Demostração - Aula 4

Configuração inicial para a aula

- Download do arquivo **(cbow_s300.zip)** de Word Embeddings Pré-treinadas em Portugûes (CBOW)
  - Repositório original: http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc
  - Repositório do professor: https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip
- Descompacte o arquivo (conforme exemplo abaixo)
- Grave localmente ou no seu drive do Google Drive (conforme exemplo abaixo)
- Exsecute os demais passos de processamento usando a biblioteca Gensim

```
# Instalação do pacote Gensim e dependência
!pip install gensim==4.3.2 scipy==1.10.1 numpy==1.23.5 POT==0.4.0 --quiet
```

Obs.: pode ser necessário reiniciar a sessão e executar a instalação novamente.

```
import gensim
print(gensim.__version__)
```

```
4.3.2
```

```
# dependências de bibliotecas que vamos utilizar na Demo
!python -m spacy download pt_core_news_sm --quiet
!python -m spacy download pt_core_news_lg --quiet
!python -m spacy download en_core_web_lg --quiet
```

```
✓ Download and installation successful
You can now load the package via spacy.load('pt_core_news_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
✓ Download and installation successful
You can now load the package via spacy.load('pt_core_news_lg')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.7/400.7 MB 3.7 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_lg')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

Obs: pode ser necessário reunicar o runtime (ambiente) do colab

```
# Download do arquivo no repositório do professor
!wget 'https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip'
!ls -la
# veja o nome do arquivo compactado salvo pelo download
```

```
--2025-06-05 00:54:40--  https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip
Resolving dados-ml-pln.s3-sa-east-1.amazonaws.com (dados-ml-pln.s3-sa-east-1.amazonaws.com)...
Connecting to dados-ml-pln.s3-sa-east-1.amazonaws.com (dados-ml-pln.s3-sa-east-1.amazonaws.com)
HTTP request sent, awaiting response... 200 OK
Length: 929305948 (886M) [application/zip]
Saving to: ‘cbow_s300.zip’

cbow_s300.zip       100%[===================>] 886.25M  9.76MB/s    in 95s

2025-06-05 00:56:17 (9.32 MB/s) - ‘cbow_s300.zip’ saved [929305948/929305948]

total 907548
drwxr-xr-x 1 root root      4096 Jun  5 00:54 .
drwxr-xr-x 1 root root      4096 Jun  5 00:47 ..
-rw-r--r-- 1 root root 929305948 Feb 16  2021 cbow_s300.zip
drwxr-xr-x 4 root root      4096 Jun  3 14:04 .config
drwxr-xr-x 1 root root      4096 Jun  3 14:04 sample_data
```

```
# Descompactação do arquivo
!unzip 'cbow_s300.zip' # subistitua com nome do arquivo
!ls -la
```

```
Archive:  cbow_s300.zip
  inflating: cbow_s300.txt
total 3501348
drwxr-xr-x 1 root root       4096 Jun  5 00:56 .
drwxr-xr-x 1 root root       4096 Jun  5 00:47 ..
-rw-r--r-- 1 root root 2656045531 Oct  4  2018 cbow_s300.txt
-rw-r--r-- 1 root root  929305948 Feb 16  2021 cbow_s300.zip
drwxr-xr-x 4 root root       4096 Jun  3 14:04 .config
drwxr-xr-x 1 root root       4096 Jun  3 14:04 sample_data
```

```
# Load do modelo pelo Gensim
from gensim.models import KeyedVectors

model_cbow = KeyedVectors.load_word2vec_format('cbow_s300.txt')


model_cbow
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7c7f3efe3c90>
```

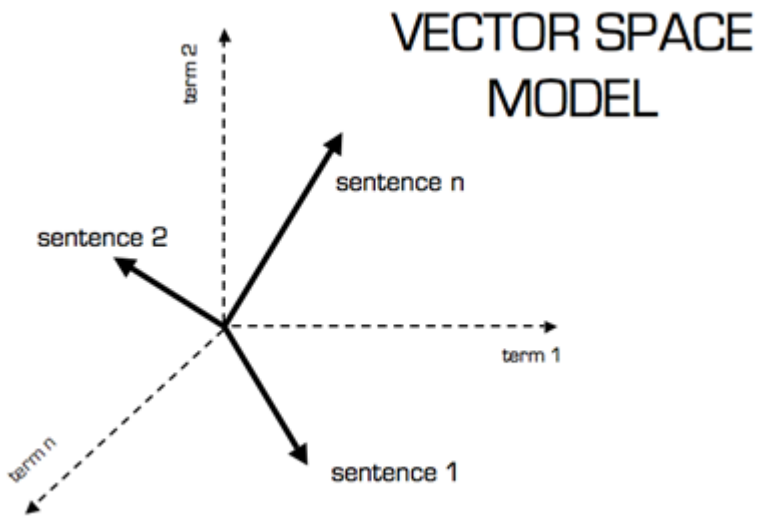
Obs: Mantenha sua sessão do Colab ativa até o exercício.

Análise Semântica em NLP

Como falamos, o próximo passo é realizar uma análise semântica dos dados, onde

buscaremos entender o significado de cada palavra e como a mesma interage no texto!

Outra técnica utilizada e considerada mais eficaz nesse processo é o **mapa de vetores multidimensional**, onde cada palavra é representada por coordenadas de forma matemática (números) e visa representar cada palavra ou sentença. Dessa forma conseguimos calcular a distância de cada vetor e analisar a proximidade linguística de cada palavra ou sentença e associar a um determinado contexto como exemplo. Veja abaixo um exemplo simbólico da representação desse vetor:



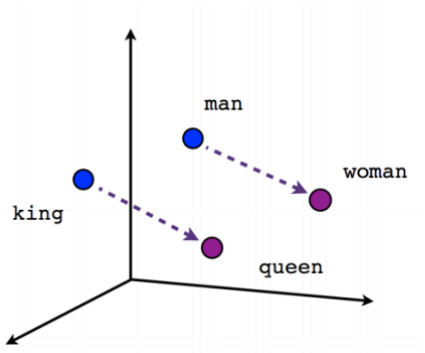
Word Embeddings

Word Embedding é conjunto de modelos para mineração de textos, ou seja, é mais uma técnica de pré-processamento em NLP, onde os textos são transformados e **as palavras representadas por um vetor na forma numérica**, ou seja, em uma representação matemática de cada palavra.

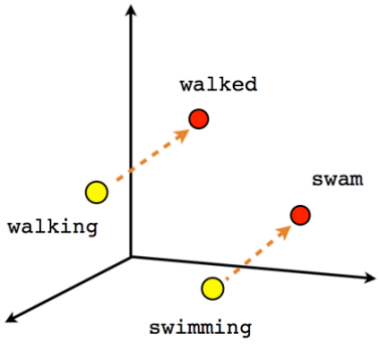
Até aqui vimos implementações usando modelos com **bag of words**, onde as palavras são representadas em vetores grandes e esparsos que podem representar todo o corpus ou documento, as implementações com modelos de **word embeddings** utilizam representações de **vetores**

densos de tamanho fixo que são capazes de armazenar informações sobre o contexto e significado dos documentos.

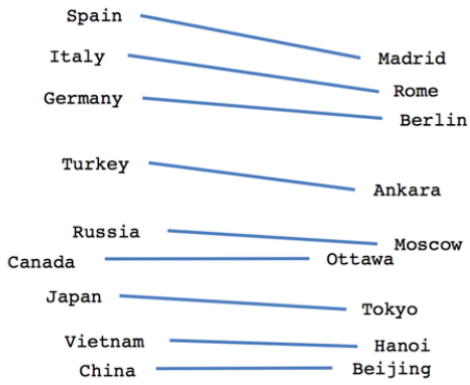
Depois de definido a técnica e o vetor na abordagem com word embedding, cada palavra é representada por um ponto em um espaço multidimensional (**embedding space**) e como falamos, cada palavra é representada de forma numérica no vetor, que na verdade são os pontos/dimensões de cada palavra.



Male-Female



Verb tense



Country-Capital

Exemplo de visualização de um Embedding: https://projector.tensorflow.org/

- Existem dois modelos mais conhecidos de aplicação Word Embeddings, são eles:

-

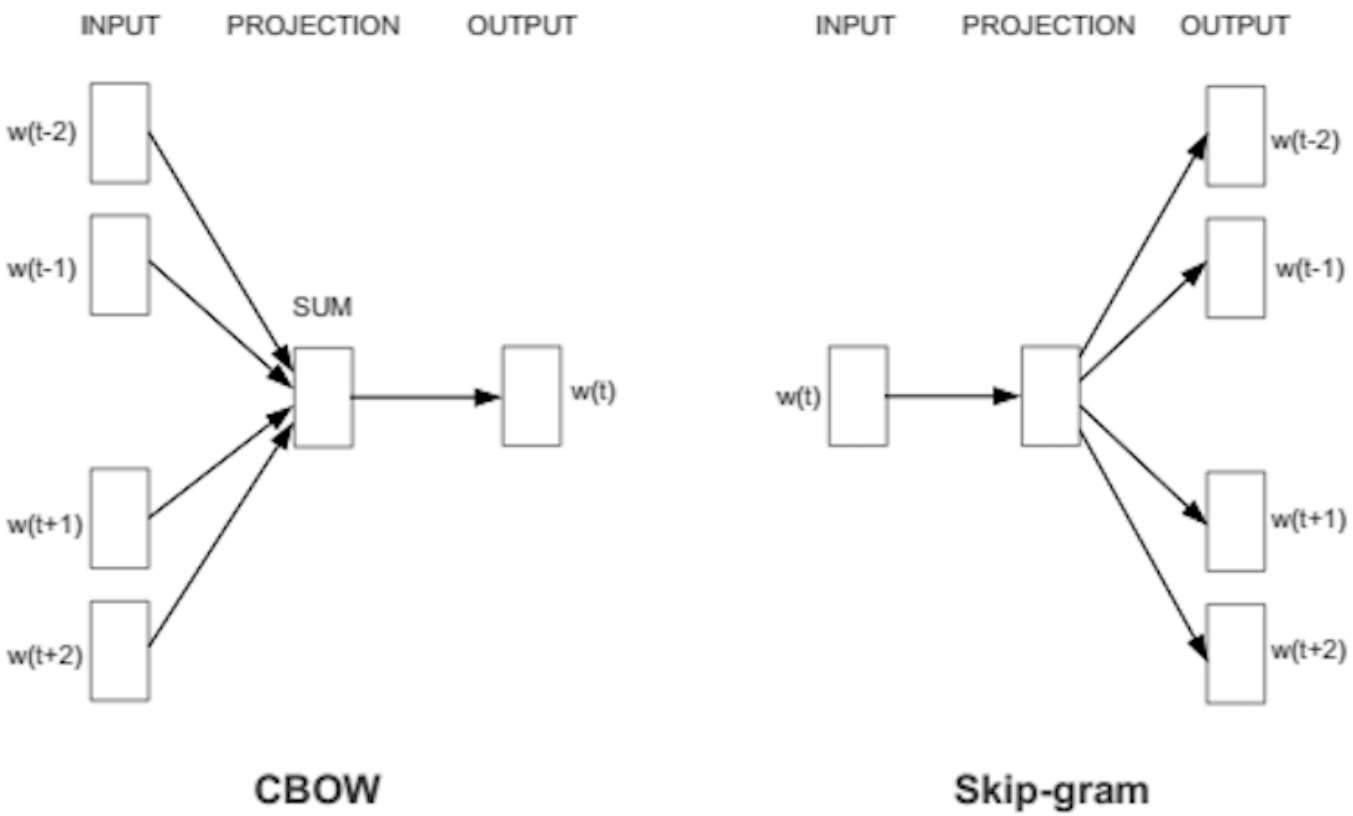
Word2Vec, que foi desenvolvido por Tomas Mikolov e pode utilizar algoritmos diferentes e são baseados em redes neurais, são eles:

-

Continuous Bag-of-Words (CBOW): A ideia do algoritmo Continuous bag of words é prever qual a palavra que estamos buscando a partir de um determinado contexto. Para isto a entrada da rede é um vetor one-hot encoded que represente as palavras do contexto e a sua saída é a palavra que estamos buscando.

-

Skip-Gram: A abordagem do Skip-gram é a inversa, tomando como ponto de partida uma determinada palavra, o objetivo é prever o contexto do qual esta palavra veio. Para isto a entrada da rede é um vetor one-hot encoded que represente a palavra que estamos buscando e a sua saída, as palavras do contexto.



Outro modelo conhecido:

-

GloVe (Global Vectors for Word Representation), desenvolvido por Pennington em Stanford. É uma extensão do método Word2Vec, no entanto, ao invés de usar uma janela fixa para configurar o contexto, este método constrói a matriz co-occurrence usando estatísticas extraídas do próprio texto.

https://wiki.pathmind.com/word2vec

Sendo mais especifico, **Word Embeddings** é um modelo de rede neural para representar dados com um grande número de classes de forma mais eficiente. Os

Embeddings melhoram muito a capacidade das redes neurais de aprender com dados desse tipo, representando os dados com vetores de menor dimensão.

Word Embeddings em particular são interessantes porque as redes são capazes de aprender relações semânticas entre as palavras. Por exemplo, os embeddings saberão que o equivalente masculino de uma rainha é um rei.

Para quem se interessar em saber mais sobre Word Embeddings, abaixo estão boas referências!

1. Uma visão geral conceitual muito boa do word2vec de Chris McCormick
2. Primeiro artigo sobre word2vec de Mikolov et al
3. Video de ilustração e implementação em Tensorflow
4. Modelos de Word Embeddings treinados em pt-br

Como o Word2Vec funciona?

```
from IPython.display import YouTubeVideo
YouTubeVideo("64qSgA66P-8")
```

https://www.youtube.com/watch?v=64qSgA66P-8

Outros modelos treinados com em Word2vec

Modelo treinado para saúde: https://www.inf.pucrs.br/linatural/wordpress/recursos-e-ferramentas/word-embeddings-para-saude/
Modelos treinados com bases genéricas: http://www.nilc.icmc.usp.br/embeddings

Word2Vec com Spacy

Vamos utilizar os modelos em inglês e em português do SpaCy.
Mais para frente vamos explorar outras formas de criar nosso próprio modelo Word2Vec.
O modelo em português foi disponibilizado recentemente com o Word2Vec treinado.

```
#!python -m spacy download en_core_web_lg
# reinicie o Runtime do colab
```

A forma mais simples de explorar a similaridade de palavras atravez do Word2Vec é usando o método `.similarity`.

```
import spacy

en = spacy.load('en_core_web_lg')

tokens = en(u'banana cat dog')

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
banana banana 1.0
banana cat 0.28154364228248596
banana dog 0.2432764321565628
cat banana 0.28154364228248596
cat cat 1.0
cat dog 0.8016854524612427
dog banana 0.2432764321565628
dog cat 0.8016854524612427
dog dog 1.0
```

```
'''
# Outra forma de carregar o Spacy
import en_core_web_lg
en = en_core_web_lg.load()
'''
```

```
'\n# Outra forma de carregar o Spacy\nimport en_core_web_lg\nen = en_core_web_lg.load()\n'
```

Outro exemplo de similaridade:

```
import spacy
import numpy as np

en = spacy.load('en_core_web_lg')

def most_similar(word):
  ms = en.vocab.vectors.most_similar(np.asarray([en.vocab.vectors[en.vocab.strings[word]]]), n=10)
  palavras_similares = [en.vocab.strings[w] for w in ms[0][0]]
  distances = ms[2]
  return palavras_similares

most_similar('dog')
```

```
['doG',
 'DOGS',
 'PUPPY',
 'PET',
 'CAT',
 'PUPPIES',
 'CANINE',
 'PuP',
 'CATs',
 'TERRIER']
```

Spacy com Word2Vec em Português

O Spacy implementou novos modelos além do padrão sem word2vec (pt = pt_core_news_sm)

pt_core_news_sm = https://spacy.io/models/pt / https://spacy.io/models/pt#pt_core_news_sm
pt_core_news_md = https://spacy.io/models/pt#pt_core_news_md
pt_core_news_lg = https://spacy.io/models/pt#pt_core_news_lg

Exemplo 1: usando o Spacy em potuguês, porém o modelo pt (pt_core_news_sm) não é efetivo, pois não tem implementação do word2vec

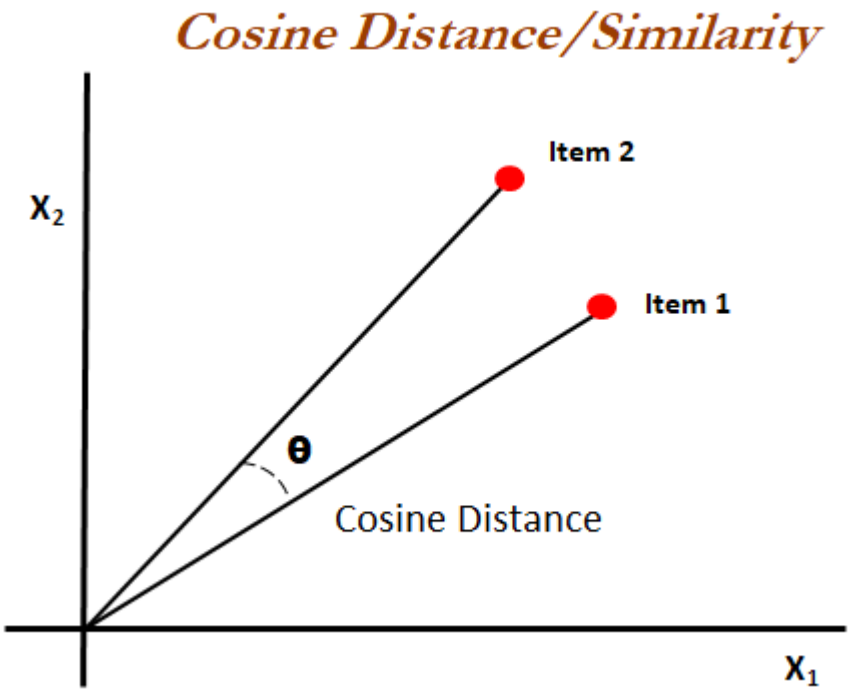
```
import spacy
pt = spacy.load('pt_core_news_sm')

tokens = pt(u'banana cachorro gato')

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
banana banana 1.0
banana cachorro 0.3028274476528168
banana gato 0.47098642587661743
cachorro banana 0.3028274476528168
cachorro cachorro 1.0
cachorro gato 0.34933236241340637
gato banana 0.47098642587661743
gato cachorro 0.34933236241340637
gato gato 1.0
<ipython-input-13-cf27e42794dd>:8: UserWarning: [W007] The model you're using has no word vecto
  print(token1.text, token2.text, token1.similarity(token2))
```

```
# sando o Spacy em potuguês com o modelo de word2vec
import spacy
import pt_core_news_lg

pt = spacy.load('pt_core_news_lg')

tokens = pt(u'banana cachorro gato')

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
banana banana 1.0
banana cachorro 0.3059433102607727
banana gato 0.1974039524793625
cachorro banana 0.3059433102607727
cachorro cachorro 1.0
cachorro gato 0.8368197083473206
gato banana 0.1974039524793625
gato cachorro 0.8368197083473206
gato gato 1.0
```

Dado que cada palavra tem seu vetor de representação que leva em consideração o contexto em que a mesma esta inserida, é possível calcular o cosseno de similaridade!



```
import spacy
import numpy as np

nlp = spacy.load('pt_core_news_lg')

king = nlp.vocab['rei'].vector
man = nlp.vocab['homem'].vector
woman = nlp.vocab['mulher'].vector

new_vector = (king - man) + woman

new_vector
```

```
array([-5.1596103e+00,  4.0269017e-01,  6.8826002e-01,  2.0923100e+00,
       -3.3991599e+00,  2.2988601e+00,  2.8415298e+00,  1.5295998e+00,
       -8.6486006e-01,  1.5775001e+00, -1.0536999e+00,  5.8993397e+00,
       -1.7286000e+00,  2.8081002e+00,  5.0857997e+00, -5.2581902e+00,
        8.8554001e-01,  9.7709984e-02,  5.3111701e+00,  3.2613010e+00,
        2.8769970e-02,  2.2991521e+00, -5.0180502e+00,  4.6739397e+00,
        3.2807398e+00,  2.9376006e-01, -5.4242201e+00,  6.4201002e+00,
       -3.5190964e-01,  6.0002298e+00, -5.4868102e+00,  6.0858002e+00,
        6.4404005e-01, -1.5342201e+00,  1.7698501e+00,  5.9997001e+00,
       -3.9427900e+00,  2.5545342e+00,  8.8311994e-01,  7.2575998e-01,
        8.6780000e-01,  1.7244401e+00, -2.9703999e+00,  2.8733499e+00,
       -6.7003999e+00,  2.7103001e-01, -6.0971999e+00, -4.6140800e+00,
        3.6470008e-01,  3.9329300e+00,  8.8900328e-03,  2.7811100e+00,
       -6.8189001e-01,  7.1632099e-01,  1.7795570e+00, -5.7809000e+00,
       -3.6637003e+00,  1.9374399e+00, -1.3269100e+00,  1.3980900e+00,
        3.6801202e+00,  5.7137674e-01, -5.9291983e-01, -2.1740019e-01,
        1.1818030e+00,  1.9008999e+00, -1.2523600e+00,  4.3011999e+00,
        3.8584001e+00, -2.2632999e+00,  1.3681599e+00, -3.5306001e+00,
        1.3671302e+00,  4.3278599e+00,  5.6618500e+00, -2.4131000e+00,
       -9.8459959e-02,  3.4782004e+00,  1.3069868e-02, -3.7364998e+00,
       -4.7317801e+00,  3.0272999e+00,  4.1271358e+00, -3.9905000e+00,
        1.1991100e+00, -2.9036996e+00,  9.1439998e-01,  4.9478202e+00,
        3.1025999e+00, -3.4898996e+00, -5.5092998e+00, -8.1239998e-01,
       -9.2847002e-01,  2.6115999e+00,  3.3021500e+00,  4.2576303e+00,
        5.0273995e+00,  1.1496999e+00,  1.6625000e+00,  1.0042700e+01,
       -2.0107002e+00, -3.6367102e+00,  1.1073000e+00, -6.4030600e+00,
        4.0870011e-01, -1.2549000e+00, -1.6673499e+00,  4.2532997e+00,
       -2.6193994e-01, -2.5583005e-01, -3.7159199e-01,  1.5639300e+00,
        1.5659000e+00,  7.5301003e-01,  1.2185299e+00,  3.0523100e+00,
        7.7386808e+00,  3.5755703e+00, -4.6429319e+00, -7.5980501e+00,
       -4.2889099e+00,  5.9297991e-01,  1.4299300e+00,  3.7204099e+00,
       -1.1813998e-01,  2.2832801e+00, -2.5623000e+00, -2.4509001e+00,
        1.2072999e+00, -1.7256200e+00,  6.0022998e+00,  6.7115898e+00,
       -3.1299999e+00,  3.6381307e+00, -4.0124798e-01,  2.2478998e-01,
        2.9452405e+00,  6.9130001e+00,  2.1719999e+00, -3.1794801e+00,
       -4.0039001e+00,  1.3091699e+00,  5.2094002e+00, -6.2248998e+00,
        6.5597801e+00,  1.2225499e+00,  3.0184999e+00, -1.8262599e+00,
        1.1732000e+00,  2.5098000e+00, -1.2658000e+00, -4.1151997e-01,
        2.6639502e+00,  1.1803410e+00, -4.7339916e-02,  3.5578201e+00,
        1.7214901e+00,  7.2959971e-01, -2.1668000e+00,  7.3441701e+00,
       -2.4946001e+00, -6.0966001e+00, -1.4605200e+00,  5.9718001e-01,
       -5.8313203e+00,  3.9506001e+00, -1.2304301e+00, -2.7287002e+00,
        1.2920300e+00,  2.3986101e+00,  1.8969983e-02,  4.5638007e-01,
        3.0881000e+00,  3.8819997e+00,  3.4389999e+00,  8.1779509e+00,
       -2.6531699e+00, -2.5803897e+00,  2.9619992e+00,  1.9282600e+00,
       -1.9056000e+00, -1.1291200e+00,  2.4125581e+00,  4.2262203e-01,
        7.6812000e+00,  4.2845001e+00,  1.4490900e+00,  8.0080009e-01,
       -1.5319901e+00,  2.4748998e+00, -4.5808001e+00, -6.3475990e-01,
        4.0395403e+00, -1.1035001e+00, -2.7441998e+00, -2.6484299e+00,
       -3.0567701e+00,  2.4668801e+00, -2.5382931e+00,  7.1608996e-01,
```

```
 4.0681000e+00, -4.5581999e+00, -3.4568200e+00,  3.4996202e+00,
-3.4413993e-02, -1.8657000e+00, -2.8102400e+00,  3.7445300e+00,
-1.1332002e+00, -1.1934000e-01, -2.4672501e+00,  5.2364001e+00,
-3.1613903e+00, -3.9178009e+00,  3.4194005e+00,  2.9004002e+00,
 1.9878299e+00,  1.9030499e+00,  4.5528999e-01,  6.0665005e-01,
-4.9922800e+00,  4.9487400e+00,  4.6080298e+00,  5.2190018e-01,
-2.0911998e-01, -8.0938997e+00,  4.8339987e-01,  1.5968899e+00,
 9.8179960e-01, -3.1984000e+00,  2.4599199e+00, -3.1288400e+00,
```

```
from scipy import spatial

new_vector = (king - man) + woman

cosine_similarity = lambda x, y: 1 - spatial.distance.cosine(x, y)
computed_similarities = []

for word in nlp.vocab:
    if word.has_vector:
        if word.is_lower:
            if word.is_alpha:
                similarity = cosine_similarity(new_vector, word.vector)
                computed_similarities.append((word.text, similarity))

computed_similarities = sorted(computed_similarities, key=lambda item: -item[1])

computed_similarities[:4]
```

```
[('rei', 0.8917216062545776),
 ('mulher', 0.36437276005744934),
 ('homem', 0.17508389055728912),
 ('dom', 0.14074578881263733)]
```

Acredito que estão se perguntando, onde vamos usar isso e como, certo?

Você pode usar este modelo em inúmeras aplicações, por exemplo, pesquisa de produtos ou serviços por meio de suas descrições ou até mesmo em aplicações mais complexas de análise de sentimentos e classificação de documentos.

Word2Vec com Gensim

Gensim é mais uma das libs disponíveis em Python para tarefas de NLP! Ao contrários das outras opções, essa é uma ferramenta mais especializada em tarefas de modelagem de tópicos e análise de similaridade!

Sua instalação é muito simples!

```
# Instalação do pacote Gensim e dependência
#!pip install gensim==4.3.2


import gensim
print(gensim.__version__)
```

```
4.3.2
```

Configuração inicial para a aula

- Download do arquivo **(cbow_s300.zip)** de Word Embeddings Pré-treinadas em Portugûes (CBOW)
  - Repositório original: http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc
  - Repositório do professor: https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip
- Descompacte o arquivo (conforme exemplo abaixo)
- Grave localmente ou no seu drive do Google Drive (conforme exemplo abaixo)
- Exsecute os demais passos de processamento usando a biblioteca Gensim

```
'''
# Download do arquivo no repositório do professor
!wget 'https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip'
!ls -la
# veja o nome do arquivo compactado salvo pelo download

# Descompactação do arquivo
!unzip 'cbow_s300.zip' # subistitua com nome do arquivo
!ls -la

# Load do modelo pelo Gensim
from gensim.models import KeyedVectors

model = KeyedVectors.load_word2vec_format('cbow_s300.txt')
'''
```

```
'\n# Download do arquivo no repositório do professor\n!wget 'https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip'\n!ls -la\n# veja o nome do arquivo compactado salvo pelo download\n\n# Descompactação do arquivo\n!unzip 'cbow_s300.zip' # subistitua com nome do arquivo\n!ls -la\n\n# Load do modelo pelo Gensim\nfrom gensim.models import KeyedVectors\n\nmodel = KeyedVectors.load_word2vec_format('cbow_s300.txt')\n'
```

Análise com CBOW em Português

```
model_cbow
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7c7f3efe3c90>
```

O código acima vai baixar o modelo treinado de Word2Vec usando CBOW com 300 posições em português.

Com o modelo disponível, podemos carrega-lo com a ajuda da classe `KeyedVectors` do Gensim e com esse modelo podemos fazer operações de análise semântica em português, veja:

```
# cada palavra é representada por um vetor de 300 posições
#model_cbow['cachorro']
model_cbow['gato']
```

```
array([ 0.080862, -0.115363,  0.375795,  0.114992, -0.036125, -0.090976,
       -0.202319, -0.297874,  0.063109, -0.322301, -0.311339, -0.145936,
        0.024642, -0.177522, -0.036947,  0.175441, -0.303248, -0.269762,
       -0.009697,  0.206333,  0.022104,  0.133072,  0.028073,  0.580374,
       -0.018641, -0.052175,  0.099459,  0.304987,  0.091809,  0.229643,
        0.187417, -0.129141, -0.06451 ,  0.096126,  0.118515, -0.040087,
       -0.019711, -0.078362, -0.078762,  0.326551,  0.163371,  0.074987,
       -0.07541 ,  0.178085, -0.129488,  0.334529,  0.02373 , -0.100327,
       -0.175369, -0.05598 ,  0.001715,  0.04005 , -0.187697,  0.057138,
        0.141195, -0.051365,  0.108957,  0.027061,  0.073435, -0.218011,
       -0.154617, -0.112563,  0.11286 ,  0.052005,  0.194585, -0.158321,
       -0.028461, -0.159067,  0.255324, -0.120326, -0.018783, -0.101366,
       -0.010514, -0.359594,  0.18899 ,  0.152701,  0.008957, -0.32782 ,
       -0.049077,  0.193421, -0.506094,  0.273253,  0.118434,  0.020687,
       -0.007004,  0.120561, -0.102476, -0.137148,  0.160295,  0.203798,
       -0.035664,  0.206802,  0.051614,  0.023802,  0.072326, -0.255317,
        0.108902,  0.256363,  0.096631, -0.05506 , -0.103569, -0.099239,
        0.240546, -0.475533,  0.12571 ,  0.15199 , -0.133593, -0.129161,
        0.03448 , -0.274172, -0.140714,  0.128492,  0.091799, -0.417719,
       -0.144971, -0.20176 ,  0.294562, -0.089292, -0.247063,  0.218979,
       -0.041262, -0.087564, -0.222953,  0.407738,  0.073559,  0.359935,
       -0.550719,  0.268232, -0.113363,  0.264407, -0.15111 ,  0.096838,
       -0.098241, -0.13263 ,  0.226412, -0.237834, -0.043622,  0.010272,
       -0.005372, -0.005596, -0.125739, -0.07468 ,  0.111457,  0.483734,
        0.18154 ,  0.060745,  0.080259, -0.086287,  0.060554, -0.01853 ,
       -0.072493,  0.462677, -0.021376, -0.227065, -0.017391, -0.12795 ,
       -0.00856 , -0.268554,  0.033225, -0.221947, -0.042683, -0.150671,
       -0.316048, -0.100047, -0.011011, -0.051026,  0.193161, -0.135101,
        0.311013,  0.009614, -0.049199,  0.03064 ,  0.020238,  0.110492,
       -0.166699, -0.25759 ,  0.021513,  0.035831, -0.147476, -0.02352 ,
       -0.069174,  0.092565,  0.126039,  0.098785, -0.10664 , -0.267832,
       -0.085589,  0.106156, -0.475714, -0.281237, -0.191292, -0.045994,
        0.034782,  0.325081,  0.207448, -0.118044,  0.123979, -0.124524,
        0.029986,  0.189371, -0.076906,  0.064082,  0.128751,  0.110474,
        0.060246,  0.21221 , -0.348762,  0.177161,  0.050718,  0.174875,
        0.314554, -0.197033,  0.40307 ,  0.178446, -0.119776,  0.064152,
        0.123859,  0.105608, -0.142715,  0.013134, -0.106779, -0.080555,
        0.041903,  0.044268,  0.069299,  0.038801, -0.246099,  0.064015,
       -0.063686,  0.062941,  0.330322,  0.035453,  0.132418, -0.064306,
       -0.437633,  0.011375, -0.065053,  0.153669, -0.415462,  0.216196,
       -0.137441, -0.064196, -0.206851,  0.043492,  0.150559, -0.175313,
       -0.127303,  0.177043,  0.072571,  0.095479,  0.107375, -0.051058,
        0.108318,  0.026019,  0.128047,  0.107983,  0.205101, -0.028385,
       -0.001803,  0.115481, -0.185445, -0.200445,  0.080524, -0.440575,
        0.117048,  0.026652, -0.314364,  0.050413, -0.086179,  0.093026,
        0.203785, -0.264977,  0.052264,  0.579669,  0.254156,  0.034244,
       -0.132928,  0.06536 , -0.240848,  0.107611,  0.340128, -0.076827,
        0.310936,  0.165077,  0.135   ,  0.007123,  0.068744, -0.281193,
```

```
          -0.341439,  0.084884, -0.203549, -0.00928 , -0.030283, -0.324593,
           0.067859,  0.156023,  0.218471,  0.002674, -0.058456, -0.276578],
         dtype=float32)
```

```
len(model_cbow['cachorro'])
```

```
300
```

Análise de similaridade

```
model_cbow.similarity('cachorro', 'gato')
```

```
0.5658017
```

```
model_cbow.similarity('cachorro', 'banana')
```

```
0.21468098
```

```
pairs = [
    ('carro', 'jipe'),
    ('carro', 'avião'),
    ('carro', 'bicicleta'),
    ('carro', 'cereal'),
    ('carro', 'filosofia'),
]
for w1, w2 in pairs:
    print('%r\t%r\t%.2f' % (w1, w2, model_cbow.similarity(w1, w2)))
```

```
'carro' 'jipe'  0.68
'carro' 'avião' 0.61
'carro' 'bicicleta'     0.42
'carro' 'cereal'        0.24
'carro' 'filosofia'     0.05
```

```
model_cbow.most_similar_cosmul(positive=['cachorro'], topn=4)
#model_cbow.most_similar(positive=['cachorro'], topn=4)
```

```
[('cão', 0.8745832443237305),
 ('cachorrinho', 0.8369466662406921),
 ('gambá', 0.8245760798454285),
 ('cãozinho', 0.8212025761604309)]
```

```
model_cbow.most_similar_cosmul(positive=['gol'])
```

```
[('golo', 0.882580041885376),
 ('golaço', 0.8568756580352783),
 ('hat-trick', 0.8438656330108643),
 ('golaã§o', 0.8257437348365784),
 ('gol.', 0.819081723690033),
 ('tento', 0.8054290413856506),
 ('pênalti', 0.7992749810218811),
 ('gols', 0.7916743159294128),
 ('penalti', 0.7705233097076416),
 ('pãªnalti', 0.7599165439605713)]
```

```
model_cbow.most_similar_cosmul(positive=['gol','carro','vw'])
```

```
[('veículo', 0.3571850061416626),
 ('jetta', 0.3335595428943634),
 ('fusca', 0.3269665539264679),
 ('passat', 0.3232665956020355),
 ('suv', 0.3186461627483368),
 ('volkswagen', 0.3184221386909485),
 ('chevette', 0.31768789887428284),
 ('caminhão', 0.31184300780296326),
 ('jipe', 0.31020066142082214),
 ('superdesportivo', 0.3077678978443146)]
```

```
model_cbow.most_similar_cosmul(positive=['rei'])
```

```
[('monarca', 0.8525514602661133),
 ('imperador', 0.8470561504364014),
 ('príncipe-herdeiro', 0.8317481875419617),
 ('ex-rei', 0.8250480890274048),
 ('príncipe', 0.8220226168632507),
 ('príncipe-regente', 0.8072745203971863),
 ('faraó', 0.805770993232727),
 ('sacro-imperador', 0.8054810166358948),
 ('cardeal-rei', 0.8019851446151733),
 ('duque', 0.8011463284492493)]
```

```
model_cbow.most_similar_cosmul(positive=['rei', 'mulher'], negative=['homem'])
```

```
[('meia-irmã', 0.9702044129371643),
 ('esposa', 0.9664881229400635),
 ('rainha', 0.9587218165397644),
 ('princesa', 0.9533653259277344),
 ('sobrinha', 0.9444015026092529),
 ('filha', 0.925252377986908),
 ('irmã', 0.9192134141921997),
 ('neta', 0.9184019565582275),
 ('rainha-consorte', 0.9000980854034424),
 ('dama-de-companhia', 0.893262505531311)]
```

```
model_cbow.most_similar_cosmul(positive=['rainha', 'homem'], negative=['mulher'])
```

```
[('rei', 0.8870742321014404),
 ('monarca', 0.8194573521614075),
 ('epigrama', 0.8095493316650391),
 ('mito', 0.8062155246734619),
 ('cardeal-rei', 0.7996988892555237),
 ('faraó', 0.797191321849823),
 ('pro-blema', 0.7968766093254089),
 ('fontanário', 0.7921295762062073),
 ('mago', 0.791897714138031),
 ('vídeo-clipe', 0.7916083335876465)]
```

```
model_cbow.doesnt_match(['fogo', 'água', 'terra', 'mar', 'ar', 'carro'])
```

```
'carro'
```

```
model_cbow.doesnt_match("cachorro almoço gato passarinho".split())
```

```
'almoço'
```

```
# pode ser necessário instalar essa dependência do Gensim
!pip install POT==0.4.0 --quiet
```

```
model_cbow.wmdistance('falou', 'taguarelou')
```

```
0.4503943517881298
```

```
model_cbow.wmdistance('Batatinha quando nasce', 'Pirulito que bate bate')
```

```
0.6247409742175248
```

```
model_cbow.wmdistance('Batatinha quando nasce', 'Batatinha se esparrama')
```

```
0.44451569085446885
```

Comece a programar ou gere código com IA.

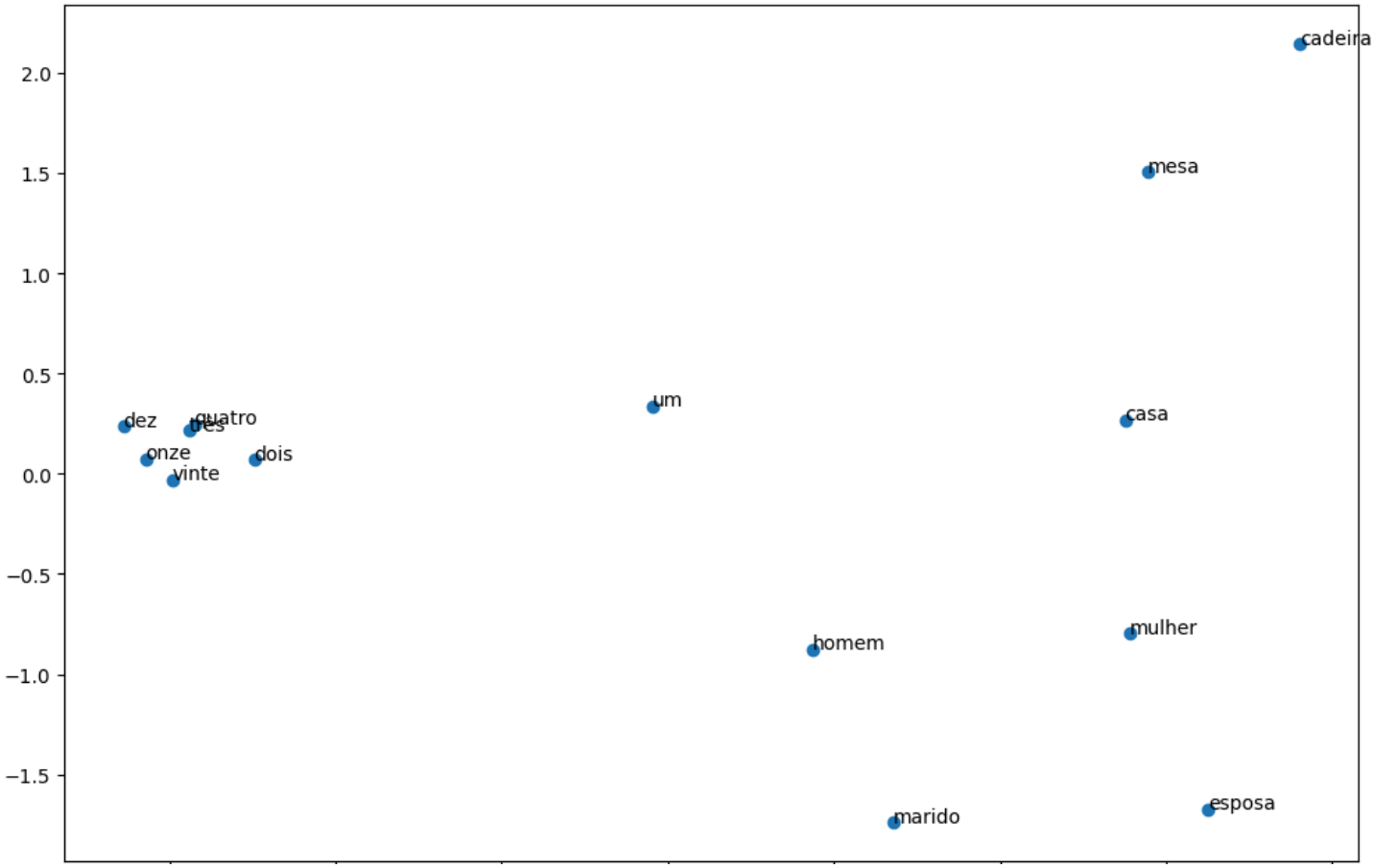
Análise de Componente Principal - PCA (Principal Component Analysis)

```
palavras = ['um', 'dois', 'três', 'quatro', 'dez', 'onze', 'vinte', 'homem', 'mulher', 'marido', 'e
```

```
# Selecionares os dois principais componentes e os mostraremos em um scatter plot.
from sklearn.decomposition import PCA
import numpy as np

sample_vectors = np.array([model_cbow[palavra] for palavra in palavras])
pca = PCA(n_components=2)
result = pca.fit_transform(sample_vectors)
result
```

```
array([[-0.0453743 ,  0.33772764],
       [-1.2444258 ,  0.07335746],
       [-1.4411286 ,  0.21697314],
       [-1.4246814 ,  0.24888164],
       [-1.6386641 ,  0.23907901],
       [-1.5718281 ,  0.07380209],
       [-1.4920576 , -0.02855898],
       [ 0.43695548, -0.8767222 ],
       [ 1.3898493 , -0.79357296],
       [ 0.67999953, -1.7405599 ],
       [ 1.6267622 , -1.6740521 ],
       [ 1.3778124 ,  0.26950768],
       [ 1.4439391 ,  1.5093452 ],
       [ 1.9028431 ,  2.1447928 ]], dtype=float32)
```

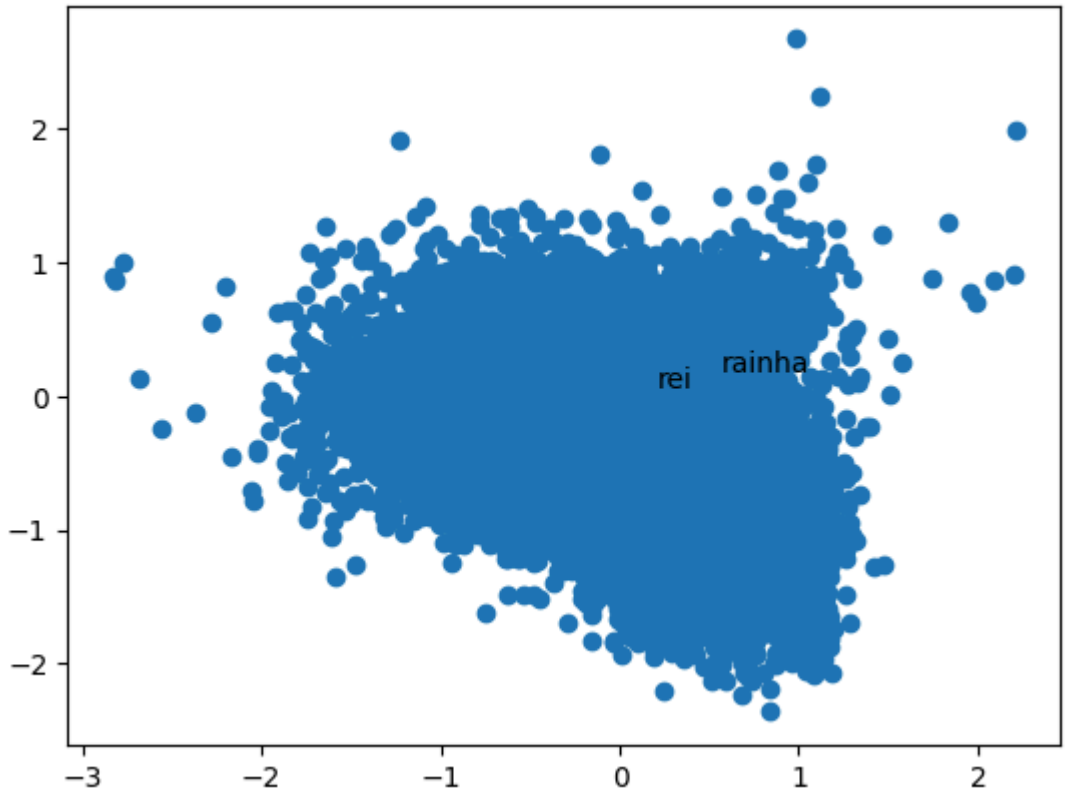
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plt.scatter(result[:,0], result[:,1])
for i, word in enumerate(palavras):
    plt.annotate(word, xy=(result[i, 0], result[i, 1]))
plt.show()
```



[Mais sobre PCA]

```
# Nesta etapa vamos usar PCA (Principal Component Analysis) para visualizar o modelo embedding que
# Selecionares os dois principais componentes e os mostraremos em um scatter plot.

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

X = model_cbow[model_cbow.index_to_key]
pca = PCA(n_components=2)
result = pca.fit_transform(X)

# criando o scatter plot (Gráfico de dispersão)
plt.scatter(result[:, 0], result[:, 1])

# selecionamos apenas duas palavras para aparecer na visualização
words = ['rei','rainha']

print(words)
for i, word in enumerate(words):
    plt.annotate(word, xy=(result[i, 0], result[i, 1]))
plt.show()
```

```
['rei', 'rainha']
```



Exercício

1. Crie uma função que substitua uma ou mais palavras de um texto um uma palavra similar com base na sua classe gramatical.

Dicas:

- Crie uma função que receba como parâmetro: a frase, a classe gramatical e o sentido (palavra no mesmo sentido - positivo)
- Use o Spacy para localizar a marcação POS Tag
- Use o modelo Word2Vec que treinanmos para localizar a similaridade

```
# resposta
#!python -m spacy download pt_core_news_sm --quiet
#model_cbow.most_similar_cosmul(positive=['alto'], topn=10)[0][0]
```

```
import spacy
# Certifique-se de que o gensim está instalado se você estiver usando um modelo Word2Vec do Gensim
# Ex: from gensim.models import Word2Vec

# Tenta carregar o modelo de português do SpaCy. Se não existir, baixa e instala.
try:
    nlp = spacy.load('pt_core_news_sm')
except OSError:
    print("Modelo 'pt_core_news_sm' do SpaCy não encontrado. Baixando e instalando...")
    print("Pode ser necessário reiniciar o kernel do Jupyter/Colab após a instalação.")
    spacy.cli.download('pt_core_news_sm')
    nlp = spacy.load('pt_core_news_sm')

def _encontrar_palavras_similares(modelo_w2v, palavra_referencia, topn=10):
    """
    Função auxiliar para buscar palavras similares em um modelo Word2Vec (Gensim).
    Tenta ser compatível com diferentes APIs do Gensim.
    Levanta KeyError se a palavra não estiver no vocabulário.
    Levanta AttributeError se um método de similaridade compatível não for encontrado.
    """
    # Prioriza o método 'most_similar_cosmul' se existir (conforme dica do usuário)
    if hasattr(modelo_w2v, 'most_similar_cosmul'):
        return modelo_w2v.most_similar_cosmul(positive=[palavra_referencia], topn=topn)

    # API comum do Gensim (modelo completo com .wv ou KeyedVectors diretamente)
    if hasattr(modelo_w2v, 'wv') and hasattr(modelo_w2v.wv, 'most_similar'):
        return modelo_w2v.wv.most_similar(positive=[palavra_referencia], topn=topn)

    # API mais antiga do Gensim (método de similaridade direto no objeto do modelo)
    # ou se 'modelo_w2v' for uma instância de KeyedVectors
    if hasattr(modelo_w2v, 'most_similar'):
        return modelo_w2v.most_similar(positive=[palavra_referencia], topn=topn)

    raise AttributeError("O modelo Word2Vec fornecido não possui um método de similaridade compatív


def substituir_palavras_com_similar_e_pos(
    texto_original: str,
    classe_gramatical_alvo: str,
    modelo_w2v, # Seu modelo Word2Vec treinado, ex: model_cbow
    palavra_sentido_referencia: str = None
    ):
    """
    Substitui palavras em um texto por palavras similares com base na classe gramatical.

    Args:
        texto_original (str): A frase ou texto a ser processado.
        classe_gramatical_alvo (str): A classe gramatical alvo para substituição
                                      (ex: "ADJ", "NOUN", "VERB" - tags POS do SpaCy).
        modelo_w2v: O modelo Word2Vec treinado (Gensim).
        palavra_sentido_referencia (str, opcional): Uma palavra para guiar a busca por
                                                    similaridade. Se fornecida, busca uma
                                                    palavra similar a esta com a POS alvo,
                                                    e a usa para todas as substituições.
                                                    Se None, busca similares para cada
                                                    palavra individualmente.

    Returns:
        str: O texto com as palavras substituídas.
```

```
    """
    doc_texto = nlp(texto_original)
    palavra_substituta_unica = None # Usada se palavra_sentido_referencia for fornecida

    # CENÁRIO 1: palavra_sentido_referencia é fornecida.
    # Encontra UMA palavra para substituir todas as ocorrências da classe_gramatical_alvo.
    if palavra_sentido_referencia:
        try:
            palavras_candidatas = _encontrar_palavras_similares(modelo_w2v, palavra_sentido_referer
            for candidata_str, _ in palavras_candidatas:
                # Evita substituir pela própria palavra de referência
                if candidata_str.lower() == palavra_sentido_referencia.lower():
                    continue

                doc_candidata = nlp(candidata_str)
                # Verifica se a candidata é uma única palavra e tem a POS correta
                if doc_candidata and len(doc_candidata) == 1 and doc_candidata[0].pos_ == classe_gr
                    palavra_substituta_unica = candidata_str
                    print(f"Info: Usando '{palavra_substituta_unica}' (similar a '{palavra_sentido_
                    break
            if not palavra_substituta_unica:
                print(f"Info: Nenhuma palavra similar a '{palavra_sentido_referencia}' com POS '{cl

        except KeyError:
            print(f"Aviso: A palavra de sentido '{palavra_sentido_referencia}' não foi encontrada r
        except AttributeError as e:
            print(f"Erro com o modelo Word2Vec: {e}")
        except Exception as e_geral:
            print(f"Erro inesperado ao processar palavra de sentido '{palavra_sentido_referencia}':


    # Constrói o texto final, token por token
    texto_modificado_partes = []
    for token in doc_texto:
        palavra_final_para_token = token.text # Palavra original por padrão

        if token.pos_ == classe_gramatical_alvo:
            palavra_substituta_escolhida = None

            if palavra_sentido_referencia and palavra_substituta_unica:
                # CENÁRIO 1 está ativo e uma palavra substituta única foi encontrada
                palavra_substituta_escolhida = palavra_substituta_unica

            elif not palavra_sentido_referencia:
                # CENÁRIO 2: Nenhuma palavra_sentido_referencia. Busca similar para o token atual.
                lemma_do_token = token.lemma_
                try:
                    palavras_candidatas_individuais = _encontrar_palavras_similares(modelo_w2v, lem
                    for candidata_individual_str, _ in palavras_candidatas_individuais:
                        # Evita substituir pela própria palavra (ou seu lemma)
                        if candidata_individual_str.lower() == lemma_do_token.lower():
                            continue

                        doc_candidata_individual = nlp(candidata_individual_str)
                        if doc_candidata_individual and len(doc_candidata_individual) == 1 and \
                           doc_candidata_individual[0].pos_ == classe_gramatical_alvo:
                            palavra_substituta_escolhida = candidata_individual_str
                            break # Encontrou substituta para este token
```

```
            except KeyError:
                # Lemma não está no vocabulário, mantém palavra original. Silencioso para não p
                pass
            except AttributeError as e:
                # Problema com o modelo, já reportado ou será no primeiro uso. Silencioso aqui.
                pass
            except Exception as e_geral_token:
                 print(f"Erro inesperado ao processar token '{token.text}': {e_geral_token}")

        # Se uma substituta foi escolhida (de qualquer cenário), aplica capitalização
        if palavra_substituta_escolhida:
            palavra_final_para_token = palavra_substituta_escolhida
            # Tenta preservar a capitalização original
            if token.text.istitle(): # Ex: "Palavra"
                palavra_final_para_token = palavra_final_para_token.capitalize()
            elif token.text.isupper() and len(token.text) > 1: # Ex: "SIGLA" ou "PALAVRA"
                palavra_final_para_token = palavra_final_para_token.upper()
            # else: mantém a capitalização da palavra similar (geralmente minúscula)

    texto_modificado_partes.append(palavra_final_para_token + token.whitespace_)

return "".join(texto_modificado_partes).strip()
```

```
# Exemplo de uso (assumindo que model_cbow está carregado):

# --- Cenário 1: Com palavra_sentido_referencia ---
# Suponha que model_cbow tenha 'rápido' e 'veloz' (ADJ) como similar
frase1 = "O carro é rápido e o avião é rápido também."
print(f"Original: {frase1}")
modificada1 = substituir_palavras_com_similar_e_pos(
    frase1,
    "ADJ",
    model_cbow, # Seu modelo carregado
    palavra_sentido_referencia="veloz"
)
print(f"Modificada (com referência 'veloz' para ADJ): {modificada1}\n")

# --- Cenário 2: Sem palavra_sentido_referencia ---
# Substitui cada adjetivo por um similar individualmente
frase2 = "Ele era um homem alto e forte."
print(f"Original: {frase2}")
# Suponha que 'alto' tenha 'elevado' (ADJ) como similar
# E 'forte' tenha 'robusto' (ADJ) como similar
modificada2 = substituir_palavras_com_similar_e_pos(
    frase2,
    "ADJ",
    model_cbow # Seu modelo carregado
)
print(f"Modificada (ADJs individuais): {modificada2}\n")

frase3 = "A casa bonita tem um jardim grande."
print(f"Original: {frase3}")
# Suponha 'bonita' -> 'bela' (ADJ) e 'grande' -> 'enorme' (ADJ)
modificada3 = substituir_palavras_com_similar_e_pos(
    frase3,
    "ADJ",
    model_cbow
)
print(f"Modificada (ADJs individuais): {modificada3}\n")

frase4 = "Eu gosto de comer uma maçã saborosa."
print(f"Original: {frase4}")
# Tentar substituir VERB usando "ingerir" como referência de sentido
modificada4 = substituir_palavras_com_similar_e_pos(
    frase4,
    "VERB", # Alvo é VERB
    model_cbow,
    palavra_sentido_referencia="ingerir" # Palavra de sentido
)
print(f"Modificada (VERB com referência 'ingerir'): {modificada4}")
```

```
Original: O carro é rápido e o avião é rápido também.
Info: Usando 'manobrável' (similar a 'veloz') para substituir palavras com POS 'ADJ'.
Modificada (com referência 'veloz' para ADJ): O carro é manobrável e o avião é manobrável tambe

Original: Ele era um homem alto e forte.
Modificada (ADJs individuais): Ele era um homem altíssimo e fraco.

Original: A casa bonita tem um jardim grande.
Modificada (ADJs individuais): A casa divertido tem um jardim enorme.

Original: Eu gosto de comer uma maçã saborosa.
```

```
Info: Usando 'consumir' (similar a 'ingerir') para substituir palavras com POS 'VERB'.
Modificada (VERB com referência 'ingerir'): Eu consumir de consumir uma maçã saborosa.
```

Criando nosso próprio modelo de similaridade

Vamos usar modelos de Word Embedding através de vetores.

```
import pandas as pd
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
from gensim.models import Word2Vec
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
# Vamos usar nosso dataframe de produtos
df = pd.read_csv("https://dados-ml-pln.s3-sa-east-1.amazonaws.com/produtos.csv", delimiter=";", enc
df.dropna(inplace=True)
df["texto"] = df['nome'] + " " + df['descricao']
```

```
# Tokenizando o texto
df['tokens'] = df.texto.apply(word_tokenize)
```

```
# Treinando nosso modelo com o nosso vocabulário
modelo_produto = Word2Vec(df['tokens'], vector_size=100, min_count=5, sg=0)

# Treina o modelo. Note que estamos setando o parametro sd = 0, ou seja, o algoritmo utilizado sera
# Outro ponto de destaque é que quando setamos o parametro min_count para 1 estamos forcando o uso
```

size: (default 100) The number of dimensions of the embedding, e.g. the length of the dense vector to represent each token (word).
window: (default 5) The maximum distance between a target word and words around the target word.
min_count: (default 5) The minimum count of words to consider when training the model; words with an occurrence less than this count will be ignored.
workers: (default 3) The number of threads to use while training.
sg: (default 0 or CBOW) The training algorithm, either CBOW (0) or skip gram (1).

Referência: https://machinelearningmastery.com/develop-word-embeddings-python-gensim/

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

# Salva o modelo
modelo_produto.save('/content/gdrive/MyDrive/model.bin')
!ls -la /content/gdrive/MyDrive/model.bin

# Carrega o modelo
modelo_produto = Word2Vec.load('/content/gdrive/MyDrive/model.bin')
'''
```

```
'\nfrom google.colab import drive\ndrive.mount('/content/gdrive')\n\n# Salva o modelo\nmodelo_produto.save('/content/gdrive/MyDrive/model.bin')\n!ls -la /content/gdrive/MyDrive/model.bin\n\n# Carrega o modelo\nmodelo_produto = Word2Vec.load('/content/gdrive/MyDrive/model.bin')\n'
```

```
# Salva o modelo
modelo_produto.save('model.bin')
```

```
!ls -la
```

```
total 3510216
drwxr-xr-x 1 root root       4096 Jun  5 01:01 .
drwxr-xr-x 1 root root       4096 Jun  5 00:47 ..
-rw-r--r-- 1 root root 2656045531 Oct  4  2018 cbow_s300.txt
-rw-r--r-- 1 root root  929305948 Feb 16  2021 cbow_s300.zip
drwxr-xr-x 4 root root       4096 Jun  3 14:04 .config
-rw-r--r-- 1 root root    9079923 Jun  5 01:01 model.bin
drwxr-xr-x 1 root root       4096 Jun  3 14:04 sample_data
```

```
# Carrega o modelo
model_novo = Word2Vec.load('model.bin')

# Exibe os parametros do meodelo treinado
print(model_novo)
```

```
Word2Vec<vocab=10930, vector_size=100, alpha=0.025>
```

```
# Buscando similaridade de palavras com o nosso vocabulário
model_novo.wv.most_similar(positive=['maquiagem'], topn=10)
```

```
[('base', 0.9171795845031738),
 ('auxilia', 0.9105656147003174),
 ('profissional', 0.9032392501831055),
 ('aplicar', 0.8821938633918762),
 ('corretivo', 0.8760757446289062),
 ('aplicação', 0.8662863373756409),
 ('pó', 0.8633875846862793),
 ('cerdas', 0.8537936806678772),
 ('pincel', 0.8510565161705017),
 ('contorno', 0.8487333059310913)]
```

```
# Buscando similaridade de palavras com o nosso vocabulário
model_novo.wv.most_similar_cosmul(positive=['romance'])
model_novo.wv.most_similar_cosmul(positive=['mistério'])
```

```
[('humano', 0.9723255038261414),
 ('ataque', 0.9710063934326172),
 ('último', 0.9702814817428589),
 ('leitor', 0.9695239663124084),
 ('verdadeiro', 0.9693001508712769),
 ('menino', 0.9671421647071838),
 ('sob', 0.9664801955223083),
 ('noite', 0.9662364721298218),
 ('misterioso', 0.9660230875015259),
 ('inesperado', 0.9651471972465515)]
```

```
!pip install POT==0.4.0 --quiet
```

```
# Comparando duas palavras
model_novo.wv.wmdistance('maquiagem', 'base')
```

```
0.4345238171405993
```

```
# Comparando duas palavras
model_novo.wv.wmdistance('maquiagem', 'brinquedo')
```

```
0.7503230348468197
```

Extra!

Outra forma de criar modelos de similaridade (mais simples)

Usando matrix de similaridade e TF-IDF

Referência: https://hackinganalytics.com/2020/03/01/comparando-textos-com-tf-idf-e-cosine-similarity-no-gensim/

Sou autorizado a replicar esse conteúdo

```
import pandas as pd
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
from gensim.corpora import Dictionary
from gensim.models import TfidfModel
from gensim import similarities

#import gensim.downloader as api
```

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

```
# Cria os dados de treino. Exemplo simples para fins didático.
lista = [
    [1, 'Este servico e oferecido em Sao Paulo'],
    [2, 'Este servico e oferecido no Rio de Janeiro'],
    [3, 'Este servico e oferecido em Tocantins'],
    [4, 'Este servico e oferecido na Bahia']]

df = pd.DataFrame(data = lista, columns = ['id', 'descicao'])
df.head()
```

	id	descicao
0	1	Este servico e oferecido em Sao Paulo
1	2	Este servico e oferecido no Rio de Janeiro
2	3	Este servico e oferecido em Tocantins
3	4	Este servico e oferecido na Bahia

```
# Transforma a descrição em tokens
df['tokens'] = df.descicao.apply(word_tokenize)

# Cria um dicionario com os tokens do dataframe
dct = Dictionary(df['tokens'])

# Converte corpus para o formato BOW (Bag Of Words)
corpus = [dct.doc2bow(line) for line in df['tokens']]

# Cria um modelo TF-IDF usando o gensim
model = TfidfModel(corpus)
```

Analisando cada objeto criado

```
# Sentença em tokens
df['tokens']
```

	tokens
0	[Este, servico, e, oferecido, em, Sao, Paulo]
1	[Este, servico, e, oferecido, no, Rio, de, Jan...
2	[Este, servico, e, oferecido, em, Tocantins]
3	[Este, servico, e, oferecido, na, Bahia]

dtype: object

```
# Cada plavra com sua identificação (id)
dct.token2id
```

```
{'Este': 0,
 'Paulo': 1,
 'Sao': 2,
 'e': 3,
 'em': 4,
 'oferecido': 5,
```

```
      'servico': 6,
      'Janeiro': 7,
      'Rio': 8,
      'de': 9,
      'no': 10,
      'Tocantins': 11,
      'Bahia': 12,
      'na': 13}
```

```
# Contagem de termos simples
corpus
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)],
 [(0, 1), (3, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1)],
 [(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (11, 1)],
 [(0, 1), (3, 1), (5, 1), (6, 1), (12, 1), (13, 1)]]
```

```
# Deswcrição: Este servico e oferecido em Sao Paulo
corpus[0]
```

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)]
```

```
# TF-IDF da deswcrição "Este servico e oferecido em Sao Paulo"
model[corpus[0]]
```

```
[(1, 0.6666666666666666), (2, 0.6666666666666666), (4, 0.3333333333333333)]
```

```
# Outro forma de salvar o modelo para reutilizar e produtizar
dct.save('dct')
```