Análise de sentimento de frases

aplicação para avaliação real time para plataforma de atendimento

Alunos: Heinrych Matheus e Luan Weba Soares

1. Descrição

Este projeto visa desenvolver um sistema de análise de sentimentos para avaliar mensagens de atendimento ao cliente em uma empresa de atendimento chamada POLI. O sistema será treinado em um conjunto de dados de avaliações de atendimento ao cliente e será capaz de atribuir um sentimento a cada mensagem recebida. Este processo será realizado a cada 10 minutos para as últimas 10 mensagens recebidas. Os resultados da análise de sentimento serão salvos na coluna "status" de uma tabela chamada "contacs".

2. Passos envolvidos

2.1. Preparação dos dados

Os dados foram adquiridos de um conjunto de dados do Kaggle intitulado "Brazilian Portuguese Sentiment Analysis Datasets". Este conjunto de dados contém uma ampla variedade de avaliações provenientes de cinco lojas de comércio eletrônico diferentes (Olist, Buscapé, B2W, UTLC-Apps e UTLC-Apps). Cada avaliação foi categorizada com um valor de classificação, sendo 1 para indicar sentimentos positivos e 0 para indicar sentimentos negativos.

Antes de serem fornecidos ao modelo, os dados passaram por uma etapa de pré-processamento que envolveu a remoção de palavras que poderiam interferir no processo de classificação. Os detalhes deste pré-processamento foram registrados na tabela a seguir:

Pré Processamento
Remoção de #, @, URL
Remoção de stop words
Conversão de palavras para caixa baixa
Tokenização
Marcação de fala POS
Stemming
Normalização das palavras

Figura 1: Tabela de Pré-Processamento

2.2. Construção do modelo

O modelo usado para treinamento foi o BERT, o modelo BERT significa "Bidirectional Encoder Representations from Transformers" (Representações de Codificador Bidirecional a partir de Transformadores), é um modelo de linguagem pré-treino desenvolvido pela Google. Ele é baseado na arquitetura Transformer e é projetado para entender o contexto das palavras em uma sentença de maneira bidirecional, ou seja, considerando as palavras à esquerda e à direita de uma palavra em particular. Para fazer tal feito ele possui algumas fases.

Antes de falarmos do modelo BERT falaremos da arquitetura que é usado, ele usa a arquitetura de transformer, nessa arquitetura existem 2 processos principais, o encoder e decoder

O encoder é responsável por processar a sequência de entrada e criar representações contextuais para cada elemento da sequência

O decoder gera sequências de saída com base nas representações contextuais geradas pelo encoder, muito utilizado em tarefas de geração de sequências, permitindo que o modelo crie traduções, legendas, resumos e outras formas de conteúdo textual.

Um exemplo no nosso caso, enquanto o encoder entende se a review do produto é boa ou ruim, o decoder gera uma mensagem de resposta dizendo “obrigado por ter gostado do produto X devido às funções Y e Z, realmente elas são muito boas devido às características k

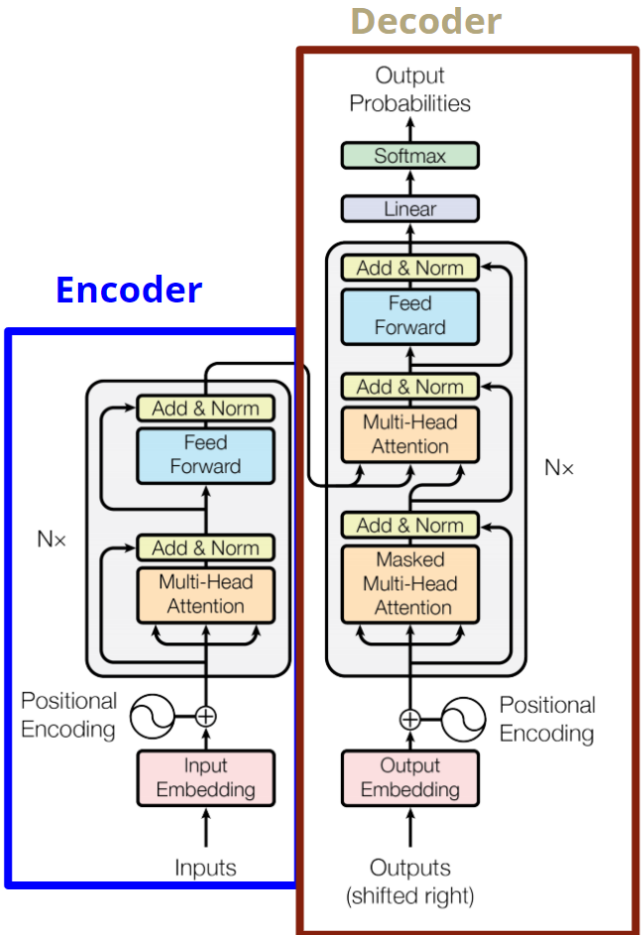


Figura 2: Arquitetura do transformer

Com a arquitetura explicada, podemos explicar o BERT propriamente dito, primeiro o modelo BERT é pré-treinado em grandes quantidades de texto, no nosso caso pegamos esse pré treinamento de um site chamado hugging face com um banco onde tinha várias palavras em portugues. Durante o pré-treinamento, o modelo aprende a prever as palavras seguintes em uma sentença. Mas em vez de prever as palavras sequencialmente, BERT considera todas as palavras em uma sentença de forma simultânea (daí o termo "bidirecional"). Isso permite que BERT capture melhor as nuances de contexto e relacionamentos entre as palavras.

Outra forma para incluir o contexto nas palavras, o BERT usa uma técnica usada durante o pré-treinamento do BERT é a máscara de palavra" (word masking) e Predição da próxima sentença. No mascaramento, algumas palavras nas sentenças de entrada são aleatoriamente mascaradas, e o modelo é treinado para prever essas palavras mascaradas com base no contexto das palavras ao redor. Já na predição da próxima sentença, o modelo compara duas sentenças e responde se uma sentença vem depois da outra como verdadeiro ou falso.

Depois do pré-treinamento, o BERT pode ser afinado (fine-tuned) para tarefas específicas de processamento de linguagem, como classificação de texto, reconhecimento de entidades nomeadas, análise de sentimentos, entre outras. Para fazer isso, o BERT é alimentado com um conjunto de dados de treinamento específico para a tarefa desejada e ajustado para aprender a realizar a tarefa, no nosso caso, o conjunto de dados foi um banco com 50000 reviews classificadas como positiva e negativa.

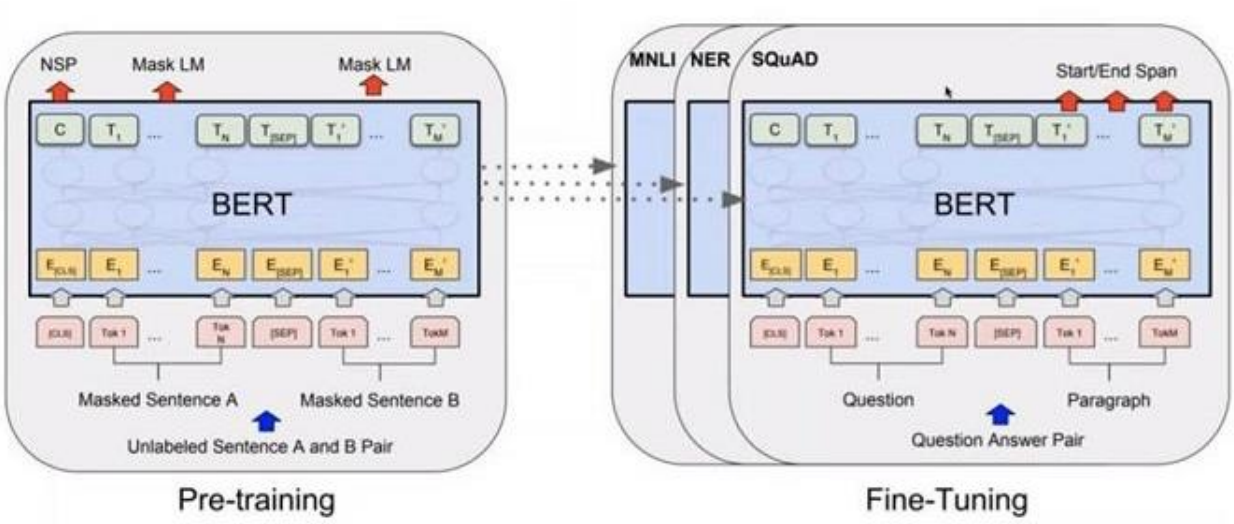


Figura 3: Arquitetura do modelo BERT

2.3. Treinamento do modelo

Com o intuito de replicar o procedimento de treinamento delineado no artigo do BERT, adotaremos a implementação do otimizador Adam (AdamW) disponibilizada pelo Hugging Face, juntamente com um agendador linear, dispensando, contudo, a fase de aquecimento (warmup).

Para configurar os hiperparâmetros do modelo, seguiremos as sugestões dos autores do BERT: Tamanho do lote (batch size): 16, 32 e Taxa de aprendizado (Adam): 5e-5, 3e-5, 2e-5. Abaixo conseguimos ver a acurácia do modelo

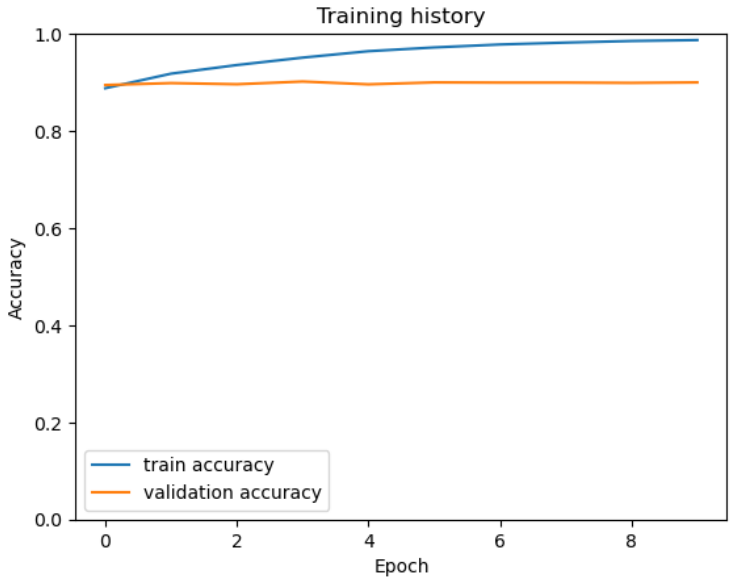


Figura 4: Acurácia do treino e validação

Após as 10 épocas, a precisão durante o treinamento começa a se aproximar de 100%.

2.4. Ajuste fino do modelo

Para avaliar outras métricas, foi utilizado o método da matriz de confusão.

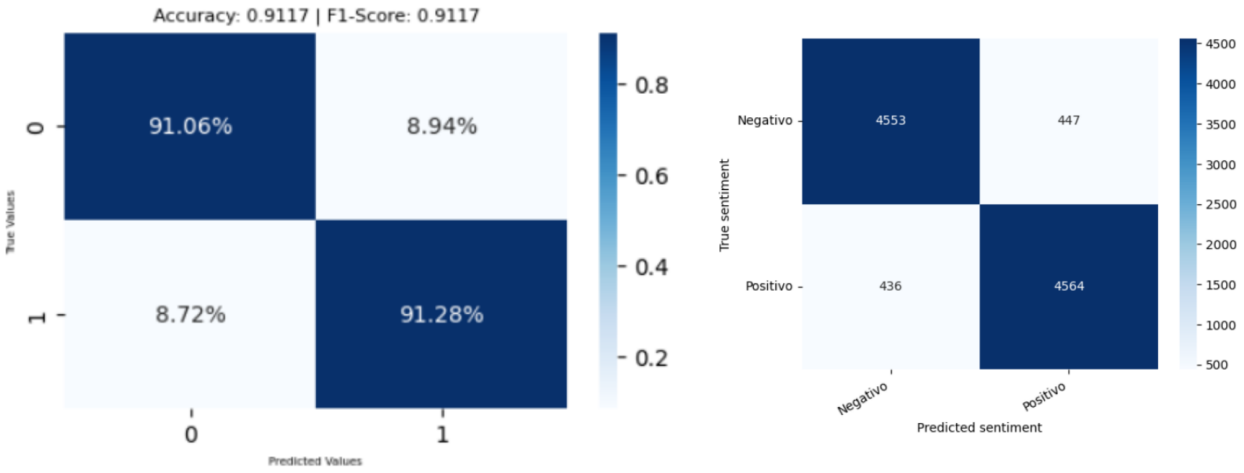
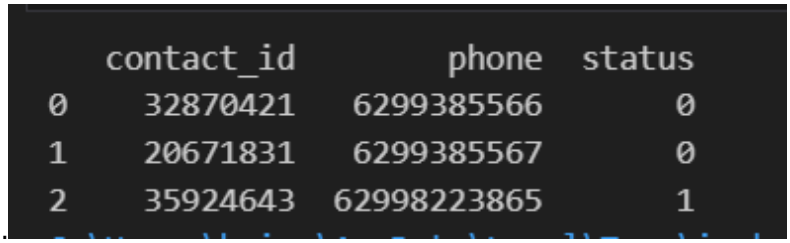


Figura 5: Matriz de confusão

A análise das previsões feitas nos dados de teste revela um equilíbrio de acurácia entre os sentimentos positivo e negativo.

2.5. Implantação do modelo

Foi realizado um processo automatizado para análise de sentimentos em mensagens de texto e atualização do status de contatos foi implementado utilizando um banco de dados PostgreSQL. Primeiramente, os dados das tabelas "messages" e "contacts" são recuperados do banco e carregados em DataFrames do Pandas. Em seguida, para cada contato, as 10 mensagens mais recentes são selecionadas e passadas por um modelo de análise de sentimentos, que realiza previsões sobre o conteúdo emocional das mensagens. Se ao menos uma das mensagens é considerada negativa, o status do respectivo contato é atualizado para "0" na tabela "contacts", indicando um possível sentimento negativo. Caso contrário, o status é atualizado para "1". Essa abordagem automatizada possibilita a identificação rápida de contatos com possíveis sentimentos negativos com base nas mensagens que enviaram, facilitando a priorização e ação adequada em relação a esses contatos. Abaixo segue a atualização de alguns exemplos de contatos da tabela contacts



	contact_id	phone	status
0	32870421	6299385566	0
1	20671831	6299385567	0
2	35924643	62998223865	1

Figura 6: Status dos contatos

2.5.1. Teste para contato com mudança de status negativo

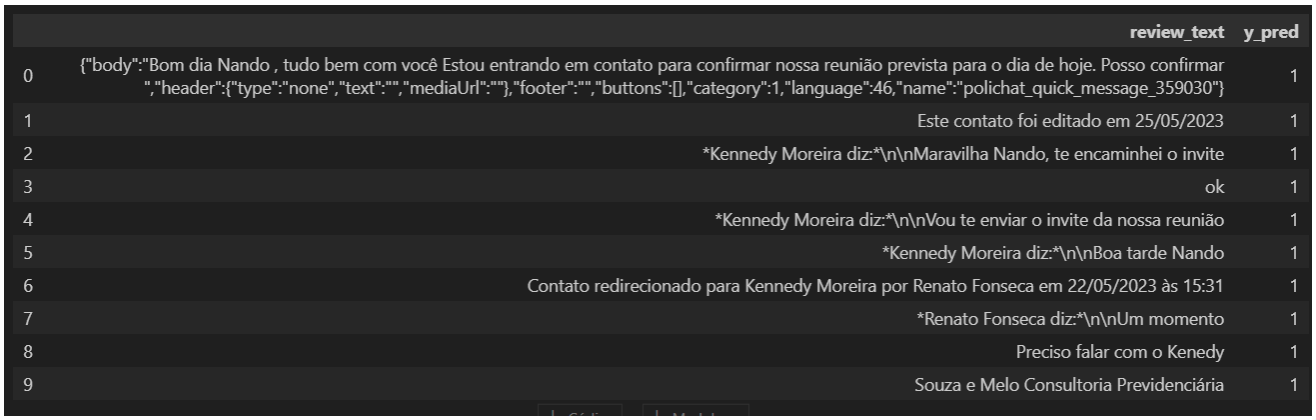
Para esse teste o contato considerado foi de id correspondente a 20671831.

	review_text	y_pred
0	*Juscelino diz:*\n\nGrato pelo resposta, qualquer situação, pode nos acionar. Tenha um excelente dia!!!	0
2	Hoje pela manhã está funcionando perfeitamente, obg	1
4	*Juscelino diz:*\n\nBom dia, Leandro. Tudo bem Tivemos um retorno positivo do nosso time técnico em relação a estabilidade da plataforma para hoje. Pode me contar como está a usabilidade hoje para você, por gentileza	1
6	*Luciana Rodrigues diz:*\n\nOlá Leandro, boa tarde! Me chamo Luciana faço parte do Time de Sucesso do Cliente na Poli! Primeiramente gostaria de pedir desculpas em nome da nossa empresa e reforçar com você que já identificamos a causa da instabilidade que afetou o dia de hoje e o nosso time técnico já está operando para retomar a normalização dos serviços. Lamentamos muito o ocorrido e reforçamos que estamos totalmente dedicados em solucionar por completo essa questão. Em breve estaremos operando dentro da normalidade. Qualquer dúvida, estamos aqui para poder te auxiliar!	0
8	NaN	0

Figura 7: Exemplo de classificação de mensagens

2.5.2. Teste para contato sem mudança de status

Para esse teste o contato considerado foi de id correspondente a 35924643.



	review_text	y_pred
0	{"body":"Bom dia Nando , tudo bem com você Estou entrando em contato para confirmar nossa reunião prevista para o dia de hoje. Posso confirmar ","header":{"type":"none","text":"","mediaUrl":""},"footer":"","buttons":[],"category":1,"language":46,"name":"polichat_quick_message_359030"}	1
1	Este contato foi editado em 25/05/2023	1
2	*Kennedy Moreira diz:*\n\nMaravilha Nando, te encaminhei o invite	1
3	ok	1
4	*Kennedy Moreira diz:*\n\nVou te enviar o invite da nossa reunião	1
5	*Kennedy Moreira diz:*\n\nBoa tarde Nando	1
6	Contato redirecionado para Kennedy Moreira por Renato Fonseca em 22/05/2023 às 15:31	1
7	*Renato Fonseca diz:*\n\nUm momento	1
8	Preciso falar com o Kenedy	1
9	Souza e Melo Consultoria Previdenciária	1

Figura 8: Exemplo de classificação de mensagens

3. Códigos

3.1. Pré-processamento

```
import nltk
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
from nltk.corpus import stopwords
from unidecode import unidecode
import unicodedata
import unidecode
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')  # Baixar o recurso do WordNet
nltk.download('rslp')
nltk.download('sentiwordnet')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
from unidecode import unidecode
import os
os.environ["NUMBA_CUDA_DRIVER"] = "1"


class preprocess:

    def fix_encoding(text):
        vogais = ["a", "e", "i", "o", "u"]
        cedilha = "c"

        text        =        text.replace("√°",        vogais[0]).replace("√†",
vogais[0]).replace("√¢",    vogais[0]).replace("√Å",    vogais[0]).replace("√É",
vogais[0]).replace("√£", vogais[0])
        text        =        text.replace("√©",        vogais[1]).replace("√™",
vogais[1]).replace("√â", vogais[1])
        text = text.replace("√ß", cedilha).replace("√á", cedilha)
```

```
        text = text.replace("√≠", vogais[2]).replace("√ç", vogais[2])
        text = text.replace("√≥", vogais[3]).replace("√µ", vogais[3]).replace("√¥", vogais[3]).replace("√ì", vogais[3])
        text = text.replace("√∫", vogais[4]).replace("√ö", vogais[4])

        text = text.replace("(c)", cedilha)

        entrada_normalizada = unicodedata.normalize('NFKD', text).encode('ASCII', 'ignore').decode('ASCII')

        return entrada_normalizada

    def expand_abbreviations(text):
        abbreviation_map = {
            'vc': 'voce',
            'tb': 'tambem',
            'pq': 'porque',
            'tbm': 'tambem',
            'q': 'que',
            'n': 'nao',
            'td': 'tudo',
            'pnc': 'pau no cu',
            'ta': 'esta',
            'blz': 'beleza',
            'vlw': 'valeu',
            'bjs': 'beijos',
            'flw': 'falou',
            'eh': 'e',
            'vdd': 'verdade',
            'dps': 'depois',
            'msm': 'mesmo',
            'qq': 'qualquer',
            'pf': 'por favor',
            't+' : 'ate mais',
            'sdd': 'saudade',
            'fzd': 'fazendo',
            'c': 'com',
            'd': 'de',
            'p': 'para',
            'vc': 'voce',
```

```
            'vc': 'voces',
            'tbm': 'tambem',
            'mt': 'muito',
            'pq': 'por que',
            'hj': 'hoje',
            'q': 'que',
            'cmg': 'comigo',
            'fds': 'fim de semana',
            'qnd': 'quando',
            'pqp':"puta que pariu",
            'n':'nao',
            'vsf':'vai se fuder',
            'vsfd':'vai se fuder',
            'sifoda':'se foda',
            'gnt':'gente',
            'vou t matar':'vou te matar',
            'gstz':'gostosa',
            'tdas':'todas',
            'tou':'estou',
            'to':'estou',
            'ta':'esta',
            'vox':'voce',
            'putaa':'puta',
            'muie':'mulher',
            'fdp':'filha da puta', #v8
            'calaboca':'cala boca' #v8
        }

        words = text.split()
        expanded_words = []

        for word in words:
            if word.lower() in abbreviation_map:
                expanded_words.append(abbreviation_map[word.lower()])
            else:
                expanded_words.append(word)

        return ' '.join(expanded_words)

    def preprocess_text(df):
        # Define a lista de stopwords em português
```

```
        stopwords_portuguese = stopwords.words('portuguese')
        stopwords_portuguese += ['http', 'https', 'www', 'com', 'br', 'rt']

        lemmatizer = WordNetLemmatizer()

        # Aplica as transformações em cada texto do dataframe
        df['texto_processado'] = df['review_text'].str.lower()  # Converte todo
o texto para minúsculo # [pandas]
        df['texto_processado']                                              =
df['texto_processado'].apply(preprocess.expand_abbreviations)    #    [pandas]
substituição de termos por replace
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
preprocess.fix_encoding(text)) # [pandas]
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
re.sub(r'(.)\1{2,}', r'\1\1', text))  # Remove caracteres sequenciais repetidos
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
word_tokenize(text))  # Tokeniza o texto
        df['texto_processado'] = df['texto_processado'].apply(lambda tokens:
[word   for   word   in   tokens   if   not   word.startswith('@')   and   not
word.startswith('#')])  # Remove tokens iniciados com @ e #
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
[unidecode(word) for word in text])  # Remove acentos
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
[re.sub(r'[^\w\s]',  '',  token)  for  token  in  text])    #  Remove  caracteres
especiais
        df['texto_processado'] = df['texto_processado'].apply(lambda tokens:
[word for word in tokens if word.isalpha()])
        df['texto_processado']  =  df['texto_processado'].apply(lambda  text:
[lemmatizer.lemmatize(word) for word in text])  # Realiza lematização
        df['frase'] = df['texto_processado'].apply(lambda text: ' '.join(text))
# Reconstroi o texto

        df = df.drop_duplicates(subset=['frase'])  # Remove duplicatas
        df = df.dropna(subset=['frase'])  # Remove linhas com valores nulos
        df = df.dropna(subset=['polarity'])

        return df


    def preprocess_text_stop(df):
        # Define a list of stopwords in Portuguese
```

```
        stopwords_portuguese = stopwords.words('portuguese')
        stopwords_portuguese += ['http', 'https', 'www', 'com', 'br', 'rt']

        lemmatizer = WordNetLemmatizer()

        # Make a copy of the dataframe to avoid SettingWithCopyWarning
        df_copy = df.copy()

        # Apply transformations to each text in the dataframe
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['review_text'].str.lower()  # Convert all text to lowercase
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(preprocess.expand_abbreviations)
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(preprocess.fix_encoding)
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda  text:  re.sub(r'(.)\1{2,}',  r'\1\1',
text))  # Remove sequential repeated characters

        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda  text:  [unidecode(word)  for  word  in
word_tokenize(text)])  # Remove accents
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda   tokens:   [re.sub(r'[^\w\s]',   '',
token) for token in tokens])  # Remove special characters
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda tokens: [word for word in tokens if
word.isalpha()])
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda  tokens:  [lemmatizer.lemmatize(word)
for word in tokens])  # Perform lemmatization
        df_copy.loc[:,                  'texto_processado']                 =
df_copy['texto_processado'].apply(lambda tokens: [word for word in tokens if
word  not  in  stopwords_portuguese  and  not  word.startswith('@')  and  not
word.startswith('#')])  # Remove tokens that are not words or that are stopwords
        df_copy.loc[:,  'frase']  =  df_copy['texto_processado'].apply(lambda
tokens: ' '.join(tokens))  # Reconstruct the text

        df_copy   =   df_copy.drop_duplicates(subset=['frase'])      #   Remove
duplicates
```

```
        df_copy = df_copy.dropna(subset=['frase'])   # Remove lines with null
values

        return df_copy
```

3.2. Processando o texto

```
#!/usr/bin/env python
# coding: utf-8

# ## pre processar o lexico inicial

# In[1]:


import torch


# In[2]:


print(torch.cuda.is_available())


# In[1]:


import sys
sys.path.insert(0, '../Codigos')
import codigos
from nltk import ngrams
import importlib
importlib.reload(codigos)

import pandas as pd


# In[7]:
```

```
from sklearn.utils import resample
import pandas as pd

new_data          =                   pd.read_csv(r"D:\Users\heinr\Desktop\TCC-
Adriano\lexicos\base_processada_sem_stop.csv")
new_data = new_data.dropna(subset=['polarity','frase'])
new_data = new_data.drop_duplicates(subset=['frase'])
df_majority = new_data[new_data["polarity"] == 1]
df_minority = new_data[new_data["polarity"] == 0]
df_majority_downsampled = resample(df_majority,
                                   replace=True,
                                   n_samples=len(df_minority),
                                   random_state=42)
df_balanced_manual = pd.concat([df_majority_downsampled, df_minority])
df_balanced_manual.groupby(['polarity']).size()


# In[11]:


df_balanced_manual.to_csv("base_balanceada.csv",index=False)


# ## preprocess para o lexico inicial

# In[2]:


import zipfile
import pandas as pd
#           zip_file_path           =           r'D:\Users\heinr\Desktop\TCC-
Adriano\corpus\concatenated.csv.zip'
# # Descompactar o arquivo ZIP
# with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
#     zip_ref.extractall()


# In[3]:


# Especificar o caminho do arquivo CSV descompactado
```

```
csv_file_path = r'D:\Users\heinr\Desktop\TCC-Adriano\lexicos\concatenated.csv'

# Ler o CSV usando pandas
df = pd.read_csv(csv_file_path)


# In[4]:


df.columns


# In[5]:


dataset = df.dropna(subset=['review_text', 'polarity'])
aux_processado = codigos.preprocess.preprocess_text_stop(dataset)


# In[6]:


aux_processado.to_csv("base_processada_sem_stop.csv", index=False)


# In[7]:


# aux_processado.to_csv("base_processada_com_stop.csv", index=False)


# In[12]:


aux_processado.columns


# In[13]:


aux_final = aux_processado[['frase','polarity']]
```

```
# In[15]:


aux_final.to_csv("base_processada_sem_stop.csv", index=False)


# # processamento da validação

# In[6]:


import torch
print(torch.cuda.is_available())


# In[2]:


# Especificar o caminho do arquivo CSV descompactado
csv_file_path = r'D:\Users\heinr\Desktop\TCC-Adriano\lexicos\df_valid2.csv'

# Ler o CSV usando pandas
df_val = pd.read_csv(csv_file_path)


# In[4]:


dataset = df_val.dropna(subset=['review_text', 'polarity'])
aux_processado = codigos.preprocess.preprocess_text_stop(dataset)


# In[ ]:


aux_processado[['frase','polarity']].to_csv("base_validacao2.csv",
index=False)
```

```
# In[15]:


dataset = df_val.dropna(subset=['body', 'target'])
dataset = dataset.rename(columns={'body':'review_text', 'target':'polarity'})
aux_processado = codigos.preprocess.preprocess_text_stop(dataset)


# In[16]:


aux_processado[['frase','polarity']].to_csv("base_validacao.csv", index=False)
```

Definição e treinamento do modelo

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split #

import nltk
from nltk.corpus import stopwords
from nltk.classify import SklearnClassifier

from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import torch.nn as nn


from subprocess import check_output
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
import seaborn as sns
```

```
# In[2]:


def printa_matriz(y_ori, y_pred):

    cm_train = confusion_matrix(y_ori,y_pred, normalize='true')
    accuracy = accuracy_score(y_ori,y_pred)
    f1 = f1_score(y_ori,y_pred, average='weighted')


    plt.figure(figsize=(5, 3))
    plt.title(f'Accuracy: {round(accuracy, 4)} | F1-Score: {round(f1, 4)}',
size=8)
    sns.heatmap(cm_train, annot=True, fmt='.2%', cmap='Blues')
    plt.xlabel('Predicted Values', size=5)
    plt.ylabel('True Values', size=5)
    plt.show()


# In[3]:


def show_confusion_matrix(confusion_matrix):
  hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
  hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(),           rotation=0,
ha='right')
  hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(),          rotation=30,
ha='right')
  plt.ylabel('True sentiment')
  plt.xlabel('Predicted sentiment');


# ## puxando a base

# In[4]:


data = pd.read_csv(r'../corpus_aplicativo\review_app_sem_stop.csv')
data = data.dropna(subset=['polarity','frase'])
```

```
data = data[['frase','polarity']]


# ## sample dos dados

# In[5]:


num_samples_per_class = 50000  # reshape dos dados, grande dms

df_zero = data[data.polarity == 0.0].sample(num_samples_per_class)
df_one = data[data.polarity == 1.0].sample(num_samples_per_class)

data = pd.concat([df_zero, df_one])

print(data.groupby(['polarity']).size())


# # BERT

# In[6]:


#!pip install -qq transformers
import transformers
from     transformers     import     BertModel,     BertTokenizer,     AdamW,
get_linear_schedule_with_warmup


# In[7]:


PRE_TRAINED_MODEL_NAME = 'neuralmind/bert-base-portuguese-cased'

tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)


# In[8]:


token_lens = []
```

```
for txt in data.frase:
  tokens = tokenizer.encode_plus(txt,
                                 max_length=512,
                                 truncation=True,
                                 padding='max_length')
  token_lens.append(len(tokens["input_ids"]))


# In[9]:


import torch
import numpy as np
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)


# In[10]:


class GPReviewDataset(Dataset):

  def __init__(self, reviews, targets, tokenizer, max_len):
    self.reviews = reviews
    self.targets = targets
    self.tokenizer = tokenizer
    self.max_len = max_len

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

  def __getitem__(self, item):
    review = str(self.reviews[item])
    target = self.targets[item]

    encoding = self.tokenizer.encode_plus(
      review,
```

```
      add_special_tokens=True,
      max_length=self.max_len,
      truncation=True, # Adicione esta linha se necessário
      return_token_type_ids=False,
      padding='max_length',
      return_attention_mask=True,
      return_tensors='pt',
    )

    return {
      'frase': review,
      'input_ids': encoding['input_ids'].flatten(),
      'attention_mask': encoding['attention_mask'].flatten(),
      'targets': torch.tensor(target, dtype=torch.long)
    }


# In[11]:


from sklearn.model_selection import train_test_split

# Primeiro, divida data em train e temporary (test + val) datasets
data_train, data_temp = train_test_split(data,
                                         test_size=0.2,
                                         stratify=data['polarity'],
                                         random_state=RANDOM_SEED)

# Em seguida, divida o dataset temporary em test e val datasets
data_val, data_test = train_test_split(data_temp,
                                       test_size=0.5,
                                       stratify=data_temp['polarity'],
                                       random_state=RANDOM_SEED)


# In[12]:


import multiprocessing

num_cores = multiprocessing.cpu_count()
```

```
num_workers = min(num_cores, 0)
print(f'Number of available CPU cores: {num_workers}')

def create_data_loader(data, tokenizer, max_len, batch_size):
  ds = GPReviewDataset(
    reviews=data.frase.to_numpy(),
    targets=data.polarity.to_numpy(),
    tokenizer=tokenizer,
    max_len=max_len
  )

  return DataLoader(
    ds,
    batch_size=batch_size,
    num_workers=num_workers
  )



# ## Gerando as bases para a relização do treinamento

# In[13]:


BATCH_SIZE = 16
MAX_LEN = 160
train_data_loader   =   create_data_loader(data_train,   tokenizer,   MAX_LEN,
BATCH_SIZE)
val_data_loader = create_data_loader(data_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader    =    create_data_loader(data_test,    tokenizer,    MAX_LEN,
BATCH_SIZE)

data_valid2 = pd.read_csv('../lexicos/base_validacao.csv')
data_valid2 = data_valid2.dropna(subset=['polarity','frase'])
valid_data_loader2   =   create_data_loader(data_valid2,   tokenizer,   MAX_LEN,
BATCH_SIZE)


data_valid = pd.read_csv('../lexicos/base_validacao2.csv')
data_valid = data_valid.dropna(subset=['polarity','frase'])
```

```
valid_data_loader = create_data_loader(data_valid, tokenizer, MAX_LEN,
BATCH_SIZE)


# In[14]:


len(train_data_loader)


# In[15]:


dataf = next(iter(train_data_loader))
dataf.keys()


# In[16]:


print(dataf['input_ids'].shape)
print(dataf['attention_mask'].shape)
print(dataf['targets'].shape)


# In[17]:


import torch.nn as nn
class SentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(SentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,
return_dict=False)
    self.drop = nn.Dropout(p=0.3)
    self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    _, pooled_output = self.bert(
      input_ids=input_ids,
```

```
      attention_mask=attention_mask
    )
    output = self.drop(pooled_output)
    return self.out(output)


# In[18]:


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)


# In[19]:


model = SentimentClassifier(2)
model = model.to(device)


# ## Definição do modelo

# In[20]:


EPOCHS = 10

optimizer = AdamW(model.parameters(), lr=3e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS

scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,
  num_training_steps=total_steps
)

loss_fn = nn.CrossEntropyLoss().to(device)


# ## definições e treinamento
```

```
# In[21]:


from tqdm.notebook import tqdm

def train_epoch(
  model,
  data_loader,
  loss_fn,
  optimizer,
  device,
  scheduler,
  n_examples
):
  model = model.train()

  losses = []
  correct_predictions = 0

  # Aqui estamos usando tqdm para envolver o data_loader e mostrar uma barra
de progresso
  for d in tqdm(data_loader, desc='Training'):
    input_ids = d["input_ids"].to(device)
    attention_mask = d["attention_mask"].to(device)
    targets = d["targets"].to(device)

    outputs = model(
      input_ids=input_ids,
      attention_mask=attention_mask
    )

    _, preds = torch.max(outputs, dim=1)
    loss = loss_fn(outputs, targets)

    correct_predictions += torch.sum(preds == targets)
    losses.append(loss.item())

    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    scheduler.step()
```

```
    optimizer.zero_grad()

  return correct_predictions.double() / n_examples, np.mean(losses)


# In[22]:


def eval_model(model, data_loader, loss_fn, device, n_examples):
  model = model.eval()

  losses = []
  correct_predictions = 0

  with torch.no_grad():
    for d in data_loader:
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["targets"].to(device)

      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      _, preds = torch.max(outputs, dim=1)

      loss = loss_fn(outputs, targets)

      correct_predictions += torch.sum(preds == targets)
      losses.append(loss.item())

  return correct_predictions.double() / n_examples, np.mean(losses)


# In[24]:


get_ipython().run_cell_magic('time', '', "from collections import
defaultdict\nhistory = defaultdict(list)\nbest_accuracy = 0\n\nfor epoch in
range(EPOCHS):\n\n  print(f'Epoch {epoch + 1}/{EPOCHS}')\n  print('-' * 10)\n\n
train_acc, train_loss = train_epoch(\n    model,\n    train_data_loader,    \n
```

```
loss_fn, \n    optimizer, \n    device, \n    scheduler, \n    len(data_train)\n
)\n\n   print(f'Train loss {train_loss} accuracy {train_acc}')\n\n   val_acc,
val_loss = eval_model(\n      model,\n      val_data_loader,\n      loss_fn, \n
device, \n     len(data_val)\n   )\n\n  print(f'Val    loss {val_loss} accuracy
{val_acc}')\n\n     valid_acc,  valid_loss  =  eval_model(\n          model,\n
valid_data_loader,\n    loss_fn, \n    device, \n    len(data_valid)\n   )\n\n
print(f'Valid     loss  {valid_acc}  accuracy  {valid_loss}')\n\n   valid_acc2,
valid_loss2 = eval_model(\n     model,\n     valid_data_loader2,\n     loss_fn,
\n   device, \n   len(data_valid2)\n  )\n\n print(f'Valid 2  loss {valid_acc2}
accuracy {valid_loss2}') \n   \n  test_acc, test_loss = eval_model( ## adicionei
para ver o comportamento\n    model, \n    test_data_loader, \n    loss_fn, \n
device, \n    len(data_test)\n  )\n  \n  print(f'Test  loss {test_loss} accuracy
{test_acc}')     ##     adicionei     para     ver     o     comportamento\n\n
history['train_acc'].append(train_acc)\n
history['train_loss'].append(train_loss)\n
history['val_acc'].append(val_acc)\n  history['val_loss'].append(val_loss)\n\n
if   val_acc   >   best_accuracy:\n            torch.save(model.state_dict(),
'best_model_state.bin')\n    best_accuracy = val_acc\n")


# ## Realização dos testes

# In[ ]:


train_acc_np = np.array([t.item() for t in history['train_acc']])
val_acc_np = np.array([t.item() for t in history['val_acc']])

plt.plot(train_acc_np, label='train accuracy')
plt.plot(val_acc_np, label='validation accuracy')

plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1])
plt.show()


# In[29]:
```

```
import torch.nn.functional as F


# In[26]:


class_names = ['Negativo', 'Positivo']
# Certifique-se de que o modelo está inicializado
model = SentimentClassifier(len(class_names))
model = model.to(device)

# Carregando o modelo
model.load_state_dict(torch.load(r'D:\Users\heinr\Desktop\TCC-
Adriano\classsificação_bert\best_model_state.bin'))


# In[27]:


def get_predictions(model, data_loader):
  model = model.eval()

  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []

  with torch.no_grad():
    for d in data_loader:

      texts = d["frase"]
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["targets"].to(device)

      outputs = model(
        input_ids=input_ids,
        attention_mask=attention_mask
      )
      _, preds = torch.max(outputs, dim=1)
```

```
      probs = F.softmax(outputs, dim=1)

      review_texts.extend(texts)
      predictions.extend(preds)
      prediction_probs.extend(probs)
      real_values.extend(targets)

  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs, real_values


# In[30]:


y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)


# In[31]:


from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names=class_names))


# In[32]:


from sklearn.metrics import confusion_matrix


# In[33]:


printa_matriz(y_test, y_pred)
```

```
# In[34]:


cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)


# ## Validação

# In[35]:


data_valid = pd.read_csv(r'../lexicos/base_validacao.csv')
data_valid = data_valid.dropna(subset=['polarity','frase'])
valid_data_loader   =   create_data_loader(data_valid,   tokenizer,   MAX_LEN,
BATCH_SIZE)


# In[45]:


y_review_texts_val, y_pred_val, y_pred_probs_val, y_val = get_predictions(
  model,
  valid_data_loader
)


# In[46]:


cm = confusion_matrix(y_val, y_pred_val)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)


# In[47]:


printa_matriz(y_val, y_pred_val)
```

```
# In[48]:


df_valid = pd.DataFrame({'frase': y_review_texts_val, 'y_val': y_val,
'y_pred_val': y_pred_val})


# In[49]:


df_valid.to_csv("validacao_classificada.csv",index=False)


# ## validacao 02

# In[41]:


BATCH_SIZE = 16


data_valid2 = pd.read_csv(r'../lexicos/base_validacao2.csv')
data_valid2 = data_valid2.dropna(subset=['polarity','frase'])
valid_data_loader2 = create_data_loader(data_valid2, tokenizer, MAX_LEN,
BATCH_SIZE)


# In[42]:


y_review_texts_val2, y_pred_val2, y_pred_probs_val2, y_val2 = get_predictions(
  model,
  valid_data_loader2
)


# In[43]:
```

```
cm2 = confusion_matrix(y_val2, y_pred_val2)
df_cm2 = pd.DataFrame(cm2, index=class_names, columns=class_names)
show_confusion_matrix(df_cm2)


# In[37]:


printa_matriz(y_val2, y_pred_val2)


# In[ ]:


df_valid2 = pd.DataFrame({'frase': y_review_texts_val2, 'y_val': y_val2,
'y_pred_val': y_pred_val2})


# In[ ]:


df_valid2.to_csv("validacao_classificada2.csv",index=False)
```

3.3. Implantação do modelo

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from transformers import BertModel, BertTokenizer, AdamW,
get_linear_schedule_with_warmup
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import torch.nn.functional as F

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


# In[21]:


import sys
sys.path.insert(0, '../Codigos')
import codigos
from nltk import ngrams
import importlib
importlib.reload(codigos)

import pandas as pd


# In[3]:


import torch
import numpy as np
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import torch.nn as nn

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)


# In[4]:


def show_confusion_matrix(confusion_matrix):
  hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
```

```
  hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(),           rotation=0,
ha='right')
  hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(),          rotation=30,
ha='right')
  plt.ylabel('True sentiment')
  plt.xlabel('Predicted sentiment');


# In[5]:


class GPReviewDataset(Dataset):
    def __init__(self, reviews, tokenizer, max_len, targets=None):
        self.reviews = reviews
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.targets = targets

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

    def __getitem__(self, item):
        review = str(self.reviews[item])
        encoding = self.tokenizer.encode_plus(
            review,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
        )
        target = self.targets[item] if self.targets is not None else 0

        return {
            'review_text': review,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'targets': torch.tensor(target, dtype=torch.long)
        }
```

```
# In[6]:


import multiprocessing

num_cores = multiprocessing.cpu_count()
num_workers = min(num_cores, 0)
print(f'Number of available CPU cores: {num_workers}')

def create_data_loader(data, tokenizer, max_len, batch_size):
  ds = GPReviewDataset(
    reviews=data.frase.to_numpy(),
    targets=data.polarity.to_numpy(),
    tokenizer=tokenizer,
    max_len=max_len
  )

  return DataLoader(
    ds,
    batch_size=batch_size,
    num_workers=num_workers
  )

def create_data_loader2(data, tokenizer, max_len, batch_size):
  ds = GPReviewDataset(
    reviews=data.frase.to_numpy(),
    tokenizer=tokenizer,
    max_len=max_len
  )

  return DataLoader(
    ds,
    batch_size=batch_size,
    num_workers=num_workers
  )



# In[7]:
```

```
BATCH_SIZE = 16
MAX_LEN = 160


data_valid2 = pd.read_csv('../lexicos/base_validacao.csv')
data_valid2 = data_valid2.dropna(subset=['polarity','frase'])
valid_data_loader2   =   create_data_loader(data_valid2,   tokenizer,   MAX_LEN,
BATCH_SIZE)


data_valid = pd.read_csv('../lexicos/base_validacao2.csv')
data_valid = data_valid.dropna(subset=['polarity','frase'])
valid_data_loader   =   create_data_loader(data_valid,   tokenizer,   MAX_LEN,
BATCH_SIZE)


# In[8]:


class SentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(SentimentClassifier, self).__init__()
    self.bert       =       BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,
return_dict=False)
    self.drop = nn.Dropout(p=0.3)
    #The last_hidden_state is a sequence of hidden states of the last layer of
the model
    self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    _, pooled_output = self.bert(
      input_ids=input_ids,
      attention_mask=attention_mask
    )
    output = self.drop(pooled_output)
    return self.out(output)


# In[46]:
```

```
import torch
import torch.nn.functional as F

def get_predictions(model, data_loader):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.eval()

    review_texts = []
    predictions = []
    prediction_probs = []
    real_values = []

    with torch.no_grad():
        for d in data_loader:
            texts = d["review_text"]  # Altere "frase" para "review_text"
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )
            _, preds = torch.max(outputs, dim=1)

            probs = F.softmax(outputs, dim=1)

            review_texts.extend(texts)
            predictions.extend(preds.tolist())
            prediction_probs.extend(probs.tolist())
            real_values.extend(targets.tolist())

    predictions = torch.tensor(predictions).cpu()
    prediction_probs = torch.tensor(prediction_probs).cpu()
    real_values = torch.tensor(real_values).cpu()
    return review_texts, predictions, prediction_probs, real_values


# In[10]:
```

```
class_names = ['Negativo', 'Positivo']


# In[11]:


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)


# In[12]:


# Certifique-se de que o modelo está inicializado
model = SentimentClassifier(len(class_names))
model = model.to(device)

# Carregando o modelo
model.load_state_dict(torch.load(r'D:\Users\heinr\Desktop\TCC-
Adriano\classsificação_bert\best_model_state.bin'))


# In[13]:


import torch.nn.functional as F


# In[14]:


y_review_texts_val, y_pred_val, y_pred_probs_val, y_val = get_predictions(
  model,
  valid_data_loader
)


# In[15]:
```

```
cm = confusion_matrix(y_val, y_pred_val)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)


# # Criando e povoando o banco

# In[16]:


import psycopg2
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)

cursor = conn.cursor()


# ### messages

# In[16]:


import psycopg2

# Conecte-se ao seu banco de dados PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)
cur = conn.cursor()

# Crie a tabela 'messages'
```

```
cur.execute("""
    CREATE TABLE IF NOT EXISTS messages (
        id SERIAL PRIMARY KEY,
        contact_id INT NOT NULL,
        body TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
""")

# Confirme as alterações
conn.commit()

# Feche o cursor e a conexão
cur.close()
conn.close()


# ### contacts

# In[17]:


import psycopg2

# Conecte-se ao seu banco de dados PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)
cur = conn.cursor()

# Crie a tabela 'messages'
cur.execute("""
    CREATE TABLE IF NOT EXISTS contacts (
        contact_id SERIAL PRIMARY KEY,
        phone VARCHAR(15),
        status INT DEFAULT 1
    );
```

```
""")

# Confirme as alterações
conn.commit()

# Feche o cursor e a conexão
cur.close()
conn.close()


# ### inserindo valores na contacts

# In[67]:


# Abra uma nova conexão e cursor
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)
cur = conn.cursor()

# Lista de contatos a inserir, onde cada tupla contém o contact_id e o phone
contacts_to_insert = [
    (35924643,'62998223865')
    #(20235323,'62998223366')
    # (32870421, '6299385566'),
    # (20671831, '6299385567')
]

# Insira múltiplos valores de uma só vez
cur.executemany("INSERT INTO contacts (contact_id, phone) VALUES (%s, %s);",
contacts_to_insert)

# Confirme as alterações
conn.commit()

# Feche o cursor e a conexão
```

```
cur.close()
conn.close()


# In[19]:


import pandas as pd
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)

# Lendo os dados da tabela 'messages' usando Pandas
query = "SELECT * FROM contacts" # Ajuste a query de acordo com suas necessidades
df = pd.read_sql(query, conn)

# Feche a conexão
conn.close()

# Agora 'df' contém os dados da tabela 'messages'
print(df)


# ### inserindo valores na messages

# In[35]:


# import pandas as pd
# import psycopg2

# # Supondo que o DataFrame já esteja carregado como df
#           df           =           pd.read_csv(r"D:\Users\heinr\Desktop\TCC-
Adriano\classsificação_bert\data_valid3.csv")
```

```
#        #        df        =        pd.read_csv(r"D:\Users\heinr\Desktop\TCC-
Adriano\classsificação_bert\data_valid1.csv")

# # Conecte-se ao seu banco de dados PostgreSQL
# conn = psycopg2.connect(
#     host="localhost",
#     port=5432,
#     user="postgres",
#     password="aluno",
#     database="projeto_inova"
# )

# # Crie um cursor para executar comandos SQL
# cur = conn.cursor()

# # Insira as linhas do DataFrame na tabela 'messages'
# for index, row in df.iterrows():
#     cur.execute("""
#         INSERT INTO messages (contact_id, body, created_at)
#         VALUES (%s, %s, %s);
#     """, (row['contact_id'], row['body'], row['created_at']))

# # Confirme as alterações
# conn.commit()

# # Feche o cursor e a conexão
# cur.close()
# conn.close()


# In[36]:


import pandas as pd
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
```

```
    password="aluno",
    database="projeto_inova"
)

# Lendo os dados da tabela 'messages' usando Pandas
query = "SELECT * FROM messages" # Ajuste a query de acordo com suas necessidades
df = pd.read_sql(query, conn)

# Feche a conexão
conn.close()

# Agora 'df' contém os dados da tabela 'messages'
print(df)


# ## Atualizando o banco

# #### limpando o status

# In[61]:


import pandas as pd
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)
cur = conn.cursor()

# Lendo os dados da tabela 'messages' usando Pandas
cur.execute("UPDATE contacts SET status = 1;")
conn.commit()
# Feche a conexão
conn.close()
```

```
# Agora 'df' contém os dados da tabela 'messages'


# In[62]:


import pandas as pd
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)

# Lendo os dados da tabela 'messages' usando Pandas
query = "SELECT * FROM contacts"
df = pd.read_sql(query, conn)

# Feche a conexão
conn.close()

# Agora 'df' contém os dados da tabela 'messages'
print(df)


# #### atualizando status

# In[94]:


import psycopg2

BATCH_SIZE = 16
MAX_LEN = 160

# Conecte-se ao banco de dados PostgreSQL
conn = psycopg2.connect(
```

```
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)
cur = conn.cursor()

# 1. Recupere todos os IDs únicos de contatos
cur.execute("SELECT DISTINCT contact_id FROM contacts;")
contact_ids = [row[0] for row in cur.fetchall()]

i = 0
# 2. Para cada ID de contato, recupere as 10 últimas mensagens
for contact_id in contact_ids:
    i += 1
    query = """
        SELECT body FROM messages
        WHERE contact_id = %s
        ORDER BY created_at DESC
        LIMIT 10;
    """
    cur.execute(query, (contact_id,))
    df = pd.read_sql(query, conn, params=(contact_id,))
    df.rename(columns={'body': 'review_text'}, inplace=True)
    df = codigos.preprocess.preprocess_text_stop(df)
    messages = [row[0] for row in cur.fetchall()]

    # Prepare o DataLoader com as 10 mensagens
    valid_data_loader = create_data_loader2(df, tokenizer, MAX_LEN, BATCH_SIZE)

    # 3. Avalie essas mensagens com seu modelo
    y_review_texts_val,     y_pred_val,     y_pred_probs_val,     y_val     =
get_predictions(model, valid_data_loader)

    df['y_pred'] = y_pred_val
    pd.set_option('display.max_colwidth', None)
    print(df[['frase','y_pred']])
    match i:
        case 1:
            a = df
```

```
        case 2:
            b = df
        case 3:
            c = df

    # 4. Se pelo menos uma previsão for zero, atualize o status do contato
    if any(pred == 0 for pred in y_pred_val):
        cur.execute("UPDATE contacts SET status = 0 WHERE contact_id = %s;",
(contact_id,))
    else:
        cur.execute("UPDATE contacts SET status = 1 WHERE contact_id = %s;",
(contact_id,))

# Confirme as alterações
conn.commit()

# Feche o cursor e a conexão
cur.close()
conn.close()


# In[87]:


b.columns


# In[96]:


c[['review_text','y_pred']]


# In[80]:


contact_ids


# In[69]:
```

```
y_pred_val


# In[91]:


import pandas as pd
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    user="postgres",
    password="aluno",
    database="projeto_inova"
)

# Lendo os dados da tabela 'messages' usando Pandas
query = "SELECT * FROM contacts" # Ajuste a query de acordo com suas necessidades
df = pd.read_sql(query, conn)

# Feche a conexão
conn.close()

# Agora 'df' contém os dados da tabela 'messages'
print(df)
```