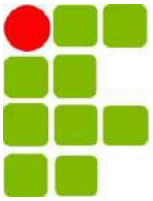


Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

Campus Campina Grande

Coordenação do Curso Superior de Bacharelado em Engenharia

de Computação

Um Estudo comparativo de algoritmos de Aprendizado de Máquina na detecção de discurso de ódio na Rede Social Twitter

Wesley Wevertton de Azevedo Palmeira

Orientador: Prof Dr. Marcelo José Siqueira Coutinho de Almeida.

Campina Grande, Novembro de 2022

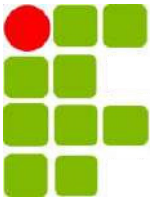
Wesley Wevertton de Azevedo Palmeira

P172e Palmeira, Wesley Wevertton de Azevedo.
Um estudo comparativo de algoritmos de aprendizado de máquina na detecção de discurso de ódio na rede social *Twitter*. - Campina Grande, 2022.
52 f. : il.

Trabalho de Conclusão de Curso (Curso de Graduação em Engenharia de Computação) - Instituto Federal da Paraíba, 2022.
Orientador: Prof. Dr. Marcelo José Siqueira Coutinho de Almeida.

1.Inteligência artificial 2. Discurso de ódio 3. Aprendizagem de máquina I. Almeida, Marcelo José Siqueira II. Título.

CDU 004



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

Campus Campina Grande

Coordenação do Curso Superior de Bacharelado em Engenharia

de Computação

Um Estudo comparativo de algoritmos de Aprendizado de Máquina na detecção de discurso de ódio na Rede Social Twitter

Wesley Wevertton de Azevedo Palmeira

Monografia apresentada à Coordenação do Curso Superior de Bacharelado em Engenharia de Computação do IFPB - Campus Campina Grande, como requisito parcial para conclusão do curso de Bacharelado em Engenharia de Computação.

Orientador: Prof Dr. Marcelo José Siqueira Coutinho de Almeida

Campina Grande, novembro de 2022

Um Estudo comparativo de algoritmos de Aprendizado de Máquina na detecção de discurso de ódio na Rede Social Twitter

Wesley Wevertton de Azevedo Palmeira

Marcelo José Siqueira Coutinho de Almeida

Orientador

Igor Barbosa da Costa

Membro da Banca

Elmano Ramalho Cavalcanti

Membro da Banca

Campina Grande, Paraíba, Brasil

novembro/2022

"Nunca se compare com ninguém neste mundo. Caso o faça, entenda que você estará insultando a si mesmo". (Bill Gates)

"Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar aonde a maioria não chega, faça o que a maioria não faz.". (Bill Gates)

"As máquinas me surpreendem muito frequentemente.". (Alan Turing).

Agradecimentos

Não poderia iniciar de outra forma a não ser agradecendo primeiramente a minha Mãe e meu Pai, ambos sempre presentes em todos os momentos da minha vida, acompanhando cada etapa vencida e me apoiando sempre em minhas decisões, cada um com seu jeito, mas, ambos com um propósito em comum de **me educar**, não importando a distância. Sempre tive o privilégio de ser motivado pelos meus pais, a prezar pela educação e sempre busquei dar o melhor de mim para isso, então nada mais justo do que começar agradecendo as pessoas que me colocaram no mundo e me guiaram para que eu pudesse ser uma pessoa melhor dia após dia. Também gostaria de agradecer a minhas avós que sempre me deram bastante suporte e me ensinam diariamente o significado da palavra amor, em especial a minha avó Materna Dona Eliete, que durante boa parte da minha trajetória dos estudos fez o papel de minha Mãe e me deu todo suporte para que eu conseguisse seguir com dos desafios diários, sem você nada disso seria possível, obrigado.

Camila minha esposa, a pessoa hoje que é minha base e está presente no meu dia a dia como suporte principal da nossa família, te agradeço demais por ser essa pessoa maravilhosa e contribuir tanto para que eu possa melhorar a cada dia, tenho orgulho demais de você e da mulher incrível que você é, me inspiro muito em suas atitudes a fim de me tornar uma pessoa melhor todos os dias. Todas as minhas conquistas só são possíveis porque tenho o seu suporte, te amo demais.

Amigos são essenciais, na minha vida sempre fui uma pessoa de poucos amigos, mas sempre tive o privilégio de ter amigos fiéis e que se provaram todos os dias que são verdadeiros parceiros para todos os tipos de momentos. Obrigado a todos em especial a Alessandro, Bê, Bode, Allan e Claudin, em vários momentos da minha vida vocês provaram serem verdadeiros parceiros.

O IFPB desde o primeiro dia que tive o contato se mostrou como uma instituição pública que abraça e que te faz sentir em uma segunda casa, cheguei de outro curso/instituição para cursar Engenharia de Computação e foi impactante, me senti realmente abraçado, agradeço demais a todos que fazem parte dessa instituição e principalmente ao meu orientador e amigo Marcelo, desde o início sempre se mostrou disposto a me ajudar em minhas pesquisas e sempre me apoiou em vários momentos me guiando sempre, a sua escolha como orientador, não vou em vão, pode ter certeza. Muito obrigado

Resumo

Com o aumento de usuários nas redes sociais surge a necessidade de criar regras de utilização, pois muitos se sentem livres para falarem o que quiserem e muitas vezes isso pode prejudicar outras pessoas. Uma das formas de lidar com essas situações é identificar e banir pessoas que disseminam ódio em seus comentários. Visto que, o volume de textos gerados é bastante elevado, utilizar de algoritmos que consigam identificar essa disseminação de ódio, é uma prática que pode ajudar bastante as redes sociais a banirem esse tipo de usuário. O presente trabalho se propôs a desenvolver e analisar algoritmos que sejam capazes de identificar discurso de ódio em tweets por meio do aprendizado de máquina e processamento de linguagem natural. Dentre os vários algoritmos de aprendizado de máquina disponíveis no mercado, foram escolhidos o Máquina Vetor de Suporte (SVM), Regressão Logística, Floresta Aleatória e Naive Bayes, ajustando os hiperparâmetros para o contexto do trabalho, para a escolha dos algoritmos, consideramos a facilidade de implementação, tempo de execução e familiarização com os mesmos.

Palavras-chave: Inteligência Artificial, Aprendizagem de Máquina, Discurso de ódio, Twitter.

Abstract

With the increase in users on social networks, the need to create usage rules arises, as many feel free to say whatever they want and this can often harm other people. One of the ways to deal with these situations is to identify and ban people who spread hate in their comments. Since the volume of texts generated is quite high, using algorithms that can identify this dissemination of hate is a practice that can help social networks a lot to ban this type of user. The present work proposes to develop and analyze algorithms that are able to identify hate speech in tweets through machine learning and natural language processing. Among the various machine learning algorithms available on the market, the Support Vector Machine (SVM), Logistic Regression, Random Forest and Naive Bayes were chosen, adjusting the hyperparameters to the context of the work, for the choice of algorithms, we considered the ease of implementation, runtime and familiarization with them.

Keywords: Artificial Intelligence, Machine Learning, Hate Speech, Twitter

Sumário

Lista de Abreviaturas

RSV - Redes Sociais Virtuais

DO - Discurso de Ódio

SL - Shallow learning

DL - Deep Learning

CNN - Convutional Neural Network

RNN - Recurrent Neural Network

API - Application Programming Interface

HSC - Hate Speech Classification

UI - User Interface

Lista de Figuras

Capítulo 1

Introdução

Neste capítulo trataremos sobre as considerações gerais sobre o tema apresentado bem como a problemática que estamos tentando resolver além da motivação do estudo para a resolução do problema e os objetivos que pretendemos atingir com o desenvolvimento deste trabalho.

1.1 - Considerações gerais

São notórios os benefícios que o desenvolvimento tecnológico trouxe para a humanidade, facilitando cada vez mais a vida de milhões de pessoas, no seu dia a dia e na sua convivência em sociedade. A criação das RSV, fez com que houvesse uma quebra de barreiras, possibilitando a comunicação de pessoas ao redor do globo e o convívio diário de geração e consumo de conteúdo educativo, entretenimento, profissional, como vários outros, acaba por influenciar a vida de milhões de pessoas.

Esses conteúdos, que trafegam diariamente aos dispositivos com acesso à Internet, são aproveitados das mais diversas maneiras, como por exemplo para aprender, ensinar, divertir, ganhar dinheiro, cozinhar, jogar, etc. Muitas pessoas, conhecidas como influenciadores digitais, vivem exclusivamente para gerar conteúdo para as RSV..

Muitos veem o conteúdo presente em uma RSV como uma verdade absoluta e é quando pessoas de má índole aproveitam esse fanatismo e fatos que possam estar acontecendo (como crises econômicas e/ou guerras) para propagar mensagens odiosas contra grupos de pessoas. Essa propagação do DO acaba por incitar frustrações, raiva e medo em parte da população, gerando assim o ódio direcionado ao grupo alvo das pessoas com más intenções.

1.2 - Apresentação do problema

O ódio direcionado a etnias, religiões e grupos de pessoas não é um assunto da atualidade[2]. Tal ódio provocou danos irreparáveis à humanidade, como as milhões de mortes causadas pelas inquisições, as colonizações das Américas e da África, o genocídio armênio, o Holocausto, como vários outros que a História conta. Porém, com o mundo globalizado e criação das RSV, a proporção do alcance que esse ódio tomou, dificilmente pode ser calculada.

Nos ambientes das RSV, pessoas más intencionadas aproveitam do poder de divulgação e do anonimato para propagar o DO para degradar, denegrir e agredir grupos como judeus, negros, LGBTQIA+, como vários outros grupos. Com a criação de perfis falsos e de robôs (bots), o poder de divulgação acaba aumentando cada vez mais, criando assim uma grande quantidade de dados que necessitam ser excluídos desses ambientes que influenciam a vida de muitas pessoas.

Para evitar que essas mensagens odiosas se iniciem ou que tomem proporções que possam causar violências a pessoas ou a grupos é necessário investigar as técnicas e ferramentas utilizadas pelas RSV e avaliá-las para verificar sua eficiência.

1.3 - Motivação da solução proposta

Nas últimas décadas, com a popularização da Internet como principal forma de comunicação entre as pessoas nos seus mais diversos ambientes pessoais e produtivos, inúmeras Redes Sociais surgiram como forma de propiciar a criação de ambientes coletivos. Alguns exemplos atualmente existentes dessas redes são Instagram, Twitter e Facebook. Nesses ambientes usuários passaram a publicar suas experiências, opiniões e sentimentos sobre os mais diversos assuntos de forma bastante livre, sem nenhuma moderação. No entanto, muitas dessas publicações podem prejudicar vários usuários mesmo que de maneira virtual.

Com o crescimento das RSV, preocupações com políticas de utilização começaram a surgir a fim de construir um ambiente mais agradável para todos que o compõem. As principais Redes Sociais mundiais chegaram a um acordo quanto a um código de conduta da União Europeia obrigando-se a avaliar a maioria das notificações válidas para a remoção de DO ilegal que seja postado em seus serviços, num prazo de 24 horas[2].

Apesar da proibição da propagação de DO em redes sociais sua ocorrência tem crescido nos últimos anos, como podemos observar, em uma reportagem do G1 [3] que mostra que o Facebook divulgou que identificou 269 milhões de conteúdos com discurso de ódio no quarto trimestre de 2020. Além disso, podemos também citar o famoso caso de uma universitária

paulista que em 2010 publicou um discurso de ódio contra nordestinos utilizando a rede social Twitter [4].

Segundo a professora da FGV Direito SP Clarissa Gross, em uma entrevista para o Nexo Jornal [5], podemos caracterizar um DO, como um discurso que traz consigo três características principais:

1) Possui conteúdo discriminatório;
2) Se dirige a um grupo geral de pessoas, e não a uma pessoa específica;
3) É um discurso proferido no debate público de ideias.

Em termos de classificação quanto à sua manifestação, o DO pode ser de vários tipos, seja ele geográfico, gênero, ideológico, raça, etnia, etc. Um estudo feito pelo site Gênero Número [8] em 2018, traz uma imagem com as principais denúncias pelo disque 100 e por meio de Boletins de Ocorrência na polícia civil, comparando os números de crimes de ódio classificados por tipos.

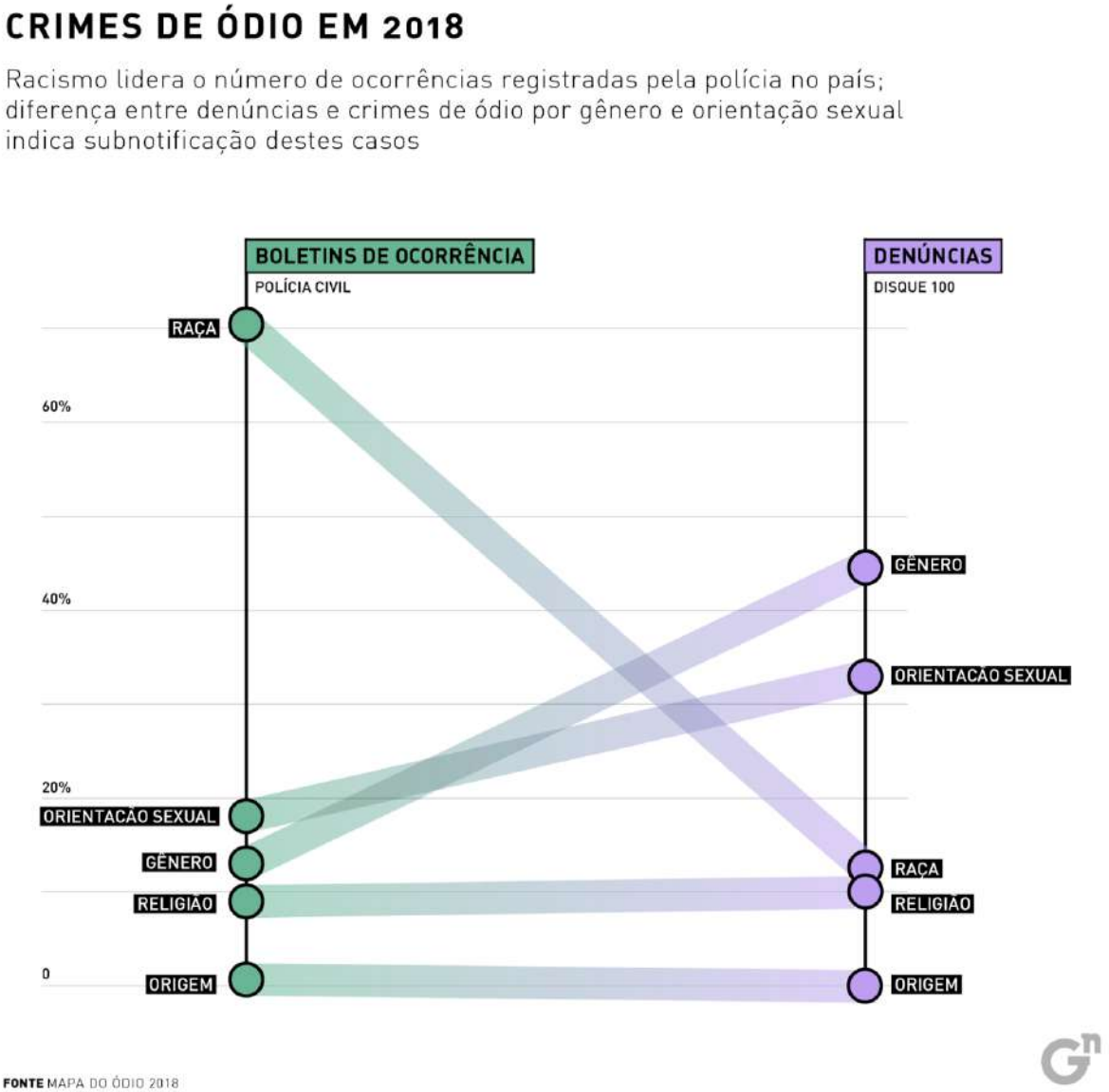


Figura 1 - Crimes de ódio em 2018. Fonte: Gênero e número[8]

DO pode se manifestar em uma RSV de diversas maneiras, a depender dos recursos disponíveis na ferramenta que estamos analisando. Para o escopo deste trabalho utilizaremos como ferramenta de análise e aquisição de dados a RSV twitter.

O DO tem a característica de não ser inócuo, porém pode gerar danos físicos e psicológicos naqueles que sofrem a agressão, como por exemplo, depressão, pressão alta, respiração acelerada e insônia [6]. Podemos destacar que naqueles que em que chega a causar uma depressão grave, pode também levar ao suícidio e/ou atitude de terrosrismo/genocídio seguida de suicídio. Partindo do pressuposto da causa de um terrorismo/genocídio é facil notar as diversas outras consequências que podem se geradas, seja ela de cunho econômico, político e/ou social.

Como já citado no trabalho, as redes sociais proíbem postagens que contenham discursos de ódio, e com isso, caso seja identificada alguma postagem que tenha esse tipo de conteúdo, o banimento total ou parcial da conta do usuário, por parte da RSV, tem sido uma política interessante para diminuir a propagação de DO, porém dependendo do discurso em si, pessoas que se sintam prejudicadas podem além de denunciar a postagem, como também, caso julgue necessário, buscar seus direitos constitucionais [7].

1.4 - Objetivos Geral e Específicos

1.4.1 - Objetivo Geral

Analisar o desempenho de alguns algoritmos de aprendizado de máquina supervisionado por classificação na identificação de discurso de ódio na rede social virtual Twitter.

1.4.2 - Objetivos Específicos

1) Coletar, armazenar e pré-processar tweets a partir da API fornecida pelo Twitter;
2) Encontrar um corpus contendo textos de discurso de ódio previamente classificados para utilizarmos como base de treino dos algoritmos;
3) Construir um dashboard com a implementação e análise de alguns algoritmos de aprendizado de máquina aplicados na identificação do discurso de ódio em tweets.

Capítulo 2

Fundamentação Teórica

Neste Capítulo trataremos de toda a parte teórica que servirá de base para o desenvolvimento do trabalho. Começaremos fazendo uma revisão bibliográfica, depois apresentaremos o estado da arte do tema, além de alguns conceitos básicos necessários para o entendimento do trabalho.

2.1 - Revisão Bibliográfica

Antes da criação de mecanismos de detecção automática, era necessário que os usuários verificassem os posts que supostamente violavam as regras da política da RSV e denunciassem[16], posteriormente, funcionários iriam verificar e avaliar o post, se realmente violassem suas políticas seriam retiradas.

Com a grande quantidade de plataformas de redes digitais e de usuários, houve a necessidade da criação de soluções automatizadas para resolverem os problemas relacionados ao DO[9], como por exemplo a empresa Facebook(atualmente o Meta) que em sua plataforma necessita retirar semanalmente por volta de 66.000 de posts que violam suas políticas[10].

Técnicas de aprendizado de máquina (mais recentemente aprendizado profundo) são de extrema importância para análise e detecção de DO, como descrito em (Alshalan, Raghad, et al. 2020) cujo artigo explica a análise de Tweets com a utilização de modelos CNN (Cable News Network) pré-treinadas, onde cada tweet foi dado uma pontuação entre 0 e 1, com 1 sendo os textos mais odiosos. O estudo mostra que 547.554 tweets foram analisados e 11.743 foram classificados como DO, porém o estudo foi realizado em uma área do globo e sobre um único assunto da pandemia da COVID-19. Se a área e os assuntos forem ampliados a detecção será mais complicada.

Em (BAYDOGAN, Cem et al. 2022), o estudo foi realizado com várias aproximações baseadas em SL (Shallow Learning) e DL (Deep Learning), a cada aproximação são utilizados vários algoritmos do estado da arte e utilizados em base de dados diferentes onde foram

analisados por diferentes critérios e avaliados. Viu que as aproximações DL foram muito melhores que as SL e os algoritmos com maior acurácia foram os RNN(78%) e CNN(76%). Isso mostra que com a utilização de vários algoritmos pode-se obter quais são os mais eficazes na detecção de DO e mesmo que haja um escalonamento dos dados, a acurácia pode se manter a mesma ou perder um pouco seu valor, porém se mantendo a um nível satisfatório.

Com essa preocupação em tornar as RSV um ambiente agradável, as próprias plataformas (twitter, facebook e instagram), tem subsistemas com algoritmos capazes de detectarem postagens e comentários que infrinjam suas políticas, porém, a precisão dos mesmos não é ideal, muitas vezes deixando passar comentários e postagem que atingem alguém negativamente de alguma forma, como também, os dados dessas coletas não são apresentados publicamente, são privados para a própria empresa que fornece a RSV.

Temos algumas ferramentas que realizam esta análise, como por exemplo a HateLab [12] que teve uma parceria recente com o Twitter [13]. Como também temos ferramentas OpenSource mais amadoras como por exemplo o “Twitter Hate Speech Detection” [14].

Também temos alguns trabalhos publicados (recentes) que abordam o tema [15][16], porém percebemos aqui que não conseguimos decidir qual a melhor abordagem para realizar essa identificação e qualificação, bem como, boa parte (maioria) das aplicações existentes e trabalhos publicados não abordam a língua portuguesa, por uma série de questões a respeito da dificuldade de análise de discurso usando a mesma.

2.2 - Processamento de Linguagem Natural

O processamento de linguagem natural é uma vertente da inteligência artificial que trata sobre a capacidade de entender, interpretar e manipular a linguagem humana por meio da computação. Podemos destacar várias aplicações do PLN como por exemplo a classificação de sentimentos, detecção de *fake news* e identificação de discurso de ódio (tema do nosso trabalho).

Várias técnicas da aprendizagem de máquina são utilizadas quando queremos construir uma aplicação que tenha processamento de linguagem natural, por isso a mesma se define como uma área complexa que necessita de um estudo aprofundado para conseguir aplicá-la.

A escolha da língua que será "processada" influencia diretamente na complexidade do desenvolvimento da aplicação que utilizará PLN, isso porquê, línguas que contenham muitos tempos verbais, vocabulário extenso e vários homônimos, parônimos e sinônimos (português por exemplo) tornam o reconhecimento de padrões mais complexo, logo para a uma máquina ou

algoritmo ser capaz de interpretar determinado texto/expressão, é necessário o desenvolvimento de um algoritmo mais robusto e que tenha um treinamento mais complexo.

2.3 - Mineração de dados

A mineração de dados é o processo de descoberta de informações em grande volumes de dados com o objetivo de gerar conhecimento. Esse conhecimento é gerado por meio de análise matemática para derivar padrões e tendências que existem nos dados. Em geral, esses padrões não são facilmente observáveis pelo fato das relações serem muito complexas ou por existir um volume muito grande de dados.

Um processo necessário para entendermos a mineração de dados é definir o conceito de **dado**, **informação** e **conhecimento**. O dado propriamente dito, se define como um texto não estruturado coletado de alguma fonte, como por exemplo: "Hoje é segunda-feira, 30 C, dia de jogo do Brasil contra Argetina". Percebemos que é uma frase na qual não estruturamos nada ainda, porém a partir dela eu consigo extrair **informações** como por exemplo: "Dia da semana = segunda-feira, Temperatura = 30 C, Evento = Brasil vs Argentina", é importante enxergarmos que a partir de um **dado** conseguimos obter algumas informações de Dia da semana, temperatura e evento que ocorrerá no dia, já o **conhecimento**, pode ser obtido a partir das informações existentes, como por exemplo: "Hoje teremos mais movimento no bar devido ao jogo do brasil, bom aumentar o estoque". Observamos então que, a partir da informação do evento, conseguimos obter o conhecimento que devemos aumentar o estoque.

2.4 - Aprendizado de máquina

Podemos definir aprendizado como o ato de descobrir novos fatos por meio de observação e/ou experiência, melhorar desempenho ao realizar determinadas tarefas e ainda organizar novo conhecimento por meio de representações efetivas e gerais.

"O aprendizado de máquina é um método de análise de dados que automatiza a construção de modelos analíticos. É um ramo da inteligência artificial baseado na ideia de que sistemas podem aprender com dados, identificar padrões e tomar decisões com o mínimo de intervenção humana" SAS[18]. Percebemos então que a máquina precisa dos dados corretos para identificar padrões e assim aprender. Na Figura 2, temos um exemplo comparativo de como seria

o aprendizado tradicional e como seria um aprendizado de máquina, a fim de que fique mais claro a diferença.

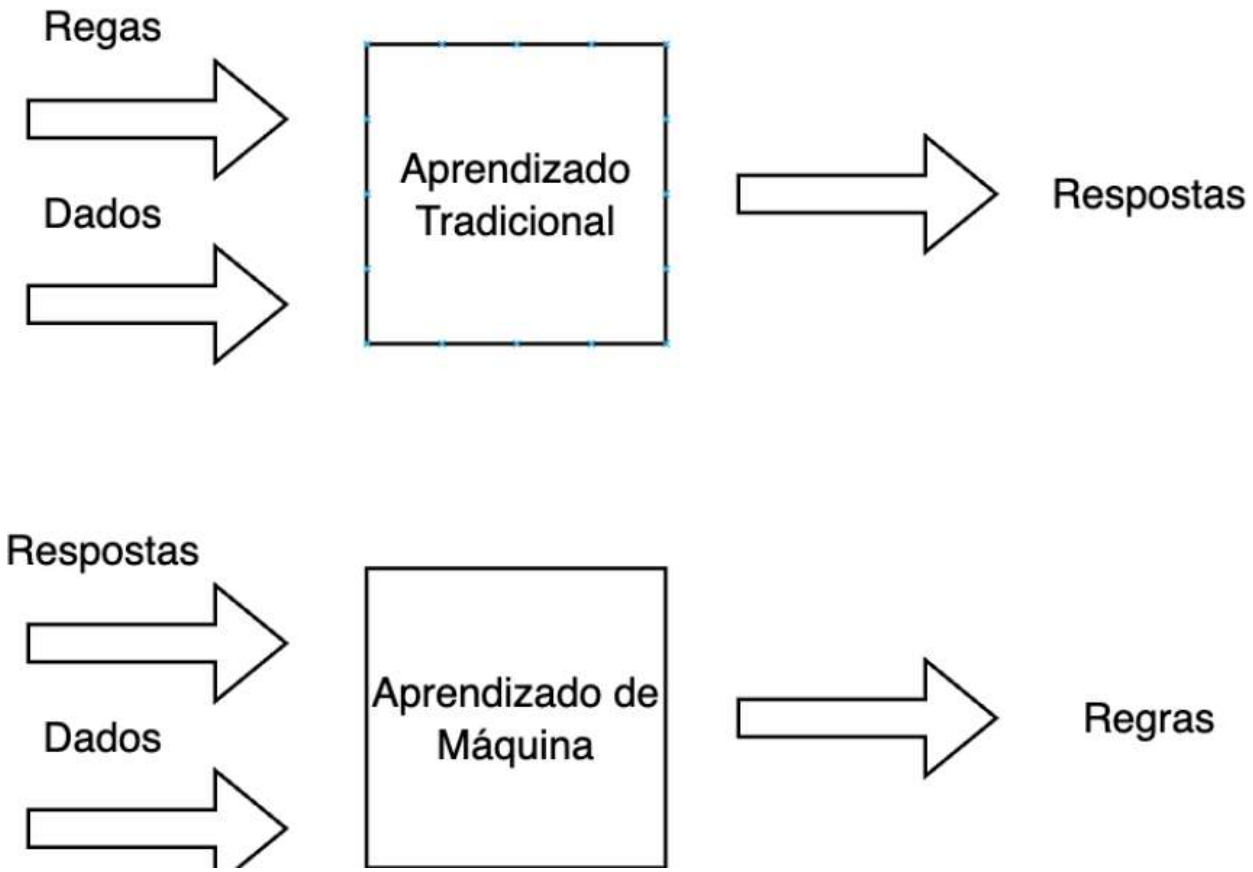


Figura 2 - Aprendizado tradicional vs Aprendizado de máquina

O Aprendizado de máquina é dividido em três grandes tipos [14], de acordo com a Figura 3. No **aprendizado supervisionado**, o algoritmo aprende com dados usados para treinamento com os quais já se sabe a solução, nesse aprendizado usamos uma base de dados que foi previamente classificada um bom exemplo seria nosso trabalho, a fim de encontrar tweets que se classificam ou não como discurso de ódio, treinamos os algoritmos anteriormente com uma base de dados pré-existente, já no **aprendizado não supervisionado**, o algoritmo não possui a necessidade de ter essa base de dados classificada anteriormente, nele busca-se encontrar o padrão entre as diferentes amostras e separar os que possuem as mesmas características, um bom exemplo seria os algoritmos de recomendação da Netflix, que trazem agrupamentos de clientes a partir de suas experiências anteriores. Temos ainda o **aprendizado por reforço** que se baseia no recebimento de recompensas ou penalidades a partir de uma decisão, o objetivo do algoritmo nesse caso, é tomar as melhores decisões para obter o maior número de recompensas possíveis, esse tipo de aprendizado é geralmente usado em robótica e em jogos, um exemplo claro é a aplicação do AlphaGo[19].

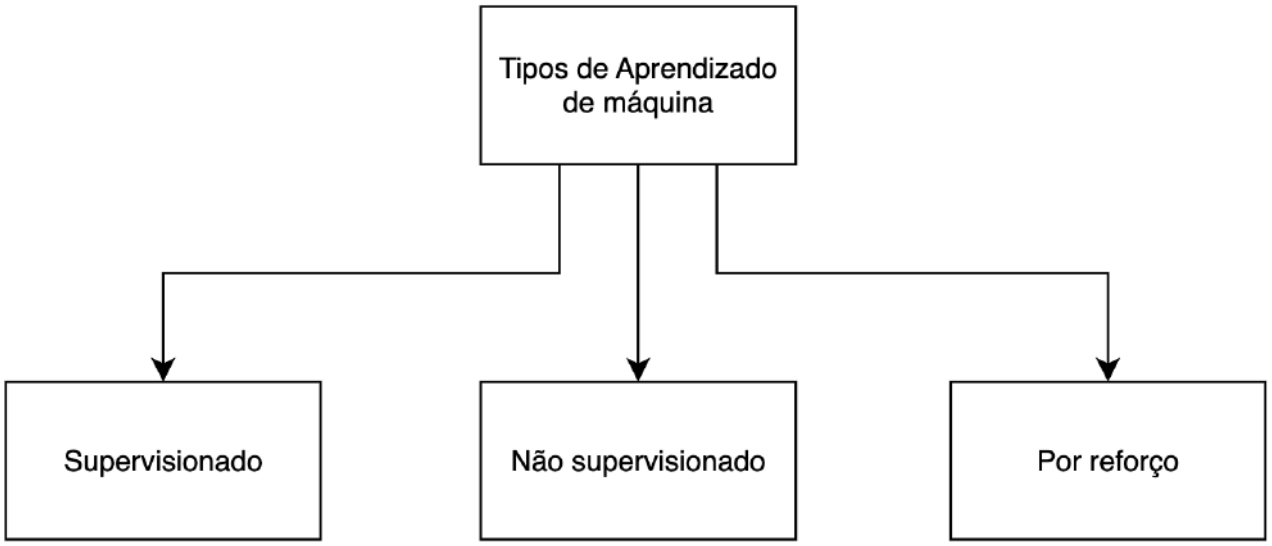


Figura 3 - Tipos de aprendizado de máquina

2.5 - Algoritmos de aprendizado supervisionado

O aprendizado supervisionado se divide em dois grandes grupos de acordo com a forma que se dará a implementação do algoritmo, na Figura 4, temos os detalhes dessa divisão e quais algoritmos podem ser utilizados em cada tipo de aprendizado. É importante notar que o mesmo algoritmo pode ser utilizado em um aprendizado por **classificação** e **regressão**, tudo depende da sua implementação.

No aprendizado supervisionado por **classificação** o algoritmo é treinado para classificar os dados de entrada em variáveis discretas, como por exemplo, um conjunto de dados que contém apenas um rótulo indicando se a foto é ou não de um cachorro, temos aí a classificação discreta "sim"ou "não".

Em contraste com a classificação, no aprendizado por **regressão**, o algoritmo é treinado para prever uma saída a partir de uma faixa contínua de valores possíveis. Por exemplo, um algoritmo capaz de prever o preço de um imóvel, nesse caso o valor não é mais discreto e sim contínuo.

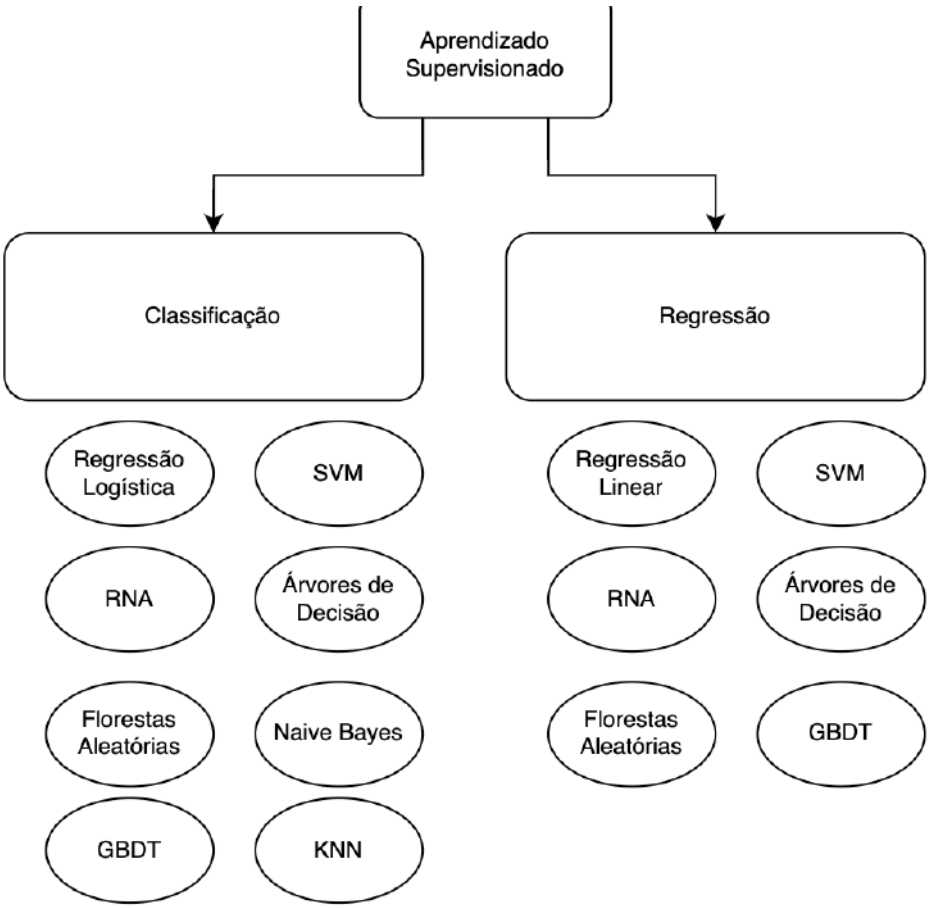


Figura 4 - Aprendizado supervisionado

Capítulo 3

Metodologia

Neste Capítulo será apresentada a metodologia utilizada para desenvolver o trabalho proposto.

Para o planejamento de uma aplicação que atenda aos objetivos propostos, foram necessárias a criação de uma arquitetura básica e utilização de tecnologias e ferramentas que facilitem a criação da plataforma.

O foco para a criação da arquitetura foi a coleta de dados do Twitter, através de API 's, analisá-los com algoritmos de aprendizado de máquina, já treinados com tweets que contenham o DO, obter os resultados e processar para que sejam demonstrados ao usuário. Na Figura 5 contém informações visuais sobre a arquitetura macro da aplicação. Ao longo do capítulo detalharemos cada parte dessa aplicação demonstrando como foi feita a implementação de cada etapa a fim de se obter os resultados esperados.

Foi escolhida a linguagem de programação Python para desenvolvimento do trabalho, pois a mesma tem várias bibliotecas de inteligência artificial, tratamento e visualização de dados que agilizam o desenvolvimento, tendo em vista que o desenvolvedor passa a ser um usuário, tendo a responsabilidade apenas de entender como usar o que já está implementado nas bibliotecas.

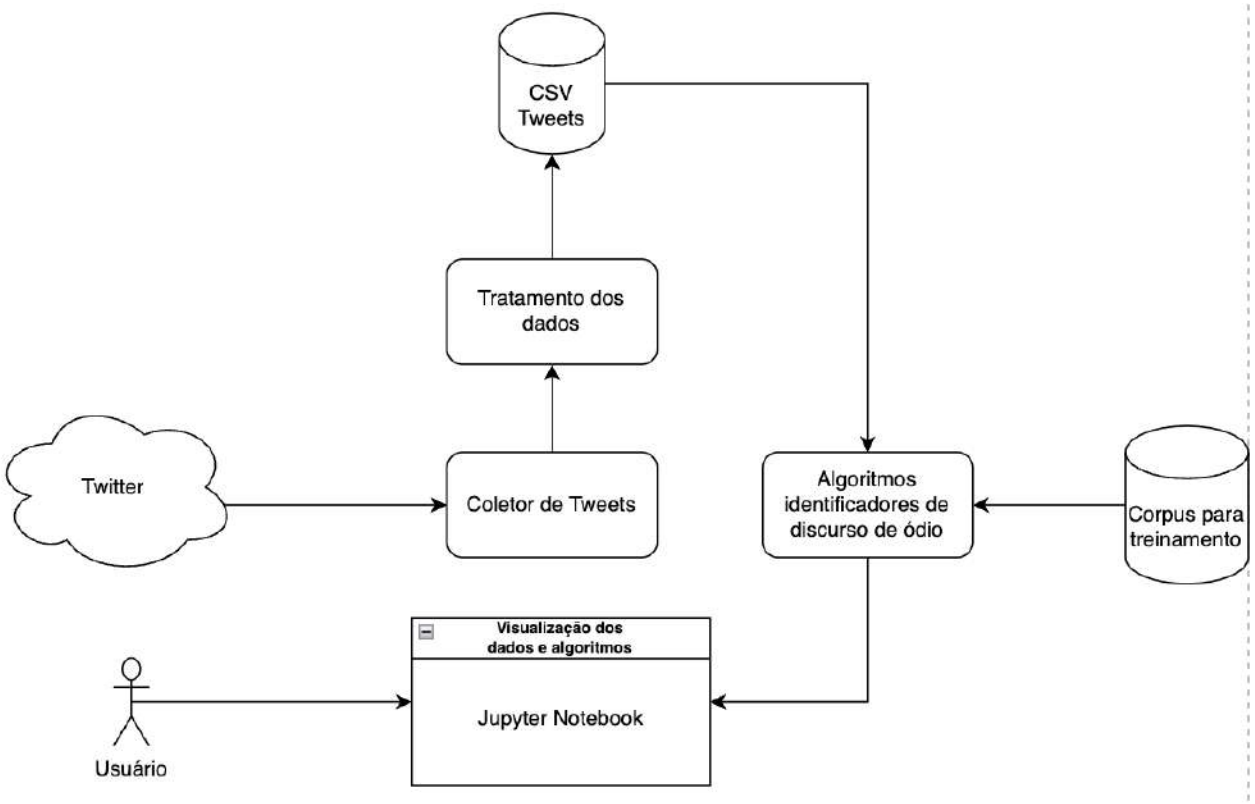


Figura 5 - Arquitetura da aplicação

3.1 - API do Twitter

O Twitter disponibiliza uma plataforma para desenvolvedores [21] na qual podemos via API RESTful[17], coletar várias informações da plataforma de acordo com um determinado limite e com o tipo de projeto. Existem três tipos de projeto que se pode criar: "Essential", "Elevated" e "Academic Research". O Primeiro tem aprovação quase que instantânea, já os outros dois exigem uma análise por parte da equipe do twitter para que você possa obter acesso.

Para se conectar à API do twitter existem diversas ferramentas e maneiras diferentes. A escolhida neste trabalho foi a biblioteca Tweepy[20], que tem uma implementação bastante simples e uma interface simplificada, a fim de facilitar para o programador no momento do desenvolvimento.

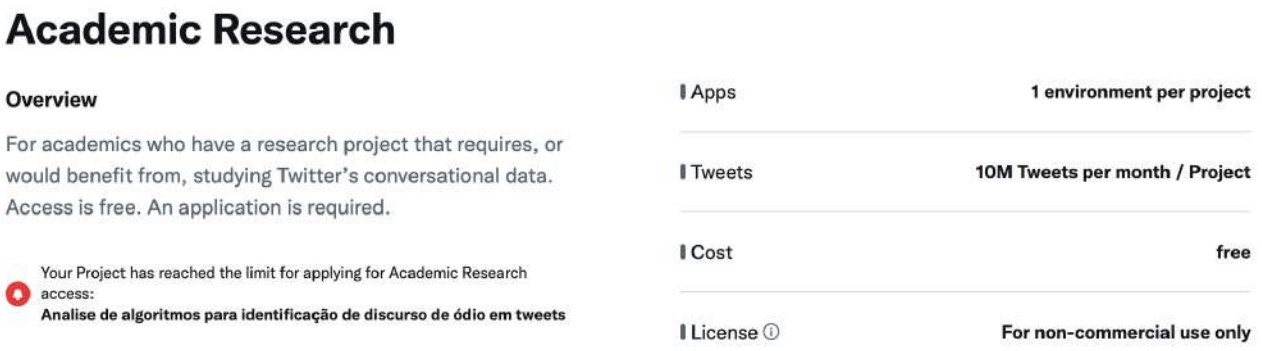


Figura 6 - Twitter Academic Research License

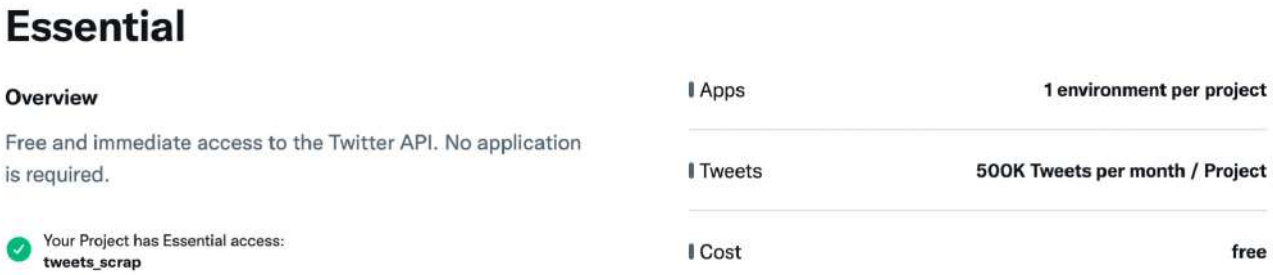


Figura 7 - Twitter Essential License

Para autenticação como conta de desenvolvedor na aplicação é exigido uma série de chaves que são obtidas ao criar a conta no twitter. Para isso criamos um arquivo contendo essas informações como mostra a Figura 9 e, criamos um método em python utilizando o auxílio do Tweepy para criar um cliente autenticado capaz de realizar requisições na API do twitter, assim como demonstrado na Figura 10.

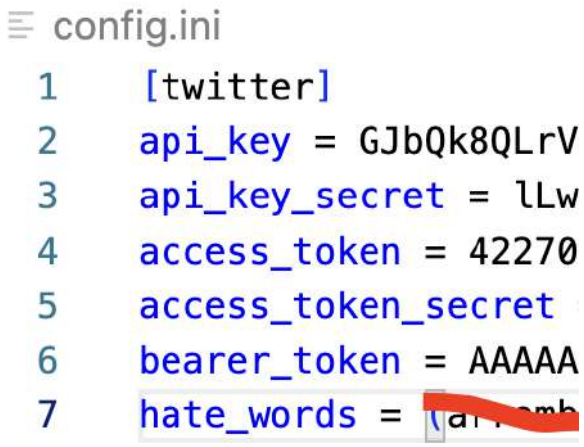


Figura 9 - Arquivo de configuração do coletor

```python
import configparser
import tweepy

def authenticate():

    # read config
    config = configparser.ConfigParser(interpolation=None)
    config.read('config.ini')

    access_token = config['twitter']['access_token']
    access_token_secret = config['twitter']['access_token_secret']
    bearer_token = config['twitter']['bearer_token']

    # authenticate
    client = tweepy.Client(bearer_token=bearer_token,
                           access_token=access_token,
                           access_token_secret=access_token_secret)
```

Figura 10 - Autenticação do coletor

3.2 - Coletor de tweets

Para a coleta dos tweets foi criado um conjunto de "palavras chave de ódio", disponíveis no arquivo de configuração como mostra a Figura 11, com base em um levantamento das palavras mais utilizadas nos tweets da base de treinamento [22]. Para a busca dos dados foi utilizada a API de desenvolvimento do twitter em conjunto com o tweepy. Para buscar um *tweet* na base de dados do twitter, existem diversas maneiras e estratégias, seja por palavras chave, data, região dentre outras. Para o escopo deste trabalho o tweet coletado contém quatro características:

1) Língua portuguesa;
2) Não é um retweet;
3) Não é uma resposta;
4) Contém uma das "palavras chave de ódio".

A implementação dessa coleta pode ser conferida na Figura 12, é importante mostrar que o twitter limita o consumo de tweets em no máximo 100 por vez, então, para atingir a meta mensal de coleta (a fim de se obter um grande volume de dados), esse script é executado várias vezes até que o limite seja atingido.

config.ini

```
1    [twitter]
2    api_key = GJbQk8QLrVD
3    api_key_secret = lLwG
4    access_token = 422708
5    access_token_secret =
6    bearer_token = AAAAAA
7    hate_words = (arromba
```

Figura 11 - Palavras chave de ódio

collector.py > ...

```
1    from auth import *
2    from normalizer import *
3    from tokenizer import create_tokens
4    import configparser
5    import pandas as pd
6
7    client = authenticate()
8    config = configparser.ConfigParser(interpolation=None)
9    config.read('config.ini')
0
1    query = config['twitter']['hate_words'] + ' lang:pt -is:retweet -is:reply
2    tweets = client.search_recent_tweets(query=query, max_results=100)
```

Figura 12 - Coletor de tweets

3.3 - Tratamento dos dados

Para facilitar na convergência dos algoritmos e facilitar o reconhecimento de padrões é necessário que os dados sejam previamente tratados e pré-processados antes de serem enviados para análise dos algoritmos de classificação. Este tratamento é divido em duas principais etapas: Tokenização e Normalização. Somente após essas duas etapas é que o nosso tweet é escrito em uma base de dados para ser analisado em seguida. Podemos conferir essa etapa na Figura 13.

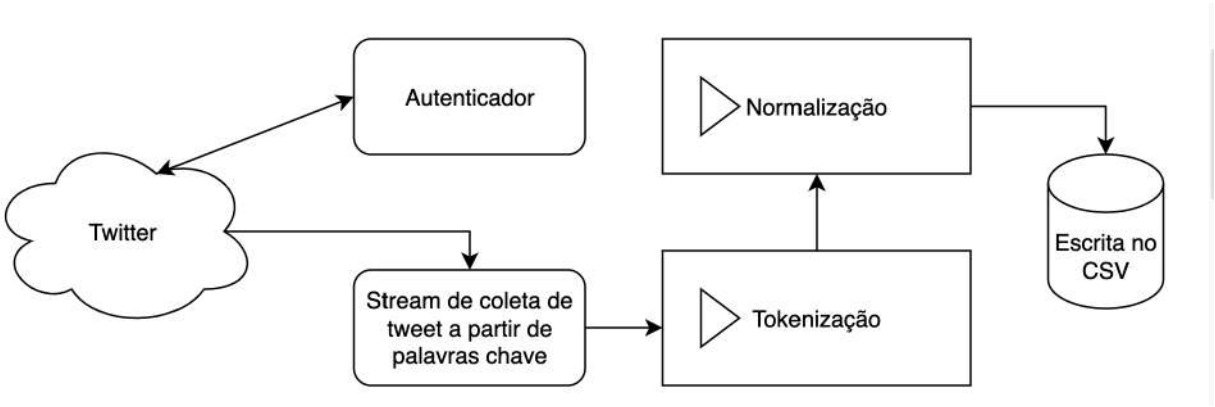


Figura 13 - Coleta e Tratamento dos Tweets

3.3.1 - Tokenização

A tokenização é o processo de dividir um texto em palavras ou tokens individuais. Durante o processo de tokenização, são removidos itens que não trazem significado para o texto, como por exemplo pontuação e caracteres especiais. O objetivo da tokenização é simplificar os dados preparando-os para a etapa de **normalização**[49].

Para implementação da tokenização utilizamos o auxílio da biblioteca NLTK[23] que já tem um método pronto para realizar esta etapa. A implementação dessa etapa é bem simples, foi criado um método em python chamado *create_tokens* que recebe a lista de tweets coletados pelo coletor e realiza a tokenização. Podemos conferir a implementação na Figura 14.

tokenizer.py > create_tokens

```
1   import nltk
2
3   def create_tokens(lista_tweets):
4       tokens = [nltk.word_tokenize(tweet) for tweet in lista_tweets]
5
6       return tokens
```

Figura 14 - Implementação da tokenização

3.3.2 - Normalização

A normalização dos tweets é feita para que o processo de análise seja mais preciso e tem a característica de manter um padrão nos textos retirando itens que sejam desnecessários para o

entendimento do mesmo. No processo de normalização do presente trabalho foram utilizadas as técnicas de: **Remoção de StopWords**, **Remoção de acentos**, **Remoção de HashTags**, **Remoção do usuário**, **Remoção de URLS** e **Remoção do Unicode**.

3.3.2.1 - Remoção de StopWords

As palavras vazias ou StopWords[25], no processamento de linguagem natural, são palavras que geralmente trazem muito pouco ou nenhum significado para o texto, por isso são removidas no momento do pré-processamento. A biblioteca do NLTK já tem um corpus com todas as stopwords da língua portuguesa[24], então basicamente fizemos um algoritmo para verificar e retirar essas palavras do tweet coletado. Podemos conferir essa implementação na Figura 15.

normalizer.py > ...

```
import re
import nltk
import string
import unicodedata

def remove_stopwords(lista_tweets):
    stopwords = nltk.corpus.stopwords.words("portuguese")

    regex = re.compile('[%s]' % re.escape(string.punctuation))
    nova_lista = []

    for index in range(len(lista_tweets)):
        nova_lista.append([])
        for palavra in lista_tweets[index]:
            nova_palavra = regex.sub(u'', palavra)
            if not nova_palavra == u'' and nova_palavra not in stopwords and not bool(re.search(r'\d', palavra)) and palavra != "RT"
                nova_lista[index].append(nova_palavra)

    return nova_lista
```

Figura 15 - Remoção de StopWords

3.3.2.2 - Remoção de Acentos

A acentuação, assim como as StopWords, não trazem significado relevante para o texto, então removê-los é uma boa prática, além de melhorar o desempenho dos algoritmos diminuindo o tempo de convergência. Na Figura 16 temos a implementação da remoção de acentos dos tweets.

```
def remove_acentos(lista_tweets):
    texto_limpo = []
    for indice in range(len(lista_tweets)):
        texto_limpo.append([])
        for palavra in lista_tweets[indice]:
            nfkd = unicodedata.normalize('NFKD', palavra)
            palavra_sem_acento = u''.join([c for c in nfkd if not unicodedata.combining(c
            q = re.sub('[^a-zA-Z0-9 \\\]', ' ', palavra_sem_acento)

            texto_limpo[indice].append(q.lower().strip())

    return texto_limpo
```

Figura 16 - Remoção de Acentos

3.3.2.3 - Remoção de Usuários, URLS e Unicode

Além da acentuação e StopWords, os tweets, ao serem coletados via API do twitter, trazem consigo algumas outras informações irrelevantes para a análise do significado do texto para nossos algoritmos, são elas: O nome do **usuário** que está publicando este tweet, a **URL** deste tweet e alguns **Unicodes**[26] que em nosso contexto são basicamente os emojis ou símbolos diferentes do alfabeto português. Na Figura 16 podemos conferir as implementações deste pré-processamento do tweet.

3.3.3 - Criação do Dataframe e armazenamento dos Tweets

Para armazenar os tweets coletados, escolhemos escrevê-los em um arquivo .csv[27], pois é um formato próprio para transferência de informações entre aplicações diferentes, além de ser um tipo de arquivo bastante leve. Uma vez que o tweet já foi devidamente pré-processado, utilizamos o auxílio da biblioteca pandas[28] para criação de um Dataframe[29] e consequentemente escrita dos tweets no arquivo .csv para serem analisados posteriormente.

No momento da criação do Dataframe, criamos duas colunas, "Tweet tratado" e "Tweet Original", com o objetivo de comparar como fica a tweet após todos o trabalho de pré-processamento e como era ele anteriormente a essa etapa, apesar de que para a análise do algoritmo em si, só o utilizaremos a coluna dos tweets tratados, é interessante que tenhamos o dado não processado, para fins de consulta e comparações futuras.

Na Figura 17 podemos conferir a implementação dessa etapa do desenvolvimento e na Figura 18, um exemplo de um tweet armazenado do arquivo .csv.

```python
def remove_usuario(lista_tweets):
    novos_tweets = []

    for tweet in lista_tweets:
        texto = re.sub(r"@\S+", "", tweet)
        novos_tweets.append(texto)

    return novos_tweets


def remove_urls(lista_tweets):
    novos_tweets = []

    for tweet in lista_tweets:
        texto = re.sub(r"http\S+", "", tweet["text"])
        novos_tweets.append(texto)

    return novos_tweets


def remove_unicode(lista_tweets):
    nova_lista = []
    for indice, palavras in enumerate(lista_tweets):
        nova_lista.append([])
        for palavra in palavras:
            if palavra != '':
                nova_lista[indice].append(str(palavra
    return nova_lista
```

Figura 16 - Remoção de Usuários, URLS e Unicodes.

```
columns = ['Tweet Tratado', 'Tweet Original']
data = []
tweets_original = []
tweets_normalized = []

for tweet in tweets.data:
    if(tweet is not None):
        tweets_original.append(tweet)

tweets_normalized = remove_urls(tweets_original)
tweets_normalized = remove_hashtag(tweets_normalized)
tweets_normalized = remove_usuario(tweets_normalized)
tweets_normalized = create_tokens(tweets_normalized)
tweets_normalized = remove_stopwords(tweets_normalized)
tweets_normalized = remove_acentos(tweets_normalized)
tweets_normalized = remove_unicode(tweets_normalized)

for i in range(len(tweets_normalized)):
    data.append([tweets_normalized[i],tweets_original[i]])


df = pd.DataFrame(data, columns=columns)

df.to_csv('tweets.csv', mode='a', index=False)
```

Figura 17 - Criação do Datraframe e armazenamento dos Tweets

Tweet Tratado	Tweet Original
['filha', 'puta', 'canta', 'mal', 'pra', 'caralho', 'fome', 'cantando', 'alto', 'vagao', 'gente', 'senso']	tem uma filha da puta que canta mal pra caralho de fome cantando alto no vagão. tem gente que não tem senso

Figura 18 - Tweet Tratado vs Tweet Original

3.4 - Corpus para treinamento

Como comentamos anteriormente nas seções 2.6 e 2.7, para o aprendizado de máquina supervisionado, é necessário que tenhamos um volume de dados anteriormente classificados ou com suas respectivas respostas para que os algoritmos sejam treinados. Neste trabalho utilizamos o corpus open source montado pelo Engenheiro de Software Rogers de Pelle [22], o OffComBR, pois é um corpus completo contendo exemplos de comentários classificados como discurso de ódio com exemplos de racismo, sexismo, homofobia, xenofobia, intolerância religiosa e xingamentos no geral.

O corpus do Rogers de Pelle foi montado com 1265 comentários de notícias de esporte e política do *g1.globo*, dos quais 420 eram ofensivos, dos quais a maioria eram insultos.

Para utilizar o corpus, realizamos a mesma etapa de pré-processamento que fizemos na coleta de dados, com o objetivo de manter o padrão em ambos os dados (Treinamento e Análise). Na Figura 19, podemos observar a implementação desse tratamento.

convertTrain.py > ...

```python
from csv import writer
from csv import reader
from normalizer import *
from tokenizer import create_tokens
import pandas as pd

with open('treinamento.csv', 'r') as read_obj, \
    open('treinamento_1.csv', 'w', newline='') as write_obj:
    csv_reader = reader(read_obj)
    csv_writer = writer(write_obj)
    lista_tweets = []
    classification = []
    data = []
    columns = ['Classificacao','Tweet Tratado', 'Tweet Original']
    for row in csv_reader:
        lista_tweets.append(row[0].split(';')[2])
        classification.append(row[0].split(';')[1])

    tweets_normalized = remove_hashtag(lista_tweets)
    tweets_normalized = remove_usuario(tweets_normalized)
    tweets_normalized = create_tokens(tweets_normalized)
    tweets_normalized = remove_stopwords(tweets_normalized)
    tweets_normalized = remove_acentos(tweets_normalized)
    tweets_normalized = remove_unicode(tweets_normalized)


    for i in range(len(tweets_normalized)):
        if len(tweets_normalized[i]) > 0 and i > 0:
            data.append([classification[i],' '.join(tweets_normalized[i]),lista_tweets[i]])


    df = pd.DataFrame(data, columns=columns)
    df.to_csv('treinamento_1.csv', mode='a', index=False)
```

Figura 19 - Tratamento do Corpus

3.5 - Revisão Bibliográfica

Observando o formato dos nossos dados, percebemos que se trata de um dado com classificação discreta, ou o tweet é classificado como discurso de ódio ou não é, ou seja, "sim"ou "não", esta é a forma que está no nosso corpus de treinamento e a forma que vamos treinar nossos algoritmos, logo, como falamos no nos capítulos 2.6 e 2.7 faz sentido utilizarmos o aprendizado supervisionado por classificação.

A biblioteca Scikit-Learn[30] para Python, é uma biblioteca que contém a implementação de uma série de algoritmos de aprendizado de máquina sendo tarefa do desenvolvedor, apenas entender essas implementações e como usá-las para o seu contexto.

Dado o tipo dos algoritmos que vamos utilizar, escolhemos 4 algoritmos cuja implementação é relativamente simples e a biblioteca Scikit-Learn os contém, são eles: **Máquina de Vetor de Suporte** (SVM)[31], **Regressão Logística**[32], **Floresta Aleatória**[33] e **Naive Bayes**[34].

3.5.1 - Máquina de Vetor de Suporte (SVM)

Imaginemos nosso conjunto de dados traçado em um plano cartesiano de duas dimensões, conforme a Figura 20, o SVM é um algoritmo que busca traçar uma reta que divide nosso conjunto de dados em grupos de classificação distintos, a descoberta dessa reta, significa dizer que conseguimos dizer qual a classe do nosso dado, ou seja, de qual lado da reta ele está.

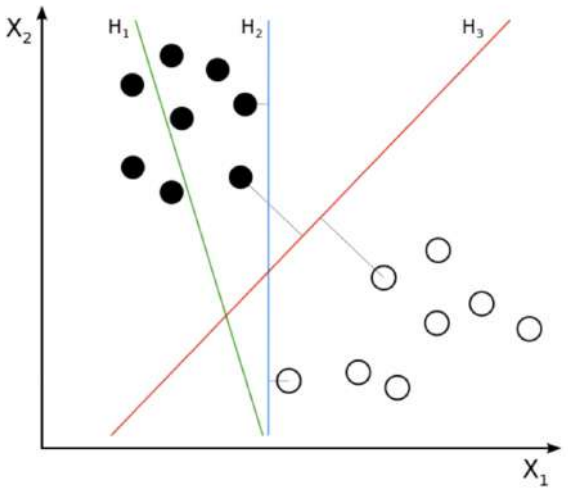


Figura 20 - Demonstração gráfica do SVM. Fonte: Wikimedia Commons[35].

Podemos conferir na Figura 21, a implementação mais simples possível de um algoritmo de SVM utilizando a biblioteca scikit-learn, claro que para nosso contexto utilizamos alguns parâmetros a mais e uma abordagem um pouco diferente, que discutiremos com mais detalhes no próximo capítulo.

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_features=4, random_state=0)
>>> clf = make_pipeline(StandardScaler(),
...                     LinearSVC(random_state=0, tol=1e-5))
>>> clf.fit(X, y)
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('linearsvc', LinearSVC(random_state=0, tol=1e-05))])

>>> print(clf.named_steps['linearsvc'].coef_)
[[0.141...   0.526... 0.679... 0.493...]]

>>> print(clf.named_steps['linearsvc'].intercept_)
[0.1693...]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

Figura 21 - Demonstração do SVM do Scikit-Learn

3.5.2 - Regressão Logística

A Regressão Logística é um modelo estatístico, geralmente é utilizado para determinar probabilidades, ele mostra a relação entre as amostras para calcular a probabilidade de alguma delas ocorrer. Podemos conferir sua definição graficamente na Figura 22.

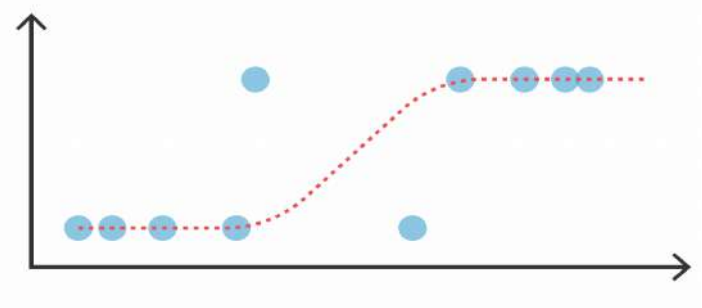


Figura 22 - Demonstração gráfica da Regressão Logística. Fonte: Tibco[36].

Sua implementação no Scikit-Learn, tem uma série de parâmetros obrigatórios, porém apenas o estado randômico inicial é necessário para utilização do algoritmo, podemos conferir a sua implementação mais básica na Figura 23.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Figura 23 - Demonstração de Regressão Logística do Scikit-Learn

3.5.3 - Floresta Aleatória

Uma floresta aleatória é um aleatória possui esse nome porque é um algoritmo formado por um conjunto de árvores de decisão[37], de maneira geral, todos os dados dessas árvores são mesclados para garantir as previsões mais precisas, enquanto uma única árvore tem um resultado menos preciso e uma gama de valores reduzidos, uma floresta aumenta aumenta essa precisão combinando os valores de todas essas árvores. Na Figura 24, temos um demonstrativo básico de uma floresta simples com apenas três árvores de decisão.

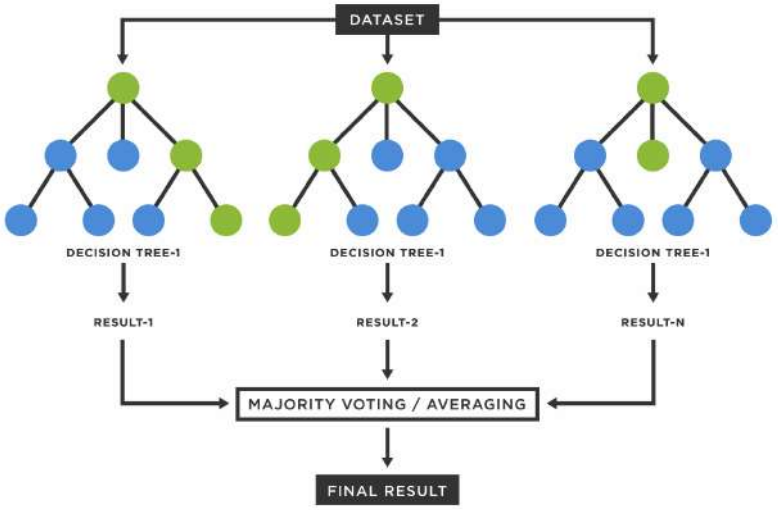


Figura 24 - Demonstração de Gráfica Random Forest. Fonte: Tibco [38]

Para implementar esse algoritmo usando o Scikit-Learn, temos vários hiperparâmetros disponíveis, no qual são configurados de acordo com o contexto de utilização. Podemos conferir um exemplo de implementação na Figura 25.

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

Figura 25 - Demonstração de Regressão Logística do Scikit-Learn

3.5.4 - Naive Bayes

O algoritmo de Naive Bayes[39] foi desenvolvido de acordo com analogia direta ao teorema da probabilidade, descrito por Thomas Bayes[40], conhecido como Teorema Bayes[41]. Podemos conferir a definição matemática do Teorema de Bayes na Figura 26.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

Figura 26 - Demonstração matemática do Teorema de Bayes

Para utilizar este Teorema no classificador Naive Bayes, são feitas algumas manipulações matemáticas gerando uma nova expressão conhecida como Naive Bayes Gaussiano [42]. A expressão matemática resultando pode ser conferida na Figura 27.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figura 27 - Naive Bayes Gaussiano

A implementação do algoritmo de Naive Bayes utilizando o Scikit-Learn, tem uma série de parâmetros que podem ser utilizados em diferentes contextos, um exemplo de utilização

deste, pode ser conferido na Figura 28, com o exemplo padrão que está na documentação da biblioteca [42].

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

Figura 28 - Implementação do Naive Bayes Gaussiano

3.5.5 - TF IDF

O termo TFIDF[43] é uma abreviação para *term frequency–inverse document frequency* ou em português *frequência do termo–inverso da frequência nos documentos*, é uma medida estatística que tem o objetivo de indicar a importância de uma palavra ou termo em relação a todo o contexto, ou seja, é um valor que aumenta proporcionalmente ao número de ocorrências dessa palavra ou termo.

Utilizamos esse conceito para criar uma matriz de *sample vs features* de acordo com a relevância da palavra em todo o contexto/documento e utilizá-la como entrada de treinamento dos nossos algoritmos. Mostraremos mais detalhadamente essa implementação no Capítulo 4, onde trataremos dos resultados obtidos.

3.6 - Jupyter Notebook

O Jupyter Notebook [44] é um software livre para a linguagem de programação Python que visa criar um ambiente computacional web para criação de documentos. Nele podemos executar códigos em python e ver logo em seguida o resultado. O documento em si é estruturado no formato JSON e pode conter uma mescla de código e texto (usando Markdown[45]) além dos resultados das execuções. Os arquivos do jupyter geralmente tem a extensão ".ipynb" e são executados no browser do usuário ao rodar o projeto. Uma alternativa muito usada ao Jupyter

Notebook é o Google Colab[46], que também tem o mesmo princípio de funcionamento porém este roda sempre na cloud, não sendo necessário executar nada localmente.

Na Figura 29 temos um exemplo de um "Hello World" no Jupyter Notebook.

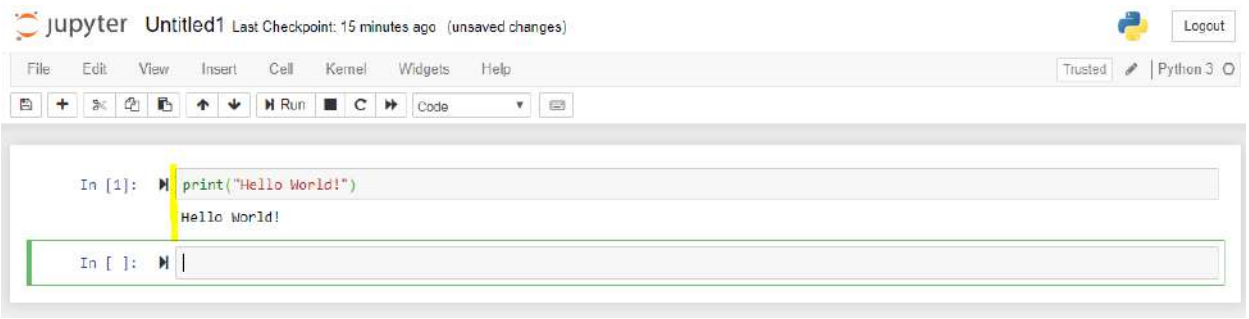


Figura 29 - Hello World Jupyter Notebook

Capítulo 4

Resultados

No final deste trabalho foram obtidos todos os resultados esperados no planejamento inicial. Mostramos esses resultados em um jupyter notebook no qual pode ser acesso localmente no computador do usuário.

4.1 - Resultado da coleta de dados

Com o objetivo de obtermos maior variedade de dados, com os mais diversos assuntos, as coletas foram segmentadas mensalmente durante os meses de **Abril 2022** até **Novembro 2022** executando o script de coleta uma vez por dia da semana (5 dias), totalizando 20 vezes por mês. Por mês foram coletados 2.000 tweets, com um **total de 16.000 tweets coletados**. Na Figura 30 temos o demonstrativo do arquivo .csv contendo os tweets coletados.

Para averiguar se os tweets coletados fizeram sentido, foi feito um demonstrativo gráfico (Figura 31), em que observamos as palavras mais comuns em todo o dataset, algumas palavras foram manualmente apagadas por questões éticas do presente trabalho. Vemos que o dataset está fazendo sentido pois a maioria das palavras fazem parte de palavras que possivelmente trazem um discurso de ódio.

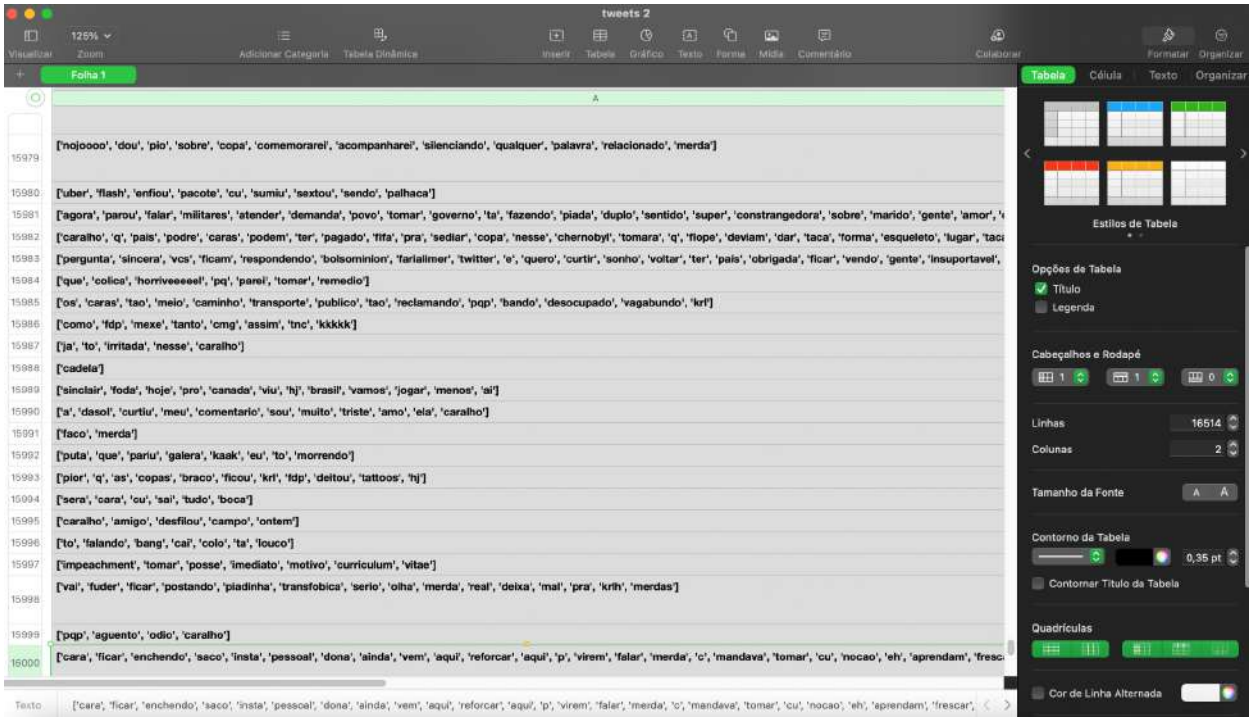


Figura 30 - Arquivo contendo os tweets coletados

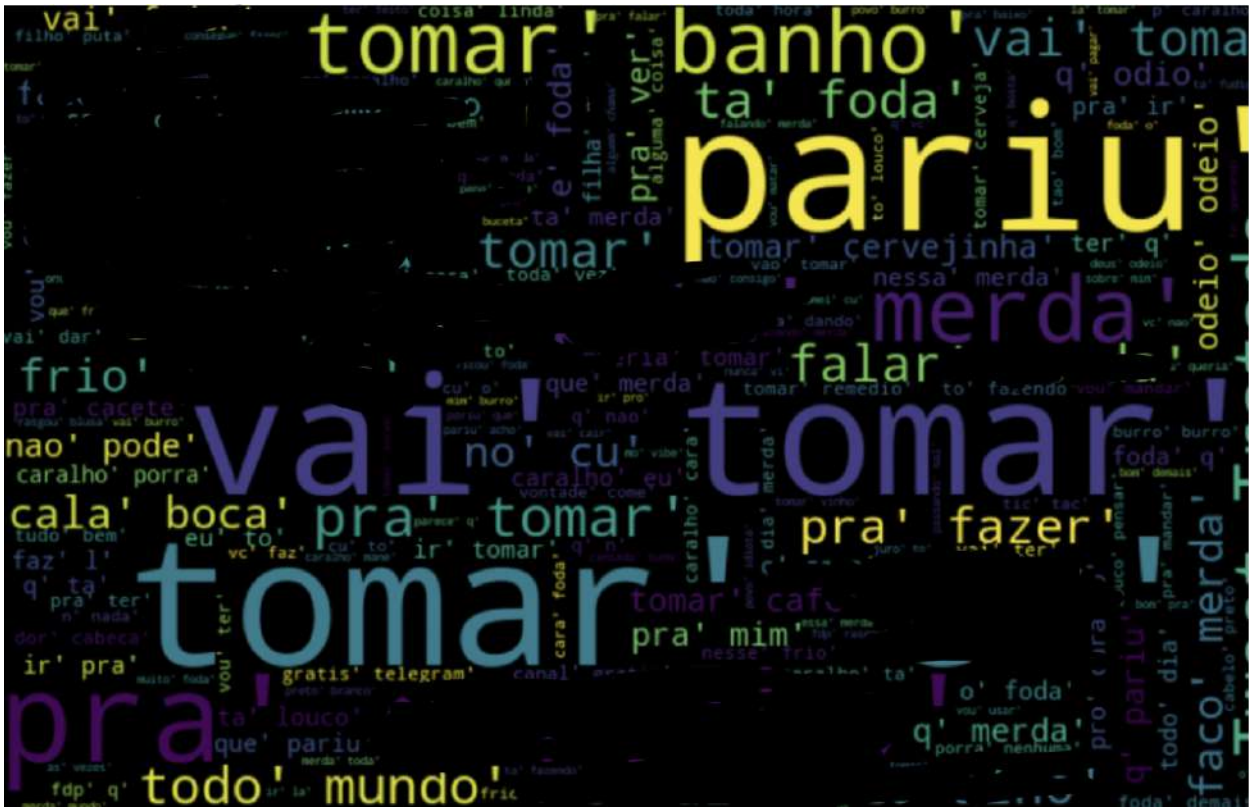


Figura 31 - Principais palavras dos tweets coletados

4.2 - Análise do corpus de treinamento

Além da qualidade dos dados coletados para análise é muito importante garantir a qualidade do corpus escolhido para treinamento. Para isso foi feita uma análise dos dados de treinamento utilizando a mesma abordagem dos dados coletados. Na Figura 32 podemos observar a maioria das palavras que aparecem no corpus de treinamento, já na Figura 33, observamos algumas palavras que contém os textos que foram qualificados como DO.

Ao observarmos as palavras dos textos qualificados como DO, vemos que o dataset de treinamento faz bastante sentido, pois algumas palavras que podem qualificar um texto de discurso de ódio aparecem no gráfico, como por exemplo: "burro", "bosta", "idiota", "anta", etc.

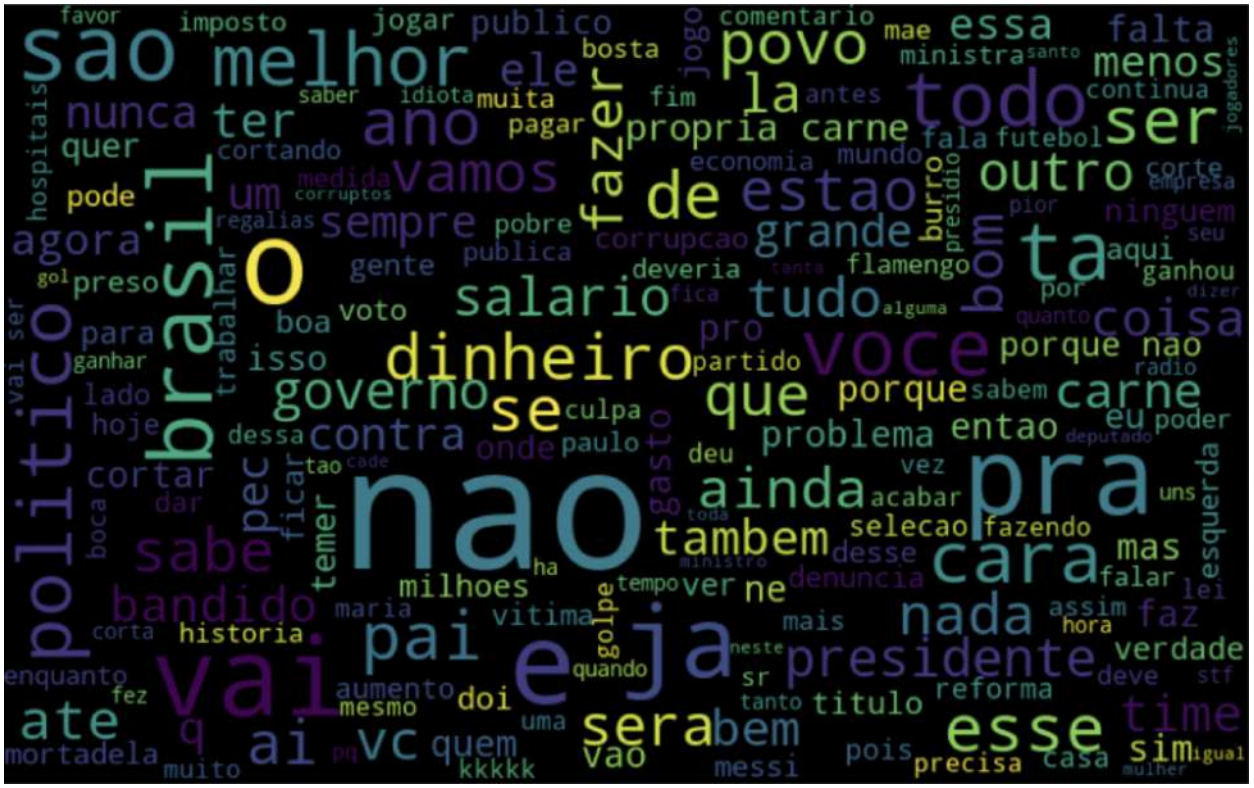


Figura 32 - Principais palavras do corpus de treinamento

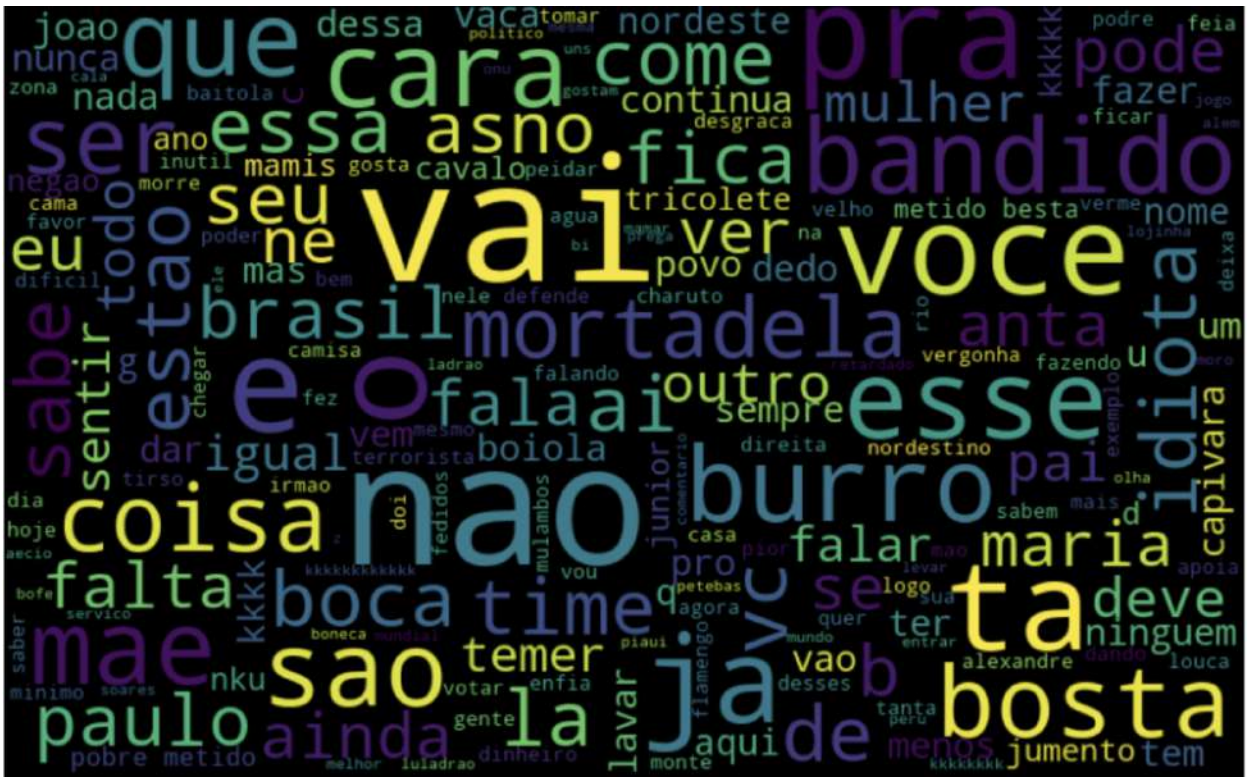


Figura 33 - Principais palavras dos textos qualificados como DO

4.3 - Treinamento e implementação dos algoritmos

Na etapa de desenvolvimento dos algoritmos, foi utilizada a biblioteca scikit-learn, primeiramente criamos uma matriz TFIDF com o corpus de treinamento, essa implementação pode ser conferida na Figura 34. Utilizamos o método TfidfVectorizer [47], podemos conferir a implementação na Figura 34. Os parâmetros foram escolhidos de acordo com uma análise de trabalhos que utilizam a mesma técnica, principalmente o Twitter-Hate-Speech-Detection[14]. O parâmetro **ngram_range** é utilizado para considerar bigramas e unigramas na criação da matriz, os parâmetros **max_df** e **min_df** para limitarem a ocorrência de um termo encontrado (desconsiderar se estiver presente em mais de 75% dos tweets e menos de 5 ocorrências) e o **max_features** para limitar as colunas da matriz em 10.000 com o objetivo de se proteger contra um **estouro de processamento**.

```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2),max_df=0.75, min_df=5, max_features=10000)
tfidf = tfidf_vectorizer.fit_transform(train['Tweet Tratado'])
```

Figura 34 - Implementação do TFIDF

Após a criação da Matriz TFIDF começamos o treinamento dos algoritmos seguindo o fluxo da Figura 35.

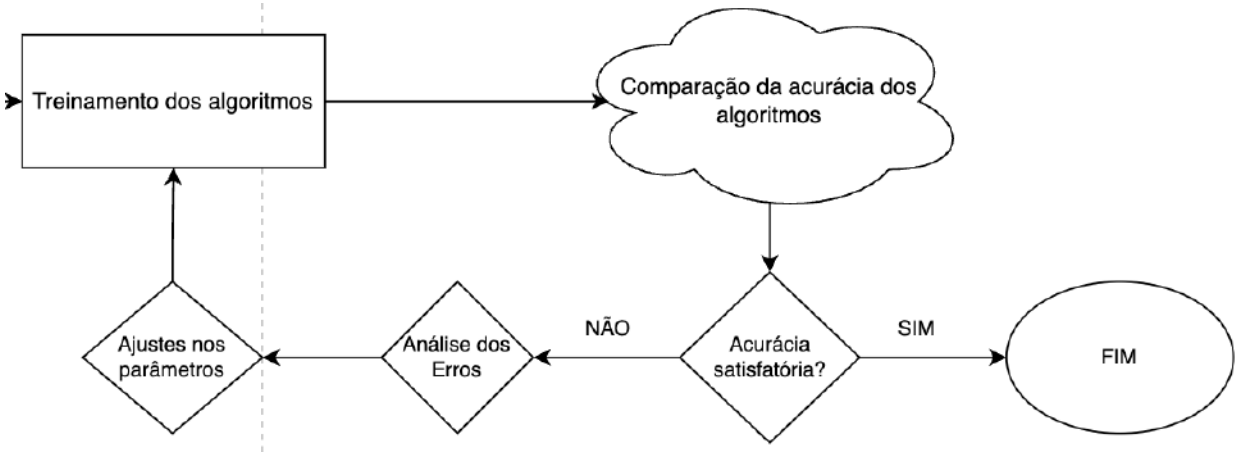


Figura 35 - Fluxo de treinamento dos algoritmos

Para nosso escopo consideramos uma **acurácia[48] satisfatória acima de 0.75 ou 75%**, ao obter esse resultado mínimo para todos os algoritmos paramos os treinamentos. Na Figuras 36, 37, 38 e 39 temos os resultados finais da implementação de cada algoritmo. É importante notar que a acurácia de todos, convergiu para mais de 75%.

Na Figura 40 temos um gráfico comparativo da acurácia dos quatro algoritmos implementados, percebemos que o de menor desempenho para o nosso contexto foi o de Naive Bayes, porém ainda assim conseguimos uma boa acurácia de mais de 75%.

```
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
rf=RandomForestClassifier()
rf.fit(X_train_tfidf,y_train)
y_preds = rf.predict(X_test_tfidf)
```

Figura 36 - Floresta Aleatória - Implementação final

```
X = tfidf
y = trains['Classificação']
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
model = LogisticRegression().fit(X_train_tfidf,y_train)
y_preds = model.predict(X_test_tfidf)
```

Figura 37 - Regressão Logística - Implementação final

```
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X.toarray(), y, random_state=42, test_size=0.2)
nb=GaussianNB()
nb.fit(X_train_tfidf,y_train)
y_preds = nb.predict(X_test_tfidf)
```

Figura 38 - Naive Bayes - Implementação final

```
X_train_tfidf, X_test_tfidf, y_train, y_test = train_test_split(X.toarray(), y, random_state=42, test_size=0.2)
support = LinearSVC(random_state=20)
support.fit(X_train_tfidf,y_train)
y_preds = support.predict(X_test_tfidf)
```

Figura 39 - SVM - Implementação final

Acurácia obtida por algoritmo

Algoritmo	Acurácia
Regressão Logística	89%
Floresta Aleatória	88%
Naive Bayes	75%
SVM	89%

Figura 40 - Comparação da acurácia dos algoritmos

4.4 - Classificação dos tweets coletados

Em relação a classificação dos tweets coletados, executamos cada algoritmo na base de dados coletada e obtemos resultados demonstrados nas Figura 41. Nota-se que o número de tweets identificados como discurso de ódio aumentou proporcionalmente a acurácia dos algoritmos. Esse resultado, é um resultado quantitativo em relação a toda a base, nesse momento nenhum tweet da base foi rotulado.

Para averiguar a acurácia obtida na base de testes, foram **rotulados manualmente 100 tweets da base**, escolhidos aleatoriamente, foram escolhidos **50 tweets rotulados como discurso de ódio** e **50 tweets rotulados como NÃO discurso de ódio**. Os algoritmos foram executados nessa pequena parcela de dados, obtendo-se os resultados demonstrados nas Figura 42 e 43.

Em geral podemos dizer que o resultado foi satisfatório, outros estudos podem aprimorar mais ainda os resultados obtidos como por exemplo construindo um corpus mais robusto e com mais registros para treinamento dos algoritmos.

Tweets classificados como discurso de ódio por algoritmo

Algoritmo	É DO	Não é DO
Regressão Logística	4.174	11.826
Floresta Aleatória	3.271	12.729
Naive Bayes	2.258	13.742
SVM	6.258	9.742

Figura 41 - Tweets classificados como discurso de ódio por algoritmo (Base sem rótulo)

Algoritmo	Acurácia
Regressão Logística	78%
Floresta Aleatória	77%
Naive Bayes	68%
SVM	79%

Figura 42 - Acurácia dos algoritmos na base rotulada

Matriz de Confusão dos Algoritmos

Algoritmo	VP	FP	VN	FN
Regressão Logística	36	14	42	8
Floresta Aleatória	40	10	37	13
Naive Bayes	29	21	39	11
SVM	34	16	37	13

Figura 43 - Matriz de Confusão dos algoritmos na base rotulada

Capítulo 5

Considerações Finais e Sugestões para Trabalhos Futuros

Ao longo deste trabalho foi identificado que a rede social virtual twitter, está servindo para disseminação de ódio relativamente alta. Se observarmos o algoritmo com maior acurácia obtida (SVM), o número de tweets classificados como DO chega próximo a 40%, um número assustador já que prezamos (idealmente) por uma sociedade mais amorosa e sem tanto ódio.

O **TFIDF** se mostrou muito eficiente na implementação dos nossos algoritmos e em conjunto com o método *fit* do *Scikit-learn* a implementação ficou bastante simples.

Apesar de termos conseguido uma acurácia satisfatória nos algoritmos implementados, acredito que podemos melhorar a mesma com outros testes a serem feitos, como futuro desenvolvimento, podemos pensar na construção de um corpus próprio, com uma base de dados maior e além disso o uso de outros algoritmos de aprendizado de máquina e uso de algoritmos de aprendizado profundo.

O Jupyter Notebook atendeu bem para demonstração dos resultados e foi capaz de gerar visualizações interessantes, porém podemos pensar em melhorar o trabalho, construindo uma interface web amigável para o usuário final, no qual ela seja dinâmica e o usuário possa decidir quais visualizações deseja analisar. Além disso, consiga por exemplo, coletar os tweets a partir de palavras chaves que ele define e criar o próprio corpus.

Referências Bibliográficas

[1] NANDI, José Adelmo Becker et al. O combate ao discurso de ódio nas redes sociais. 2018. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/187510/O_Combate_ao_Discurso_de_Odio_nas_Redes_Sociais.pdf?sequence=1. Acesso em 22 fev. 2022.

[2] Facebook, YouTube, Twitter and Microsoft sign EU hate speech code. The Guardian, 2016. Disponível em: https://www.theguardian.com/technology/2016/may/31/facebook-youtube-twitter-microsoft-eu-hate-speech-code . Acesso em 22 fev. 2022.

[3] Conteúdo identificado como discurso de ódio no Facebook sobe 389% em 1 ano, mas rede diz que visualizações caíram. G1, 2021. Disponível em: https://g1.globo.com/economia/tecnologia/noticia/2021/02/11/facebook-identifica-269-milhoes-de-conteudos-com-discurso-de-odio-no-4o-trimestre-de-2020.ghtml. Acesso em 22 fev. 2022.

[4] MARTINS, Anna Clara Lehmann. Discurso de ódio em redes sociais e reconhecimento do outro: o caso M. Revista Direito GV, v. 15, 2019.

[5] O que é discurso de ódio? | Nexo Políticas Públicas. Youtube, 2021. Disponível em: https://www.youtube.com/watch?v=KeWp9wIi4SI. Acesso em 22 fev. 2022.

[6] Discurso de ódio. Wikipedia, 2021. Disponível em: https://pt.wikipedia.org/wiki/Discurso_de_%C3%B3dio. Acesso em 22 fev. 2022.

[7] Política contra propagação de ódio. Twitter. Disponível em: https://help.twitter.com/pt/rules-and-policies/hateful-conduct-policy. Acesso em 22 fev. 2022.

[8] da Silva, Vitória Régia. Brasil do ódio. Generonumero, 2019. Disponível em: https://www.generonumero.media/crime-de-odio/. Acesso em 23 fev. 2022.

[9] Alshalan, Raghad, et al. “Detection of Hate Speech in COVID-19-Related Tweets in the Arab Region: Deep Learning and Topic Modeling Approach.” Journal of Medical Internet Research, vol. 22, no. 12, 2020, p. e22609.

[10] Decide What Is Hate Speech in an Online Global Community?. Facebook, 2017. Disponível em: https://about.fb.com/news/2017/06/hard-questions-hate-speech/. Acesso em 23 fev. 2022.

[11] BAYDOGAN, Cem et al. Deep-Cov19-Hate: A Textual-Based Novel Approach for Automatic Detection of Hate Speech in Online Social Networks throughout COVID-19 with Shallow and Deep Learning Models. Tehnički vjesnik, v. 29, n. 1, p. 149-156, 2022.

[12] Hatelab. Hatelab. Disponível em: https://hatelab.net/. Acesso em 22 fev. 2022.

[13] HateLab: Understanding toxic conversations to combat them. Twitter. Disponível em: https://developer.twitter.com/en/community/success-stories/hatelab. Acesso em 22 fev. 2022.

[14] Twitter-Hate-Speech-Detection. Github, 2020. Disponível em: https://github.com/vedant-95/Twitter-Hate-Speech-Detection. Acesso em 22 fev. 2022.

[15] Detecção Automática de Discurso de Ódio em Textos do Twitter. UFRRJ, 2021. Disponível em: https://www.cc.ufrrj.br/wp-content/uploads/2018/07/Trabalho_de_Conclus_o_de_Curso___Victor_Diniz1-compactado.pdf. Acesso em 22 fev. 2022.

[16] Classificação Automática de Discursos de Ódio em Textos do Twitter. UFRPE, 2019. Disponível em: https://www.repository.ufrpe.br/bitstream/123456789/2439/1/tcc_robsonmuriloferreiradonascimento.pdf. Acesso em 22 fev. 2022.

[17] Rivera, Pablo. Tweepy: An easy-to-use Python library for accessing the Twitter API. Tweepy, 2021. Disponível em: https://www.tweepy.org. Acesso em 21 mar. 2022.

[18] Machine Learning, O que é e qual sua importância?. Disponível em: https://www.sas.com/pt_br/insights/analytics/machine-learning.html#:~:text=O%20aprendizado%20de%20m%C3%A1quina%20(em,o%20m%C3%ADnimo%20de%20interven%C3%A7%C3%A3o%20humana. Acesso em 22 mar. 2022.

[19] AlphaGo. Disponível em: https://www.deepmind.com/research/highlighted-research/alphago. Acesso em 23 mar 2022.

[20] Tweepy. Disponível em:
https://www.tweepy.org/. Acesso em 24 mar 2022.

[21] Twitter Developer. Disponível em:
https://developer.twitter.com/en. Acesso em 24 mar 2022.

[22] Corpus de treinamento. Disponível em:
https://github.com/rogersdepelle/OffComBR. Acesso em 25 mar 2022

[23] NLTK. Disponível em:
https://www.nltk.org/. Acesso em 27 mar 2022.

[24] StopWords, NLTK. Disponível em:
https://www.nltk.org/howto/portuguese_en.html. Acesso em 29 mar 2022.

[25] StopWords. Disponível em:
https://pt.wikipedia.org/wiki/Palavra_vazia. Acesso em 29 mar 2022.

[26] Unicode. Disponível em:
https://pt.wikipedia.org/wiki/Unicode. Acesso em 30 mar 2022.
[27] Arquivos CSV. Disponível em:
https://rockcontent.com/br/blog/csv/. Acesso em 2 Abr 2022.

[28] Pandas, python. Disponível em:
https://pandas.pydata.org/. Acesso em 2 Abr 2022.

[29] Dataframes, Pandas. Disponível em:

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html. Acesso em 2 Abr 2022.

[30] Scikit-Learn. Disponível em:
https://scikit-learn.org/stable/. Acesso em 4 Abr 2022.

[31] SVM, SciKit-Learn. Disponível em:
https://scikit-learn.org/stable/modules/svm.html. Acesso em 5 Abr 2022.
[32] Regressão Logística, SciKit-Learn. Disponível em:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression. Acesso em 5 Abr 2022.

[33] Floresta Aleatória, SciKit-Learn. Disponível em:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforestclassifier#sklearn.ensemble.RandomForestClassifier. Acesso em 5 Abr 2022.

[34] Naive Bayes, SciKit-Learn. Disponível em:
https://scikit-learn.org/stable/modules/naive_bayes.html. Acesso em 5 Abr 2022.

[35] Wikimedia commons. Disponível em:
https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg. Acesso em 6 Abr 2022.

[36] Tibco, Regressão Logística. Disponível em:
https://www.tibco.com/pt-br/reference-center/what-is-logistic-regression. Acesso em 6 Abr 2022.

[37] Árvore de decisão. Disponível em:
https://pt.wikipedia.org/wiki/%C3%81rvore_de_decis%C3%A3o. Acesso em 6 Abr 2022.

[38] Floresta Aleatória. Disponível em:
https://www.tibco.com/pt-br/reference-center/what-is-a-random-forest. Acesso em 7 Abr 2022.

[39] Naive Bayes Classifier. Disponível em:
https://en.wikipedia.org/wiki/Naive_Bayes_classifier. Acesso em 8 Abr 2022.

[40] Thomas Bayes. Disponível em:
https://pt.wikipedia.org/wiki/Thomas_Bayes. Acesso em 8 Abr 2022.

[41] Teorema de Bayes. Disponível em:
https://pt.wikipedia.org/wiki/Teorema_de_Bayes. Acesso em 8 Abr 2022.

[42] Naive Bayes Gaussiano. Disponível em:
https://scikit-learn.org/stable/modules/naive_bayes.html. Acesso em 8 Abr 2022.

[43] TFIDF. Disponível em:
https://pt.wikipedia.org/wiki/Tf%E2%80%93idf. Acesso em 9 Abr 2022.

[44] Jupyter Notebook. Disponível em:
https://jupyter.org/. Acesso em 15 mar 2022.

[45] Markdown. Disponível em:
https://pt.wikipedia.org/wiki/Markdown. Acesso em 15 mar 2022.

[46] Google Colab. Disponível em:
https://colab.research.google.com/. Acesso em 15 mar 2022.

[47] TfidfVectorizer. Disponível em:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Acesso 15 jun 2022.

[48] Acurácia, ClearSale. Disponível em:
https://blogbr.clear.sale/conheca-e-saiba-como-aplicar-a-acuracia. Acesso 20 jun 2022.

[49] Normalização, Wikipédia. Disponível em:
https://pt.wikipedia.org/wiki/Normaliza%C3%A7%C3%A3o_de_dados. Acesso 20 jun 2022.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Campina Grande
R. Tranqüilino Coelho Lemos, 671, Dinamérica, CEP 58432-300, Campina Grande (PB)
CNPJ: 10.783.898/0003-37 - Telefone: (83) 2102.6200

Documento Digitalizado Ostensivo (Público)

TCC FINAL COM A FICHA

Assunto: TCC FINAL COM A FICHA
Assinado por: Wesley Azevedo
Tipo do Documento: Tese
Situação: Finalizado
Nível de Acesso: Ostensivo (Público)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Wesley Weverton de Azevedo Palmeira, ALUNO (201621250039) DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO - CAMPINA GRANDE**, em 03/01/2023 12:16:41.

Este documento foi armazenado no SUAP em 03/01/2023. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 712000
Código de Autenticação: 9dd784849d