# Trabalho de conclusão de curso de **Pós Graduação Business Intelligence Master" na Pontifícia Universidade Católica do Rio de Janeiro**

## Aluno: André Luis Mendes Teixeira

## Matrícula: 211101300

## Orientadora: Evelyn Batista

Introdução

A extração de dados do Twitter pode ser uma ferramenta valiosa para as empresas públicas que buscam compreender melhor o sentimento e as necessidades dos cidadãos. Com bilhões de tweets sendo publicados diariamente, o Twitter oferece uma riqueza de informações que as empresas públicas podem usar para melhorar a visão e o objetivo da empresa para a sociedade.

No caso da Eletrobrás, a extração de dados do Twitter pode ser uma ferramenta de apoio importante permitindo que a empresa compreenda melhor a opinião pública sobre a sua atuação no setor de energia e avalie a efetividade de suas estratégias de comunicação, além de se manter atualizada sobre as tendências do setor de energia.

A Interface de Programação de Aplicação (API) do Twitter é um conjunto de ferramentas e interfaces de programação que permitem que desenvolvedores criem aplicativos que se integram com a plataforma do Twitter. Ao longo do tempo, a API do Twitter passou por várias mudanças de contexto que afetaram os recursos disponíveis para os desenvolvedores e a forma como esses recursos podem ser acessados.

Ferramentas utilizadas

Para a consulta , extração e tratamento dos dados, usamos as seguintes ferramentas:

- Google Colaboratory: Também conhecido como “Colab”, é um serviço de nuvem gratuito hospedado pelo Google para incentivar a pesquisa de aprendizado de máquina e inteligência artificial.

- Linguagem Python: é uma linguagem de programação interpretada; ou seja, não precisa ser compilada antes de ser executada, tornando de fácil escrita e teste. A linguagem tem sido muito utilizada em diversas áreas de tecnologia e muito difundida em análise de dados e inteligência artificial.

Bibliotecas

As bibliotecas listadas abaixo foram importadas para possibilitar a extração, tratamento e análise dos dados, além do treinamento de modelos.

import tweepy

import numpy as np

import pandas as pd

import re

import string

import matplotlib.pyplot as plt

from textblob import TextBlob

#from google.colab                    import drive

from nltk.corpus                     import stopwords

nltk.download('punkt')

from nltk.tokenize                   import word\_tokenize

import nltk

nltk.download('vader\_lexicon')

from collections import Counter

from nltk.tag import pos\_tag

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes             import MultinomialNB

from sklearn                         import metrics

from sklearn.model\_selection         import cross\_val\_predict

from sklearn.ensemble                import RandomForestClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

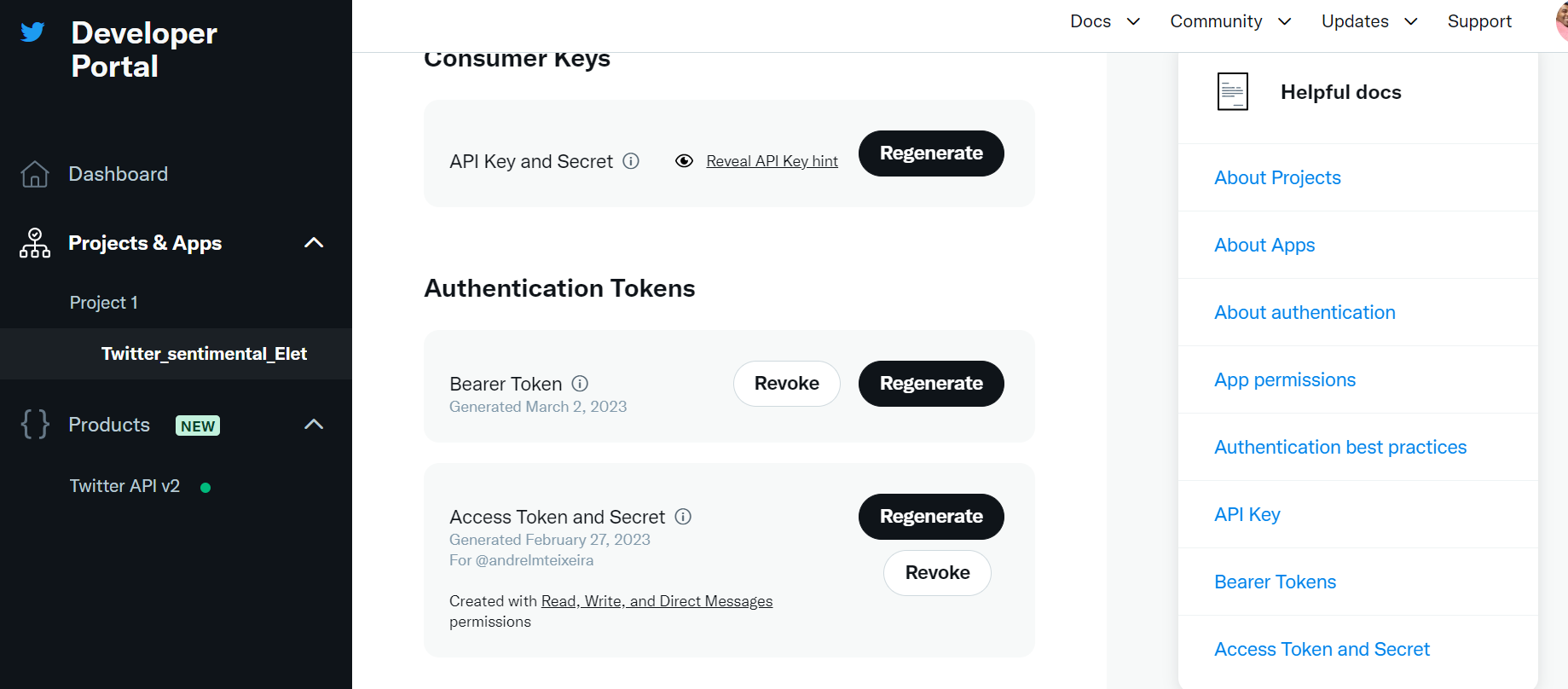
from sklearn.metrics import accuracy\_score

Extração dos dados

Para realizar a extração dos dados para este estudo, usamos a biblioteca do Python TWEEPY que serve para acessar a API do Twitter, A versão 2 do Tweepy introduziu um novo método chamado “search\_recent\_tweets”, que permite pesquisar tweets recentes com base em determinados parâmetros de pesquisa

!pip3 install tweepy --upgrade

Para acessar a API do Twitter , primeiramente precisamos criar/ter uma conta no Twitter em seu site de desenvolvimento, <https://developer.twitter.com/en/portal/dashboard>.



E através do comando abaixo, conseguimos a autenticação necessária para acessar a API do Twitter :

#Chave

bearer\_token = "<bearer token gerada no site de desenvolvimento>"

client = tweepy.Client(bearer\_token=bearer\_token)

Após a autenticação, usamos o método da API V2 do Twitter “search\_recent\_tweets”, para extração de 2000 registros em texto e armazenamos em um dataframe e realizamos em conjunto, a análise de sentimento de textos usando o TextBlob.

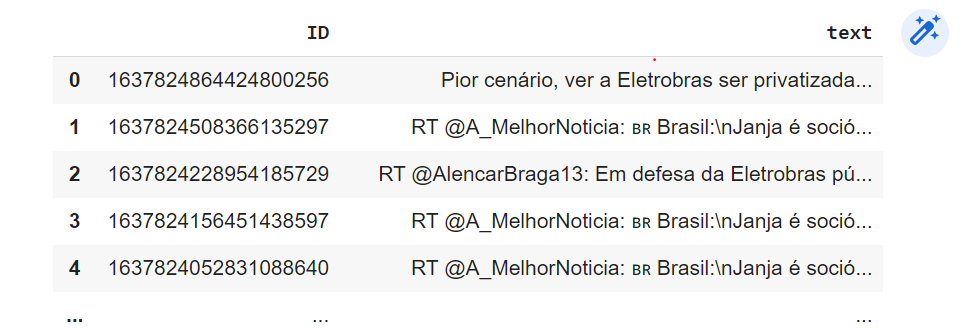
public\_tweets = tweepy.Paginator(client.search\_recent\_tweets, query='eletrobras',

                              max\_results=100).flatten(limit=2000)

Search\_recent\_tweets: O endpoint de pesquisa recente retorna Tweets dos últimos sete dias que correspondem a uma consulta de pesquisa.

OBS: O código acima permite que tenhamos um limite de 2000 linhas retornadas.

Com o apoio da biblioteca Pandas, extraímos os textos e importamos em um dataframe contendo inicialmente as colunas ID e texto.



Tratamento dos dados

Após a extração dos dados, existiu a necessidade de realizar um saneamento dos dados, uma vez que o Twitter é um campo de texto livre e muitas escritas podem vir de diversas formas, por isso, usamos ferramentas como “re.sub” e o “re.escape” que usam expressões regulares para remoção de pontuações e palavras que não tenha algum sentido. A tokenização também se faz importante neste processo de tratamento dos dados adquiridos pela API.

Análise de Sentimento

A análise de sentimento é importante para rotularmos as saídas dos textos extraídos de nossa base de dados. Para tal função , usamos uma biblioteca de processamento de linguagem natural chamada NLTK (Natural Language Toolkit), importando ferramentas para a possibilidade de análise.

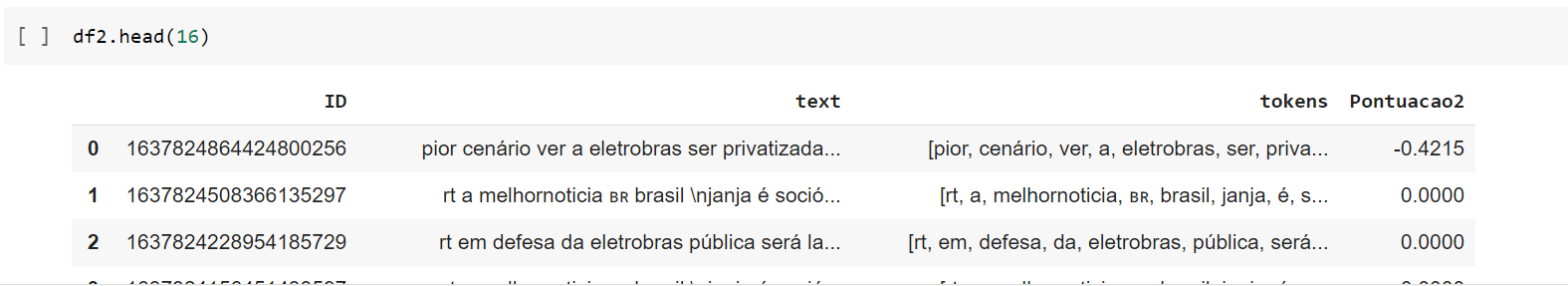
O resultado da análise foi extraído após a criação de uma função denominada “analisar\_sentimento2”:

def analisar\_sentimento2(texto):

    tokens = TweetTokenizer(texto)

    score = sia.polarity\_scores(texto)

    return score['compound']



Como o resultado é numérico, precisou-se rotular os dados para uma saída categórica :

# Define uma função de mapeamento

def mapear\_valor(valor):

    if valor > 0:

        return 'Positivo'

    elif valor < 0:

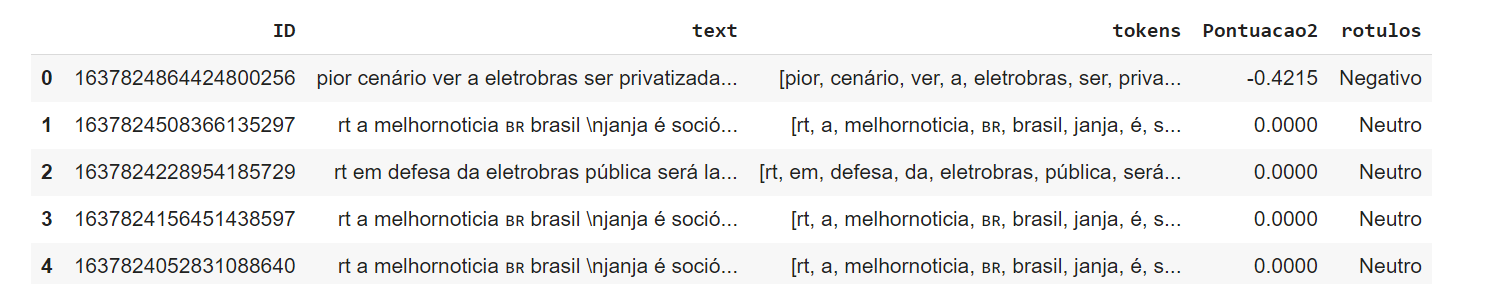
        return 'Negativo'

    else:

        return 'Neutro'

df2['rotulos'] = df2['Pontuacao2'].apply(mapear\_valor)

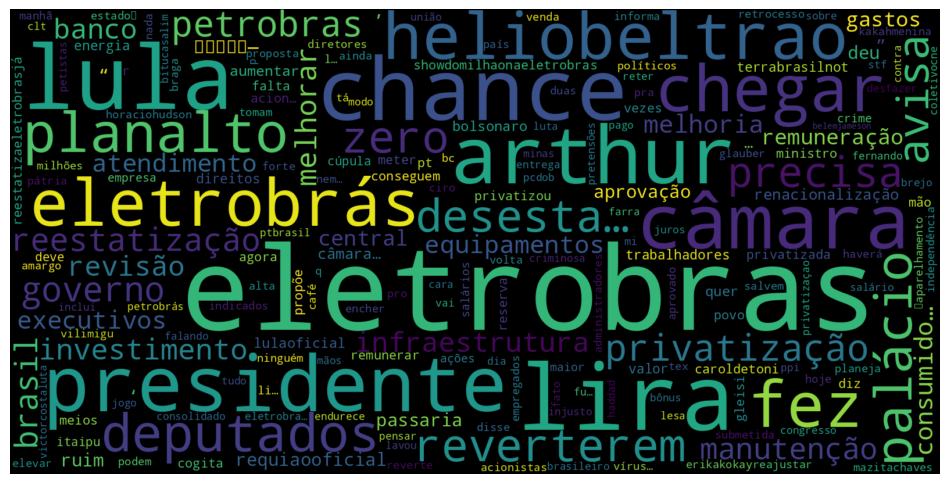
Onde: valores **maiores** que zero atribuímos no rótulo o valor “positivo”, valores **menores** que zero atribuímos no rótulo o valor “negativo” e valores **iguais** que zero atribuímos no rótulo o valor “neutro”



Nuvem de palavras

Wordcloud – em português, nuvem de palavras ou nuvem de tags – é um tipo de **visualização de dados** muito poderoso e [ferramenta de Data Science](https://sigmoidal.ai/como-aprender-data-science/) usado quando estamos trabalhando com textos, documentos, pesquisas, entre outras.

Resumidamente, é como se você estivesse **contando a frequência** com que cada palavra aparece em um texto. Com essa frequência, você define tamanhos proporcionais às palavras, dispondo-as, também, em ângulos diferentes.



Treinamento de modelos

O primeiro modelo usado foi o algoritmo Multinomial Naive Bayes que é um algoritmo de aprendizado de máquina utilizado em problemas de classificação de texto ou documentos que envolvem mais de 2 categorias.

Todos os modelos obtiveram a separação de datasets em treinamento (80%) e em teste (20%).

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df2['text'], df2['rotulos'], test\_size=0.2)

vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor = None,stop\_words = stop\_words, max\_features = 5000)

X\_train\_vectors = vectorizer.fit\_transform(X\_train)

X\_test\_vectors = vectorizer.transform(X\_test)

mnb = MultinomialNB()

mnb.fit(X\_train\_vectors, y\_train\_mnb)

y\_pred\_mnb = mnb.predict(X\_test\_mnb\_vectors)

O segundo modelo foi o algoritmo Multnomial usando Bigrams para tentar obter resultados mais precisos nos resultados de classificação.

vectorizer = CountVectorizer(ngram\_range = (1, 2),analyzer = "word", tokenizer = None, preprocessor = None,stop\_words = stop\_words, max\_features = 5000)

X\_train\_mnb2\_vectors = vectorizer.fit\_transform(X\_train)

X\_test\_mnb2\_vectors = vectorizer.transform(X\_test)

modelo = MultinomialNB()

modelo.fit(X\_train\_mnb2\_vectors, y\_train)

O terceiro modelo foi o Random Forest, este algoritmo cria várias árvores de decisão aleatórias a partir de subconjuntos aleatórios do conjunto de dados original, criando, assim, uma floresta de árvores.

vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor = None,stop\_words = stop\_words, max\_features = 5000)

train\_data\_features = vectorizer.fit\_transform(X\_train)

forest = RandomForestClassifier(n\_estimators = 100)

class\_sentimentos = y\_train.values

forest = forest.fit(train\_data\_features, class\_sentimentos)

test\_data\_features\_resultados = vectorizer.transform(X\_test)

resultados = forest.predict(test\_data\_features\_resultados)

O quarto modelo é o Support Vector Model (SVM) é um algoritmo de aprendizado de máquina supervisionado que pode ser usado para desafios de classificação ou regressão.

vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor = None,stop\_words = stop\_words, max\_features = 5000)

X\_train\_svm\_vectors = vectorizer.fit\_transform(X\_train)

X\_test\_svm\_vectors = vectorizer.transform(X\_test)

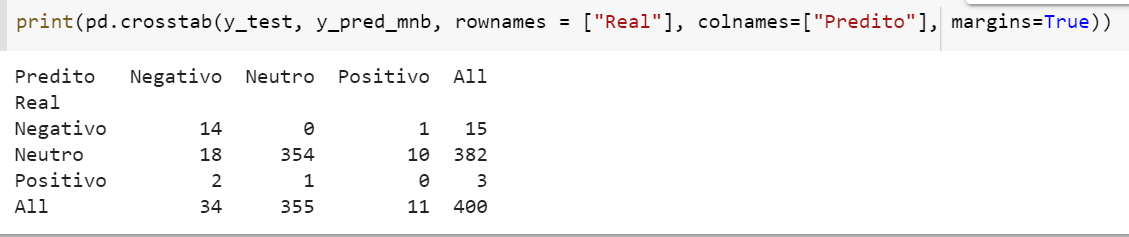
svm = SVC(kernel='linear')

svm.fit(X\_train\_svm\_vectors, y\_train)

Resultados

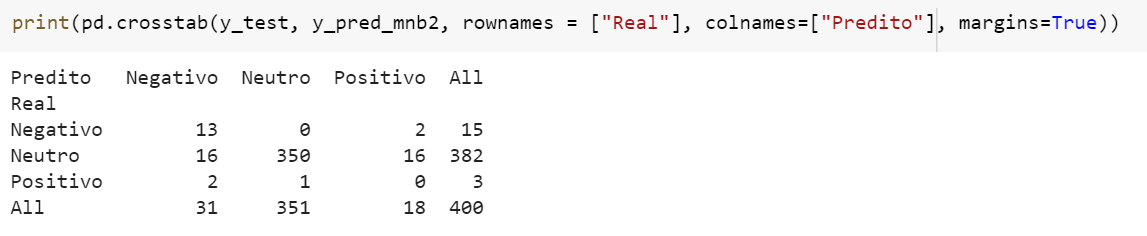
Abaixo, são apresentados as matrizes e os valores de acurácia de cada um dos modelos apresentados:

1. Multinomial Naive Bayes



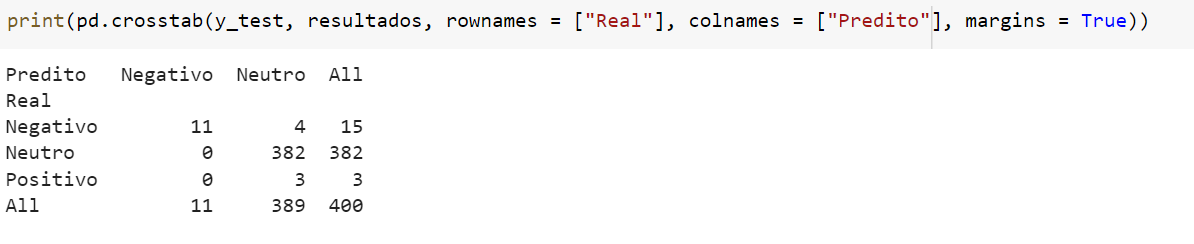
Acurácia: 0,92

1. Multinomial usando Bigrams



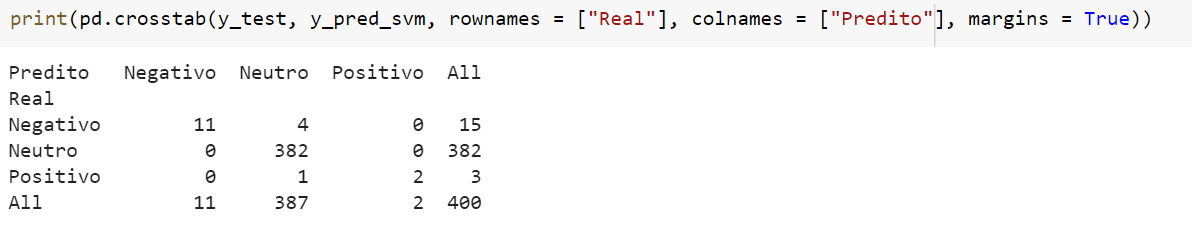
Acurácia: 0,9075

1. Random Forest



Acurácia: 0,9825

1. SVM



Acurácia: 0,9875

Conclusão

Após o treinamento dos 4 modelos de classificação de textos, o algoritmo Support Vector Machine (SVM) foi o que se mostrou mais eficaz tendo uma acurácia superior aos demais algoritmos apresentados nesse estudo.

As maiores dificuldades no projeto foram relacionadas à extração da base de dados em que as mudanças de contexto da API do Twitter tiveram que ser satisfeitas, tais como mudanças de autenticação, restrição de acesso limitando número de solicitações e determinados recursos e mudança de estrutura dos dados da API ao longo do tempo o que provocou mudanças nos modos de autenticação, contextos, parâmetros e chamadas a variáveis. Por isso, o apoio e experiência de minha orientadora e a consulta profunda da [documentação de API do Twitter](https://developer.twitter.com/en/docs/twitter-api/tools-and-libraries/v2) foram fundamentais para a evolução do projeto.

Resumidamente, as mudanças de contexto na API do Twitter são uma parte natural da evolução da plataforma e da tecnologia subjacente. Essas alterações nos alertam que os desenvolvedores precisam estar cientes dessas mudanças e atualizar seus aplicativos em conformidade para garantir que seus aplicativos continuem funcionando corretamente e de acordo com as políticas do Twitter.

Esse trabalho foi concebido através de muitos conhecimentos adquiridos do Curso de Pós-Graduação Business Intelligence Master usando conceitos de Python adquiridos nos módulos de Sistemas de Apoio à Decisão, conceitos de Localização e Uso de Informação e muitas atividades envolvidas em módulos de Processamento de Linguagem Natural e DataMining.