FACULDADE DE TECNOLOGIA DE BAURU

TECNOLOGIA EM BANCO DE DADOS

Análise e replicação de projeto de Ciência de Dados: detecção de discursos de ódio em publicações do X

Autor:
Guilherme Augusto Ribeiro Cadete

Bauru/SP
2025

Análise e replicação de projeto de Ciência de Dados: detecção de discursos de ódio em publicações do X

Autor:
Guilherme Augusto Ribeiro Cadete

Relatório de pesquisa apresentado como requisito para aprovação na disciplina Laboratório de Desenvolvimento em Banco de Dados VI do curso de Tecnologia em Banco de Dados, Faculdade de Tecnologia de Bauru.

Profa. Dra. Patricia Bellin Ribeiro

Bauru/SP
2025

SUMÁRIO

Pág.

RESUMO

Este estudo apresenta a reprodução e a análise crítica de um projeto de aprendizado de máquina desenvolvido para detectar discursos de ódio em tuítes, originalmente proposto por Aman Kharwal. Todos os procedimentos e códigos fornecidos pelo autor foram executados com sucesso utilizando Python e o ambiente Google Colab, resultando em um modelo funcional de classificação de texto baseado em técnicas de Processamento de Linguagem Natural (PLN).

Ao longo do processo de replicação, um desafio enfrentado foi a limitação das explicações sobre o funcionamento das funções e dos utilizados, o que demandou pesquisas em novas fontes de pesquisa e informação. Além disso, o projeto original não ofereceu instruções explícitas sobre como aplicar o modelo de detecção desenvolvido ao conjunto de dados para teste oferecido pelo próprio autor. Para enfrentar esse problema, implementou-se um código adicional — adaptado a partir de uma referência disponível no Stack Overflow. Os tuítes de teste, então, puderam ser classificados, e a análise qualitativa do resultado revelou tanto associações adequadas quanto previsões questionáveis. Esses achados evidenciam limitações do modelo ao lidar com ambiguidades semânticas ou com excertos que dependem de contexto para serem classificados. Percebeu-se, ainda, que as técnicas de pré-processamento utilizadas podem impactar significativamente a precisão das detecções.

A replicação indica que, embora o projeto sirva como uma estrutura introdutória para aqueles que desejam explorar a detecção de discurso de ódio baseada em PLN, ainda demanda métodos e códigos mais robustos capazes de compreender mais profundamente os tuítes. As reflexões apresentadas neste trabalho sugerem oportunidades para pesquisas futuras e para o aperfeiçoamento da relação entre tecnologias computacionais e conhecimentos sobre a linguagem e a comunicação.

ABSTRACT

This study presents the reproduction and critical analysis of a machine learning project designed to detect hate speech in tweets, originally developed by Aman Kharwal. All procedures and code provided by the author were successfully executed using Python and the Google Colab environment, resulting in a functional text classification model based on Natural Language Processing (NLP) techniques.

Throughout the replication process, one challenge encountered was the limited explanation regarding the functioning of the functions and tools used, which required additional research from external sources. Moreover, the original project did not provide explicit instructions on how to apply the developed detection model to the test dataset made available by the author. To address this issue, an additional code—adapted from a reference found on Stack Overflow — was implemented. This enabled the classification of the test tweets, and qualitative analysis of the results revealed both appropriate associations and questionable predictions. These findings highlight limitations of the model when dealing with semantic ambiguities or excerpts that rely on contextual interpretation. It was also observed that preprocessing techniques can significantly impact detection accuracy.

The replication indicates that, although the project serves as an introductory framework for those interested in exploring NLP-based hate speech detection, it still requires more robust methods and code capable of achieving deeper understanding of the tweets. The reflections presented in this work suggest opportunities for future research and for strengthening the relationship between computational technologies and knowledge from language and communication studies.

1. INTRODUÇÃO

Estima-se que o mundo gere cerca de 2,5 quintilhões de bytes de dados diariamente[1], o equivalente ao conteúdo de dois Museus do Louvre. Grande parte dessa informação provém de publicações nas mídias sociais e páginas informativas, incluindo sites governamentais, blogs de empresas e portais jornalísticos — ou seja, textos. Segundo Oliveira Mesa (2016), os dados estruturados — geralmente números e pequenos textos armazenados em tabelas — representam apenas 20% da informação produzida em uma organização. Já no montante dos dados não estruturados — como imagens, vídeos, áudios e textos — 80% é informação textual.

Em meio à tamanha quantidade de letras, palavras, frases e parágrafos, é fundamental desenvolvermos a capacidade de não apenas ler, mas interpretar os conteúdos, incluindo aqueles produzidos nas mídias sociais. É nelas que, em grande parte na atualidade, se dão os debates sobre os mais diferentes campos da sociedade, como cultura, meio ambiente e política. E entre as plataformas que se destacam como palco para o compartilhamento e a discussão de pontos de vista, está o X — o antigo Twitter, fundado em 2006, mas adquirido e transformado em 2022 pelo empresário estadunidense Elon Musk.

De fato, o X é utilizado por desde cidadãos comuns e personalidade midiáticas e políticas, do Brasil e do mundo, cujos posts geram engajamento por meio de respostas — comentários na própria publicação — e compartilhamento — quando o conteúdo original é republicado, com ou sem comentários. Essa dinâmica origina interações muitas vezes acaloradas, culminando em discursos de ódio: um tipo de comunicação que ataca ou utiliza "linguagem depreciativa ou discriminatória contra uma pessoa ou grupo com base no que é" (Kharwal, 2020), ou seja, em características como "religião, etnia, nacionalidade, raça, cor, ascendência, sexo ou outro fator de identidade" (Kharwal, 2020).

Embora a própria plataforma X ofereça mecanismos de denúncia contra essas manifestações de ódio e haja mecanismos legais para penalizar os casos em que a violência verbal se transforma em crime, faz necessário analisar esses textos, extrair entendimentos dos mesmos e classificá-los, a fim de "evitar a desinformação e a polarização adicional de tópicos tão delicados como os atuais" (González Sánchez, 2022, p.4). E para isso, o Processamento de Linguagem Natural, que une o potencial da linguística e da Inteligência Artificial, pode ser muito útil.

Aliás, conforme aponta Nascimento (2023), as aplicações destinadas a esse tipo de análise devem ser as melhores possíveis, considerando que, para que um texto seja interpretado, demanda também que seja submetido a processos de extração, transformação e carga. Em outras palavras, esses programas devem ter a capacidade de receber a informação, submetê-la a técnicas de limpeza e tratamento e armazená-la, de forma a estar sempre acessível e assim atender às necessidades do usuário.

Entre a gama de softwares capazes de garantir o cumprimento dessas etapas, alguns se destacam. É o caso da linguagem de programação Python e da plataforma Google Colab — voltada principalmente para o desenvolvimento de projetos com Python —, as

[1] MIT Technology Review — Data Literacy: a importância da alfabetização em dados em um mundo Big Data.

quais, integradas a técnicas de *machine learning*, possibilitam a criação de robustas ferramentas de análise textual.

Considerando a importância da análise textual com foco em publicações em mídias sociais, bem como a possibilidade de automatizar e otimizar esse processo mediante as tecnologias Python e Google Colab, o presente trabalho se propõe a examinar e reproduzir um projeto de Ciências de Dados, de autoria do estrategista de dados Aman Kharwal, para detecção de discursos de ódio na plataforma X. Por meio de técnicas de engenharia reversa, busca-se compreender, na prática, os processos inerentes à análise de discursos no ambiente digital e verificar a possibilidade de aplicar o modelo a outros formatos de textos para além dos produzidos nas mídias digitais.

2. OBJETIVOS

2.1 Objetivo geral

O Analisar e reproduzir, através de engenharia reversa, um projeto de ciências de dados para detecção de discursos de ódio em tuítes (posts na plataforma X) a fim de compreender as ferramentas e os métodos necessários para detecção automatizada de sentimentos em textos.

2.2 Objetivos específicos

a) Desenvolver, na prática, a aplicação de processos de ETL (extração, transformação e carregamento) com foco em bases de dados textuais.
b) Compreender a adaptabilidade dos métodos apresentados no projeto modelo para detectar sentimentos em outros formatos de texto, como conteúdos jornalísticos.
c) Observar se as detecções de discurso de ódio geradas pelo sistema implementado podem apoiar estudos em análise do discurso.

3. MATERIAIS E MÉTODOS

Os procedimentos apresentados no projeto de Aman Kharwal para detecção de discursos de ódio em tuítes podem ser compreendidos a partir do conceito ETL, cujas técnicas garantem a adequação e o uso assertivo de bases de dados em sistema de análise. O modelo, além disso, é baseado na linguagem de programação Python, que pode ser manipulada em diferentes aplicações, incluindo o Google Colab, escolhido neste trabalho.

Detalhes sobre os processos ETL, a linguagem Python e o Google Colab são apresentados a seguir, bem como o hardware e demais softwares utilizados. Por fim, esta seção descreve e comenta as instruções de Kharwal na íntegra, desde o acesso aos conjuntos de dados, que baseiam o modelo, ao seu treinamento e teste.

3.1 Extração, transformação e carga

O processo de análise de textos envolve o que Nascimento (2023) chama de extração de valor dos dados, ou seja, a captura de informações valiosas no material examinado. Para isso, os dados precisam ter qualidade, o que demanda efetuar estratégias de ETL — relativo a *extract*, *transform*, *load* (extração, transformação e carga de dados, em tradução livre) —, que tratam os conteúdos e os tornam adequados para análises.

O processo de extração se refere à captação dos dados em estado bruto, a partir de uma ou diferentes fontes, e seu armazenamento provisório até a próxima etapa, de transformação. Nela, os dados são limpos e aprimorados a fim de se tornarem

consistentes, utilizando-se de práticas como a limpeza textual por meio de expressões regulares (regex) para remoção de ruídos (menções, hashtags, URLs), conversão para minúsculas e remoção de *stopwords*. Por fim, executa-se a carga dos dados tratados no ambiente de processamento (por exemplo, na memória de trabalho de um ambiente como o Google Colab), onde estarão disponíveis e estruturados para a etapa de modelagem e análise.

As estratégias ETL de pré-processamento e mapeamento de textos ajudam a eliminar inconsistências, duplicações e erros, além de garantir que informações com origens e formatos diversos atendam com uniformidade aos requisitos da análise. Ainda segundo Nascimento (2023), essas ações otimizam o uso de modelos de processamento de linguagem natural (PLN), cujas redes neurais e algoritmos conseguem "capturar com precisão nuances da linguagem, regras gramaticais e contextos" (Nascimento, 2023, p. 8).

3.2 Google Colab

Para o desenvolvimento deste trabalho, optou-se pela utilização do Google Colab, um ambiente de desenvolvimento *online* e gratuito que executa códigos Python diretamente no navegador. Baseado na plataforma Jupyter Notebook, o Colab elimina a necessidade de complexas configurações locais de software e hardware, oferecendo previamente bibliotecas fundamentais para análise de dados, o que facilita a reprodutibilidade dos experimentos. A área de trabalho da plataforma permite a criação de campos destinados às instruções em Python e também a títulos e descrições, o que torna o Google Colab "uma ponte constante entre o código e os textos explicativos, com um servidor que compila e executa essas partes do código" (González Sánchez, 2022, p.19).

Na escolha da aplicação para executar os códigos Python, também cogitou-se usar o Jupyter Notebook. O Google Colab foi escolhido, porém, por não demandar instalação e facilitar o compartilhamento e o acesso ao projeto a partir de qualquer dispositivo, bastando acessar a conta Google que detém o mesmo. Um ponto negativo é que, diferentemente do Jupyter, o Google Colab passa por desconexões automáticas — já que roda em servidores remotos —, o que demanda recarregar as tabelas no ambiente e reexecutar os códigos já processados anteriormente.

3.3 Python

Python é uma linguagem de programação de alto nível orientada a objetos, com código aberto e uma extensa variedade de bibliotecas de alta qualidade, com termos e funções que expandem a capacidade da linguagem principal. Segundo Oliveira Mesa (2016), o modelo é de fácil leitura e aprendizado, sendo muito utilizado nas áreas de Ciência, análise de dados e Engenharia. Conta ainda com uma grande comunidade de usuários que impulsiona o crescimento e a popularidade do modelo.

A infinidade de recursos disponibilizados pelo Python permite, segundo Pedroso (2019), desde a carga ao tratamento e análise de textos — ou seja, os processos ETL. Detalhes sobre os métodos aplicados neste trabalho são apresentados a seguir.

a) Tratamento e análise de dados textuais: a linguagem Python oferece um ecossistema robusto para PLN. Neste projeto, destaca-se o uso de bibliotecas como:
    i. Pandas: utilizada para a extração e carga dos dados a partir de arquivos CSV, e para manipulação e limpeza inicial dos dados tabulares;

ii. sklearn (Scikit-learn): biblioteca fundamental para *machine learning*, com componentes para a transformação dos dados textuais e algoritmos de classificação que aprendem a detectar padrões nos dados;
iii. re (relativo a Regex, acrônimo para *regular expressions* — expressões regulares, em tradução livre): ferramenta também para a transformação (limpeza) de texto, é útil na remoção de ruídos específicos de redes sociais, como menções de usuário (@), hashtags (#), URLs e a palavra "RT" (relativa a retuíte, o compartilhamento de postagens alheias), tornando os dados consistentes para a análise.

b) *Machine learning*: trata-se de um ramo da inteligência artificial em que sistemas aprendem com dados, identificam padrões e tomam decisões com mínima intervenção humana (PEDROSO, 2019). No caso da análise de sentimentos, o modelo é treinado com exemplos já classificados (por exemplo, "ódio" ou "não-ódio") e, uma vez treinado, pode classificar novos textos com base nos padrões aprendidos, melhorando sua precisão conforme a quantidade e qualidade dos dados de treinamento.

É importante afirmar que, além das bibliotecas já oferecidas, a linguagem Python permite a criação de programas próprios, possibilitando inclusive a integração entre diferentes aplicações Python. Outro ponto a se destacar é que, conforme descrito, esta linguagem é capaz de percorrer todas as etapas ETL necessárias à análise textual computacional: extração, transformação e carga.

3.4 Hardware e demais softwares

Todo o trabalho foi executado a partir de um notebook Acer modelo Nitro 5, com processador AMD Ryzen 7 4800H, 8GB de memória RAM e sistema operacional (SO) Windows 11 Home. Foi utilizada a versão gratuita *online* do aplicativo Microsoft Excel para ler e estudar os arquivos .csv disponibilizados pelo projeto. A plataforma Google Colab, para executar os códigos Python, foi acessada a partir do navegador Brave (versão 1.84.132). O aplicativo Acrobat Reader (versão 2005.001.20756) foi usado para leitura de documentos, incluindo referências bibliográficas, em formato .pdf.

3.5 Execução das instruções

Os procedimentos descritos a seguir seguem a mesma divisão e sequência determinadas no projeto de Aman Kharwal, e são acrescidos de comentários e detalhes que ajudam a compreender o funcionamento e a lógica dos códigos, das bibliotecas e das funções.

3.5.1 Download e importação dos dados

Após a leitura atenta de todas as etapas do projeto para uma compreensão preliminar dos desafios e procedimentos, foi feito o download do conjunto de dados a partir do link disponibilizado nas instruções, que levava ao portal GitHub. O arquivo em formato .rar continha dois documentos em formato .csv, um chamado 'train' com 31.962 tuítes para treinar o sistema, e outro intitulado 'test' com 17.197 tuítes para testá-lo. Ambas as tabelas utilizavam tabulação por vírgulas. Aquela referente aos tuítes de treinamento continha três campos: *id* (atributo identificador), *label* (com valor 0 ou 1, para rotular os tuítes como 'neutro' [0] ou de 'ódio' [1]) e *tweet* (com o texto do tuíte). Já a tabela de treinamento continha apenas os campos *id* e *tweet*, ou seja, não apresentava previamente o rótulo para 'neutro' ou 'ódio'.

No Google Colab, foi iniciado um novo notebook (em outras palavras, um projeto em branco), e os arquivos .csv foram carregados no ambiente (pasta 'sample-data'). A

seguir, na célula de código foi executada a instrução disponível na figura 1, que importou a biblioteca pandas e a partir dela leu as tabelas carregadas. As linhas referentes à leitura dos arquivos (2 e 4) foram adaptadas para especificar o caminho correto dos arquivos no Google Colab.

Figura 1 – Leitura dos arquivos .csv.

```
1 import pandas as pd
2 train = pd.read_csv("/content/sample_data/train.csv")
3 print("Training Set:"% train.columns, train.shape, len(train))
4 test = pd.read_csv("/content/sample_data/test.csv")
5 print("Test Set:"% test.columns, test.shape, len(test))
```

Fonte: Elaborado pelo autor.

A resposta gerada pela execução é exibida na figura 2 e corresponde à apresentada no projeto. Finalizou-se, assim, a etapa de extração do ETL.

Figura 2 – Resultado da execução do código presente na figura 1.

```
Training Set: (31962, 3) 31962
Test Set: (17197, 2) 17197
```

Fonte: Elaborado pelo autor.

3.5.2 Data Cleaning

A etapa a seguir se refere à segunda fase do ETL, a transformação dos dados. Isso é feito com duas bibliotecas: pandas (já importada anteriormente) e re. Além da linha 1, que importa re, as demais têm o seguinte propósito:

2 *def clean_text(df, text_field)*: define (com *def*) uma função chamada 'clean_text', que receberá dois parâmetros, 'df' e 'text_field';

3 *df[text_field] = df[text_field].str.lower()*: cada campo de texto (*text_field*) recebido pela função terá seus caracteres convertidos em minúsculos (*str.lower*)[2];

4 *df[text_field] = df[text_field].apply(lambda elem: re.sub*[3]*(r"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", "", elem))*: agora, com re, cada campo de texto passa por uma segunda transformação, removendo menções a usuários, caracteres não alfanuméricos (como pontuação e emojis), links e o termo "rt". Como a função *re.sub*, por si só, não lê linha por linha da coluna, aplica-se (*apply*) a função *lambda*, que faz esse mapeamento[4];

5 *return df*: finaliza a função e retorna seu resultado, ou seja, o *dataframe* limpo;

6 *test_clean = clean_text(test, "tweet")*: cria o *dataframe* 'test_clean', cujo resultado é a execução da função 'clean_text', que recebe como parâmetros a tabela 'test' e sua coluna 'tweet';

7 *train_clean = clean_text(train, "tweet")*: cria o novo *dataframe* 'train_clean', cujo resultado será a execução da função 'clean_text', que recebe como parâmetros a tabela 'train' e sua coluna 'tweet'.

Os novos *dataframes* são criados virtualmente, ou seja, não são gravados em disco no sistema.

Figura 3 – Importação da biblioteca re e limpeza dos tuítes.

```
1 import re
```

[2] Python Software Foundation — *str.lower*, documentação oficial.
[3] Python Software Foundation — *re.sub*, documentação oficial.
[4] GeeksforGeeks — Apply a function to each row or column in Dataframe using pandas.apply().

```
2  def  clean_text(df, text_field):
3      df[text_field] = df[text_field].str.lower()
4      df[text_field] = df[text_field].apply(lambda elem:
   re.sub(r"(@[A-Za-z0-9]+)|([^0-9A-Za-z
   \t])|(\w+:\/\/\S+)|^rt|http.+?", "", elem))
5      return df
6  test_clean = clean_text(test, "tweet")
7  train_clean = clean_text(train, "tweet")
```

Fonte: Elaborado pelo autor.

O código da figura 3 foi executado com sucesso.

3.5.3 Balanceamento dos dados

Também parte do processo de transformação em ETL, a seguir foi necessário aplicar uma técnica de balanceamento de dados. Conforme explica Kharwal (2020) — e como se observa nos registros das tabelas 'train' e 'test' —, há uma quantidade consideravelmente menor de tuítes com discurso 'de ódio' em comparação aos demais. Se esses dados fossem utilizados dessa forma para o treinamento, o modelo não seria capaz de realizar uma análise correta e rotular adequadamente as frases.

Para resolver esse desequilíbrio, o autor aplica a técnica de *oversampling* (sobreamostragem, em tradução livre) para replicar os registros da classe minoritária (tuítes 'de ódio') até que ambas as categorias — 'neutros' e de 'ódio' — possuam a mesma quantidade.

O código utilizado para aplicar essa transformação é apresentado na figura 4 e executa, linha a linha, as seguintes tarefas:

1 *from sklearn.utils import resample*: importa exclusivamente a função *resample* (reamostragem, em tradução livre), que pertence ao módulo *utils*, da biblioteca scikit-learn, e será responsável pela replicação dos dados minoritários;

2 *train_majority = train_clean[train_clean.label==0]*: cria o *dataframe* 'train_majority' para armazenar os tuítes 'neutros' ('label' = 0) presentes no *dataframe* 'train_clean' criado anteriormente;

3 *train_minority = train_clean[train_clean.label==1]*: mesma lógica da linha anterior, mas o *dataframe* 'train_minority' criado vai armazenar os tuítes 'de ódio' ('label' = 1);

4-7 *train_minority_upsampled = resample(train_minority, replace=True, n_samples=len(train_majority), random_state=123)*: cria o novo *dataframe* 'train_minority_upsampled' que executa a função *resample*. A sintaxe inclui: 'train_minority' (o conjunto de onde serão sorteados os tuítes 'de ódio' a serem multiplicados), '*replace=true'* (permite que um mesmo tuíte 'de ódio' seja sorteado e replicado mais de uma vez, o que é essencial para ampliar a amostra), '*n_samples=len(train_majority)'* (define que o número final de amostras para 'train_minority' deve ser igual ao de amostras de 'train_majority', garantindo que sejam equiparados) e *'random_state=123'* (número arbitrário para garantir que a mesma sequência de sorteio se repita a cada vez que o código for executado)[5];

8 *train_upsampled = pd.concat([train_minority_upsampled, train_majority])*: cria o *dataframe* 'train_upsampled' que vai armazenar as linhas do 'train_majority' com as do 'train_minority_upsampled' (contendo as amostras recém-criadas); isso é feito por meio de concatenação vertical[6] (função *pd.concat*), ou seja, do empilhamento de tais linhas;

[5] scikit-learn developers — *resample*, documentação oficial.
[6] Hey Amit — Understanding pandas Vertical Concatenation.

9 *train_upsampled['label'].value_counts()*: conta quantos tuítes 'neutros' e 'de ódio' ('label' = 0 ou 1) há no conjunto[7], permitindo confirmar que agora ambas as classes possuem a mesma quantidade.

Nesse processo de *oversampling*, vale ressaltar que os tuítes 'de ódio' são replicados até que seu total se iguale ao montante já existente de tuítes 'neutros' — sem que esses sejam substituídos. É o que se depreende da sintaxe utilizada, que estabelece o total de tuítes majoritários como quantidade limite para a reamostragem.

Figura 4 – Balanceamento de registros com *oversampling*.

```
1 from sklearn.utils import resample
2 train_majority = train_clean[train_clean.label==0]
3 train_minority = train_clean[train_clean.label==1]
4 train_minority_upsampled = resample(train_minority,
5                                     replace=True,
6                                     n_samples=len(train_majority),
7                                     random_state=123)
8 train_upsampled = pd.concat([train_minority_upsampled,
  train_majority])
9 train_upsampled['label'].value_counts()
```

Fonte: Elaborado pelo autor.

O código foi executado e retornou o código presente na figura 5, a saber, o mesmo resultado do projeto original.

Figura 5 – Resultado da execução do código na figura 4.

```
label   count
1       29720
2       29720
```

Fonte: Elaborado pelo autor.

3.5.4 Criação do pipeline

A próxima fase do projeto foi a criação do *pipeline*: uma estrutura que encadeia diferentes etapas de processamento, desde a transformação dos dados até o treinamento do modelo. Esse arranjo funciona de fato como um tubo (tradução literal de *pipeline*), onde os dados — no caso, os tuítes — entram como texto cru e saem classificados[8] como 'neutros' ou de 'ódio'. KHARWAL (2020) explica que cria o *pipeline* antes de treinar o modelo, o que é essencial, já que essa estrutura não apenas contém, mas automatiza o fluxo completo de transformação dos dados para que o modelo seja treinado e assim aprenda a classificar os tuítes. O código para criar o *pipeline* é apresentado na figura 6 e engloba as seguintes ações, em suas respectivas linhas:

1 *from sklearn.feature_extraction.text import TfidfVectoriz*er: importa a função *TfidfVectorizer*, que atribui um número a cada palavra de acordo com sua frequência no conjunto de dados, e assim define um peso, ou seja, indica a relevância da palavra: quanto mais comum, menor é a importância, e quanto mais rara, maior é. O *TfidfVectorizer*[9] executa automaticamente o que as funções *CountVectorizer* e *TfidfTransformer* fariam separadamente. Apesar de as três funções serem importadas, o autor opta por não usar o *TfidfVectorizer* e assim executar as tarefas separadamente, conforme se observa na figura 6;

[7] Pandas — *pandas.DataFrame.value_counts*, documentação oficial.
[8] scikit-learn developers — *Pipeline*, documentação oficial.
[9] scikit-learn developers — *TfidfVectorizer*, documentação oficial.

2 *from sklearn.pipeline import Pipeline*: importa a função *pipeline*, essencial para criar essa estrutura no modelo;

3 *from sklearn.feature_extraction.text import CountVectorizer*: importa a função *CountVectorizer*[10], que atribui um número a cada palavra de acordo com sua frequência no conjunto;

4 *from sklearn.feature_extraction.text import TfidfTransformer*: importa a função *TfidfTransformer*[11], que atribui um peso a cada palavra de acordo com a numeração atribuída pelo *CountVectorizer*;

5 *from sklearn.linear_model import SGDClassifier*: importa a classe *SGDClassifier*. SGD é relativo a *Stochastic Gradient Descent* (Descida Gradiente Estocástica, em tradução livre), um método para treinar modelos que pode usar classificação binária (0 e 1)[12], como no caso deste projeto;

6-9 *pipeline_sgd = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('nb', SGDClassifier()),])*: cria um *pipeline* chamado 'pipeline_sgd' com os processamentos a serem executados em cadeia. Cada etapa é definida por um par: um nome (escolhido pelo programador, como 'vect') e a função que será executada. Neste pipeline, o modelo primeiramente atribui um número às palavras de acordo com a frequência (por meio do *CountVectorizer*), depois atribui um peso de acordo com essa contagem (com o *TfidfTransformer*) e, por fim, com os dados transformados (a partir do *SGDClassifier*), aprende a distinguir entre tuítes 'neutros' e de 'ódio'.

Figura 6 – Criação do pipeline.

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.pipeline import Pipeline
3 from sklearn.feature_extraction.text import CountVectorizer
4 from sklearn.feature_extraction.text import TfidfTransformer
5 from sklearn.linear_model import SGDClassifier
6 pipeline_sgd = Pipeline([
7     ('vect', CountVectorizer()),
8     ('tfidf',  TfidfTransformer()),
9     ('nb', SGDClassifier()),])
```

Fonte: Elaborado pelo autor.

3.5.5 Treinamento e teste do modelo

A última etapa do projeto corresponde ao processo de treinamento e avaliação do modelo é dividido em duas fases: a separação do *dataframe* e a execução propriamente dita do treinamento e do teste. Essa fase é fundamental para verificar a efetividade da classificação aplicada aos tuítes.

A primeira fase, ilustrada na figura 7, utiliza uma função que divide o *dataframe* 'train_upsampled' — composto por tuítes já rotulados com 0 ('neutros') ou 1 ('de ódio') — em dois subconjuntos: um destinado ao aprendizado do modelo e outro à sua validação.

Uma analogia ajuda a compreender: imagine-se um aluno que precisa se preparar para uma prova de Língua Portuguesa em que deve classificar frases como certas ou erradas. Por dispor de várias frases já corrigidas, ele decide separar parte delas para estudar e o restante para simular uma prova e conferir o gabarito. Quanto mais acertos obtiver na simulação da prova, mais preparado estará para enfrentar novas questões, ainda desconhecidas.

[10] scikit-learn developers — *CountVectorizer*, documentação oficial.
[11] scikit-learn developers — *TfidfTransformer*, documentação oficial.
[12] James D. McCaffrey — Binary Classification Using the scikit SGDClassifier Class.

Durante a execução do projeto, surgiu a dúvida sobre por que não se utilizou o *dataframe* 'test_clean' para a etapa de testes. A razão é que ele não possui a coluna 'label', responsável por indicar se um tuíte é 'neutro' ou de 'ódio' — seria como o aluno fazer o simulado sem ter o gabarito disponível. Assim, o modelo precisa ser inicialmente avaliado com uma base rotulada, permitindo medir sua precisão. Só então os tuítes do conjunto 'test_clean', sem rótulo e com dados inéditos, podem ser analisados em um teste final para verificar o desempenho.

As linhas do código desta etapa — cuja sintaxe segue um padrão comum em modelos de *machine learning* — têm a seguinte função:

1 *from sklearn.model_selection import train_test_split*: importa a função *train_test_split* do módulo *model_selection* pertencente à biblioteca sklearn; é ela a responsável por dividir o conjunto de dados entre treino e teste[13];

2-5 *X_train, X_test, y_train, y_test = train_test_split(train_upsampled['tweet'], train_upsampled['label'],random_state = 0)*: nesta sintaxe, o mecanismo de divisão (*split*) recebe os tuítes (linha 3) e seus rótulos (0 ou 1). O *split* devolve quatro objetos (linha 2): como num plano cartesiano, os tuítes são posicionados como eixo X e os rótulos como eixo y (é comum em *pipelines* de *machine learning* que *features*, como o campo *tweet*, sejam chamados de X, e os rótulos, como o campo *label*, de y[14]). Os tuítes (X) de treino são embaralhados com os rótulos (y) de treino. E os tuítes (X) de teste são embaralhados com os rótulos (y) de teste. O parâmetro *random_state* recebe o valor de 0 (à escolha do programador) e vai determinar a lógica de embaralhamento dos dados antes da divisão — vale utilizar um valor específico para que o modelo funcione de forma padronizada.

Há parâmetros que o autor não utiliza: *test_size* e *train_size*, que determinam quanto do conjunto será usado para treinamento e quanto para teste, e *shuffle*, para escolher se os dados são embaralhados antes da divisão de train/test, o que evita que o modelo seja treinado erroneamente sob a influência da ordem dos dados. Como não foram determinados, os parâmetros recebem os valores padrão: 25% do conjunto é direcionado para teste, e *shuffle* é executado como *true*, determinando que os dados sejam embaralhados antes da divisão train/test.

Durante a execução do projeto, também se questionou a divisão train/test: no processo de *oversampling*, os tuítes da amostra foram replicados, ou seja, o conjunto ficou repleto de dados repetidos. Ao serem divididos, haveria uma chance de haver o mesmo tuíte tanto no grupo de treino quanto no de teste (erro chamado de *data leakage*[15]), o que poderia corromper o modelo. O ideal seria primeiramente fazer o *split*, e só depois aplicar o *oversampling* exclusivamente no grupo de treino.

Figura 7 – Divisão do conjunto de dados.

```
1  from sklearn.model_selection import train_test_split
2  X_train, X_test, y_train, y_test = train_test_split(
3                                          train_upsampled['tweet'],
4                                          train_upsampled['label'],
5                                          random_state = 0)
```

Fonte: Elaborado pelo autor.

[13] scikit-learn developers — *train_test_split*, documentação oficial.
[14] Chandra P. Shekhar — Decoding X and y in Machine Learning.
[15] scikit-learn developers — *Data leakage*, documentação oficial.

O último código do modelo, exposto na figura 8, serve para de fato treinar e testar o modelo, a partir do conjunto dividido na fase anterior, e representa a fase final do processo de ETL: o carregamento dos dados já tratados. Cada linha tem a seguinte função:

1 *model = pipeline_sgd.fit(X_train, y_train)*: cria-se o modelo 'model', cujo resultado vai ser o 'pipeline_sgd' ajustado (ou treinado) com base nos tuítes direcionados para treinamento. O método *.fit* é o responsável por esse processo de aprendizado[16]: ele analisa os tuítes contidos em 'X_train' e seus respectivos rótulos ('y_train'), encontrando padrões que permitam distinguir os tuítes 'neutros' dos 'de ódio';

2 *y_predict = model.predict(X_test)*: aplica ao modelo 'model' a função *.predict*, que utiliza o 'pipeline_sgd' ajustado para prever os rótulos dos tuítes do conjunto de teste ('X_test')[17]. Essas previsões são armazenadas na variável 'y_predict';

3 *from sklearn.metrics import f1_score*: importa a função *f1_score* do módulo *metrics* pertencente à biblioteca sklearn; sua tarefa é avaliar o desempenho de modelos de classificação;

4 *f1_score(y_test, y_predict)*: calcula o *F1 score*, comparando os rótulos reais dos tuítes de teste ('y_test') com os rótulos previstos ('y_predict'). O *F1 score* é uma métrica para modelos de classificação que combina dois atributos: precisão e revocação[18]. Tomando as classificações usadas neste modelo (tuítes 'neutros' e 'de ódio'), com a precisão avalia-se, entre os tuítes classificados como 'de ódio', quantos são realmente 'de ódio'. Já com a revocação, observa-se, dentre todos os tuítes 'de ódio' (originalmente rotulados como 1), quantos o modelo conseguiu identificar. A fórmula é:

$$F1 = 2 * \frac{(\text{Precisão} * \text{Revocação})}{(\text{Precisão} + \text{Revocação})}$$

O resultado pode variar de 0 a 1: quanto mais perto de 1, melhor é o desempenho do sistema desenvolvido. Portanto, o resultado de 0.96 — gerado na execução e que corresponde ao projeto original, conforme figura 9 — indica que o modelo tem uma boa performance, com 96% de assertividade.

Figura 8 – Treinamento e teste do modelo.

```
1 model = pipeline_sgd.fit(X_train, y_train)
2 y_predict = model.predict(X_test)
3 from sklearn.metrics import f1_score
4 f1_score(y_test, y_predict)
```

Fonte: Elaborado pelo autor.

Figura 9 – Resultado da execução do código na figura 8.

```
0.9694707372350353
```

Fonte: Elaborado pelo autor.

[16] GeeksforGeeks — What is fit() method in Python's Scikit-Learn?
[17] Jason Brownlee — How to Make Predictions with scikit-learn.
[18] Mario Filho — Precisão, Recall e F1 Score Em Machine Learning.

4. RESULTADOS E DISCUSSÃO

Todos os procedimentos e códigos disponibilizados por Aman Kharwal no projeto para detecção de discursos de ódio em tuítes foram executados com sucesso. Ao fim da replicação, no entanto, percebeu-se que não foram inclusas orientações para a utilização da base de dados 'test' — disponibilizada pelo autor no início das instruções, submetida a limpeza e, em tese, destinada a colocar em prática o modelo desenvolvido.

Além disso, ainda que a realização das etapas tenha ocorrido com sucesso, é válido afirmar que o projeto carece de explicações mais aprofundadas sobre o funcionamento dos códigos, bem como das linguagens, funções e classes contidas nos mesmos.

Essa escassez de elucidações, somada à não apresentação do método para aplicar o modelo de detecção à base 'test', permitem afirmar que o sistema em questão demanda um conhecimento considerável sobre machine learning, linguagem python e tecnologias de PLN.

Optou-se, então, por desenvolver um código que permitisse aplicar o modelo treinado aos tuítes listados na tabela 'test'. A partir da adaptação de um exemplo disponível na plataforma Stack Overflow, chegou-se às instruções apresentadas na figura 10, descritas a seguir:

1 *test_predictions = model.predict(test_clean['tweet'])*: gera-se o objeto 'test_predictions', cujo resultado será a aplicação do modelo treinado ('model.predict') à série (equivalente a coluna) 'tweet' do *dataframe* 'test_clean' — produzido, por sua vez, no início das instruções a partir da limpeza dos tuítes da tabela 'test';

2 *test_clean['predicted_label'] = test_predictions:* os resultados de 'test_predictions' são adicionados como uma nova coluna ao *dataframe* 'test_clean';

3 test_clean.to_csv('test_predicted.csv', index=False): o *dataframe* 'test_clean' é exportado em formato .csv e a tabela é disponibilizada nos arquivos do Google Colab.

Figura 10 – código para aplicação do modelo treinado à tabela 'test'.

```
1  test_predictions = model.predict(test_clean['tweet'])
2  test_clean['predicted_label'] = test_predictions
3  test_clean.to_csv('test_predicted.csv', index=False)
```

Fonte: Elaborado pelo autor.

O código foi executado com sucesso e a tabela 'test_clean' gerada foi baixada afim de verificar como os tuítes foram classificados. Para exemplo, a figura 11 apresenta os cinco primeiros rotulados como 'neutro' ['predicted_label' = 0], e a figura 12, os cinco primeiro detectados como 'de ódio' ['predicted_label' = 1].

Figura 11 – Exemplos de tuítes classificados como 'neutros'.

1	studiolife aislife requires passion dedication willpower to find newmaterials
2	safe ways to heal your acne altwaystoheal healthy healing
3	is the hp and the cursed child book up for reservations already if yes where if no when harrypotter pottermore favorite
4	3rd bihday to my amazing hilarious nephew eli ahmir uncle dave loves you and misses
5	choose to be momtips

Fonte: Elaborado pelo autor.

Figura 12 – Exemplos de tuítes classificados como 'de ódio'.

1	white supremacists want everyone to see the new birds movie and heres why
2	thought factory bbc neutrality on right wing fascism politics media blm brexit trump leadership gt3
3	chick gets fucked hottest naked lady
4	suppo the taiji fisherman no bullying no racism tweet4taiji thecove seashepherd
5	i say we because im speaking collectively ive always known 2016 showed a lot and

Fonte: Elaborado pelo autor.

Um olhar superficial sobre os tuítes classificados como 'de ódio' permite reconhecer a relação desse rótulo com expressões de cunho político, social e sexual como *"supremacists"* ("supremacistas", em tradução livre), *"fascism"* ("fascismo", em tradução livre), *"fucked hottest naked lady"* ("conteúdo sexual explícito", em tradução interpretativa) e *"bullying"*. Essas palavras estão nos quatro primeiros tuítes. No caso do quinto exemplo, entretanto, é mais difícil identificar qual termo teria motivado sua classificação como discurso odioso. Poderia ser *"speaking collectively"* ("falando coletivamente", em tradução livre), mas a carga semântica desse recorte — desconsiderando possíveis contextos específicos — é distante dos demais termos presentes nos quatro tuítes anteriores.

Considerando que esse tuíte pode ter sido classificado erroneamente, que a linguagem é subjetiva e que discursos de ódio podem estar nas entrelinhas de um texto, estaria o modelo apto a captar nuances? Ou, ao contrário do que pode ter ocorrido com o quinto tuíte, se sairia malsucedido com postagens em que a retórica odiosa não está evidente?

Nota-se também que os tuítes, após passarem pela limpeza, ficaram repletos de palavras confusas, já que o código removeu pontuações, *stopwords* e outros caracteres dispensáveis sem, no entanto, substituí-los por um espaço em branco. Esse resultado levantou mais uma dúvida: o modelo poderia falhar em classificar tuítes com discurso de ódio em razão de palavras-chave terem sido concatenadas formando expressões que não existem?

Reconhece-se que o projeto de Aman Kharwal pode ter caráter introdutório, com um modelo básico de PNL a ser aprimorado por usuários interessados pelo tema. Porém, entende-se que as indagações levantadas neste trabalho são pertinentes e poderiam subsidiar futuros estudos sobre o tema, bem como a elaboração de códigos mais robustos. Seria válido, assim, associar as funcionalidades das tecnologias utilizadas a noções mais amplas do campo da comunicação e da linguística.

5. CONCLUSÃO

A reprodução do projeto de detecção de discursos de ódio em tuítes permitiu compreender, na prática, o funcionamento de um modelo básico de PLN aplicado à classificação automática de textos. A execução das instruções publicadas por Aman Kharwal demonstrou que é possível construir um *pipeline* de aprendizado supervisionado utilizando ferramentas amplamente acessíveis, como Google Colab, Python, Pandas e Scikit-Learn, resultando em um modelo capaz de rotular novos dados com razoável eficiência.

Contudo, a experiência evidenciou alguns desafios relevantes. Embora todas as etapas tenham sido executadas com sucesso, a ausência de explicações mais detalhadas sobre o funcionamento de funções, métodos e classes utilizadas dificultou o entendimento completo do processo, especialmente para iniciantes. Soma-se a isso a inexistência, nas instruções originais, de uma orientação explícita sobre como aplicar o modelo treinado à base 'test', o que exigiu a adaptação de um código externo para realizar a classificação dos novos tuítes e exportar o resultado.

A análise qualitativa dos dados classificados revelou que, apesar de identificar expressões comumente associadas ao discurso de ódio, o modelo apresenta limitações quando o sentido ofensivo depende de contexto ou subjetividade. A dificuldade em determinar os critérios que levaram à classificação de determinados tuítes, bem como os problemas decorrentes da etapa de limpeza — que resultou em palavras concatenadas e perda de clareza semântica — reforçam que soluções automáticas desse tipo ainda enfrentam barreiras importantes na interpretação de linguagem humana.

Considera-se, portanto, que o projeto cumpre um papel introdutório significativo, oferecendo uma base útil para experimentação e estudo. No entanto, a reprodução realizada demonstra que, para alcançar resultados mais consistentes, seria necessário aprofundar aspectos técnicos, aprimorar etapas de pré-processamento e desenvolver abordagens capazes de capturar nuances discursivas. Espera-se que as reflexões aqui apresentadas contribuam para futuras investigações e inspirem avanços que aproximem tecnologias de PLN de discussões éticas e comunicacionais mais amplas, especialmente no enfrentamento responsável ao discurso de ódio nas redes sociais.

6. REFERÊNCIAS

AMIT, H. **Understanding pandas Vertical Concatenation**. Medium. Publicado em: 30 mar. 2025. Disponível em: <https://medium.com/@heyamit10/understanding-pandas-vertical-concatenation-f8671760f8fe>. Acesso em: 13 nov. 2025.

BROWNLEE, J. **How to Make Predictions with scikit-learn**. Publicado em: 10 jan. 2020. Disponível em: <https://machinelearningmastery.com/make-predictions-scikit-learn/>. Acesso em: 13 nov. 2025.

FILHO, M. **Precisão, Recall e F1 Score Em Machine Learning**. Publicado em 3 jan. 2023. Disponível em: <https://mariofilho.com/precisao-recall-e-f1-score-em-machine-learning/>. Acesso em: 13 nov. 2025.

GEEKSFORGEEKS. **Apply a function to each row or column in Dataframe using pandas.apply()**. Publicado em: 15 jul. 2025. Disponível em: <https://www.geeksforgeeks.org/python/apply-a-function-to-each-row-or-column-in-dataframe-using-pandas-apply/>. Acesso em: 13 nov. 2025.

GEEKSFORGEEKS. **What is fit() method in Python's Scikit-Learn?** Publicado em: 23 jul. 2025. Disponível em: <https://www.geeksforgeeks.org/machine-learning/fit-method-in-pythons-scikit-lear/>. Acesso em: 13 nov. 2025.

GONZÁLEZ SÁNCHEZ, M. I. **Research and analysis of hate and other emotions in social media**. Trabalho de Conclusão de Curso (Gradução em Engenharia Informática)

– Faculdade de Matemática da Universidade de Barcelona, 2022. Disponível em: <https://hdl.handle.net/2445/197252>. Acesso em: 11 set. 2025.

KHARWAL, A. Hate Speech Detection Model. AmanXai. Disponível em: <https://amanxai.com/2020/08/19/hate-speech-detection-model/>. Acesso em: 11 set. 2025.

MCCAFFREY, J. D. **Binary Classification Using the scikit SGDClassifier Class**. Publicado em: 29 mar. 2023. Disponível em: <https://jamesmccaffreyblog.com/2023/03/29/binary-classification-using-the-scikit-sgdclassifier-class/>. Acesso em: 13 nov. 2025.

MIT TECHNOLOGY REVIEW. **Data Literacy: a importância da alfabetização em dados em um mundo Big Data**. 24 out. 2022. Disponível em: <https://mittechreview.com.br/data-literacy-a-importancia-da-alfabetizacao-em-dados-em-um-mundo-big-data/>. Acesso em: 11 set. 2025.

NASCIMENTO, A. J. **Ingestão e processamento de dados textuais do Reddit: uma solução de qualidade e disponibilidade**. 2023. 12 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, Campina Grande, 2023. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/36709>. Acesso em: 11 set. 2025.

OLIVEIRA MESA, L. F. **Análisis de datos de periódicos digitales**. 2016. Trabalho de Conclusão de Curso (Graduação em Engenharia Informática) – Universidade Complutense de Madrid, Faculdade de Informática, Madrid, 2016. Disponível em: <https://docta.ucm.es/entities/publication/27f7c111-aa33-4c1a-b4f4-0aa750f38e3e>. Acesso em: 11 set. 2025.

PANDAS. **pandas.DataFrame.value_counts — pandas 2.3.3 documentation**. Disponível em: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html>. Acesso em: 13 nov. 2025.

PYTHON SOFTWARE FOUNDATION. **Built-in Types — Python 3.14.0 Documentation**. Seção str.lower. Disponível em: <https://docs.python.org/3/library/stdtypes.html#str.lower>. Acesso em: 13 nov. 2025.

PYTHON SOFTWARE FOUNDATION. **re — Regular expression operations — Python 3.14.0 documentation**. Seção re.sub. Disponível em: <https://docs.python.org/3/library/re.html#re.sub>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **11. Common pitfalls and recommended practices — scikit-learn 1.7.2 documentation**. Seção 11.2. Data leakage. Disponível em: <https://scikit-learn.org/stable/common_pitfalls.html#data-leakage>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **CountVectorizer — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-

learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **Pipeline — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **resample — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **TfidfTransformer — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **TfidfVectorizer — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html>. Acesso em: 13 nov. 2025.

SCIKIT-LEARN DEVELOPERS. **train_test_split — scikit-learn 1.7.2 documentation**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>. Acesso em: 13 nov. 2025.

SHEKHAR, C. **Decoding X and y in Machine Learning**. LinkedIn. Publicado em: 1º maio 2024. Disponível em: <https://www.linkedin.com/pulse/decoding-x-y-machine-learning-chandra-prakash-nmsjf/>. Acesso em: 13 nov. 2013.