PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Guylerme Velasco de Souza Figueiredo

Análise de Sentimento em Tweets utilizando NLP para Casos de COVID

Orientador: Leonardo Mendonza, PhD.

Julho/2021

Resumo

Num cenário de pandemia, muitas pessoas têm externalizado seus sentimentos e opiniões em relação ao COVID e a tudo que o cenário pandêmico nos impõe. Com base neste cenário, o presente trabalho propõe criar um mecanismo automatizado que seja capaz de capturar informações das redes sociais e utilizar tanto NLP quanto Machine Learning para produzir uma visão geral quanto ao sentimento das pessoas ao realizar uma publicação na plataforma de mídia social.

Palavras-chaves: NLP, ML, Análise de Sentimento

Abstract

In a pandemic scenario, many people have externalized their feelings and opinions regarding COVID and everything that the pandemic scenario imposes on us. Based on this scenario, this paper proposes to create an automated mechanism that is able to capture information from social networks and use both NLP and Machine Learning to create an overview of how people feel when they post on the social media platform.

Keywords: NLP, ML, Sentiment Analysis

Sumário

1. Introdução

Atualmente as redes sociais, como o Twitter, tomaram um lugar de grande importância na vida das pessoas. Hoje, os comentários e postagens nas plataformas de mídia social influencia, forma opiniões e muitas vezes até mudam o comportamento das pessoas. Isso, à medida que o aprendizado de máquina se torna cada vez mais popular e importante, assim como o processamento de linguagem natural (NLP), temos que lidar com isso, analisar e pesquisar as emoções nessas plataformas.

Há muitas maneiras de abordar um tópico, desde a análise "pura" baseada em dicionário até o aprendizado profundo "mais sério", redes neurais. Ao construir algoritmos de aprendizagem e classificadores, nos esforçamos para rotular os tweets relevantes com a polaridade emocional apropriada.

O objetivo principal deste trabalho é desenvolver um modelo de predição de emoções focando na relação entre palavras, rotulando entradas específicas, em oposição à decomposição usual de 'positiva' e 'negativa', obtemos uma escala muito mais ampla para previsões mais precisas. No entanto, no ponto de foco, não há um conjunto de dados maior, mas o modelo devidamente treinado analisa com um conjunto de dados recém-extraído que corresponde à tendência atual (temas de coronavírus agora) e número de compilação do conjunto de dados (número de tweets eliminados) que estreitando o círculo uma grande quantidade de dados em um tópico mais restrito. Desta forma, não indicamos apenas que os dados devem ser positivos ou negativos, também fornecemos uma análise mais detalhada dos níveis emocionais. Isso pode fornecer dados mais precisos do que a análise de conjuntos de dados maiores, já que a mineração recente está sempre disponível, para que você possa obter resultados muito mais rápidos e precisos.

Sabendo que plataformas como o Twitter produzem dados no contexto big data, ou seja, grandes volumes, em velocidade alta e em uma variedade considerável, é necessário utilizar uma plataforma tecnológica específica para tratar este problema. No nosso caso utilizamos a plataforma big data da Azure para provimento da solução.

2. Trabalhos relacionados

O método, que descrito por (Ortis, Farinella, Torrisi, & Battiato, 2018) usa texto extraído da descrição de diferentes imagens em vez de entradas clássicas do usuário. Em seguida, define um espaço de incorporação multimodal com base nas propriedades do texto. O exame emocional sendo realizado por uma Máquina de Vetores de Suporte supervisionada.

Este estudo explora as técnicas de (Leskovec, 2011) para modelar, analisar e otimizar as mídias sociais. Primeiro, eles nos mostram como coletar grandes quantidades de dados de mídia social. Em seguida, continuará a discutir métodos para obter e rastrear informações e como construir modelos de previsão para disseminação e inclusão de informações. Finalmente, eles discutem métodos para monitorar o fluxo de emoções na rede e o desenvolvimento da polarização.

3. Obtendo dados do Twitter

Para poder acessar os dados do Twitter de forma programática, precisamos criar e registrar um aplicativo no site de desenvolvedores do Twitter para autenticação e, a partir daí, podemos acessar os dados usando a API do Twitter.

3.1. Registrando aplicativo

Para registrar o aplicativo do Twitter, precisamos criar um aplicativo https://apps.twitter.com/. Ao registrar o aplicativo, receberemos consumer_key e consumer_secret_key. Em seguida, na página de configuração do aplicativo, obteremos access_token e access_token_secret, que serão usados para obter acesso ao Twitter em nome de nosso aplicativo. Devemos manter esses tokens de autenticação privados, pois podem ser mal utilizados. A prática recomendada é criar um arquivo de configuração separado e manter esses tokens

3.2. Acessando Dados

O Twitter fornece APIs REST para se conectar com seu serviço. Usaremos uma biblioteca python para acessar a API REST do Twitter chamada Tweepy. Ele fornece métodos de wrapper para acessar facilmente a API REST do twitter.
Para instalar o Tweepy, podemos usar o comando abaixo.
pip install tweepy

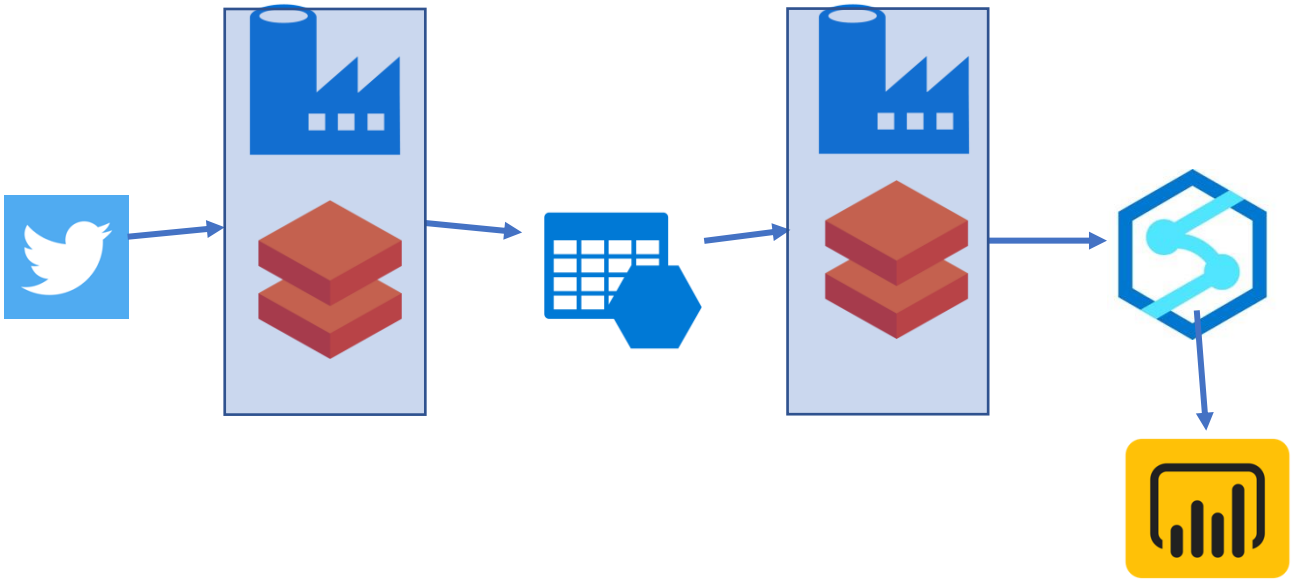
Para autorizar nosso aplicativo a acessar o Twitter em nosso nome, precisamos usar a interface OAuth. O código abaixo usará o método OAuthHandler tweepy e nossos tokens de configuração para fornecer acesso ao Twitter. (Ediger, et al.)

```
import tweepy
from tweepy import OAuthHandler
consumer_key = 'YOUR-CONSUMER-KEY'
consumer_secret = 'YOUR-CONSUMERSECRET'
access_token = 'YOUR-ACCESS-TOKEN'
access_secret = 'YOUR-ACCESS-SECRET'
auth=OAuthHandler(consumer_key,consu mer_secret)
auth.set_access_token(access_token,a ccess_secret)
api = tweepy.API(auth)
```

A variável api agora é nosso ponto de entrada para a maioria das operações que podemos realizar com o Twitter.

4. NLP para Análise de Sentimentos

Neste trabalho utilizamos infraestrutura em nuvem da Azure para criação dos recursos para implementação da solução.



Em uma visão geral temos um código python sendo executado dentro do Azure Databricks que funciona como crawler. Essa aplicação se conecta no Twitter e captura os dados que possuem COVID dentro do seu conjunto de termos.

Orquestrado pelo Azure Data Factory, o Databricks consome os dados do Twitter e depois armazena em um Data Lake Storage Gen 2 os dados.

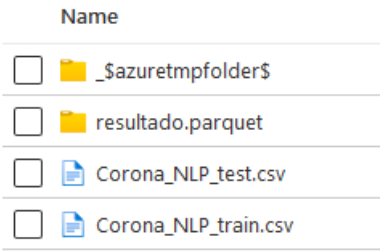


Figura 1 - Azure Storage Account com os dados capturados

O passo seguinte era realizar a marcação sobre os dados e dividir o conjunto de dados em dois grupos, um para treino e outro para teste do modelo.

Com isso, seguimos criando um notebook no Azure Databricks que irá realizar a transformação do dado.

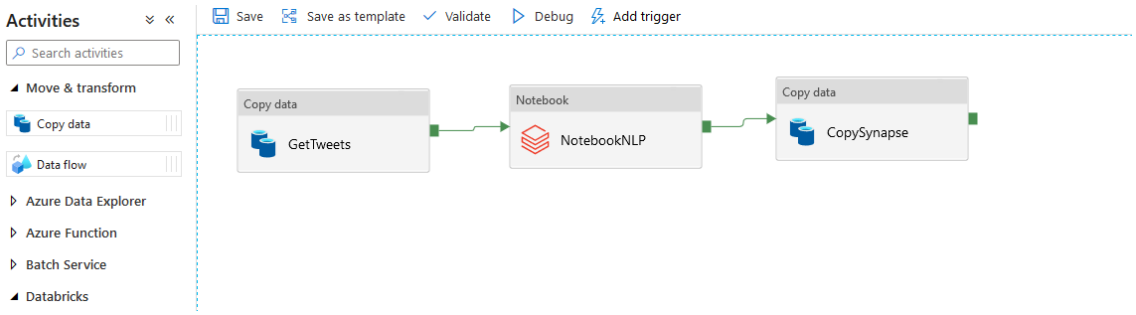


Figura 2 - Pipeline no Azure Data Factory para transformação do dado

Neste notebook, a partir dos dados obtidos fazemos um tratamento nos dados, onde utilizamos NLP para realizar a classificação e marcação das palavras-chaves que serão utilizadas na análise de sentimento.

Uma abordagem alternativa ao NLP seria utilizar um conjunto definido de palavras e classificar o tweet com base nas palavras contidas, porém entendemos que usando NLP temos mais flexibilidade e maior potencial no desenvolvimento da solução.

Uma vez que as palavras são classificadas, e foram retiradas do texto marcações e outros caracteres considerados desnecessários para o trabalho, podemos usar um algoritmo de Machine Learning para classificar o tweet quanto ao sentimento positivo ou negativo.

Inicialmente utilizamos três abordagens: MultinomialNB, Radom Forest e SGD Classifier. Dentro do nosso caso,o SGD Classifier se mostrou mais assertivo no conjunto de dados que estamos tratando.

Desta forma, após treinar o modelo, utilizamos o próprio Databricks, orquestrado pelo Data Factory, para executar o modelo sobre os dados carregados e escrever os resultados em um arquivo no formato PARQUET, dentro do Data Lake Storage.

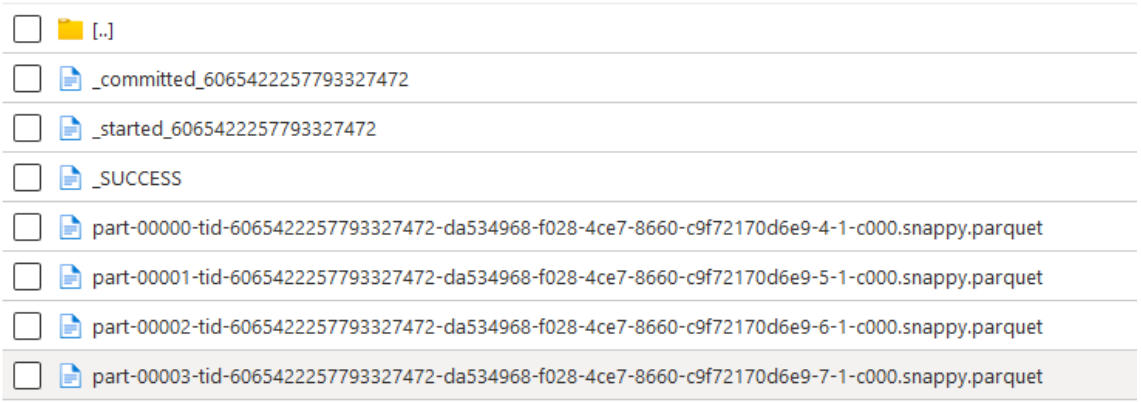


Figura 3 - Arquivos parquet com o resultado da classificação

5. Consumo dos dados de Resultado

Uma vez que os dados em formato parquet ficam com os dados dos resultados, o Data Factory copia esses dados para dentro do Synapse Analytics. O Synapse tem uma estrutura relacional onde irá disponibilizar os dados para consumo.

No Synapse, foi criada uma tabela que ficam armazenados os dados dos resultados, importados a partir dos arquivos parquet, que vão ser utilizados posteriormente por uma ferramenta de dashboard.

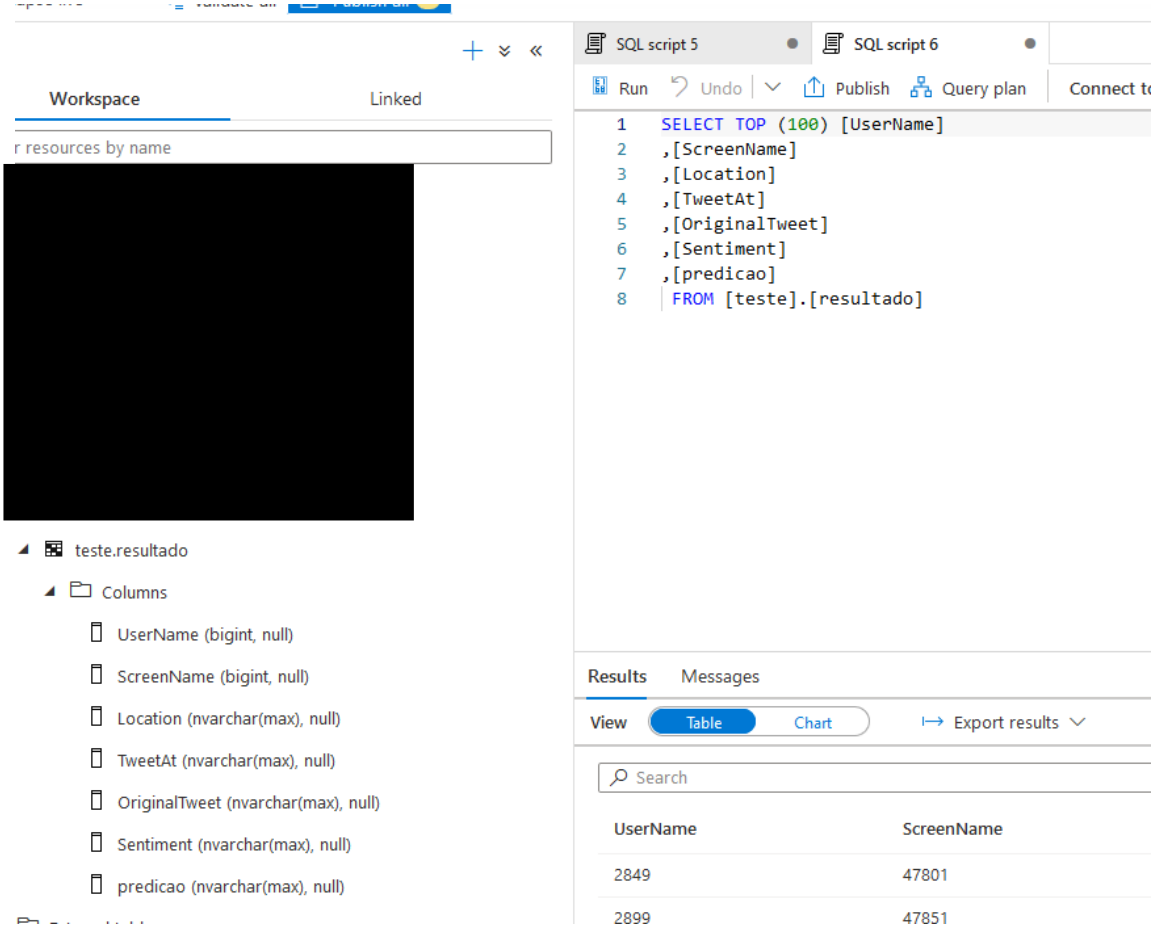


Figura 4 - Tabela de resultados no Synapse

6. Painel Analítico sobre os dados

A última etapa do processo de construção da solução é elaborar uma forma de visualizar os dados produzidos a partir da análise de sentimentos.

Seguindo a proposta arquitetural feita neste trabalho, criamos um painel no Power BI que irá se conectar ao Synapse Analytics e assim disponibilizar a visualização dos dados.

O painel tem por objetivo expor as principais informações referentes aos dados coletados e posteriormente processados. Essas informações são basicamente:

- O texto no tweet
- A visão de onde esse tweet foi feito, geograficamente falando
- A visão de classificação do sentimento dos tweets
- E uma distribuição por data de postagem

A figura abaixo representa o painel construído, lembrando que como se trata de uma solução que garante que o painel seja atualizado a medida que os tweets são coletados, processados e classificados.

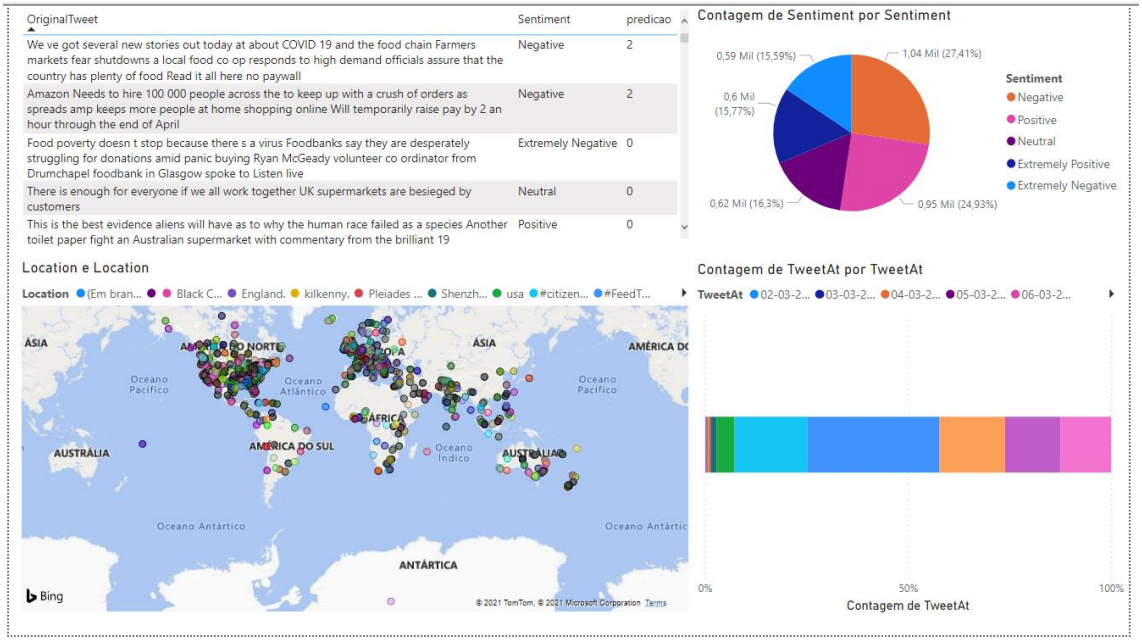


Figura 5 - Painel analítico

7. Conclusão

Neste trabalho implementamos uma solução Big Data que captura dados do Twitter e o submete a um processo de tratamento usando NLP e posteriormente a um processo de classificação utilizando Machine Learning. Por fim, o resultado produzido é exporto em um painel analítico.

Utilizando os recursos disponíveis na Azure, criamos uma solução capaz de capturar tweets usando o Azure Databricks executando um código python. Em posse desses dados o Databricks, orquestrado pelo Azure Data Factory, fizemos um tratamento do conjunto de dados, e usamos NLP para identificação de palavras.

Com isso, ainda no Databricks, usamos o SGD Classifier como algoritmo de Machine Learning que pudesse classificar o sentimento do tweet, quanto ser positivo ou negativo, para o contexto de COVID.

Com esse processamento concluído, criamos um painel no Power BI que consome os resultados produzidos e armazenados em uma tabela no Synapse.

Essa solução se mostrou eficaz e flexível podendo trazer, em tempo real, uma análise sobre o que tem sido comentado nas redes sociais sobre o assunto COVID.

Bibliografia

Balahur, A. (2013). Sentiment analysis in social media texts. *4th workshop on computational approaches to subjectivity, sentiment and social media analysis.*

Ediger, D., Jiang, K., Riedy, J., Bader, D. A., Corley, C., Farber, R., & Reynolds, W. N. (s.d.). Massive Social Network Analysis: Mining Twitter for Social Good. *39th International Conference on Parallel Processing 2010*, (pp. 583-593).

Leskovec, J. (2011). Social media analytics: tracking, modeling and predicting the flow of information through networks. *20th international conference companion on World wide web*, (pp. 277-278).

Ortis, A., Farinella, G. M., Torrisi, G., & Battiato, S. (2018). Visual sentiment analysis based on on objective text description of images. *2018 international conference on content-based multimedia indexing (CBMI)*, (pp. 1-6).

Anexo I – Código para criação da tabela no Synapse Analytics

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [teste].[resultado]
(
    [UserName] [bigint]  NULL,
    [ScreenName] [bigint]  NULL,
    [Location] [nvarchar](max)  NULL,
    [TweetAt] [nvarchar](max)  NULL,
    [OriginalTweet] [nvarchar](max)  NULL,
    [Sentiment] [nvarchar](max)  NULL,
    [predicao] [nvarchar](max)  NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    HEAP
)
GO
```

Anexo II – Notebook Databricks

```
# Databricks notebook source
STORAGE_ACCOUNT_POPE = 'xxx'
SASTOKEN_POPE= 'xxxx'

# COMMAND ----------

# MAGIC %sh
# MAGIC pip install nltk
# MAGIC pip install --upgrade pip

# COMMAND ----------

# MAGIC %sh
# MAGIC pip install beautifulsoup4

# COMMAND ----------

# MAGIC %sh
# MAGIC pip install lxml

# COMMAND ----------

#imports necessários
from pyspark.sql import Window
from pyspark.sql.functions import *
import pyspark.sql.functions as Function
from datetime import timedelta
from pyspark.sql.types import *
from pyspark.sql import SparkSession, Row
import datetime
import numpy as np
```

```
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import string
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier



from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

# COMMAND ----------
```

```
# Unmount todos os volumes.
def delete_mounts(root_mountpoint):
  mnts = dbutils.fs.ls(root_mountpoint + '/')
  for mnt in mnts:
    try:
      dbutils.fs.unmount(mnt[0])
    except Exception as e:
      if "Directory not mounted" in str(e):
        print(mnt[0] , " - Não estava montado !")
        pass
      else:
        print(e)


# COMMAND ----------

# Função para montar os volumes
def
create_mount(nome_container,path_mountpoint,STORAGE_ACCOUNT,SASTOKEN)
:
  CONTAINER = nome_container
  MOUNTPOINT = path_mountpoint
  SOURCE                                                              =
"wasbs://{container}@{storage_acct}.blob.core.windows.net/".format(container=CON
TAINER, storage_acct=STORAGE_ACCOUNT)
  URI                                                                 =
"fs.azure.sas.{container}.{storage_acct}.blob.core.windows.net".format(container=CO
NTAINER, storage_acct=STORAGE_ACCOUNT)

  try:
    dbutils.fs.mount(
    source=SOURCE,
    mount_point=path_mountpoint,
```

```
    extra_configs={URI:SASTOKEN})
  except Exception as e:
    if "Directory already mounted" in str(e):
      print("Já estava montado !")
      pass # Ignore error if already mounted.
    else:
      raise e
  return path_mountpoint

# COMMAND ----------

def criar_tabela_temporaria(mount,fonte,owner,tabela,alias, tipo):
  if owner is None:
    path_tabela = mount + '/' + fonte +  '/' + tabela
  else: path_tabela = mount + '/' + fonte +  '/' +  owner + '/' + tabela
  if tipo == 'parquet':
    df = spark.read.parquet(path_tabela)
  else: df = spark.read.options(header='true').csv(path_tabela)
  df.createOrReplaceTempView(alias)

# COMMAND ----------

#monta os containers
TESTE =
create_mount('teste','/mnt/teste',STORAGE_ACCOUNT_POPE,SASTOKEN_POPE)

# COMMAND ----------

TESTE

# COMMAND ----------

dbutils.fs.refreshMounts()
```

```
# COMMAND ----------

train_dataset = pd.read_csv("/dbfs/mnt/teste/Corona_NLP_train.csv",encoding="latin")
test_dataset = pd.read_csv("/dbfs/mnt/teste/Corona_NLP_test.csv",encoding="latin")

# COMMAND ----------

train_dataset.head()

# COMMAND ----------

test_dataset.head()

# COMMAND ----------

print(train_dataset.columns)
print(test_dataset.columns)

# COMMAND ----------

train_dataset["Sentiment"].unique()

# COMMAND ----------

def classes_def(x):
    if x ==  "Extremely Positive":
        return "2"
    elif x == "Extremely Negative":
        return "0"
    elif x == "Negative":
        return "0"
    elif x ==  "Positive":
        return "2"
    else:
```

```
        return "1"


train_dataset['class']=train_dataset['Sentiment'].apply(lambda x:classes_def(x))

# COMMAND ----------

train_dataset["class"].value_counts(normalize= True)

# COMMAND ----------

nltk.download('stopwords')

# COMMAND ----------

from bs4 import BeautifulSoup
STOPWORDS = set(stopwords.words('english'))

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
from tqdm import tqdm
preprocessed_tweets = []
# tqdm is for printing the status bar
for sentance in tqdm(train_dataset['OriginalTweet'].values):
    sentance = re.sub(r'https?://\S+|www\.\S+', r'', sentance) # remove URLS
    sentance = re.sub(r'<.*?>', r'', sentance) # remove HTML
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub(r'\d+', '', sentance).strip() # remove number
    sentance = re.sub(r"[^\w\s\d]","", sentance) # remove pnctuations
    sentance = re.sub(r'@\w+','', sentance) # remove mentions
    sentance = re.sub(r'#\w+','', sentance) # remove hash
    sentance = re.sub(r"\s+"," ", sentance).strip() # remove space
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)

    sentance = ' '.join([e.lower() for e in sentance.split() if e.lower() not in STOPWORDS])
    preprocessed_tweets.append(sentance.strip())

# COMMAND ----------

# MAGIC %md
# MAGIC ###TF-IDF###

# COMMAND ----------

tf_idf_vect = TfidfVectorizer(min_df=10)
tf_idf_vect.fit(preprocessed_tweets)
print("alguns recursos de exemplo (palavras exclusivas no corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_tweets)
print("o tipo de vetorizador de contagem ",type(final_tf_idf))
```

```
print("a forma do vetorizador TFIDF de texto ",final_tf_idf.get_shape())
print("o número de palavras únicas, incluindo unigramas e bigramas ", final_tf_idf.get_shape()[1])

# COMMAND ----------

X = final_tf_idf
y = train_dataset["class"].tolist()

X_train, X_test, y_train, y_test = train_test_split(X.tocsr(), y, test_size= 0.33, stratify=y, random_state=42)

# COMMAND ----------

# MAGIC %md
# MAGIC ###MultinomialNB###

# COMMAND ----------

grid_params ={'alpha':[10**x for x in range(-4,4)]}
alpha_log = [math.log(x,10) for x in grid_params["alpha"]]

MultinomialNB_model = GridSearchCV(MultinomialNB(),grid_params,
                scoring = 'accuracy', cv=10,n_jobs=-1, return_train_score=True)
MultinomialNB_model.fit(X_train, y_train)

# COMMAND ----------

results = pd.DataFrame.from_dict(MultinomialNB_model.cv_results_)
results = results.sort_values(['param_alpha'])

plt.plot(alpha_log, results["mean_train_score"], label='Acurácia Treinamento')
plt.plot(alpha_log, results["mean_test_score"].values, label='Acurácia CV')
```

```
plt.scatter(alpha_log, results["mean_train_score"].values, label='Pontos de Acurácia de Treinamento')
plt.scatter(alpha_log, results["mean_test_score"].values, label='Pontos Acurácia CV')

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("Acurácia")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
print(MultinomialNB_model.best_estimator_)

# COMMAND ----------

MultinomialNB_model = MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
MultinomialNB_model.fit(X_train,y_train)

y_pred = MultinomialNB_model.predict(X_test)
cm=confusion_matrix(y_test, y_pred)
cm_df=pd.DataFrame(cm,index=[0,1,2],columns=[0,1,2])
print("Accuracy:",accuracy_score(y_test, y_pred))

sns.set(font_scale=1.4,color_codes=True,palette="deep")
sns.heatmap(cm_df,annot=True,annot_kws={"size":16},fmt="d",cmap="YlGnBu")
plt.title("Matriz de confusão")
plt.xlabel("Valor Previsto")
plt.ylabel("Valor Real")

# COMMAND ----------

print(metrics.classification_report(y_test, y_pred,
                                    target_names= train_dataset['class'].unique()))

# COMMAND ----------
```

```
# MAGIC %md
# MAGIC ###Random Forest###

# COMMAND ----------

max_depth = [1,5,10,50]
n_estimators = [5,10,100,500]
grid_params ={'max_depth':max_depth,'n_estimators':n_estimators}

RandomFoest_model = GridSearchCV(RandomForestClassifier(class_weight = 'balanced'), grid_params,
                scoring = 'accuracy', cv=10,n_jobs=-1, return_train_score=True)
RandomFoest_model.fit(X_train, y_train)

results = pd.DataFrame.from_dict(RandomFoest_model.cv_results_)
print(RandomFoest_model.best_estimator_)

# COMMAND ----------

from mpl_toolkits.mplot3d import Axes3D
import matplotlib

max_depth = [1,1,1,1,5,5,5,5,10,10,10,10,50,50,50,50]
n_estimators = [5,10,100,500,5,10,100,500,5,10,100,500,5,10,100,500]
mean_train_score = list(results["mean_train_score"].values)
mean_test_score = list(results["mean_test_score"].values)

fig = matplotlib.pyplot.figure(figsize=(12,6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(max_depth, n_estimators, mean_train_score, c='r', marker='o')
ax.scatter(max_depth, n_estimators, mean_test_score, c='b', marker='o')
```

```
ax.set_xlabel('max_depth ')
ax.set_ylabel('n_estimators')
ax.set_zlabel('Accuracy')

# COMMAND ----------

RandomFoest_model = RandomForestClassifier(bootstrap=True,
class_weight='balanced',
        criterion='gini', max_depth=50, max_features='auto',
        max_leaf_nodes=None, min_impurity_decrease=0.0,
        min_impurity_split=None, min_samples_leaf=1,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        n_estimators=500, n_jobs=1, oob_score=False, random_state=None,
        verbose=0, warm_start=False)
RandomFoest_model.fit(X_train,y_train)

y_pred = RandomFoest_model.predict(X_test)
cm=confusion_matrix(y_test, y_pred)
cm_df=pd.DataFrame(cm,index=[0,1,2],columns=[0,1,2])
print("Accuracy:",accuracy_score(y_test, y_pred))

sns.set(font_scale=1.4,color_codes=True,palette="deep")
sns.heatmap(cm_df,annot=True,annot_kws={"size":16},fmt="d",cmap="YlGnBu")
plt.title("Matriz de Confusao")
plt.xlabel("Valor Previsto")
plt.ylabel("Valor Real")

# COMMAND ----------

print(metrics.classification_report(y_test, y_pred,
                        target_names= train_dataset['class'].unique()))

# COMMAND ----------
```

```
# MAGIC %md
# MAGIC ###SGD Classifier###

# COMMAND ----------

alpha = [10**x for x in range(-4,4)]
penalty = ["l1","l2"]
grid_params ={'alpha':alpha,'penalty':penalty}
alpha_log = [math.log(x,10) for x in grid_params["alpha"]]

SGDClassifier_model = GridSearchCV(SGDClassifier(class_weight= 'balanced'), grid_params,
                scoring = 'accuracy', cv=10,n_jobs=-1, return_train_score=True)
SGDClassifier_model.fit(X_train, y_train)

results = pd.DataFrame.from_dict(SGDClassifier_model.cv_results_)
results = results.sort_values(['param_alpha'])

print(SGDClassifier_model.best_estimator_)

# COMMAND ----------

SGDClassifier_model = SGDClassifier(class_weight='balanced', penalty='l1')
SGDClassifier_model.fit(X_train,y_train)

y_pred = SGDClassifier_model.predict(X_test)
cm=confusion_matrix(y_test, y_pred)
cm_df=pd.DataFrame(cm,index=[0,1,2],columns=[0,1,2])
print("Accuracy:",accuracy_score(y_test, y_pred))

sns.set(font_scale=1.4,color_codes=True,palette="deep")
sns.heatmap(cm_df,annot=True,annot_kws={"size":16},fmt="d",cmap="YlGnBu")
plt.title("Matriz de Confusao")
plt.xlabel("Valor Previsto")
```

```
plt.ylabel("Valor Real")

# COMMAND ----------

print(metrics.classification_report(y_test, y_pred,
                                    target_names= train_dataset['class'].unique()))

# COMMAND ----------

print(test_dataset,y_pred)


# COMMAND ----------

new_series = pd.Series(y_pred)

df= pd.merge(test_dataset, new_series.rename('predicao'), left_index=True, right_index=True)

# COMMAND ----------

df


# COMMAND ----------

def equivalent_type(f):
    if f == 'datetime64[ns]': return TimestampType()
    elif f == 'int64': return LongType()
    elif f == 'int32': return IntegerType()
    elif f == 'float64': return FloatType()
    else: return StringType()

def define_structure(string, format_type):
```

```
    try: typo = equivalent_type(format_type)
    except: typo = StringType()
    return StructField(string, typo)

# Given pandas dataframe, it will return a spark's dataframe.
def pandas_to_spark(pandas_df):
    columns = list(pandas_df.columns)
    types = list(pandas_df.dtypes)
    struct_list = []
    for column, typo in zip(columns, types):
      struct_list.append(define_structure(column, typo))
    p_schema = StructType(struct_list)
    return sqlContext.createDataFrame(pandas_df, p_schema)

# COMMAND ----------

spark_df = pandas_to_spark(df)

# COMMAND ----------

spark_df.write.parquet(TESTE+'/resultado.parquet')

# COMMAND ----------
```