

▼ Demonstração - Aula 5

Ativar sessão com GPU (TPU - T4)

Até aqui vimos como usar **word embeddings generativos** (pré-treinados) focando em análise de similaridade e manuseio do espaço vetorial de alguns algoritmos como Word2Vec.

E se tentarmos usar essa "inteligência" para resolver nosso problema de classificação?

▼ Análise com skip-gram em Português

```
# Instalação do pacote Gensim e dependência
!pip install gensim==4.3.2 scipy==1.10.1 numpy==1.26.4 --quiet
```

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

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

```
4.3.2
```

Download Word Embeddings Pré-treinadas em Português (skip-gram)

- Repositório original: <http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc>

```
# Download do arquivo no repositório do professor
!wget 'https://dados-ml-pln.s3-sa-east-1.amazonaws.com/skip_s300.zip'
```

```
# Descompactação do arquivo
!unzip 'skip_s300.zip' # substitua com nome do arquivo
```

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

```
model_skip = KeyedVectors.load_word2vec_format('skip_s300.txt')
model_skip
```

```
--2025-06-04 23:25:46-- https://dados-ml-pln.s3-sa-east-1.amazonaws.com/skip_s300.zip
Resolving dados-ml-pln.s3-sa-east-1.amazonaws.com (dados-ml-pln.s3-sa-east-1.amazonaws.com)... 3.5.233.153, 52.95.165.15, 52.95.164.110, ...
Connecting to dados-ml-pln.s3-sa-east-1.amazonaws.com (dados-ml-pln.s3-sa-east-1.amazonaws.com)|3.5.233.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 958619745 (914M) [application/zip]
Saving to: 'skip_s300.zip.1'

skip_s300.zip.1      100%[=====>] 914.21M  11.5MB/s   in 86s

2025-06-04 23:27:13 (10.6 MB/s) - 'skip_s300.zip.1' saved [958619745/958619745]

Archive:  skip_s300.zip
replace skip_s300.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: skip_s300.txt
<gensim.models.keyedvectors.KeyedVectors at 0x7e233640ee10>
```

```
!ls -la
```

```
total 4465744
drwxr-xr-x 1 root root      4096 Jun  4 23:27 .
drwxr-xr-x 1 root root      4096 Jun  4 23:11 ..
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
-rw-r--r-- 1 root root 2655642222 Oct  4 2018 skip_s300.txt
-rw-r--r-- 1 root root 958619745 Nov 30 2023 skip_s300.zip
-rw-r--r-- 1 root root 958619745 Nov 30 2023 skip_s300.zip.1
```

Load do Word2Vec com CBOW

```
# Download do arquivo no repositório do professor
!wget 'https://dados-ml-pln.s3-sa-east-1.amazonaws.com/cbow_s300.zip'
```

```
# Descompactação do arquivo
!unzip 'cbow_s300.zip' # substitua com nome do arquivo
```

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

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

```
model_skip
```

```
<gensim.models.keyedvectors.KeyedVectors at 0x7e233640ee10>
```

Análise de similaridade

```
model_skip.similarity('maçã', 'uva')
```

```
0.6507031
```

```
model_skip.similarity('maçã', 'carro')
```

```
0.074699655
```

```
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_skip.similarity(w1, w2)))
```

```
'carro' 'jipe' 0.71
'carro' 'avião' 0.50
'carro' 'bicicleta' 0.55
'carro' 'cereal' 0.08
'carro' 'filosofia' -0.01
```

```
model_skip.most_similar(positive=['carro', 'jipe'], topn=3)
```

```
[('furgão', 0.7599552273750305),
 ('caminhão', 0.7516525387763977),
 ('veículo', 0.7478904724121094)]
```

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

```
<gensim.models.keyedvectors.KeyedVectors at 0x7e233640ee10>
```

```
model_skip.most_similar(positive=['rainha', 'homem'], negative=['mulher'], topn=3)
```

```
[('rei', 0.5894271731376648),
 ('monarca', 0.49123090505599976),
 ('guardião', 0.46204233169555664)]
```

```
model_skip.most_similar(positive=['rei', 'mulher'], negative=['homem'], topn=10)
```

```
[('rainha', 0.660095751285553),
 ('consorte', 0.6526049375534058),
 ('esposa', 0.6504772305488586),
 ('sobrinha', 0.6446163654327393),
 ('princesa', 0.6398769617080688),
 ('filha', 0.6342788338661194),
 ('rainha-viúva', 0.6339502334594727),
 ('primogênita', 0.6332842707633972),
 ('princesa-eleitora', 0.6240091323852539),
 ('meia-irmã', 0.6229892373085022)]
```

```
palavras = ['um', 'dois', 'três', 'quatro', 'dez', 'onze', 'vinte', 'homem', 'mulher', 'marido', 'esposa', 'casa', 'mesa', 'cadeira']
```

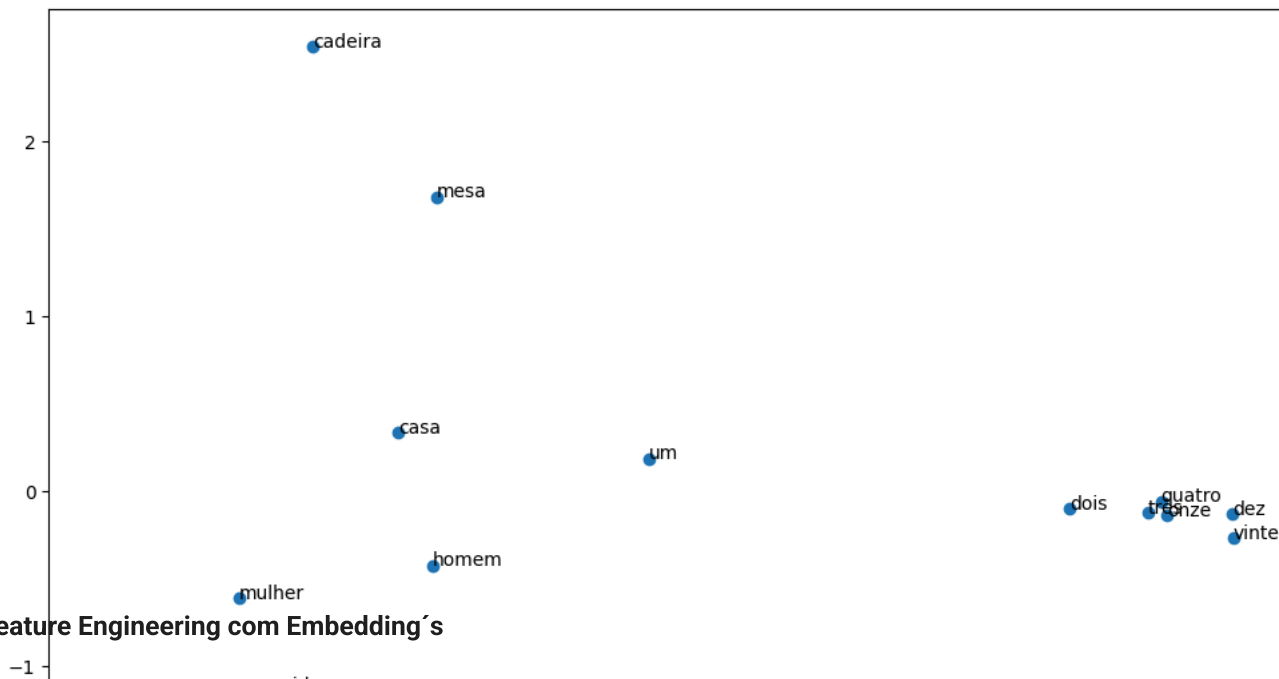
```
from sklearn.decomposition import PCA
import numpy as np
```

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

```
array([[ -0.17956805,  0.18375202],
 [ 1.22293358, -0.09905089],
 [ 1.4864911, -0.1221025 ],
 [ 1.5300087, -0.05887511],
 [ 1.7701347, -0.13250092],
 [ 1.5516897, -0.14122166],
 [ 1.7721732, -0.26937428],
 [-0.9042849, -0.42676726],
 [-1.5513343, -0.61161923],
 [-1.4905175, -1.1370462 ],
 [-1.9972197, -1.73543  ],
 [-1.0187491,  0.33234295],
 [-0.8883214,  1.6803375 ],
 [-1.303439,  2.5375557 ]], dtype=float32)
```

```
import matplotlib.pyplot as plt
```

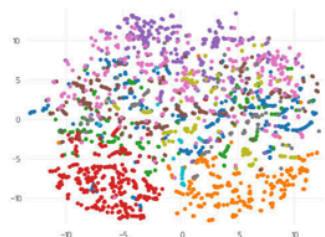
```
plt.figure(figsize=(12,8))
plt.scatter(result[:,0], result[:,1])
for i, word in enumerate(palavras):
    plt.annotate(word, xy=(result[i, 0], result[i, 1]))
plt.show()
```



Feature Engineering com Embedding's

A ideia é criar um vetor de features com características de um Embedding generativo e aplicar no nosso problema de classificação de textos.

Word Embedding pré-treinado



Corpus de documentos rotulados



Processo de feature engineering

Vector-space representation

	united	nations	peace
Doc 1	6	9	16
Doc 2	18	13	9
Doc 3	42	17	5
Doc 4	13	11	10

Classificador com word2vec

Criando um Dataset com base em reviews

```
# Install Google play scraper: https://github.com/JoMingyu/google-play-scraper
# Google-Play-Scraper provides APIs to easily crawl the Google Play Store for Python without any external dependencies!
!pip install google_play_scraper --quiet
```

```
import pandas as pd
from tqdm import tqdm
from google_play_scraper import Sort, reviews, app
import numpy as np
```

```
apps_ids = [
    'br.com.brainweb.ifood',
    'com.cerveceriamodelo.modelonow',
    'com.mcdo.mcdonalds',
    'habibs.alphacode.com.br',
    'com.xiaojukeji.didi.brazil.customer',
    'com.ubercab.eats',
    'com.grability.rappi',
    'burgerking.com.br.appandroid',
    'com.vanuatu.aiqfome'
]
```

```
SCORE_SAMPLES = 100
SCORE3_FACTOR = 1; TARGET_COLUMN = 'score'
```

```
#TARGET_COLUMN = 'sentiment'; SCORE3_FACTOR = 2
```

```
app_reviews = []
```

```
for app_id in tqdm(apps_ids):
    for score in list(range(1, 6)):
        for sort_order in [Sort.MOST_RELEVANT, Sort.NEWEST]:
            rvs, _ = reviews(
                app_id,
                lang='pt',
                country='br',
                sort=sort_order,
                count= SCORE3_FACTOR * SCORE_SAMPLES if score == 3 else SCORE_SAMPLES,
                filter_score_with=score
            )
            for r in rvs:
                r['sortOrder'] = 'most_relevant' if sort_order == Sort.MOST_RELEVANT else 'newest'
                r['appId'] = app_id
            app_reviews.extend(rvs)
```

100%|██████████| 9/9 [00:19<00:00, 2.15s/it]

```
len(app_reviews)
```


9000

```
app_reviews[0]
```

```
{'reviewId': '2c392c34-6da1-49c1-a185-05a20803a94a',
 'userName': 'Fabio Santana',
 'userImage': 'https://play-lh.googleusercontent.com/a-/ALV-UjVup99NajQB2nh5-z6-Aiiorw5WCL0fC-jRE8dA8hXvBxycd69M',
 'content': 'Ultimamente quase tudo no Ifood está caro, cupons falsos para dar a impressão de desconto, que no fundo não passa de uma enganação. Raramente faço pedidos umas três ou quatro vezes no mês, às vezes fazer compra no mercado vale a pena, desde que tenha promoção e frete grátis de resto não espero muita coisa.',
 'score': 1,
 'thumbsUpCount': 107,
 'reviewCreatedVersion': '10.67.1',
 'at': datetime.datetime(2025, 5, 6, 19, 3, 36),
 'replyContent': None,
 'repliedAt': None,
 'appVersion': '10.67.1',
 'sortOrder': 'most_relevant',
 'appId': 'br.com.brainweb.ifood'}
```


```
app_reviews_df = pd.DataFrame(app_reviews)
```

```
app_reviews_df.head()
```


	reviewId	userName	userImage	content	score	thumbsUpCount	reviewCreatedVersion	at	replyContent	replied
0	2c392c34-6da1-49c1-a185-05a20803a94a	Fabio Santana	lh.googleusercontent.com/a-/ALV-U...	Ultimamente quase tudo no lfood está caro, cup...	1	107	10.67.1	2025-05-06 19:03:36	None	N
1	8806c2ea-8359-4cca-8196-de979c07abea	ricardo alexandre	lh.googleusercontent.com/a-/ALV-U...	vou dar uma estrela, e é muito ainda!!! já é a...	1	113	10.66.0	2025-04-21 03:53:26	None	N
2	af5a1bd6-cf7a-48f2-a96c-d3d5b2224fb7	Celes	lh.googleusercontent.com/a-/ALV-U...	Disigne horrível, navegação também e promoções...	1	78	10.66.0	2025-04-19 15:34:06	None	N
3	c9686587-086c-4303-a81e-2248cf40143f	Alexandre Maehler	lh.googleusercontent.com/a/ACg8oc...	O aplicativo sempre funcionou bem, mas nós últ...	1	296	10.56.0	2025-02-13 23:48:16	None	N
4	4b8be737-9b61-471e-8ccd-23b20f5559d3	Giovanni Leal	lh.googleusercontent.com/a/ACg8oc...	Atendimento a problemas do consumidor horrendo...	1	625	10.49.0	2025-01-08 16:09:19	None	N

```
app_reviews_df['sentiment'] = 0
app_reviews_df.loc[app_reviews_df["score"] < 3, "sentiment"] = -1
app_reviews_df.loc[app_reviews_df["score"] > 3, "sentiment"] = 1
```

```
app_reviews_df.head()
```

	reviewId	userName	userImage	content	score	thumbsUpCount	reviewCreatedVersion	at	replyContent	replied
0	2c392c34-6da1-49c1-a185-05a20803a94a	Fabio Santana	lh.googleusercontent.com/a-/ALV-U...	Ultimamente quase tudo no lfood está caro, cup...	1	107	10.67.1	2025-05-06 19:03:36	None	N
1	8806c2ea-8359-4cca-8196-de979c07abea	ricardo alexandre	lh.googleusercontent.com/a-/ALV-U...	vou dar uma estrela, e é muito ainda!!! já é a...	1	113	10.66.0	2025-04-21 03:53:26	None	N
2	af5a1bd6-cf7a-48f2-a96c-d3d5b2224fb7	Celes	lh.googleusercontent.com/a-/ALV-U...	Disigne horrível, navegação também e promoções...	1	78	10.66.0	2025-04-19 15:34:06	None	N
3	c9686587-086c-4303-a81e-2248cf40143f	Alexandre Maehler	lh.googleusercontent.com/a/ACg8oc...	O aplicativo sempre funcionou bem, mas nós últ...	1	296	10.56.0	2025-02-13 23:48:16	None	N
4	4b8be737-9b61-471e-8ccd-23b20f5559d3	Giovanni Leal	lh.googleusercontent.com/a/ACg8oc...	Atendimento a problemas do consumidor horrendo...	1	625	10.49.0	2025-01-08 16:09:19	None	N

```
app_reviews_df['sentiment'].value_counts()
```

	count	
sentiment		
-1	3600	
1	3600	
0	1800	

```
df_model = app_reviews_df[['content','score','sentiment']]
```

```
# Normalização de texto
import string

import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')

# lista de stopwords do NLTK
stopwords = nltk.corpus.stopwords.words('portuguese')

# função que remove pontuação
def remove_punctuation(text):
    punctuations = string.punctuation
    table = str.maketrans({key: " " for key in punctuations})
    text = text.translate(table)
    return text

# função que normaliza o texto e remove stopwords
def norm_tokenize(text):
    text = text.lower()
    text = remove_punctuation(text)
    text = "".join([w for w in text if not w.isdigit()])
    text = word_tokenize(text)
    text = [x for x in text if x not in stopwords]
    text = [y for y in text if len(y) > 2]
    #text = " ".join([t for t in text])
    return text
```

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

```
df_model['tokens'] = df_model['content'].apply(norm_tokenize)
```

```
<ipython-input-51-a8bb658abe50>: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_model['tokens'] = df_model['content'].apply(norm_tokenize)
```

```
df_model.head()
```

	content	score	sentiment	tokens
0	Ultimamente quase tudo no ifood está caro, cup...	1	-1	[ultimamente, quase, tudo, ifood, caro, cupons...
1	vou dar uma estrela, e é muito ainda!!! já é a...	1	-1	[vou, dar, estrela, ainda, segunda, vez, faço,...
2	Disigne horrível, navegação também e promoções...	1	-1	[disigne, horrível, navegação, promoções, pra,...
3	O aplicativo sempre funcionou bem, mas nós últ...	1	-1	[aplicativo, sempre, funcionou, bem, últimos, ...

```
# Exemplo 1: Função para obter embeddings médios para cada texto
vectorizer = model_skip
```

```
def average_vector(tokens):
    vector_size = vectorizer.vector_size
    ww_res = np.zeros(vector_size)
    ctr = 1
    for word in tokens:
        if word in vectorizer:
            ctr += 1
            ww_res += vectorizer[word]
    ww_res = ww_res/ctr
    return ww_res
```

```
#vectorizer['maquiagem']
```

```
# Exemplo 2: Função para obter embeddings médios para cada texto
```

```
...
vectorizer = model_skip

def average_vector(words):
    vectors = [vectorizer[word] for word in words if word in vectorizer]
    if vectors:
        return sum(vectors) / len(vectors)
    else:
        vector_size = vectorizer.vector_size
        wv_res = np.zeros(vector_size)
        return wv_res
...
```

```
<img alt="code icon" data-bbox="21 154 41 174"/> \nvectorizer = model_skip\n\ndef average_vector(words):\n    vectors = [vectorizer[word] for word in words if word in vectorizer]\n    if vectors:\n
```

```
# Tokenizar e obter embeddings médios para cada texto
df_model['vector'] = df_model['tokens'].apply(average_vector)
```

```
<img alt="code icon" data-bbox="21 248 41 268"/> <ipython-input-55-e1e7b11c2590>:2: 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_model['vector'] = df_model['tokens'].apply(average_vector)
```

```
#df_model['vector'][0]
```

```
df_model.head()
```

	content	score	sentiment	tokens	vector
0	Ultimamente quase tudo no lfood está caro, cup...	1	-1	[ultimamente, quase, tudo, ifood, caro, cupons...	[-0.017724334035139892, 0.016495222998653643, ...
1	vou dar uma estrela, e é muito ainda!!! já é a...	1	-1	[vou, dar, estrela, ainda, segunda, vez, faço,...	[-0.02211865014396608, -0.0684307242743671, -0...
2	Disigne horrível, navegação também e promoções	1	-1	[disigne, horrível, navegação, promoções, pra	[-0.028950500444625504, -0.07825767648464535, ...

```
x = df_model['vector'].to_list()
y = df_model[TARGET_COLUMN].to_list()
```

```
len(x[0])
```

```
<img alt="code icon" data-bbox="21 740 41 760"/> 300
```

Clique duas vezes (ou pressione "Enter") para editar

```
from sklearn.model_selection import train_test_split
# divisão da amostra entre treino e teste
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=y, random_state = 42)
```

✓ Treina modelo

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
# treinamento do modelo árvore de decisão com o dataframe de treino
model = RandomForestClassifier(random_state=42)
model.fit(x_train, y_train)
```

```
# escoragem da classificação na amostra de teste
y_pred = model.predict(x_test)
```

```
# Avaliação do modelo
```

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix
```

```
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```

precision    recall  f1-score   support

     1       0.45      0.56      0.50      540
     2       0.39      0.39      0.39      540
     3       0.41      0.34      0.37      540
     4       0.43      0.36      0.39      540
     5       0.60      0.63      0.61      540

 accuracy          0.46      2700
  macro avg       0.45      0.46      0.45      2700
 weighted avg     0.45      0.46      0.45      2700

0.457037037037037
```

Comprando o resultado com vetorização Bag of Words

```
def token_to_text(tokens):
    return " ".join(tokens)
```

```
df_model['norm_content'] = df_model['tokens'].apply(token_to_text)
```

```

<ipython-input-63-12ec7bc66d6f>:4: 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_model['norm_content'] = df_model['tokens'].apply(token_to_text)
```

```
df_model.head()
```

	content	score	sentiment	tokens	vector	norm_content
0	Ultimamente quase tudo no lfood está caro, cup...	1	-1	[ultimamente, quase, tudo, ifood, caro, cupons...	[-0.017724334035139892, 0.016495222998653643, ...	ultimamente quase tudo ifood caro cupons falso...
1	vou dar uma estrela, e é muito ainda!!! já é a...	1	-1	[vou, dar, estrela, ainda, segunda, vez, faço...	[-0.02211865014396608, -0.0684307242743671, -0...	vou dar estrela ainda segunda vez faço cancela...
2	Disigne horrível, navegação também e promoções...	1	-1	[disigne, horrível, navegação, promoções, pra...	[-0.028950500444625504, -0.07825767648464535, ...	disigne horrível navegação promoções pra algum...
3	O aplicativo sempre funcionou bem, mas nós últ...	1	-1	[aplicativo, sempre, funcionou, bem, últimos, ...	[0.017615938858528222, -0.014666019741692866, ...	aplicativo sempre funcionou bem últimos meses ...
4	Atendimento a problemas do	1	1	[atendimento, problemas,	[-0.027385065230824377,	atendimento problemas


```

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

```

```

df_train, df_test = train_test_split(
    df_model,
    test_size = 0.3,
    random_state = 42
)

```

```

vect = CountVectorizer()
vect.fit(df_train.norm_content)
x_train = vect.transform(df_train.norm_content)
x_test = vect.transform(df_test.norm_content)
y_train = df_train[TARGET_COLUMN]
y_test = df_test[TARGET_COLUMN]

```

```

# treinamento do modelo árvore de decisão com o dataframe de treino
model = RandomForestClassifier(random_state=42)
model.fit(x_train, y_train)

```

```

# escoragem da classificação na amostra de teste
y_pred = model.predict(x_test)

```

Avaliação do modelo

```

from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix

```

```

print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))

```

```

precision    recall  f1-score   support

     1       0.54      0.65      0.59         558
     2       0.47      0.37      0.41         535
     3       0.39      0.40      0.40         526
     4       0.45      0.36      0.40         559
     5       0.57      0.66      0.61         522

```

```

accuracy                0.49        2700
macro avg              0.48        0.49        0.48        2700
weighted avg           0.48        0.49        0.48        2700

```

```
0.4892592592592593
```

```
x_train.shape
```

```
(6300, 10272)
```

Classificador com Transformers

Referências/documentações:

- [Paper "Attention Is All You Need"](#)
- [Documentação Hugging Face](#)
- [Modulo sentence-transformers e modelo de linguagem da Hugging Face](#)

Sentence transformers

O Sentence-Transformers é uma biblioteca construída sobre o framework Transformers, e foi projetada para a geração de representações semânticas de sentenças (documentos ou pedaços de texto) de alta qualidade.

Vamos utilizar o modelo 'distiluse-base-multilingual-cased-v2', que é um modelo da família DistilBERT (uma versão mais leve e eficiente do BERT) que foi pré-treinado especificamente para a geração de embeddings de sentenças multilíngues.

```
!pip install sentence-transformers==3.2.1 transformers==4.46.3 --quiet
```

```

import sentence_transformers
print(sentence_transformers.__version__)

```

```
3.2.1
```

```

from sentence_transformers import SentenceTransformer

```

```

st = SentenceTransformer('sentence-transformers/distiluse-base-multilingual-cased-v2')
review_embeddings = st.encode(df_model['content'])

```

```
len(review_embeddings[0])
```

```
x = review_embeddings
y = df_model[TARGET_COLUMN].to_list()
```

```
from sklearn.model_selection import train_test_split
# divisão da amostra entre treino e teste
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=y, random_state = 42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
# treinamento do modelo árvore de decisão com o dataframe de treino
log_reg = RandomForestClassifier(random_state=42)
log_reg.fit(x_train, y_train)
```

```
# escoragem da classificação na amostra de teste
y_pred = log_reg.predict(x_test)
```

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix
```

```
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```

┌───┬────────┬────────┬────────┬────────┬────────┐
│   │ precision │ recall │ f1-score │ support │
├───┼────────┼────────┼────────┼────────┼────────┤
│ 1 │ 0.50      │ 0.64   │ 0.56    │ 540     │
│ 2 │ 0.43      │ 0.41   │ 0.42    │ 540     │
│ 3 │ 0.42      │ 0.37   │ 0.39    │ 540     │
│ 4 │ 0.46      │ 0.40   │ 0.43    │ 540     │
│ 5 │ 0.68      │ 0.68   │ 0.68    │ 540     │
├───┼────────┼────────┼────────┼────────┼────────┤
│ accuracy │          │          │ 0.50    │ 2700    │
│ macro avg │ 0.50     │ 0.50    │ 0.50    │ 2700    │
│ weighted avg │ 0.50     │ 0.50    │ 0.50    │ 2700    │
└───┴────────┴────────┴────────┴────────┴────────┘

0.5007407407407407
```

Exercício

Você deverá treinar o modelo de classificação do dataset de produtos [1] utilizando as duas abordagens apresentadas aqui (embeddings word2vec (CBOW ou Skip-gram) e sentence transformer e comparar os resultados.

- Remover registros com valores nulos;
- Contatenar as colunas de nome e descrição;
- Aplicar normalização da demo;
- Amostra de 30% para teste e random_state = 42.

[1] <https://dados-ml-pln.s3-sa-east-1.amazonaws.com/produtos.csv>

```
# resposta
```

```
# Dica, transformar em lista
...
```

```
vetor_embedding = st.encode(df['texto'].to_list())
```