Ambiente

```
import pandas as pd
import numpy as np
import seaborn as sns
import os
import nltk
nltk.download('punkt')
```

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

Datasets

AmericanasBR

```
#baixando os datasets
!curl https://www.inf.ufrgs.br/~viviane/DS/B2W-Reviews01_binario5000_TRAIN.csv > B2W-Reviews01_binario5000_TRAIN.csv
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1657k  100 1657k    0     0   569k      0  0:00:02  0:00:02 --:--:--  569k
```

```
df_train = pd.read_csv('B2W-Reviews01_binario5000_TRAIN.csv')
```

```
classes = df_train.label.unique()
classes
```

```
array([0, 1])
```

Entendendo as Embeddigns

A bilbioteca Gensim permite treinar e usar word embedings.

A versão da biblioteca a ser usada neste notebook é a 4.

Veja diferenças entre versão 3 e 4 neste link.

```
!pip install gensim
```

```
Collecting gensim
  Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.1 kB)
Collecting numpy<2.0,>=1.18.5 (from gensim)
  Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 3.1 MB/s eta 0:00:00
Collecting scipy<1.14.0,>=1.7.0 (from gensim)
  Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 4.4 MB/s eta 0:00:00
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
Downloading gensim-4.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.7/26.7 MB 70.6 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 91.5 MB/s eta 0:00:00
Downloading scipy-1.13.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.6/38.6 MB 12.5 MB/s eta 0:00:00
Installing collected packages: numpy, scipy, gensim
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.14.1
    Uninstalling scipy-1.14.1:
      Successfully uninstalled scipy-1.14.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
Successfully installed gensim-4.3.3 numpy-1.26.4 scipy-1.13.1
```

```
!pip install numpy==1.23.5 #Para usar o gensim
```

```
Requirement already satisfied: numpy==1.23.5 in /usr/local/lib/python3.11/dist-packages (1.23.5)
```

```
from gensim import utils
import gensim.models
```

```
gensim.__version__
```

```
'4.3.3'
```

Treinar embeddings

A biblioteca Gensim permite que você treine as embeddings do seu corpus

https://radimrehurek.com/gensim/models/word2vec.html

```
# classe para montar o dataset
class PreProcess:
    def __init__(self, docs):
            self.lista_text = docs
    def __iter__(self):
        for line in self.lista_text:
            # assume there's one document per line, tokens separated by whitespace:
            yield utils.simple_preprocess(line) # este método tokeniza e faz algum preprocessamento
            # https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html
```

Alguns parametros do Word2vec

- `vector_size` – dimensionalidade dos vetores das palavras.
- `window` – tamanho do contexto a considerar, por exemplo windows=5 irá considerar as 5 palavras à esquerda e as 5 palavras à direita da palavra atual como a janela de contexto. O modelo tentará então prever a palavra atual dado este contexto.
- `min_count` – ignora palavras com frequência total menor do que min_count.
- `sg` – o algoritmo de treinamento: 1 for skip-gram e diferente disto CBOW.

```
sentences = PreProcess(df_train['text'].values)
# assim treina o modelo usando as configurações padrão e estas especificadas aqui
model = gensim.models.Word2Vec(sentences=sentences, vector_size=100, window=5, min_count=1, epochs=20, sg=1)
```

Quando não se necessita mais do estado completo do modelo treinado (não precisa continuar treinando), `Gensim` permite separar os vetores treinados em `KeyedVectors` possibilitando salvar apenas os vetores e suas chaves (as palavras).

```
# desta forma acessa somente as palavras e seus vetores
word_vectors = model.wv
word_vectors
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7f89b7263490>
```

```
# total de palavras e as 10 primeiras
words = list(word_vectors.key_to_index)
print(f'O vocabulario contém {len(words)} palavras')
print(words[0:10])
```

```
O vocabulario contém 13905 palavras
['de', 'não', 'produto', 'que', 'muito', 'com', 'do', 'um', 'para', 'da']
```

```
# verificando o id de uma palavra:
print('id de entrega:', word_vectors.key_to_index['entrega'])
print('palavra do id 4:', word_vectors.index_to_key[4])
```

```
id de entrega: 12
palavra do id 4: muito
```

```
# ocorrências de uma palavra:
palavra = 'entrega'
palavra_cnt = word_vectors.get_vecattr(palavra, "count")
print(f'A palavra {palavra} ocorre {palavra_cnt} vezes no dataset')
```

```
A palavra entrega ocorre 1770 vezes no dataset
```

Cada palavra única do corpus é representada por um vetor de tamanho `vector_size`, que corresponde ao número de dimnesões usado no treinamento.

```
print(f"Embeddings da palavra produto com dimensão {word_vectors['produto'].shape}")
word_vectors['produto']
```

```
Embeddings da palavra produto com dimensão (100,)
array([ 0.17470533, -0.14190084,  0.06710811, -0.01138282, -0.03957819,
       -0.32239056,  0.33270392,  0.49581882, -0.40610936, -0.42311403,
       -0.24202031, -0.32410857,  0.2474162 ,  0.28356737,  0.44602737,
        0.01523447,  0.3702031 ,  0.15064284, -0.30331457, -0.62956476,
        0.3497122 ,  0.14535832,  0.3340684 , -0.16197301,  0.3481385 ,
       -0.24719895,  0.07718111,  0.02790798, -0.42369196,  0.17972656,
        0.23585007, -0.01800809,  0.04085732, -0.3840595 , -0.05642095,
        0.3282759 ,  0.33011034,  0.2763505 ,  0.17933546,  0.24010561,
        0.28481728, -0.09099186, -0.3645987 , -0.11849046,  0.00179532,
        0.18515311,  0.17709017,  0.05294485,  0.28247523, -0.04680588,
        0.12101904, -0.41603193, -0.03781983, -0.7613891 , -0.02665293,
       -0.12948911, -0.06154432,  0.17326157, -0.0212841 ,  0.23879434,
        0.00564251, -0.14533769,  0.2680177 ,  0.23609795, -0.29144403,
        0.46549645,  0.09280705,  0.16777119,  0.02625799,  0.2926556 ,
       -0.05007668,  0.23524506,  0.22846597,  0.3142368 ,  0.523102  ,
       -0.00664081, -0.03582658,  0.2807857 , -0.01140467,  0.33919466,
       -0.11837228, -0.25808263, -0.5538806 ,  0.1165751 , -0.11430335,
       -0.17623112,  0.4346967 , -0.22934772,  0.6616576 , -0.04529468,
       -0.10745972,  0.15758203,  0.3487871 , -0.18396038,  0.6279739 ,
       -0.10332917,  0.2872453 , -0.07162914,  0.04541764, -0.10560982],
      dtype=float32)
```

```
# possuem representações diferentes:
print(word_vectors['agua'][0:5])
print(word_vectors['água'][0:5])
```

```
[-0.26869255  0.58220154  0.5914807   0.23859586 -0.25641334]
[ 0.04064145  0.31516647  0.9487154   0.07355908 -0.05783588]
```

Salvando as embeddings treinadas:

```
# salva o modelo em formato binario do gensim:
model.save("word2vec.model")
```

```
# salva em formato texto somete as palavras e seus vetores de embeddings
word_vectors = model.wv
word_vectors.save_word2vec_format("word2vec.txt", binary= False)
```

Lendo os vetores

```
word_vectors = gensim.models.KeyedVectors.load_word2vec_format('word2vec.txt', binary=False)
```

```
word_vectors
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7f89b7835dd0>
```

```
print('Total de palavras: ',len(word_vectors))
print('id da palavra água:', word_vectors.key_to_index['água'])
print(word_vectors['água'][0:10])
```

```
Total de palavras:  13905
id da palavra água: 206
[ 0.04064145  0.31516647  0.9487154   0.07355908 -0.05783588 -0.35119385
  0.5878381   0.6707491  -0.24897209 -0.7746001 ]
```

```
#somente os vetores das embeddings:
vectors = word_vectors.vectors
vectors
```

```
array([[-0.07574224, -0.08806279,  0.24580932, ..., -0.47713107,
        -0.04412623,  0.15782326],
       [ 0.08310077, -0.03144601,  0.39662305, ..., -0.3478132 ,
        -0.16251448,  0.21877348],
       [ 0.17470533, -0.14190084,  0.06710811, ..., -0.07162914,
         0.04541764, -0.10560982],
       ...,
       [-0.00732837,  0.1457949 ,  0.2128637 , ..., -0.21616185,
        -0.06636345,  0.05660735],
       [-0.03472126,  0.23713963,  0.11934121, ..., -0.3356424 ,
        -0.03144514,  0.05503844],
       [-0.02665094,  0.07851093,  0.07094771, ..., -0.16345543,
        -0.11348496,  0.00252025]], dtype=float32)
```

```
# acessando o vetor da palavra água
print(vectors[206][0:10])
```

```
[ 0.04064145  0.31516647  0.9487154   0.07355908 -0.05783588 -0.35119385
  0.5878381   0.6707491  -0.24897209 -0.7746001 ]
```

Visualizando embeddings

```
from sklearn.manifold import TSNE
import plotly.express as px
```

```
%%time
tsne = TSNE(n_components=3, random_state=0)
projections = tsne.fit_transform(vectors, )
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<timed exec> in <module>

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_set_output.py in wrapped(self, X, *args, **kwargs)
    317     @wraps(f)
    318     def wrapped(self, X, *args, **kwargs):
--> 319         data_to_wrap = f(self, X, *args, **kwargs)
    320         if isinstance(data_to_wrap, tuple):
    321             # only wrap the first output for cross decomposition

                                5 frames
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py in _kl_divergence_bh(params, P, degrees_of_freedom, n_samples,
n_components, angle, skip_num_points, verbose, compute_error, num_threads)
    280
    281     grad = np.zeros(X_embedded.shape, dtype=np.float32)
--> 282     error = _barnes_hut_tsne.gradient(
    283         val_P,
    284         X_embedded,

KeyboardInterrupt:
```

```
dfP = pd.DataFrame(projections)
dfP['word'] = words
fig = px.scatter_3d(dfP, x=0, y=1, z=2,hover_data=['word'])
fig.update_traces(marker_size=3)
fig.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-b28e50820964> in <cell line: 0>()
----> 1 dfP = pd.DataFrame(projections)
      2 dfP['word'] = words
      3 fig = px.scatter_3d(dfP, x=0, y=1, z=2,hover_data=['word'])
      4 fig.update_traces(marker_size=3)
      5 fig.show()

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

Próximas etapas: Explicar o erro

Também é possível usar o Embeddings Projector do Tensorflow.

Usando embeddings já treinadas

Podemos usar word embeddings que já foram treinadas e disponibilizadas

Existem modelos disponíveis no Gensim:

```
import gensim.downloader as api
```

```
disponiveis = api.info()
disponiveis.keys()
```

```
dict_keys(['corpora', 'models'])
```

```
disponiveis['models'].keys()
```

```
dict_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'])
```

```
word_vectors = api.load("glove-wiki-gigaword-100") #128Mb
```

```
[==================================================] 100.0% 128.1/128.1MB downloaded
```

```
#é uma lista de palavras e seus vetores de embeddinsg treinados por alguém e em algum algoritmo que normlmente é especificado na nomenc
word_vectors
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7f89ae5f6610>
```

```
from gensim.models import KeyedVectors
```

```
#baixando as embeddings do NILC de http://nilc.icmc.usp.br/embeddings
!curl http://143.107.183.175:22980/download.php?file=embeddings/word2vec/cbow_s100.zip > cbow_s100.zip
!unzip -o cbow_s100.zip
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  310M  100  310M    0     0  5898k      0  0:00:53  0:00:53 --:--:-- 7744k
Archive:  cbow_s100.zip
  inflating: cbow_s100.txt
```

```
%%time
nomearq = 'cbow_s100.txt'
nilc_word_vectors = KeyedVectors.load_word2vec_format(nomearq, binary=False)  # binary=True se for um arquivo binário
print('Carregado: ',nomearq)
```

```
Carregado:  cbow_s100.txt
CPU times: user 1min 13s, sys: 2.6 s, total: 1min 15s
Wall time: 1min 15s
```

```
# total de palavras e as 10 primeiras
words = list(nilc_word_vectors.key_to_index)
print(f'O vocabulario contém {len(words)}')
```

```
O vocabulario contém 929606
```

Operações com embeddings

A similaridade entre vetores de embeddings é dada pelo cosseno. Quanto mais próximo de 1 mais similar

```
similarity = nilc_word_vectors.similarity('maçã', 'manga')
similarity
```

```
0.6305432
```

```
nilc_word_vectors.similarity('veiculo', 'carro')
```

```
0.61882997
```

```
similarity = nilc_word_vectors.similarity('mulher', 'fruta')
similarity
```

```
0.41813976
```

```
import numpy as np
from numpy.linalg import norm
```

```
A = nilc_word_vectors['maçã']
B = nilc_word_vectors['manga']
```

```
# lembrando que o cosseno é o produto escalar normalizado
cosine = np.dot(A,B)/(norm(A)*norm(B))
print("Cosine Similarity:", cosine)
```

```
Cosine Similarity: 0.6305433
```

engenheira = (engenheiro - homem) + mulher

```
#Analogia correta
result = nilc_word_vectors.most_similar(positive=['mulher', 'engenheiro'], negative=['homem'], topn=1)
print(result)
```

```
[('engenheira', 0.697005033493042)]
```

```
#Analogia incorreta com viés
result = nilc_word_vectors.most_similar(positive=['mulher', 'médico'], negative=['homem'], topn=1)
print(result)
```

```
[('enfermeira', 0.725476086139679)]
```

Material suplementar - outras embeddings

Glove: http://github.com/stanfordnlp/glove

Word2vec treinado no detalhe no Keras: https://www.tensorflow.org/tutorials/text/word2vec

Doc2Vec

https://cs.stanford.edu/~quocle/paragraph_vector.pdf

https://alvinntnu.github.io/python-notes/nlp/doc2vec.html

Exercícios para entregar

Carregue as embeddings indicadas em português para os três exercicios.

Exercício 1

A polissemia ocorre quando uma mesma palavra possui mais de um significado. Um exemplo de polissemia é a palavra “manga”, que pode ser parte de vestimenta ou uma fruta.

a) Usando a função `most_similar` do Gensim, analise o resultado para a palavra "manga". Escolha outra palavra polissêmica que exista no vocabulário e verifique as palavras mais similares.

```
manga_result = nilc_word_vectors.most_similar('manga')
laranja_result = nilc_word_vectors.most_similar('laranja')

print('Maior similaridade com manga\n', manga_result)
# O resultado de proximidade para a palavra manga trouxe tanto palavras relacionas a manga ( de roupa ) como capa, lona, lapela (tudo rel
# relacionadas com frutas ( laranja, groselha, maça )
print('\n\n')
print('Maior similaridade com laranja\n', laranja_result)
# O resultado de proximidade para a palavra laranja trouxe palavras relacionadas a frutas como limao, ameixa, groselha , mas também palav
```

```
Maior similaridade com manga
 [('lapela', 0.6999149918556213), ('cola', 0.6836391687393188), ('laranja', 0.6765139102935791), ('groselha', 0.6659492254257202),


Maior similaridade com laranja
 [('limão', 0.7387260794639587), ('castanha', 0.7295199036598206), ('creme', 0.7138504385948181), ('avelã', 0.7022291421890259), ('m
```

b) O que você observa e qual a sua hipótese para explicar esse comportamento.

> Sua análise aqui

Exercício 2

a) Faça o exercício para duas palavras:

Escolha uma palavra, um sinônimo e um antônimo da mesma. Calcule a distância euclideana e a similaridade do cosseno entre a palavra e seu sinônimo e a palavra e seu antônimo.

```
palavra = nilc_word_vectors['bom']
sinonimo = nilc_word_vectors['otimo']
antonimo = nilc_word_vectors['mau']
```

```
cosine_antonimo = np.dot(palavra,antonimo)/(norm(palavra)*norm(antonimo))
print("Cosine Similarity Antonimos:", cosine_antonimo)


cosine_sinonimo= np.dot(palavra,sinonimo)/(norm(palavra)*norm(sinonimo))
print("Cosine Similarity Sinonimos:", cosine_sinonimo)


distancia_antonimo = np.linalg.norm(palavra - antonimo)
distancia_sinonimo = np.linalg.norm(palavra - sinonimo)

print("Distancia euclidiana antonimo", distancia_antonimo)
print("Distancia euclidiana sinonimo", distancia_sinonimo)
```

```
Cosine Similarity Antonimos: 0.67988425
Cosine Similarity Sinonimos: 0.60562664
Distancia euclidiana antonimo 1.7446551
Distancia euclidiana sinonimo 1.8038198
```

+ Código + Texto

b) O que você observa e qual a sua hipótese para explicar esse comportamento.

Na similaridade entre cossenos o resultado deu relativamente alto, com 0.67 para similaridade entre antonimos e 0.60 para sinonimos. Faz sentido esses valores , pois possivelmente no corpus quando aparecia a palavra bom possivelmente as palavras mau ou alguma palavra de sinonimo de qualidade (como otimo) apareceria por perto. Já com a distancia euclidiana conseguimos afirmar que essas palavras tem uma relação até que próxima visto que para ambos resultados a distancia foi menor do que 2

Exercício 3

a) Verifique as palavras mais similares em relação às palavras "enfermeiro" e "enfermeira".

```
enfermeiro_sim = nilc_word_vectors.most_similar('enfermeiro')
enfermeira_sim = nilc_word_vectors.most_similar('enfermeira')


print('Palavras mais similares com enfermeiro\n\n', enfermeiro_sim)
print('Palavras mais similares com enfermeira\n\n', enfermeira_sim)
```

```
Palavras mais similares com enfermeiro

 [('anestesista', 0.7805444002151489), ('dentista', 0.7740222215652466), ('psicólogo', 0.7639155983924866), ('ortopedista', 0.75170
Palavras mais similares com enfermeira

 [('cabeleireira', 0.851155698299408), ('psicóloga', 0.8485087156295776), ('prostituta', 0.8362278938293457), ('faxineira', 0.83085
```

b) Você observa algum viés? Se sim, qual sua hipótese.

Existe um viés onde para a palavra enfermeiro as outras mais similares estão relacionadas a profissões esteriotipadas como profissões masculina como, por exemplo, especializações na medicina. Já para a palavra enfeirmeira é perceptível que as palavras mais similares estão relacionadas a profissões esteriotipadas como profissoes femininas como , por exemplo, cabeleireira, psicóloga, faxineira, prostituta etc..