**Análise de sentimento de frases**

**aplicação para avaliação real time para plataforma de atendimento**

**Alunos: Luan Weba Soares, Heinrych Matheus Gomes Schneider**

****

[**1. Descrição 3**](#_vlh1y1nxq4wa)

[**2. Passos envolvidos 3**](#_qrlq2e39k1r7)

[**2.1. Preparação dos dados 3**](#_pirqrvcw0i34)

[2.2. Construção do modelo 4](#_pz67vfuwi4wr)

[2.3. Treinamento do modelo 6](#_bynoqrjkn92)

[2.4. Ajuste fino do modelo 7](#_ss5a7p2d6wqv)

[2.5. Implantação do modelo 7](#_u7k40v4chvyj)

[2.5.1. Teste para contato com mudança de status negativo 8](#_1v4rypcc6egl)

[2.5.2. Teste para contato sem mudança de status 8](#_8etmzhs2mpsu)

[**3. Códigos 9**](#_tqouua5q55u9)

[3.1. Pré-processamento 9](#_hk0sb1ft1fle)

[3.2. Processando o texto 14](#_k6g8pb16xhvj)

[3.3. Implantação do modelo 35](#_d08q0y2v2eo7)

# Descrição

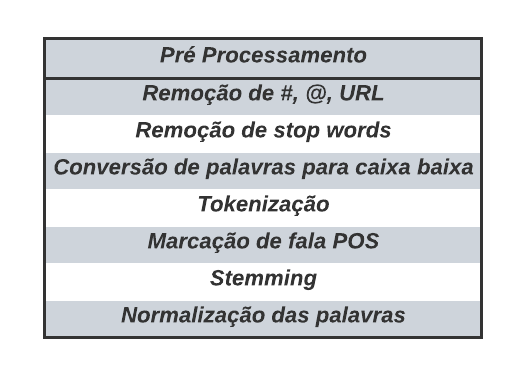
Este projeto visa desenvolver um sistema de análise de sentimentos para avaliar mensagens de atendimento ao cliente em uma empresa de atendimento chamada POLI. O sistema será treinado em um conjunto de dados de avaliações de atendimento ao cliente e será capaz de atribuir um sentimento a cada mensagem 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".

# Passos envolvidos

## 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:

  
Figura 1: Tabela de Pré-Processamento

## 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

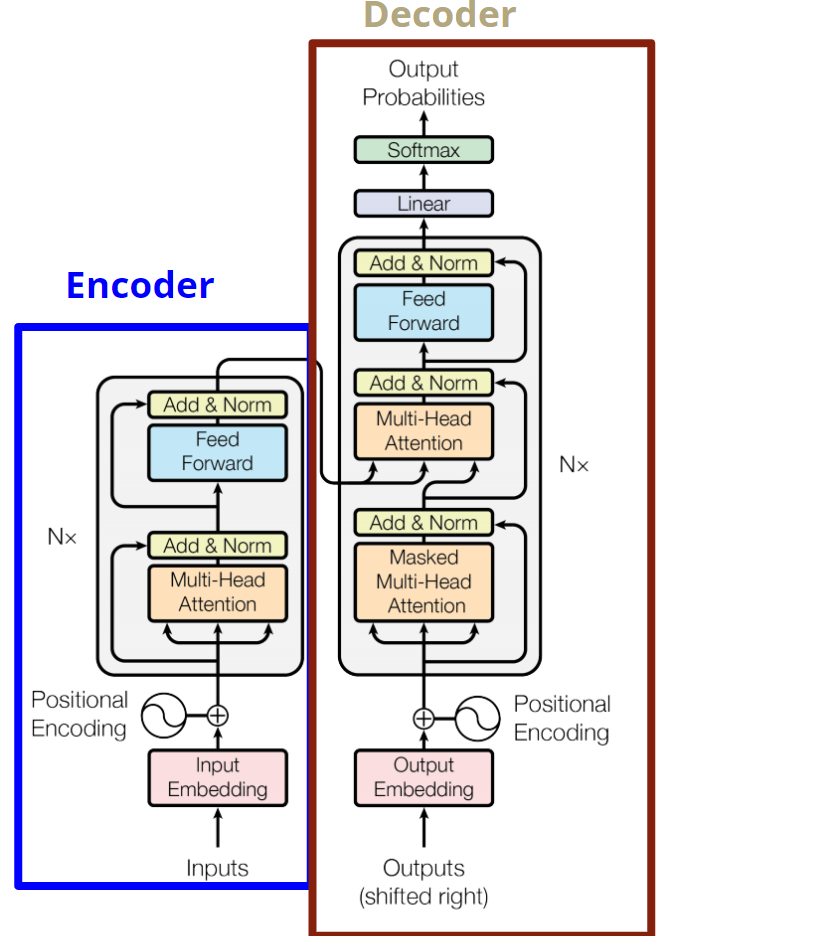


Figura 2: Arquitetura do transformer

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.

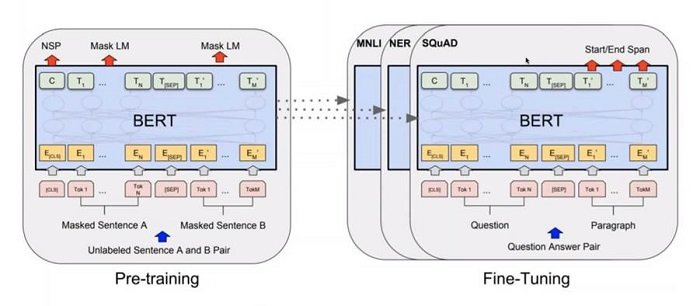


Figura 3: Arquitetura do modelo BERT

## 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

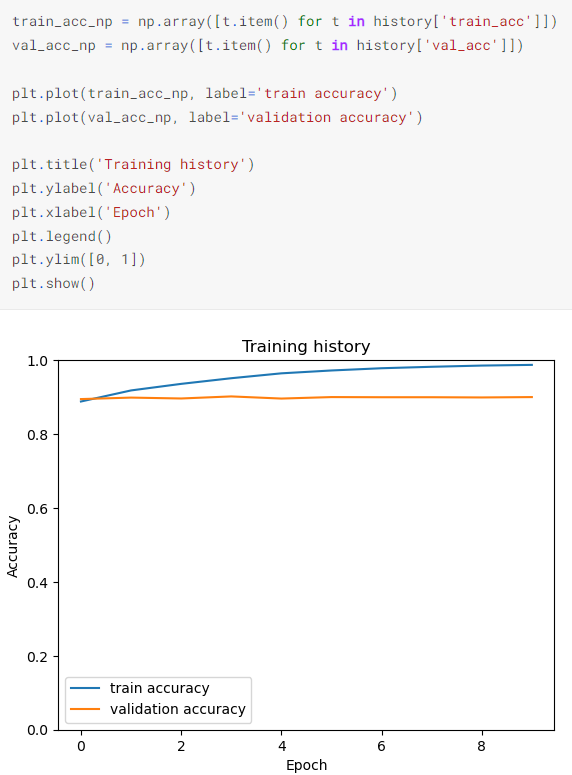


Figura 4: Acurácia do treino e validação

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

## Ajuste fino do modelo

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

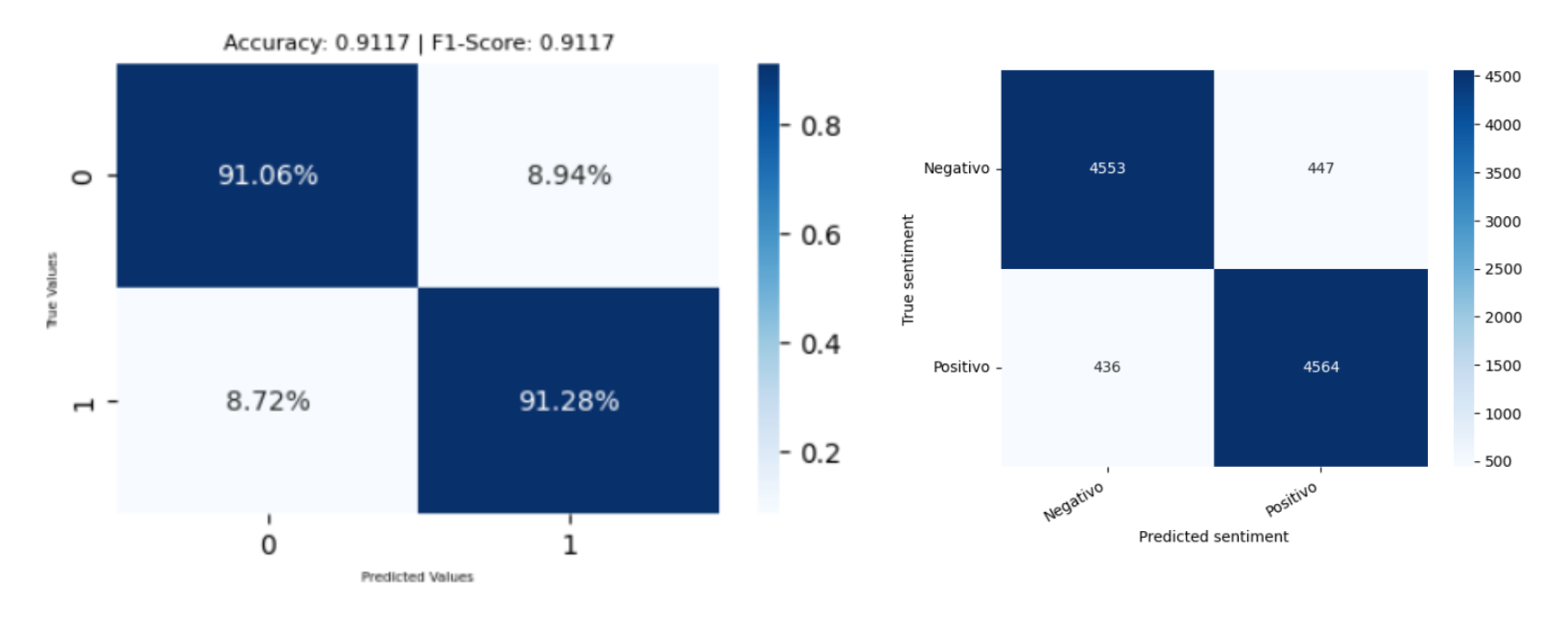


Figura 5: Matriz de confusão

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

## 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

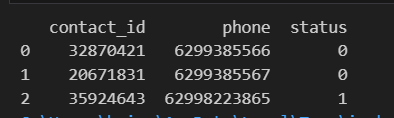
.

Figura 6: Status dos contatos

### Teste para contato com mudança de status negativo

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

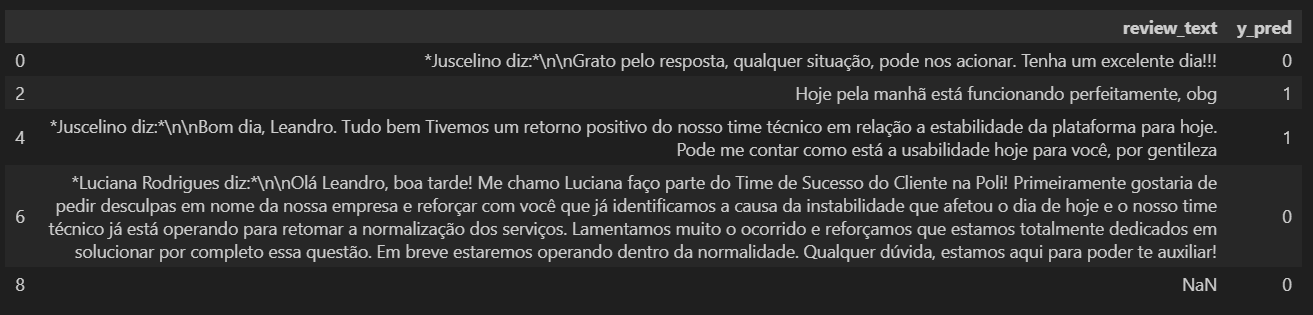


Figura 7: Exemplo de classificação de mensagens

### Teste para contato sem mudança de status

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

# 

Figura 8: Exemplo de classificação de mensagens

# 

# Códigos

## 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

## 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)

## 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)