparecem em muitos documentos são menos informativos e, portanto, menos importantes do que termos que aparecem em poucos documentos. A fórmula para o IDF de um termo t é:

$$\text{IDF}(t) = \log\left(\frac{\text{Número total de documentos no corpus}}{\text{Número de documentos que contêm o termo } t}\right)$$

Às vezes, é adicionado 1 no denominador para evitar divisão por zero.

$$\text{IDF}(t) = \log\left(\frac{\text{Número total de documentos no corpus}}{1 + \text{Número de documentos que contêm o termo } t}\right)$$

3. **TF-IDF**: é o produto do TF e do IDF. A fórmula é:

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

Assim, com a técnica BoW vista anteriormente, o TF-IDF pode ser usado para transformar documentos textuais em vetores de características (representação vetorial) que podem ser usados como entrada para algoritmos de aprendizado de máquina. A simplicidade é uma das principais vantagens do TF-IDF, pois essa forma de representação além de funcionar bem para muitas tarefas de PLN e recuperação de informação, também é fácil de implementar.

Como limitações podemos enumerar:

- **Não considera a semântica**: TF-IDF trata palavras isoladamente e não considera a relação semântica entre elas.
- **Sensível a dados ruidosos**: termos raros, erros de digitação e palavras fora de contexto podem afetar os cálculos de TF-IDF.
- **Escalabilidade**: em conjuntos de dados muito grandes, o cálculo do IDF pode ser computacionalmente caro.

TF-IDF continua sendo uma ferramenta importante em PLN, apesar de avanços em modelos baseados em aprendizado profundo, como *word embeddings* (Unidade IV deste *e-book*) e *transformers* (Pennington; Socher; Manning, 2014), que podem capturar relações semânticas mais complexas.

3.2 Exemplos

Vamos aplicar a técnica TF-IDF a um exemplo simples de textos de jornal ou revista. Suponha que temos os seguintes três documentos (D1, D2, e D3):

- **D1**: “O governo anunciou novas medidas econômicas.”
- **D2**: “A economia do país cresceu no último trimestre.”
- **D3**: “Novas políticas foram implementadas pelo governo.”

a) Passo 1: cálculo do *Term Frequency* (TF)

Primeiro, contamos a frequência de cada termo em cada documento. Para simplificação, não vamos considerar palavras comuns (*stopwords*) e vamos aplicar uma simples normalização, como transformar todas as palavras em minúsculas (Tabela 2).

Tabela 2 - Frequência dos termos (TF) para os exemplos de documentos

Documento	Frequência dos Termos	TF
D1	• "governo" aparece 1 vez • "anunciou" aparece 1 vez • "novas" aparece 1 vez • "medidas" aparece 1 vez • "econômicas" aparece 1 vez	• "governo": 1/5 • "anunciou": 1/5 • "novas": 1/5 • "medidas": 1/5 • "econômicas": 1/5
D2	• "economia" aparece 1 vez • "país" aparece 1 vez • "cresceu" aparece 1 vez • "último" aparece 1 vez • "trimestre" aparece 1 vez	• "economia": 1/5 • "país": 1/5 • "cresceu": 1/5 • "último": 1/5 • "trimestre": 1/5
D3	• "novas" aparece 1 vez • "políticas" aparece 1 vez • "implementadas" aparece 1 vez • "pelo" aparece 1 vez • "governo" aparece 1 vez	• "novas": 1/5 • "políticas": 1/5 • "implementadas": 1/5 • "pelo": 1/5 • "governo": 1/5

Fonte: autoria própria.

b) Passo 2: Cálculo do *Inverse Document Frequency* (IDF)

Agora, calculamos o IDF para cada termo. O IDF é dado por:

$$\text{IDF}(t) = \log\left(\frac{N}{1 + \text{df}(t)}\right)$$

onde N é o número total de documentos, e $df\,(t)$ é o número de documentos que contêm o termo t.

» "governo" aparece em dois documentos (D1, D3):

$$IDF(\text{“}governo\text{”}) = \log\left(\frac{3}{1+2}\right) = \log\,(1) = 0$$

» “anunciou” aparece em um documento (D1):

$$IDF(\text{“}anunciou\text{”}) = \log\left(\frac{3}{1+1}\right) = \log(1.5) \simeq 0.405$$

E o mesmo é feito para todas as outras palavras.

c) Passo 3: cálculo do TF-IDF

Multiplicamos o TF pelo IDF para cada termo em cada documento.

Tabela 3 - TF-IDF para os termos dos documentos exemplos

Documento	TF-IDF
D1	• “governo”: (1/5) x 0 = 0 • “anunciou”: (1/5) x 0.405 = 0.081 • “novas”: (1/5) x 0 = 0 • “medidas”: (1/5) x 0.405 = 0.081 • “econômicas”: (1/5) x 0.405 = 0.081
D2	• “economia”: (1/5) x 0.405 = 0.081 • “país”: (1/5) x 0.405 = 0.081 (1/5) • “cresceu”: (1/5) x 0.405 = 0.081 • “último”: (1/5) x 0.405 = 0.081 (1/5) • “trimestre”: (1/5) x 0.405 = 0.081 (1/5)
D3	• “novas”: (1/5) x 0 = 0 • “políticas”: (1/5) x 0.405 = 0.081 • “implementadas”: (1/5) x 0.405 = 0.081 • “pelo”: (1/5) x 0.405 = 0.081 • “governo”: (1/5) x 0 = 0

Fonte: autoria própria.

Após calcular os valores TF-IDF, temos uma representação mais informativa sobre a importância de cada termo em relação aos documentos. Podemos observar que termos específicos como “anunciou”, “medidas”, “econômicas” em D1 e “economia”, “país”, “cresceu” em D2 têm uma maior relevância nos respectivos documentos. Essa informação pode ser utilizada para várias tarefas de PLN, como classificação de textos, sumarização e recuperação de informação.

Notebook Colab

A. TF-IDF

O procedimento descrito pela técnica Count Vectorizer é simples e bastante poderoso, mas tem uma limitação principal. Manning et al. 2008 descrevem que nele 'todos os termos são considerados igualmente importantes quando se trata de avaliar relevância'. Portanto, uma melhoria nesse sentido seria ponderar a computação de características dos dados textuais levando em consideração a frequência de cada palavra do exemplo considerado e no corpus como um todo. Isso permitiria que os algoritmos pudessem obter informações mais relevantes, mesmo quando lidando com dados textuais muito grandes. Isso é conhecido como *Term Frequency - Inverse Document Frequency*, ou TF-IDF, e também é uma técnica *bag-of-words*.

Para ilustrar essa questão do que significa relevância, consideremos as seguintes frases em um contexto de uma tarefa de análise de sentimentos:

1. Eu adorei o último jogo do Corinthians. Foi muito melhor que no jogo anterior, com mudanças táticas importantes que impactaram no estilo de jogo do time. (Sentimento positivo)
2. Eu detestei o novo modo de jogo do Call of Duty. (Sentimento negativo)

Numa abordagem usando *Count Vectorizer*, temos o seguinte panorama:

```
[ ] corpus = [
        'Eu adorei o último jogo do Corinthians. Foi muito melhor que no jogo anterior, com mudanças táticas importantes que impactaram no estilo de jogo do time. ',
        'Eu detestei o novo modo de jogo do Call of Duty.'
    ]
```

```
[ ] from sklearn.feature_extraction.text import CountVectorizer

    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(corpus)
    print(vectorizer.get_feature_names_out())

    print(X.toarray())
```

```
['adorei' 'anterior' 'call' 'com' 'corinthians' 'de' 'detestei' 'do'
 'duty' 'estilo' 'eu' 'foi' 'impactaram' 'importantes' 'jogo' 'melhor'
 'modo' 'mudanças' 'muito' 'no' 'novo' 'of' 'que' 'time' 'táticas'
```

continua

```
 'último']
[[1 1 0 1 1 1 0 2 0 1 1 1 1 1 3 1 0 1 1 2 0 0 2 1 1 1]
 [0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0]]
```

Como todos os termos são igualmente importantes, os classificadores podem entender que o fator mais importante para que uma frase de exemplo possua sentimento positivo é ter mais ocorrências do termo 'jogo'. Usando TF-IDF podemos driblar essa questão ao atribuir um peso menor para termos que ocorrem muito no vocabulário como um todo. Formalmente, temos

$$\text{TF}(p, e) = 1 + log\ f_{p,e} \text{ [1]}$$

$$\text{IDF}(p, C) = log\ \left(1 + \frac{N}{n_p}\right) \text{ [2]}$$

em que *Term Frequency*, TF, é o termo da equação [1], *Inverse Document Frequency*, IDF, é representado pela equação [2], e ainda p é a palavra em questão, e é o exemplo de texto analisado, C é o corpus, N é o número de documentos no corpus e np é o número de ocorrências da palavra p em todos os documentos. O resultado final é a multiplicação de TF com IDF, conforme a equação [3]:

$$\text{TF-IDF}(p, e, C) = TF(p, e) * IDF(p, C) \text{ [3]}$$

Das frases de exemplo, vamos considerar o termo 'jogo', que foi o que mais apareceu no vocabulário.

$\text{TF}(jogo, frase1) = 1 + log\ f_{jogo,frase1} = 1 + log\ 3 = 1 + 0.47712 = 1.47712$(Observe que a palavra jogo apareceu 3 vezes na frase 1)

$\text{IDF}(jogo, corpus) = log\ \left(1 + \frac{2}{4}\right)$ (Observe que a palavra jogo apareceu 4 no corpus. O corpus de exemplo possui dois documentos)

.: Continuando os cálculos temos que $\text{IDF}(jogo, corpus) = log\ \left(\frac{6}{4}\right) = 0.17609$

$\text{TF-IDF}(jogo, frase1, corpus) = 1.47712 * 0.17609 = 0.2601$

Observe que TF da palavra "jogo" na frase 1 é um valor alto, pois a palavra jogo aparece 3 vezes nesta sentença, o seu IDF no corpus é baixo 0.17609. E isso é usado para ponderar esta relação.

A seguir veja alguns exemplos de uso do TF-IDF na prática com o auxílio da biblioteca sklearn

sklearn linha 1461 -> https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/feature_extraction/text.py

https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275

https://www.tablesgenerator.com/markdown_tables#

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
    # chamando a classe TfidfVectorizer biblioteca sklearn
```

A biblioteca sklearn tem um padrão de uso de suas classes em 3 fases.

1. Importa a classe desenhada; Exemplo: from sklearn.feature_extraction.text import TfidfVectorizer
2. Instância a classe desejada; Ex.:vectorizer = TfidfVectorizer()
3. Chama função fit_transform da classe instanciada

```
[ ] vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(corpus)
    print(vectorizer.get_feature_names_out())

    print(X.toarray())
```

```
['adorei' 'anterior' 'call' 'com' 'corinthians' 'de' 'detestei' 'do'
 'duty' 'estilo' 'eu' 'foi' 'impactaram' 'importantes' 'jogo' 'melhor'
 'modo' 'mudanças' 'muito' 'no' 'novo' 'of' 'que' 'time' 'táticas'
 'último']
[[0.18382334 0.18382334 0.         0.18382334 0.18382334 0.13079182
  0.         0.26158365 0.         0.18382334 0.13079182 0.18382334
  0.18382334 0.18382334 0.39237547 0.18382334 0.         0.18382334
  0.18382334 0.36764669 0.         0.         0.36764669 0.18382334
  0.18382334 0.18382334]
 [0.         0.         0.35300279 0.         0.         0.25116439
  0.35300279 0.25116439 0.35300279 0.         0.25116439 0.
  0.         0.         0.25116439 0.         0.35300279 0.
  0.         0.         0.35300279 0.35300279 0.         0.
  0.         0.        ]]
```

```
[ ] vectorizer.get_feature_names_out()
```

```
array(['adorei', 'anterior', 'call', 'com', 'corinthians', 'de',
       'detestei', 'do', 'duty', 'estilo', 'eu', 'foi', 'impactaram',
```

continua

```
       'importantes', 'jogo', 'melhor', 'modo', 'mudanças', 'muito', 'no',
       'novo', 'of', 'que', 'time', 'táticas', 'último'], dtype=object)
```

```
[ ] import pandas as pd
```

```
[ ] #Veja os tokens mais relevantes do Exemplo 'Eu detestei o novo modo de jogo do Call of Duty.'
    df = pd.DataFrame(X[1].T.todense(), index=vectorizer.get_feature_names_out(), columns=["TF-IDF"])
    df = df.sort_values('TF-IDF', ascending=False)
    print (df.head(50))
```

```
                TF-IDF
duty          0.353003
detestei      0.353003
novo          0.353003
of            0.353003
modo          0.353003
call          0.353003
de            0.251164
do            0.251164
eu            0.251164
jogo          0.251164
time          0.000000
mudanças      0.000000
que           0.000000
táticas       0.000000
no            0.000000
muito         0.000000
adorei        0.000000
importantes   0.000000
melhor        0.000000
anterior      0.000000
impactaram    0.000000
foi           0.000000
estilo        0.000000
corinthians   0.000000
com           0.000000
último        0.000000
```

B. Desafio

1. Ajuste o TfidfVectorizer para usar n-grams (bigramas, trigrama etc.) e compare os resultados com a vetorização de unigramas.

SAIBA MAIS...

Para o(a) leitor(a) mais interessado em aprofundar nos temas, sugerimos consultar este artigo:

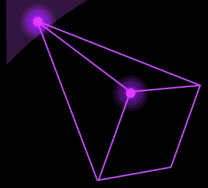
- HARISH, B. S.; REVANASIDDAPPA, M. B. A comprehensive survey on various feature selection methods to categorize text documents. International Journal of Computer Applications, v. 164, n. 8, p. 1-7, 2017.

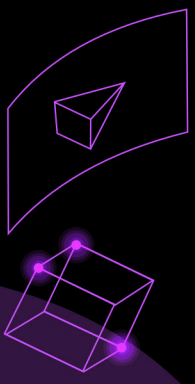
AKCIT
CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Métodos Clássicos de Processamento de Linguagem Natural

Unidade IV

Word2Vec





Unidade IV: *Word2Vec*

4.1 Introdução

Word2Vec é uma técnica popular de aprendizado de representações vetoriais de palavras, desenvolvida pelo Google®, em 2013 (Baccanella; Esuli; Sebastiani, 2010; Batsuren *et al.*, 2022). Em vez de representar palavras como índices únicos (como as representações mostradas anteriormente), Word2Vec representa cada palavra como um vetor de valores contínuos em um espaço de dimensionalidade controlada. Esses vetores capturam semelhanças semânticas entre palavras, onde palavras com significados similares ficam próximas no espaço vetorial.

Word2vec é um algoritmo para obter vetores de palavras treinando uma rede neural rasa (com apenas uma *hidden layer* - camada escondida) com duas arquiteturas possíveis:

1. ***Continuous Bag of Words* (CBOW)**: tem por objetivo prever uma palavra com base no contexto das palavras ao seu redor. Dado um contexto (as palavras ao redor de uma palavra alvo), o modelo tenta prever a palavra alvo. Por exemplo, na frase "O gato está no tapete", se a palavra alvo for "está", o contexto seria ["O", "gato", "no", "tapete"].

2. ***Skip-gram***: o objetivo do modelo *skip-gram* é prever o contexto, dado uma palavra alvo. Por exemplo, se a palavra alvo for "gato", o modelo tenta prever palavras como "O", "está", "no", "tapete".

Se você estiver se perguntando: essas tarefas são supervisionadas ou não supervisionadas? Na verdade, dizemos que são *self-supervised* ou autossupervisionadas, já que a rede aprende por *labels* (etiqueta ou rótulo), porém, não precisamos fazer a anotação ou rotulação, pois elas estão contidas no *corpus* base.

Com a rede, enfim, treinada, extraímos os *embeddings* da matriz de pesos da *hidden layer*. A dimensão dessa matriz é um hiperparâmetro; nós que definimos, que escolhemos o tamanho dos nossos *embeddings*, diminuindo assim a dimensão em relação ao vocabulário. Por levar em conta o contexto, esse algoritmo geralmente é capaz de gerar vetores com valor semântico, de modo que podemos, então, usá-los

para estabelecer relações entre as palavras: o quão semelhantes elas são, por exemplo. Podemos, inclusive, plotar esses vetores e visualizar essas relações na prática, como demonstrado na Figura 5.

4.2 Representação de Sentenças com *Word2Vec*

Embora o Word2Vec seja originalmente projetado para representar palavras, ele pode ser adaptado para representar sentenças (ou documentos) de várias maneiras, conforme nas subseções seguintes.

4.2.1 Média dos Vetores de Palavras

Para representar uma sentença, calcula-se a média dos vetores de Word2Vec das palavras que compõem a sentença. A limitação dessa abordagem é que perde-se a ordem e a estrutura da sentença, e nem todas as palavras têm o mesmo peso semântico.

Por exemplo, considere a frase "O gato está no tapete". Se os vetores Word2Vec de "O", "gato", "está", "no", "tapete" forem v_O, v_{gato}, v_{est}, v_{no} e v_{tapete} respectivamente, a representação vetorial da sentença seria:

$$v_{\text{sentença}} = \frac{v_O + v_{gato} + v_{está} + v_{no} + v_{tapete}}{5}$$

4.2.2 TF-IDF Ponderado

Em vez de calcular a média simples, calcula-se uma média ponderada dos vetores de palavras, onde os pesos são os valores TF-IDF (ver Unidade III deste *e-book*) das palavras. A vantagem é que se atribui mais peso às palavras mais importantes, segundo a métrica TF-IDF. Porém, ainda, pode-se perder alguma ordem e estrutura da sentença.

Suponha que os valores TF-IDF para "O", "gato", "está", "no", "tapete" sejam $tfidf_O$, $tfidf_{gato}$, $tfidf_{est}$, $tfidf_{no}$, $tfidf_{tapete}$, a representação vetorial da sentença seria:

$$v_{\text{sentença}} = \frac{tfidf_O \times v_O + tfidf_{gato} \times v_{gato} + tfidf_{está} \times v_{está} + tfidf_{no} \times v_{no} + tfi}{tfidf_O + tfidf_{gato} + tfidf_{está} + tfidf_{no} + tfidf_{tapete}}$$

4.2.3 Modelos Mais Complexos

Um exemplo de modelos mais complexos é o ***Doc2Vec***, que é uma extensão do Word2Vec que pode aprender vetores diretamente para sentenças, parágrafos ou documentos inteiros. A vantagem é que ele é capaz de capturar a ordem e a estrutura da sentença, porém, pode ser um pouco mais caro computacionalmente.

Como exemplo de aplicação desses modelos podemos citar:

- **Classificação de textos**: as representações vetoriais das sentenças podem ser usadas como entrada para algoritmos de aprendizado de máquina para tarefas de classificação de textos, como análise de sentimentos.
- **Similaridade de sentenças**: medir a similaridade entre sentenças, calculando a similaridade de cosseno entre seus vetores.
- **Sumarização de textos**: ajudar a identificar as sentenças mais importantes em um documento, representando cada sentença como um vetor e comparando-as.

4.3 Exemplos Didáticos

Nas subseções, a seguir, alguns exemplos do uso das teorias abordadas acima serão apresentados.

4.3.1 Representação das Palavras

Vamos considerar um *corpus* simples com três sentenças:

1. "O gato está no tapete"
2. "O cachorro está no jardim"
3. "O pássaro voa no céu"

Suponha que treinamos um modelo Word2Vec usando o algoritmo *skip-gram* ou CBOW. O resultado é que cada palavra do nosso *corpus* é mapeada para um vetor em um espaço vetorial de dimensão d (por exemplo, $d = 3$). Após o treinamento, o modelo gera vetores para cada palavra. Exemplos de vetores podem ser:

- "gato": $[0.5, 0.1, -0.3]$
- "cachorro": $[0.6, 0.2, -0.2]$

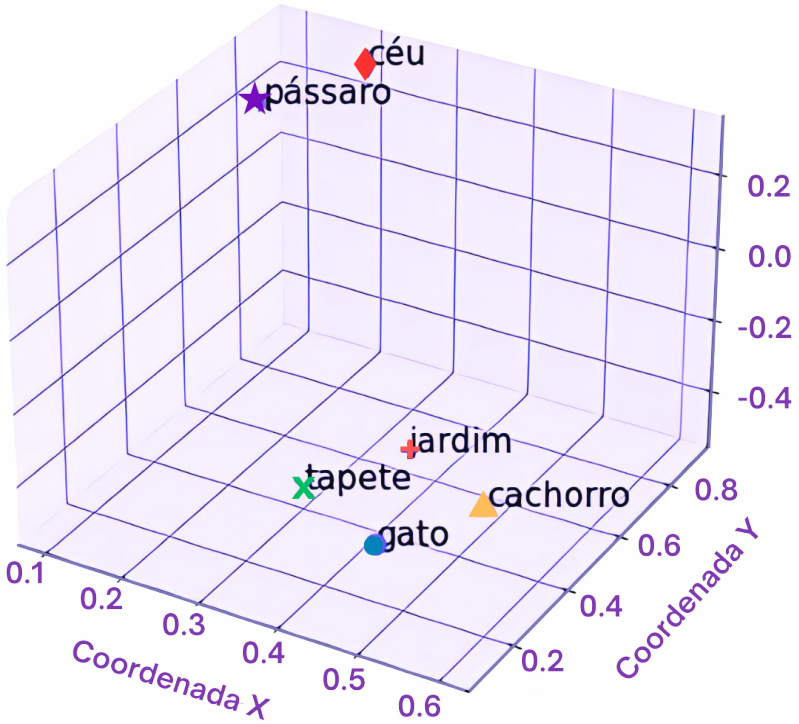
- “tapete”: $[0.3, 0.4, -0.5]$
- “jardim”: $[0.4, 0.5, -0.4]$
- “pássaro”: $[0.1, 0.8, 0.2]$
- “céu”: $[0.2, 0.9, 0.3]$

Os vetores acima são apenas exemplos, mas a ideia é que palavras com significados semelhantes fiquem próximas entre si no espaço vetorial. Por exemplo:

- “gato” e “cachorro” podem ter vetores semelhantes porque são ambos animais de estimação.
- “tapete” e “jardim” podem ter vetores diferentes, pois pertencem a contextos diferentes.

Veja o hiperplano com as palavras gato, cachorro, tapete, jardim, pássaro, céu representados com seus respectivos vetores (Figura 5). Observe que “céu” e “pássaro” são palavras que estão próximas no espaço vetorial, o que pressupõe que aparecem em um mesmo contexto.

Figura 5 - Representação vetorial das palavras “gato”, “cachorro”, “tapete”, “jardim”, “pássaro”, “céu”



Fonte: autoria própria.

Como exemplo de repositório destinado ao armazenamento e compartilhamento de vetores de palavras treinados segundo a abordagem Word2vec, podemos citar o trabalho de Hartmann *et al.* (2017). Eles disponibilizaram vetores treinados, com dimensionalidade variando de 50 a 1.000 dimensões, conforme literatura correlata.

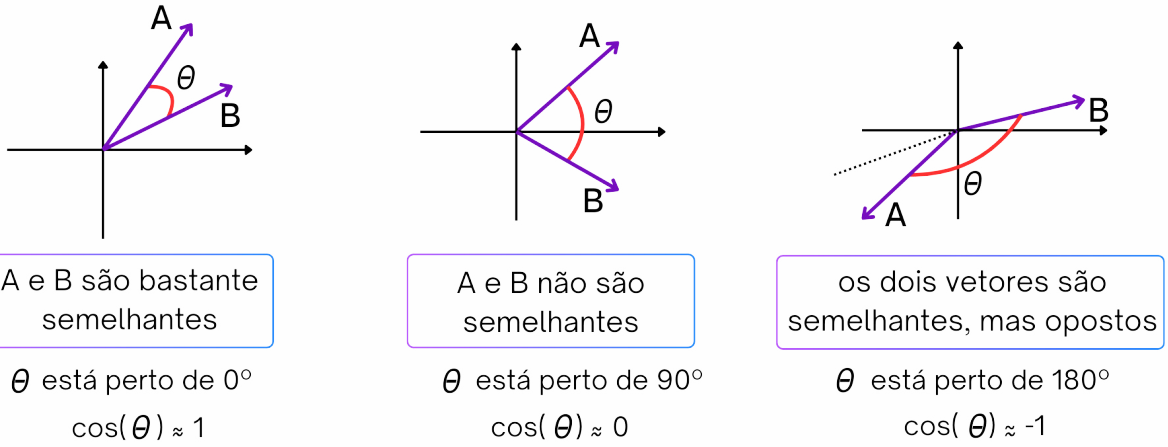
4.3.1.1 Representação de Similaridade

Uma maneira comum de medir a similaridade entre dois vetores de palavras é calcular a **similaridade do cosseno**. A similaridade do cosseno entre dois vetores A e B é dada por:

$$Similaridade(A, B) = \frac{A \cdot B}{||A||\,||B||} = cos(\theta)$$

onde $A \cdot B$ é o produto escalar dos vetores A e B, e $\|A\|\,\|B\|$ são as normas (comprimentos) dos vetores A e B, respectivamente. Se A e B são similares a similaridade de cosseno será próxima de 1, caso contrário será menor conforme Figura 6, abaixo.

Figura 6 - A medida do cosseno entre A e B reflete a similaridade entre os vetores



Fonte: adaptada de *Operations on Word Vectors* (2024).

4.3.1.2 Operações Aritméticas com Vetores

Uma das características interessantes dos vetores Word2Vec é a capacidade de realizar operações aritméticas que capturam relações semânticas. Um exemplo clássico é a operação:

$$rei - homem + mulher \simeq rainha$$

Isso significa que, no espaço vetorial, a diferença entre os vetores de "rei" e "$homem$" é similar à diferença entre os vetores de "$rainha$" e "$mulher$".

4.3.2 Representação de Sentenças

Vamos usar as palavras do nosso *corpus* simplificado para representar a sentença "O gato está no tapete". Podemos fazer isso tirando a média dos vetores das palavras na sentença. Se os vetores de palavras são (os valores indicados são apenas exemplos aleatórios):

- "O": $[0.2, 0.1, 0.4]$
- "gato": $[0.5, 0.1, -0.3]$
- "está": $[0.3, 0.3, 0.1]$
- "no": $[0.2, 0.2, 0.2]$
- "tapete": $[0.3, 0.4, -0.5]$

Observe que, para simplificar, estamos considerando apenas três dimensões. A média dos vetores é calculada da seguinte forma:

$$v_{\text{sentença}} = \frac{1}{5}([0.2, 0.1, 0.4] + [0.5, 0.1, -0.3] + [0.3, 0.3, 0.1] + [0.2, 0.2, 0.2] + [0.3, 0.4, -0.5])$$

$$v_{\text{sentença}} = \frac{1}{5}([1.5, 1.1, -0.1])$$

$$v_{\text{sentença}} = [0.3, 0.22, -0.02]$$

Dessa forma, esse vetor resultante representa a sentença "O gato está no tapete" no espaço vetorial.

Word2Vec transforma palavras em vetores de alta dimensionalidade que capturam relações semânticas de maneira eficaz. Essas representações vetoriais podem ser usadas para várias tarefas de PLN, oferecendo uma base robusta para a análise de textos.

Notebook Colab

Essa aula se baseia:

- No artigo “Distributed Representations of Words and Phrases and their Compositionality”, de Mikolov et al. (2013). Clique aqui para acesso.
- Na documentação do word2vec da biblioteca Gensim.

Objetivos de Aprendizagem

- Neste notebook mostramos como treinar um modelo Word2Vec do zero e como importar um modelo pré-treinado usando a biblioteca gensim.

Princípios de Word Embeddings

Inicialmente é necessário instalar as bibliotecas keras, tensorflow, gensim e np_utils. Para instalar esses recursos você usará o comando !pip install biblioteca.

```
[ ] !pip install keras
    !pip install tensorflow
    !pip install -U gensim
    !pip install np_utils
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (3.4.1)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras) (1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras) (13.8.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras) (3.11.0)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras) (0.12.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras) (0.4.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras) (24.1)
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras) (2.18.0)
```

continua

```
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)

Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)

Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)

Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)

Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)

Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)

Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)

Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)

Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)

Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)

Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)

Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)

Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)

Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)

Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)

Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)

Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)

Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)

Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)

Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)

Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)

Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)

Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
```

continua

```
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/
dist-packages (from tensorflow) (1.26.4)

Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/
dist-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)

Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (13.8.1)

Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.0.8)

Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.12.1)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.3.2)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.10)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/
dist-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/
dist-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)

Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/
dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.7)

Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/
local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow)
(0.7.2)

Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/
dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.0.4)

Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/
dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (2.1.5)

Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/
python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (3.0.0)

Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/
python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)

Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)

Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages
(4.3.3)

Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.10/
dist-packages (from gensim) (1.26.4)

Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.10/
dist-packages (from gensim) (1.13.1)

Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/
dist-packages (from gensim) (7.0.4)

Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages
(from smart-open>=1.8.1->gensim) (1.16.0)

Collecting np_utils

  Downloading np_utils-0.6.0.tar.gz (61 kB)

     ---------------------------------------- 62.0/62.0 kB 567.7 kB/s eta
0:00:00
```

continua

```
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.0 in /usr/local/lib/python3.10/dist-packages (from np_utils) (1.26.4)
Building wheels for collected packages: np_utils
  Building wheel for np_utils (setup.py) ... done
 Created wheel for np_utils: filename=np_utils-0.6.0-py3-none-any.whl size=56437 sha256=88b5e83a6ca29bdb51a65a8ef7dc4495bde0d49e5e932e03db3506b0d3c9f8da
  Stored in directory: /root/.cache/pip/wheels/b6/c7/50/2307607f44366dd021209f660045f8d51cb976514d30be7cc7
Successfully built np_utils
Installing collected packages: np_utils
Successfully installed np_utils-0.6.0
```

```
[ ] from keras.models import Sequential
    from keras.layers import Dense,Embedding,Activation,Dropout,SimpleRNN,BatchNormalization,RNN,Flatten,Input,LSTM,Bidirectional
    #from keras.utils.np_utils import to_categorical
    from keras.utils import to_categorical
    import tensorflow as tf
    import tensorflow_datasets as tfds
    from tensorflow import keras
    from tensorflow.keras.optimizers import SGD
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    import pandas as pd
    import numpy as np
    import gensim
    from sklearn.metrics import classification_report
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
```

Caso você queira treinar um modelo Word2vec do Zero, a biblioteca Gensim provê essa funcionalide:

Alguns comentários sobre o código a seguir:

1. Observe que estamos usando um corpus já tratado, com palavras de interesse (imagine que temos um corpus com várias sentenças já tokenizadas por espaços). Cada setença então está em um vetor e o corpus é um vetor de vetores. Entretanto, a entrada poderia vir de um arquivo no disco, da rede em tempo real, sem precisar carregar todo o seu corpus na RAM.

2. O Word2vec aceita vários parâmetros que afetam tanto a velocidade quanto a qualidade do treinamento.

3. Um deles é para podar o dicionário interno. Palavras que aparecem apenas uma ou duas vezes em um corpus de um bilhão de palavras são provavelmente erros de digitação desinteressantes e lixo. Além disso, não há dados suficientes para fazer qualquer treinamento significativo sobre essas palavras, então é melhor ignorá-las. Um valor razoável para min_count está entre 0-100, dependendo do tamanho do seu conjunto de dados.

4. O paralelismo de treinamento é tratado no número de workers, para acelerar o treinamento, default = 1 -- worker = no parallelization

5. vector_size (int, opcional) – Dimensionalidade dos vetores de palavras.

   Aqui temos a lista completa de parâmetros e valores default. https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec

6. Observe a variável model resultado: Este objeto contém essencialmente o mapeamento entre palavras e embeddings. Após o treinamento, ele pode ser usado diretamente para consultar esses embeddings de várias maneiras.

```
[ ] from gensim.models import Word2Vec
    corpus = [["hello", "world","hi","earth","sunshine","law"],["cat", "say", "meow"], ["dog", "say", "woof"]]
    model = Word2Vec(sentences=corpus, vector_size=5, window=5, min_count=1, workers=4)
```

Os vetores de palavras treinados são armazenados em uma instância KeyedVectors, como model.wv. O motivo para separar os vetores treinados em KeyedVectors é que se você não precisar mais do estado completo do modelo (não precisar continuar o treinamento), seu estado pode ser descartado, mantendo apenas os vetores e suas chaves adequadas.

Uma vez o modelo treinado podemos carregar esse modelo e realizar operações aritméticas entre termos:

```
[ ] word_vectors = model.wv
```

Observe que o tamanho dos vetores obtidos é 5, definido na etapa de treinamento. Temos um vetor denso de tamanho 5 para cada palavra.

```
[ ] len(word_vectors[0])

5
```

Aqui podemos verificar a operação de subtração dos vetores das palavras ‘dog’ e ‘cat’.

```
[ ] word_vectors['dog'] - word_vectors['cat']

array([ 0.00137478, -0.13207467, -0.22588612,  0.11582372, -0.22422102],
      dtype=float32)
```

Podemos ainda retreinar o modelo com novas palavras:

```
[ ] model.train([["dear", "bear", "cream"]], total_examples=3, epochs=1)

WARNING:gensim.models.word2vec:Effective 'alpha' higher than previous training cycles
WARNING:gensim.models.word2vec:EPOCH 0: supplied example count (1) did not equal expected count (3)
(0, 3)
```

Carregando Modelos Treinados em Outros Corpora

Existem diversos repositórios de **embeddings** pré-treinados para facilitar nossa vida. Algumas bibliotecas dispobilizam isso para nós usuários. Veja a senguir os modelos pré-treinados de **word embeddings** disponíveis na biblioteca gensim:

```
[ ] import gensim.downloader

    print(list(gensim.downloader.info()['models'].keys()))

['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300', 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50', 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300', 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200', '__testing_word2vec-matrix-synopsis']
```

Existem vários, mas vamos escolher um modelo treinado no google news com 300 dimensões e no idioma Inglês. Observe que esse processo pode ser um pouco demorado em função do tamanho deste modelo.

```
[ ] word_vectors = gensim.downloader.load('word2vec-google-news-300')
```

```
[==================================================] 100.0% 1662.8/1662.8MB
downloaded
```

Agora, vamos usar este modelo? Quais são as palavras mais similares à palavra 'car'?

```
[ ] word_vectors.most_similar('car')
```

```
[('vehicle', 0.7821096181869507),
 ('cars', 0.7423831224441528),
 ('SUV', 0.7160962224006653),
 ('minivan', 0.6907036900520325),
 ('truck', 0.6735789775848389),
 ('Car', 0.6677608489990234),
 ('Ford_Focus', 0.667320191860199),
 ('Honda_Civic', 0.6626849174499512),
 ('Jeep', 0.651133120059967),
 ('pickup_truck', 0.6441438794136047)]
```

Podemos realizar operaçõe aritméticas

```
[ ] word_vectors['airplane']-word_vectors['flight']
```

```
array([ 2.53417969e-01, -2.23144531e-01,  3.80859375e-02,  2.54394531e-01,
       -1.92871094e-01, -1.28906250e-01, -3.61328125e-02,  2.16796875e-01,
        2.08007812e-01,  1.16699219e-01,  1.44042969e-01, -1.10351562e-01,
        9.17968750e-02, -2.11425781e-01, -6.05468750e-02,  1.19140625e-01,
       -1.39404297e-01,  1.58508301e-01,  2.34375000e-02, -4.05273438e-02,
        5.56640625e-02,  5.85021973e-02,  1.89697266e-01, -2.05078125e-02,
        4.58526611e-03, -7.69042969e-02,  6.83593750e-02, -4.39453125e-03,
       -7.51953125e-02, -1.25976562e-01, -2.45117188e-01, -1.58203125e-01,
       -1.97753906e-01,  2.93212891e-01,  2.24243164e-01,  1.37695312e-01,
```

continua

```
 2.87658691e-01, -9.76562500e-02, -3.07617188e-02,  1.98242188e-01,
 7.30438232e-02,  5.85937500e-03, -1.48437500e-01,  7.95288086e-02,
-5.29785156e-02,  1.97265625e-01, -1.09863281e-01, -2.58789062e-02,
 1.17187500e-02, -1.85546875e-01, -1.45019531e-01,  2.28515625e-01,
-1.41601562e-02, -9.98535156e-02, -1.67724609e-01,  2.03125000e-01,
 9.13085938e-02, -2.61230469e-02,  2.07031250e-01,  1.75781250e-02,
 1.39648438e-01, -1.73828125e-01, -2.76184082e-01,  7.51953125e-02,
 2.00683594e-01,  8.53271484e-02,  2.92968750e-03, -9.76562500e-03,
-2.62451172e-01,  6.56738281e-02, -4.34570312e-02, -2.16552734e-01,
-1.29638672e-01,  7.36083984e-02, -1.01562500e-01, -5.37109375e-03,
-2.14843750e-02, -2.11425781e-01,  1.34277344e-01, -6.15234375e-02,
-1.34277344e-02,  1.56250000e-02, -1.03637695e-01, -2.73437500e-02,
-1.20849609e-01, -4.49218750e-02,  1.76330566e-01,  1.81579590e-01,
-4.78515625e-02,  9.86328125e-02, -1.34033203e-01,  2.93823242e-01,
 9.66796875e-02,  9.03320312e-02, -9.13085938e-02, -1.03515625e-01,
 8.71582031e-02, -2.04833984e-01,  1.93847656e-01, -1.44531250e-01,
-1.75781250e-01, -2.55859375e-01,  1.42822266e-01,  2.44140625e-02,
 1.67968750e-01,  1.46484375e-01, -1.06445312e-01, -8.44726562e-02,
-1.68945312e-01,  3.54003906e-03, -3.16406250e-01, -2.11914062e-01,
-6.73828125e-02, -1.05834961e-01,  2.36328125e-01, -4.95605469e-02,
 1.13769531e-01,  6.54296875e-02, -6.00585938e-02,  4.74853516e-02,
 1.48925781e-01, -9.57031250e-02,  3.22265625e-02, -3.43017578e-01,
 3.70483398e-02,  9.75341797e-02,  6.54296875e-02, -8.83789062e-02,
-6.83593750e-03,  1.87500000e-01,  8.00781250e-02,  3.90625000e-02,
-1.22070312e-01,  3.85742188e-01,  1.62109375e-01,  2.34375000e-01,
-7.29980469e-02, -3.03710938e-01,  1.66503906e-01, -2.49267578e-01,
-6.84814453e-02,  1.04003906e-01, -1.38671875e-01,  2.44140625e-03,
 1.80541992e-01, -2.34863281e-01,  8.00781250e-02, -1.07421875e-02,
-1.97753906e-02, -3.61328125e-02,  9.76562500e-03, -1.95800781e-01,
-1.48925781e-01, -1.61743164e-02,  2.73437500e-02,  5.04882812e-01,
 8.78906250e-02,  1.20117188e-01, -1.45996094e-01,  1.81396484e-01,
 1.85058594e-01,  1.46484375e-02, -1.32385254e-01, -1.41601562e-01,
 2.20703125e-01,  1.08764648e-01,  4.88281250e-02, -6.68945312e-02,
-9.32617188e-02,  8.88671875e-02, -7.32421875e-02, -1.14501953e-01,
-1.86523438e-01, -1.39648438e-01, -3.39355469e-02, -1.99462891e-01,
 1.95556641e-01, -1.27929688e-01, -7.61718750e-02, -4.62646484e-02,
-3.71093750e-02, -2.59765625e-01, -1.58935547e-01, -9.47265625e-02,
 1.75781250e-02, -2.04589844e-01,  5.76171875e-02,  8.76464844e-02,
-1.56250000e-02,  1.55273438e-01,  1.21582031e-01,  9.66796875e-02,
-1.70410156e-01,  6.64062500e-02,  9.76562500e-04, -1.92382812e-01,
 2.06298828e-01, -1.87011719e-01,  1.26953125e-02,  2.48046875e-01,
```

continua

```
       -8.00781250e-02, -1.65893555e-01, -2.44140625e-01, -1.33789062e-01,
       -1.99279785e-01, -8.64257812e-02,  6.66503906e-02, -2.44140625e-04,
       -1.01562500e-01,  2.05810547e-01,  1.00585938e-01, -2.05383301e-01,
        1.38916016e-01,  4.23812866e-02,  6.44531250e-02,  5.22460938e-02,
       -2.51464844e-01,  2.59765625e-01,  5.55877686e-02, -4.09240723e-02,
        2.02636719e-01,  6.54296875e-02,  2.14843750e-02, -1.92871094e-01,
        3.25195312e-01,  5.76171875e-02,  1.60156250e-01,  1.65039062e-01,
       -1.46484375e-01, -6.62841797e-02, -1.80664062e-01,  1.41113281e-01,
       -5.66406250e-02, -1.71875000e-01, -2.78808594e-01, -3.05175781e-02,
       -8.78906250e-03,  5.65795898e-02,  7.95898438e-02,  2.35595703e-01,
        1.85058594e-01, -1.04980469e-02, -1.95800781e-01,  2.65625000e-01,
        2.26684570e-01, -1.32812500e-01, -2.53906250e-02,  1.38671875e-01,
        1.33789062e-01, -3.32031250e-01, -9.93652344e-02, -8.00781250e-02,
        1.96777344e-01, -1.51855469e-01,  2.70996094e-02, -6.39648438e-02,
        1.12304688e-02, -1.68945312e-01,  2.79541016e-02, -1.98730469e-01,
        9.52148438e-03, -1.62109375e-01,  4.42504883e-02, -7.81250000e-02,
       -7.71484375e-02,  1.07421875e-01, -1.97631836e-01,  3.32031250e-02,
       -3.12500000e-02,  1.65100098e-01,  6.83593750e-03,  8.78906250e-03,
        3.05664062e-01, -2.05444336e-01, -2.72949219e-01, -9.66796875e-02,
        1.91375732e-01, -7.34863281e-02,  2.92968750e-02, -2.08251953e-01,
       -9.37500000e-02, -1.91162109e-01,  1.14257812e-01,  7.66601562e-02,
       -1.72950745e-01, -2.79296875e-01,  4.19921875e-02, -1.67480469e-01,
        1.38671875e-01, -2.63916016e-01, -3.86962891e-01, -9.93652344e-02,
        6.90917969e-02, -1.40136719e-01, -1.16210938e-01,  2.97851562e-02,
       -1.07421875e-01,  4.19921875e-02, -2.07641602e-01, -1.46484375e-02],
      dtype=float32)
```

Utilizando o método most_similar, podemos retornar as palavras mais similares por meio de uma operação aritmética: somam-se os vetores positivos e subtrai o vetor negativo. A partir do resultado, podemos obter as palavras mais similares comparando-se esse vetor resultante com os vetores das demais palavras do word *embedding* com base na similaridade cosseno. Abaixo as top 10 palavras mais similares são mostradas:

```
[ ] word_vectors.most_similar(positive=['airplane','flight'],negative=['ship'],topn=10)
```

```
[('plane', 0.6277297735214233),
 ('jet', 0.5784463882446289),
 ('flights', 0.5631440877914429),
```

continua

```
('airliner', 0.5585241913795471),
('aircraft', 0.5546182990074158),
('jetliner', 0.550014853477478),
('NOTE_Expedia_Expedia.com', 0.5478827357292175),
('airplanes', 0.5451778173446655),
('Flight', 0.5407993197441101),
('airline', 0.5332231521606445)]
```

Desafio

Reproduza os experimentos importando os embeddings treinados do Nilc disponíveis em: http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings--do-nilc. Utilize vetores de tamanho 50 para não inviabilizar seus testes.

SAIBA MAIS...

Para o(a) leitor(a) mais interessado em aprofundar o conhecimento nos temas, sugerimos a leitura dos seguintes artigos científicos:

- MIKOLOV, T. *et al.*. Efficient estimation of word representations in vector space. *In:* Proceedings of International Conference on Learning Representations Workshop (ICLR-2013), 2013.

- BOJANOWSKI, P. *et al.*. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics. ArXiv Preprint arXiv:1607, 2016.

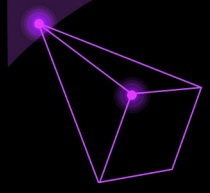
- LING, W. *et al*. Two/too simple adaptations of word2vec for syntax problems. *In:* Proceedings of the 2015 Conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies. 2015. p. 1299-1304..04606, 2015.

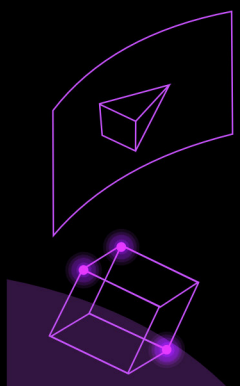
- PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: global vectors for word representation. *In:* Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. p. 1532-1543.

AKCIT
CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Métodos Clássicos de Processamento de Linguagem Natural

Unidade V
Aplicação com métodos clássicos





Unidade V: Aplicação com Métodos Clássicos

5.1 Análise de Sentimentos

A análise de sentimentos (AS) é um campo de estudo com muita popularização devido ao crescimento da Internet e do conteúdo que é gerado por seus usuários, principalmente nas redes sociais, nas quais as pessoas publicam suas opiniões em uma linguagem coloquial e em muitos casos utilizando de artifícios gráficos para tornar ainda mais sucintos seus diálogos (Speriosu et al., 2011; Colleta et al., 2014). [5-8]. A AS, também conhecida como mineração de opinião, é uma subárea do PLN que envolve a identificação e a extração de informações subjetivas em textos. Em termos mais simples, trata-se de determinar as emoções ou atitudes expressas em um texto. Muitos trabalhos abordam esse problema do ponto de vista de classificação, por exemplo, classificando um texto como positivo ou negativo ou, ainda, positivo, negativo ou neutro. Alguns trabalhos tratam como um problema de regressão, indicando que *range* (na faixa) de 0 a 1 e quanto mais próximo de 1 mais positivo.

A AS tem aplicações vastas e variadas. Empresas a utilizam para monitorar a satisfação do cliente, melhorar produtos e serviços e entender melhor a opinião pública sobre suas marcas. Nas redes sociais, a análise de sentimentos pode revelar tendências e opiniões sobre eventos atuais, produtos e personalidades. A AS pode oferecer tendências valiosas para orientar decisões estratégicas.

A AS é uma tarefa complexa e pode oferecer diversos desafios devido a fatores como:

- **Ambiguidade da linguagem**: a mesma palavra ou frase pode ter significados diferentes dependendo do contexto, isto é, palavras com múltiplos significados (polissemia) podem causar ambiguidades. Exemplo: a palavra “fria” pode ser usada para descrever a temperatura ou para expressar indiferença.
- **Sarcasmo e ironia**: identificar sarcasmo é particularmente desafiador, pois as palavras podem expressar o oposto do sentimento real. Sentenças como “Que maravilha!” podem ser genuínas ou sarcásticas.
- **Linguagem informal e gírias**: textos de redes sociais frequentemente utilizam gírias e abreviações, complicando a análise.

- **Contexto cultural**: diferentes culturas podem expressar sentimentos de maneiras distintas, afetando a precisão da análise.
- **Textos curtos:** desafios particulares surgem em tratar textos curtos, pois estes referem-se a fragmentos de texto que contêm uma quantidade limitada de palavras, como *tweets*, comentários em *posts* de *blogs*, mensagens de *chat*, e avaliações de produtos. Esses tipos de texto apresentam desafios únicos para a AS.
- **Limitação de contexto**: com poucas palavras, pode ser difícil entender o contexto completo.
- **Uso de emojis e abreviações**: a linguagem em textos curtos é frequentemente mais informal e repleta de símbolos não-textuais.
- **Alta variabilidade de tópicos**: em um único conjunto de dados, os tópicos podem variar amplamente, dificultando a criação de modelos de análise de sentimentos que se adaptem bem a todos os casos.

Existem várias abordagens para a AS, que podem ser classificadas em duas categorias principais: **baseadas em léxicos** e **baseadas em aprendizado de máquina**.

5.1.1 Abordagens Baseadas em Léxicos

As **abordagens baseadas em léxicos** utilizam dicionários de palavras associadas a sentimentos positivos ou negativos. A análise é realizada contando a frequência dessas palavras em um texto. As vantagens desse tipo de abordagem são simplicidade e facilidade de implementação e uma menor necessidade de grandes conjuntos de dados anotados, com uma definição de regras explícitas e compreensíveis. Entretanto, tratam-se de abordagens que não são eficazes em capturar nuances contextuais e expressões complexas/idiomáticas e com desempenho limitado em lidar com sarcasmo e ironia, além de serem estáticas e pouco generalizáveis.

A construção dos léxicos de sentimento pode ser de forma manual, por especialistas em linguagem ou no domínio de interesse e estes selecionam palavras com base em seu conhecimento, ou pode ser de forma automática em que algoritmos identificam palavras frequentemente associadas a sentimentos positivos ou negativos em grandes conjuntos de dados. Cada palavra no dicionário recebe um valor que representa seu sentimento. Esses valores podem ser binários (positivo ou negativo) ou contínuos (por exemplo, uma escala de -1 a 1). A atribuição pode ser feita de forma manual, em que os Especialistas atribuem valores com base em sua experiência. Ou de forma automática em que algoritmos utilizam técnicas estatísticas ou de *machine learning* para atribuir valores com base em ocorrências de palavras.

Exemplos de léxicos comuns incluem o SentiWordNet (Le; Mikolov, 2014) e o VADER (Hutto; Gilbert; Vader, 2014) e AFINN (Manning, 2022):

- **SentiWordNet:** um léxico derivado do WordNet, onde cada *synset* (grupo de sinônimos) é anotado com escores de positividade, negatividade e objetividade.
- ***Valence Aware Dictionary and sEntiment Reasoner* (VADER):** um dicionário criado especificamente para linguagem social, eficaz em textos curtos e informais, como *tweets*.
- **AFINN:** um dicionário com escores de sentimentos para palavras em inglês, variando de -5 (muito negativo) a +5 (muito positivo).

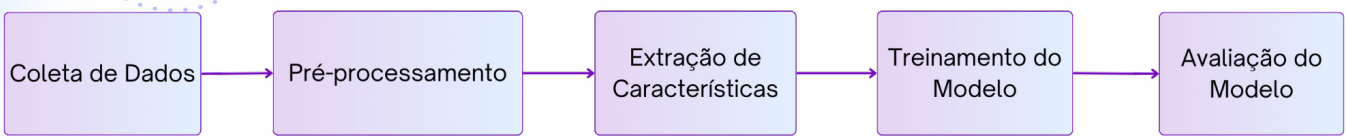
5.1.2 Abordagens Baseadas em Aprendizado de Máquina e Redes Neurais Profundas

Essas técnicas envolvem o treinamento de modelos de aprendizado de máquina em grandes conjuntos de dados anotados. Métodos populares incluem regressão logística, máquinas de vetores de suporte (*Support Vector Machines* - SVM), e redes neurais profundas (*Deep Learning*), como *Long Short-Term Memory* (LSTM) e *transformers* (Pennington *et al*., 2014), por exemplo, *Bidirectional Encoder Representations for Transformers* (BERT) (Silva; Coletta; Hruschka, 2016). Essas técnicas são mais flexíveis e podem capturar nuances linguísticas que as abordagens baseadas em léxicos podem não conseguir. As vantagens dessas abordagens envolvem a capacidade de capturar padrões complexos e contextuais, além de um melhor desempenho em lidar com variabilidade linguística e, portanto, melhor generalização. Mas, por outro lado, tais técnicas requerem grandes volumes de dados anotados para treinamento e possuem maior complexidade e custo computacional.

5.1.2.1 Pipeline de *Machine Learning* para Análise de Sentimentos

Para implementar uma abordagem baseada em aprendizado de máquina (do inglês *machine learning*) para AS, é necessário seguir um *pipeline* estruturado, como o proposto na Figura 7. Esse *pipeline,* geralmente, inclui as etapas de coleta de dados, pré-processamento, extração de características ou conversão em uma representação vetorial dos dados, treinamento do modelo, avaliação e ajuste ou refinamento (*fine--tuning*).

Figura 7 - *Pipeline* para treinamento de um modelo de análise de sentimentos baseado em aprendizado de máquina



Fonte: autoria própria.

As etapas da Figura 7 podem ser descritas conforme abaixo:

1. **Coleta de dados:** a primeira etapa é coletar um conjunto de dados de textos e rotular esses textos com sentimentos. Esses dados podem ser provenientes de avaliações de produtos, *tweets*, comentários em redes sociais, entre outros.

2. **Pré-processamento:** o pré-processamento é crucial para preparar os dados para a extração de características e treinamento do modelo. Alguns exemplos de operações que podem ser feitas no pré-processamento:

   - **Limpeza:** remover URLs, menções (@usuário), *hashtags, e-mails*, números, etc.
   - **Tokenização:** dividir o texto em palavras ou *tokens*.
   - **Normalização:** converter todas as palavras para minúsculas, remover pontuações e *stopwords*.
   - **Lematização e *stemming*:** reduzir palavras às suas formas raiz.

3. **Extração de características:** a extração de características transforma o texto pré-processado em uma representação numérica vetorial que pode ser utilizada pelos algoritmos de aprendizado de máquina. Métodos de extração:

   - ***Bag of words* (BoW):** representa o texto como uma matriz de frequências de palavras.
   - **TF-IDF:** ajusta a frequência das palavras com base na sua relevância.
   - ***Word embeddings*:** utiliza vetores densos para representar palavras (por exemplo, Word2Vec, GloVe (Silva *et al.*, 2014), *FastText* (Speriosu *et al.*, 2011).
   - ***Transformers*:** modelos pré-treinados como BERT que capturam contextos complexos.

4. **Treinamento do modelo:** com os dados convertidos em características (representação vetorial), podemos treinar o modelo de Aprendizado De Máquina. Escolher o algoritmo adequado e ajustar seus hiperparâmetros são etapas essenciais. Algoritmos clássicos para aprendizado de máquina:

- **_Naive Bayes_:** simples e eficaz para textos curtos.
- **SVM:** bom desempenho em alta dimensionalidade.
- **Redes neurais:** incluindo redes neurais recorrentes, LSTMs, e *Transformers* para captura de contextos sequenciais e complexos.

5. **Avaliação do Modelo:** avaliar o desempenho do modelo em um conjunto de dados de teste é crucial para garantir sua eficácia. Alguns exemplos de métricas de avaliação:
   - **Acurácia**: percentual de previsões corretas.
   - **Precisão**: proporção de previsões positivas corretas.
   - ***Recall***: proporção de verdadeiros positivos identificados.
   - **F1-Score**: média harmônica de precisão e *recall*.

Após a avaliação inicial, podemos ajustar o modelo para melhorar seu desempenho. Isso pode incluir a otimização de hiperparâmetros e a experimentação com diferentes técnicas de pré-processamento e extração de características. Algumas técnicas para ajuste dos modelos de classificação:

- **Validação cruzada:** dividir os dados em múltiplos subconjuntos para uma avaliação robusta.
- **Busca em grade *(grid search)*:** testar combinações de hiperparâmetros.
- **Busca aleatória *(random search)*:** explorar hiperparâmetros de maneira aleatória.

Notebook Colab

Objetivos de Aprendizagem

- Aplicar técnicas de Processamento de Linguagem Natural em Análise de sentimentos

A. Análise de Sentimentos em Comentários sobre restaurante em Inglês.

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import io
```

Vamos fazer a leitura de um arquivo com comentários sobre um restaurante. Baixe o Arquivo 'Restaurant_Reviews.tsv' em https://drive.google.com/file/d/1ulP59sOf-4jA4ziRGxNU3J_nRrPow3s_S/view?usp=drive_link e memorize onde você colocou o arquivo na sua máquina.

```
[ ] from google.colab import files
    uploaded = files.upload()
```

```
    Nenhum arquivo escolhido Upload widget is only available when the cell has
    been executed in the current browser session. Please rerun this cell to enable.
    ---------------------------------------------------------------------------
    KeyboardInterrupt                         Traceback (most recent call last)
    <ipython-input-3-21dc3c638f66> in <cell line: 2>()
          1 from google.colab import files
     ----> 2 uploaded = files.upload()

                                        3 frames
      /usr/local/lib/python3.10/dist-packages/google/colab/files.py in upload()
         67     """
         68
     ---> 69     uploaded_files = _upload_files(multiple=True)
         70     # Mapping from original filename to filename as saved locally.
         71     local_filenames = dict()

      /usr/local/lib/python3.10/dist-packages/google/colab/files.py  in  _upload_
      files(multiple)
        154
        155   # First result is always an indication that the file picker has completed.
     --> 156    result = _output.eval_js(
        157   'google.colab._files._uploadFiles("{input_id}", "{output_id}")'.format(
        158             input_id=input_id, output_id=output_id

      /usr/local/lib/python3.10/dist-packages/google/colab/output/_js.py in eval_
      js(script, ignore_result, timeout_sec)
         38   if ignore_result:
         39     return
     ---> 40   return _message.read_reply_from_input(request_id, timeout_sec)
         41
         42
```

continua

```
/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in read_
reply_from_input(message_id, timeout_sec)
    94       reply = _read_next_input_message()
    95       if reply == _NOT_READY or not isinstance(reply, dict):
---> 96         time.sleep(0.025)
    97         continue
    98       if (

KeyboardInterrupt:
```

```
[ ] df = pd.read_csv(io.BytesIO(uploaded['Restaurant_Reviews.tsv']), delimiter='\t',
    quoting=3)
    df.head()
```

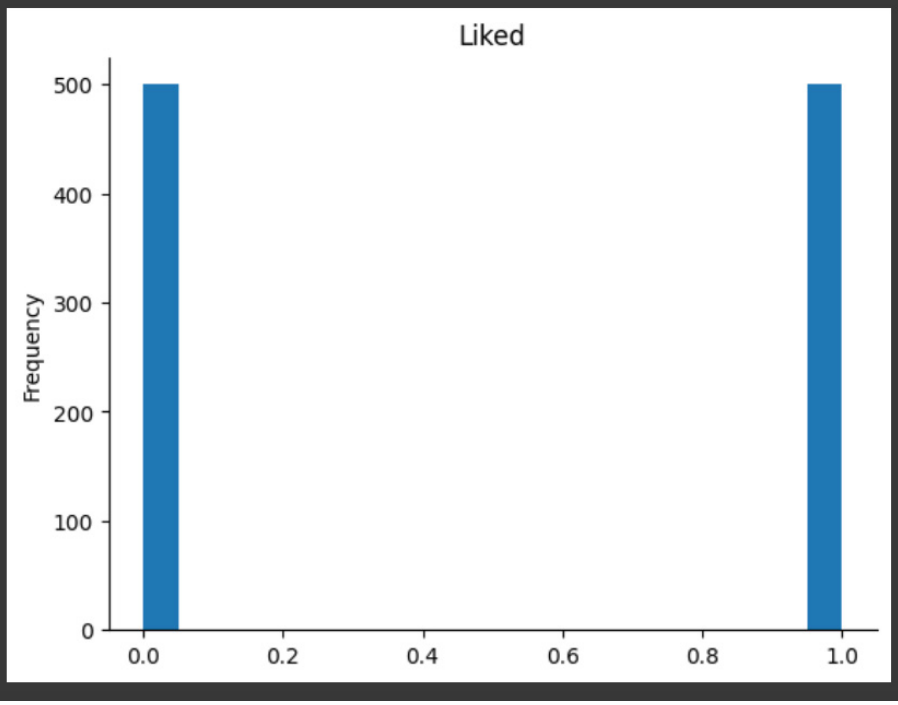
	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

Liked

```
[ ] # @title Liked

    from matplotlib import pyplot as plt
    df['Liked'].plot(kind='hist', bins=20, title='Liked')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

continua



Observem que temos duas colunas, a primeira com o comentário e a segunda com os valores 0 ou 1, sendo 1 para comentários positivos e 0 para comentários negativos. Além disso observe que temos um conjunto de dados balanceado com 500 comentários da classe positiva e 500 da classe negativa.

B. Agora podemos realizar algumas operações de pré-processmaneto, como remoção de stop-words e stemming. A função 'rep' é responsável por esse pré-processamento.

```
[ ] import nltk
    nltk.download('stopwords')

    import re
    from nltk.corpus import stopwords
    from nltk.stem import PorterStemmer
    stemmer = PorterStemmer()

    def rep(review):
        review = re.sub('[^a-zA-Z]', ' ', review)
        review = review.lower()
        reviews = review.split()
        reviews = [stemmer.stem(x) for x in reviews if x not in stopwords.words('english')]
        review = ' '.join(reviews)
        return review
```

continua

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

C. Vamos aplicar o pré-processamento definido anteriormente aos dados.

```
[ ] df['Review'] = df['Review'].apply(rep)
    df.head()
```

	Review	Liked
0	wow love place	1
1	crust good	0
2	tasti textur nasti	0
3	stop late may bank holiday rick steve recommen...	1
4	select menu great price	1

D. Precisamos converter nossos textos em uma representação vetorial

```
[ ] from sklearn.feature_extraction.text import CountVectorizer
    corpus = df['Review'].tolist()

    cv = CountVectorizer(max_features=1500)
    X = cv.fit_transform(corpus).toarray()
    y = df['Liked'].tolist()
```

E. Precisamos separar dados para treinar o modelo e dados para testar. 20% dos dados serão usados para teste.

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
    random_state=0)
```

F. Vamos usar um classificador Bayesiano:

```
[ ] from sklearn.naive_bayes import GaussianNB

    clf = GaussianNB()
    clf.fit(X_train, y_train)
    print('Accuracy:{0: .1f}%'.format(clf.score(X_test, y_test) * 100))
```

```
Accuracy: 73.0%
```

Observe que tivemos uma acurácia de 73% nos testes, mas podemos ter uma análise mais fina, e que indica que ele teve um desempenho semelhante nas classes 0 e 1.

```
[ ] from sklearn.metrics import classification_report
    target_names = ['class 0', 'class 1']
    y_pred = clf.predict(X_test)
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.82      0.57      0.67        97
     class 1       0.68      0.88      0.77       103

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.72       200
weighted avg       0.75      0.73      0.72       200
```

G. Vamos ver como o modelo está se saindo com novos textos?

```
[ ] new_review = 'This food is amazing... hummm'


    new_review = rep(new_review)
    new_review_vector = cv.transform([new_review]).toarray()
    prediction = clf.predict(new_review_vector)
    print(prediction)
```

continua

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-7ad1ffb36b44> in <cell line: 4>()
      2
      3
----> 4 new_review = rep(new_review)
      5 new_review_vector = cv.transform([new_review]).toarray()
      6 prediction = clf.predict(new_review_vector)

NameError: name 'rep' is not defined
```

SAIBA MAIS...

Para o(a) leitor(a) mais interessado em aprofundar nos temas, sugerimos a leitura dos seguintes textos:

- BIRJALI, M.; KASRI, M.; BENI-HSSANE, A.. A comprehensive survey on sentiment analysis: Approaches, challenges and trends. Knowledge-Based Systems, v. 226, p. 107134, 2021.

- LIU, B.. Sentiment analysis and opinion mining. Springer Nature, 2022.

- WANKHADE, M.; RAO, A. C. S.; KULKARNI, C.. A survey on sentiment analysis methods, applications, and challenges. Artificial Intelligence Review, v. 55, n. 7, p. 5731-5780, 2022.

CENTRO DE COMPETÊNCIA EMBRAPII EM TECNOLOGIAS IMERSIVAS

Métodos Clássicos de Processamento de Linguagem Natural

Unidade VI

Encerramento

Unidade VI: Encerramento

6.1 Recapitulação dos Pontos Principais

Ao longo deste *e-book*, exploramos diversos aspectos do PLN e suas aplicações práticas. Aqui estão os principais pontos abordados:

1. **Fundamentos do PLN**: introduzimos os conceitos básicos de PLN, incluindo a importância do pré-processamento de texto e as técnicas de tokenização, lematização e remoção de *stopwords*.
2. **Modelos de linguagem**: discutimos modelos de linguagem, como TF-IDF, *word embeddings*, Word2Vec e *transformers*, explicando como cada um contribui para a representação de texto.
3. **Análise de sentimentos**: exploramos métodos para análise de sentimentos, destacando a importância de classificar emoções em textos e suas aplicações em vários contextos.

Esse *e-book* oferece uma visão do campo do PLN, destacando tanto os fundamentos teóricos quanto as aplicações práticas. A evolução dos modelos de linguagem, especialmente com o advento de técnicas baseadas em aprendizado profundo, transformou a maneira como analisamos e processamos textos. A capacidade de capturar nuances semânticas complexas tem aberto novas possibilidades em áreas como AS, tradução automática e geração de texto.

SAIBA MAIS...

Para aprofundar ainda mais o seu conhecimento em PLN, recomendamos as seguintes ações e leituras adicionais:

❁ ASHISH, V.. Attention is all you need. Advances in Neural information Processing Systems, v. 30, p. I, 2017.

❁ DEVLIN, J. *et al.*. Bert: pre-training of deep bidirectional transformers for language understanding. ArXiv Preprint arXiv:1810.04805, 2018.

❁ MIKOLOV, T.. Efficient estimation of word representations in vector space. arXiv pre-print arXiv:1301.3781, 2013.

❁ PENNINGTON, J.; SOCHER, R.; MANNING, C. D.. Glove: global vectors for word representation. *In:* Proceedings of The 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. p. 1532-1543.

Referências

AIZAWA, A.. An information-theoretic perspective of tf–idf measures. **Information Processing & Management**, v. 39, n. 1, p. 45-65, 2003.

ASHISH, V.. Attention is all you need. **Advances in neural information processing systems**, v. 30, p. I, 2017.

BACCIANELLA, S. *et al.*. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. *In*: **Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)**, Valletta, Malta. European Language Resources Association (ELRA), p. 2200-2204, 2010.

BATSUREN, K. *et al.*. **UniMorph 4.0: universal morphology**. ArXiv Preprint arXiv:2205.03608, 2022.

BOJANOWSKI, P. *et al.*. Enriching word vectors with subword information. **Transactions of the Association for Computational Linguistics**, v. 5, p. 135-146, 2017.

BUCHHOLZ, S.; MARSI, E.. CoNLL-X shared task on multilingual dependency parsing. *In*: **Proceedings of the Tenth Conference on Computational Natural Language Learning** (CoNLL-X). 2006. p. 149-164.

CABRÉ, M. T. La terminología: representación y comunicación. **Documenta Universitaria,** 1999.

COLETTA, L. F. S. *et al.*. Combining classification and clustering for tweet sentiment analysis. In: 2014 Brazilian conference on intelligent systems. **Institute of Electrical and Electronics Engineers**, 2014. p. 210-215.

DEVLIN, J.. **Bert: pre-training of deep bidirectional transformers for language understanding.** ArXiv Preprint arXiv:1810.04805, 2018.

HARTMANN, N. *et al.*. **Portuguese word embeddings: evaluating on word analogies and natural language tasks.** ArXiv Preprint arXiv:1708.06025, 2017.

HUTTO, C.; GILBERT, E.. Vader: a parsimonious rule-based model for sentiment analysis of social media text. In: **Proceedings of the International AAAI Conference on Web and Social Media**. 2014. p. 216-225.

LE, Q.; MIKOLOV, T.. Distributed representations of sentences and documents. *In*: **International Conference on Machine Learning - Proceedings of Machine Learning Research,** 2014. p. 1188-1196.

MANNING, C. D.. Human language understanding & reasoning. **Daedalus**, v. 151, n. 2, p. 127-138, 2022. Disponível em: https://nlp.stanford.edu/~manning/papers/Daedalus_Sp22_09_Manning.pdf. Acesso em: 17 out. 2024.

MIKOLOV, T. *et al.*. Distributed representations of words and phrases and their compositionality. **Advances in Neural Information Processing Systems**, v. 26, 2013.

MIKOLOV, T.; YIH, W-T.; ZWEIG, G.. Linguistic regularities in continuous space word representations. *In*: **Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.** Stroudsburg, PA, USA: Association for Computational Linguistics (NAACL 2013), p. 746–751, 2013.

MILLER, G. A. WordNet: a lexical database for English. **Communications of the Association for Computing Machinery**, v. 38, n. 11, p. 39-41, 1995.

NIELSEN, F. Å.. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. **Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages 718 in CEUR Workshop Proceedings 93-98**. ArXiv Preprint arXiv:1103.2903, 2011.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D.. Glove: global vectors for word representation. *In*: **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)**. 2014. p. 1532-1543.

SILVA, N. F. F.; COLETTA, L. F. S.; HRUSCHKA, E. R.. A survey and comparative study of tweet sentiment analysis via semi-supervised learning. **ACM Computing Surveys (CSUR)**, v. 49, n. 1, p. 1-26, 2016.

SILVA, N. F. F.; HRUSCHKA, E. R.; HRUSCHKA JR, E. R.. Tweet sentiment analysis with classifier ensembles. **Decision Support Systems**, v. 66, p. 170-179, 2014.

SPERIOSU, M. *et al.*. Twitter polarity classification with label propagation over lexical links and the follower graph. *In*: **Proceedings of the First workshop on Unsupervised Learning in NLP**. 2011. p. 53-63.

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