GABRIEL OGA SANEFUJI
SANDRA AYUMI NIHAMA

RACISMO AMARELO: UMA ANÁLISE SOBRE DISCURSOS DE ÓDIO CONTEMPORÂNEOS POR MEIO DE APRENDIZAGEM DE MÁQUINA

São Paulo
2022

GABRIEL OGA SANEFUJI
SANDRA AYUMI NIHAMA

RACISMO AMARELO: UMA ANÁLISE SOBRE DISCURSOS DE ÓDIO CONTEMPORÂNEOS POR MEIO DE APRENDIZAGEM DE MÁQUINA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

São Paulo
2022

GABRIEL OGA SANEFUJI
SANDRA AYUMI NIHAMA

RACISMO AMARELO: UMA ANÁLISE SOBRE DISCURSOS DE ÓDIO CONTEMPORÂNEOS POR MEIO DE APRENDIZAGEM DE MÁQUINA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro de Computação.

Orientador:

Prof. Dr. Ricardo Luis de Azevedo Rocha

São Paulo
2022

Prof. Dr. Ricardo Luis de Azevedo da Rocha

Catalogação-na-publicação

Sanefuji, Gabriel
Racismo amarelo: uma análise sobre discursos de ódio contemporâneos por meio de aprendizagem de máquina / G. Sanefuji, S. A. Nihama -- São Paulo, 2022.
110 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Racismo 2.Aprendizado Computacional 3.Processamento de Linguagem Natural 4.Serviços em Nuvem I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Nihama, Sandra Ayumi

AGRADECIMENTOS

Agradecemos aos nossos pais e irmãozinhos, pelo apoio e carinho incondicionais ao longo de toda a nossa vida, nos altos e baixos. Nunca teríamos conseguido chegar até aqui sem eles.

Ao Professor Ricardo Luis de Azevedo da Rocha, pelo suporte e orientação durante todo o projeto, pelas conversas descontraídas depois das reuniões e pelas aulas (nada fáceis) de Lógica Computacional e Sistemas de Programação.

Ao Lucas Lago e ao Professor David Nemer, que tanto nos ajudaram com ideias e conhecimento, essenciais para o desenvolvimento deste projeto.

Às turmas de 2017 e 2018, de onde fizemos grandes amigos que nos acompanharam nos momentos de alegria e de sofrimento ao longo dessa longa jornada na Escola Politécnica. Aos veteranos, que mesmo distantes, continuaram contribuindo para nós podermos nos formar. Um destaque a Saori Tsuji, Marcel Kondo, Tiago Marto, Guilherme Migliati, Kenzo Shiraishi, Leonardo Cassiano, Felipe Valencia, ao "grupinho da Elétrica", do Duplo Diploma, do "Rubens Surfista", a equipe de Tênis de Mesa e do Cursinho da Poli-USP.

Uma menção especial ao Leme e ao Serginho, que em tempos de desespero, nos deram uma luz para concluir o projeto.

Agradecemos um ao outro, pela motivação, conversas, desabafos e pelo árduo trabalho que sabemos que cada um desempenhou para concluirmos o projeto.

Por fim, agradecemos à Escola Politécnica, pela nossa formação, por todas as oportunidades que nos proporcionou e pelas portas que continuam a se abrir para nós.

"O essencial é ter a coragem de arriscar e desenvolver novas ideias"

-- Zaha Hadid

RESUMO

O racismo ainda é um tema muito relevante atualmente, presente não apenas no contexto de povos com descendência africana, mas também contra populações provenientes da Ásia e de cultura indígena. Esse tipo de discurso de ódio direcionados à população asiática se tornou mais visível por meio do crescente número de comentários xenofóbicos em redes sociais durante os Jogos Olímpicos de Tóquio 2021 e o início do período da pandemia da Covid-19 . Um exemplo concreto do ódio direcionado a etnias asiáticas se dá por meio de episódio em 2021, no qual um atirador assassinou mulheres de descendência asiática que frequentavam um spa.

O trabalho aqui apresentado é um experimento em processamento de linguagem natural e *machine learning*, com o intuito de identificar comentários racistas contra asiáticos e seus descendentes na rede social Twitter. Portanto, procura-se por analisar sentimentos presentes nos comentários, assim como elementos de microagressão, de forma que o contexto e a forma como certas palavras foram usadas nas frases em estudo indiquem a presença ou não de injúrias raciais.

Com a esperança de se trazer uma conscientização, ou ao menos um meio para que a população consiga se educar, os modelos treinados e especificados ao longo do trabalho foram integrados a uma aplicação em forma de *bot* do Twitter, por meio do qual usuários tem a possibilidade de chamá-lo e verificar se uma mensagem possui alguma conotação racista contra asiáticos.

Vale destacar que o trabalho possui um enfoque ao racismo contra povos e descendentes da Ásia, uma vez que pouco se encontrou trabalhos com essa temática. Ademais, devido à sensibilidade do tema, achou-se sensato concentrar-se numa etnia apenas.

Palavras-Chave – Racismo, Asiático, Ásia, Machine Learning, Processamento de Linguagem Natural, Bot, Twitter.

ABSTRACT

Racism is still a very relevant topic today, present not only in the context of people of African descent, but also against Asian and indigenous populations. This kind of hate speech directed at the Asian population has become more visible through the increasing number of xenophobic comments on social networks during the Tokyo 2021 Olympic Games and the beginning of the Covid-19 pandemic period. A concrete example of the hatred directed at Asian ethnicities comes through an episode in 2021 in which a gunman murdered women of Asian descent who frequented a spa.

The work presented here is an experiment in natural language processing and *machine learning*, in order to identify racist comments against Asians and their descendants on the social network Twitter. Therefore, it tries to analyze the sentiments present in posts, as well as elements of microaggression, so that the context and the way certain words were used in the sentences under study indicate the presence or not of racial slurs.

With the hope of bringing awareness, or at least a way for the population to educate themselves, the models trained and specified throughout the work were integrated into a Twitter application in the form of a Twitter app, through which users have the possibility to call it and check whether a message has any racist connotation against Asians.

It is worth pointing out that the work has a focus on racism against Asian people and descendants, since few studies on this theme have been found. Moreover, given the sensitivity of the theme, it was thought sensible to concentrate on only one ethnic group.

Keywords – Racism, Asian, Asia, Machine Learning, Natural Language Processing, Bot, Twitter.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este primeiro capítulo contém as motivações, objetivos e justificativas para a realização do trabalho desenvolvido, apresentando em cada uma das seções explicações a respeito de cada um desses tópicos. Por fim, há também uma seção com o conteúdo do documento como um todo.

1.1 Motivação do projeto

O racismo, infelizmente, ainda é um tema muito recorrente. Presente na sociedade atual, afeta não apenas a população preta, mas também as parda, indígena e amarela. Durante os Jogos Olímpicos de Tóquio em 2021, a quantidade de comentários racistas e xenofóbicos contra a população amarela cresceu consideravelmente nas redes sociais. O mesmo fenômeno ocorreu no início da pandemia de Covid-19. Agressões e crimes de ódio se tornaram mais comuns e repercutiram no mundo todo, como o caso do assassinato de descendentes de asiáticos em casas de massagem em Atlanta em 2021.

A coalizão norte-americana *Stop AAPI Hate* divulgou dados alarmantes sobre ataques de ódio contra a população asiática no país (JEUNG; HORSE; CAYANAN, 2021): durante a pandemia, receberam 6603 denúncias de março de 2020 a março de 2021, sendo 2808 só neste último mês em questão. Os ataques são de diversos tipos: verbais, físicos, no local de trabalho, na rua, mesmo por meio de vandalismo.

A fim de sensibilizar e gerar mais estatísticas sobre esse tema, o projeto de formatura busca estudar a presença de comentários racistas contra asiáticos e descendentes de asiáticos na rede social *Twitter*, criando modelos de classificação de *tweets* e reconhecendo elementos de microagressões em postagens. Além disso, possui também o papel de promover a conscientização da população sobre o tema através de soluções desenvolvidas na própria rede social.

1.2 Objetivos do projeto

Tendo em vista o que foi exposto como motivação do projeto, seu objetivo geral é utilizar de métodos de aprendizado de máquina para analisar mensagens e os sentimentos que elas passam ao leitor, a fim de classificá-las como sendo de cunho racista ou não. As frases classificadas pelo projeto foram recolhidas de *datasets* públicos de comentários do Twitter e por meio do processo de web scraping na mesma rede social.

Dentre as metas do trabalho de conclusão de curso, estava a criação de um *bot* na plataforma. Um *bot* consiste numa conta automatizada, capaz de postar e responder comentários automaticamente. Para o projeto, foram elaboradas as seguintes funcionalidades, com o auxílio do professor orientador e de colaboradores externos (Lucas Lago e David Nemer, mencionados na seção de metodologia):

- Identificação de comentários racistas em *threads* do Twitter;
- Verificação das contas que o seguem na plataforma;
- Publicação de resposta com informações sobre a presença ou ausência de comentários racistas na *thread* em que foi chamado para atuar.

O racismo contra a população asiática é muito presente na sociedade ocidental e o projeto tem como intenção trazer visibilidade para o tema. Por meio do *bot*, é possível promover a conscientização dos usuários da rede social em questão, por meio de mensagens instrutivas e não agressivas. É uma forma de compartilhar os resultados técnicos do grupo – ou seja, os modelos de *Machine Learning* desenvolvidos – de modo mais abrangente e acessível, para diferentes públicos.

Apesar do projeto ter como motivação pessoal dos integrantes do grupo o racismo contra asiáticos no Brasil, devido à falta de *datasets* pertinentes para estudo em português, todo o estudo foi feito com textos em inglês. A quantidade de bases de dados públicas e, até mesmo, de *tweets* nesta língua é maior, o que contribuiu para o desenvolvimento do trabalho, uma vez que ter um grande volume de amostras para treinamento de modelos é crucial em problemas de Processamento de Linguagem Natural.

Por conta da delicadeza e sutileza no assunto abordado, o modelo utilizado e desenvolvido teve de levar em conta questões como a livre expressão e o contexto em que determinada frase é utilizada. Foram estudados modelos já existentes na literatura e possíveis adaptações e diferentes aplicações decorrentes do contexto em que serão empregados.

Desse modo, concluído o projeto, os alunos puderam aprender conceitos sobre métodos de aprendizado de máquina e processamento de linguagem natural, aplicando-os num contexto voltado à identificação de discursos racistas. Além disso, foi feita uma contribuição para a conscientização do público geral sobre o racismo amarelo através do trabalho desenvolvido.

1.3 Justificativa

Diversos estudos sobre análise de sentimentos podem ser encontrados na literatura, inclusive a partir de comentários retirados do *Twitter*. Entretanto, poucos são os voltados ao racismo, especialmente ao racismo contra amarelos. Consequentemente, poucas são as bases públicas de *tweets* disponíveis sobre esse tema. Trabalhos como (HE et al., 2021) e (VIDGEN et al., 2020) são alguns dos exemplos de análise de discursos preconceituosos.

A realização deste trabalho teve como intuito desenvolver análises sobre o tema, especificamente em relação aos asiáticos e descendentes de asiáticos. Isso, por este ser um tópico presente atualmente e de grande notoriedade, dado o relevante aumento no número de casos de racismo durante a pandemia de Covid-19 e alguns episódios dos Jogos Olímpicos de Tóquio 2021.

1.4 Organização do trabalho

Este trabalho está organizado em nove seções:

1. Esta primeira, de introdução, voltada a apresentação do tema do projeto e suas motivações;
2. De aspectos conceituais, apresentando definições de elementos importantes para a concepção do projeto, tanto no âmbito técnico quanto no social;
3. De tecnologias utilizadas, indicando as que foram usadas para as primeiras análises, extrações, processamento de dados e criação de modelos de *Machine Learning* e *Deep Learning*, e as utilizadas para o desenvolvimento do *bot*;
4. Da metodologia de trabalho, que indica o modo como o trabalho foi desenvolvido e as reuniões que o grupo teve com colaboradores e o professor orientador;
5. De projeto e implementação, indicando as decisões técnicas de projeto e detalhes sobre a implementação do mesmo;

6. De testes e resultados, apresentando os resultados obtidos com os estudos realizados;
7. De conclusões extraídas com o trabalho em questão;
8. De Próximos Passos, evidenciando futuras melhorias possíveis para o estudo;
9. E de referências, com os artigos e outras fontes úteis para estudo e entendimento do projeto como um todo.

2 ASPECTOS CONCEITUAIS

Neste capítulo, serão apresentados alguns conceitos importantes para a compreensão do documento e do projeto sendo realizado. São definições de âmbitos tanto técnicos quanto sociais, que foram necessárias para o trabalho.

2.1 Machine Learning

Em português, "Aprendizagem de Máquina", é uma área da Inteligência Artificial que se baseia na criação de modelos estatísticos em computação para interpretação de dados. Geralmente, esses modelos são procedimentos computacionais automatizados baseados em operações lógicas ou binárias, os quais aprendem uma tarefa por meio de vários exemplos. (MICHIE; SPIEGELHALTER; TAYLOR, 1999) Tais modelos têm fins de classificação ou regressão sobre novos dados de entrada.

2.1.1 Deep Learning

É uma área de estudo contida em *Machine Learning* que foca no desenvolvimento de modelos complexos. Nela, são abordadas redes neurais profundas, com várias camadas e parâmetros a serem analisados e treinados. Em geral, está unida a tecnologias de manipulação de grandes volumes de dados (*Big Data*) e de computação paralela (incluindo GPUs e TPUs).

2.1.2 Processamento de linguagem natural

É uma subárea da Inteligência Artificial que estuda a capacidade de máquinas em interpretar e compreender linguagens naturais. Ela utiliza de conceitos da Linguística, *Machine Learning* e *Deep Learning* para a criação de modelos e análises. Essas tecnologias em conjunto possibilitam que computadores processem linguagem humana na forma de texto ou dados de voz, "compreendendo"o significado inteiro das frases analisadas. Desse

modo, técnicas de processamento de linguagem natural são utilizadas em aplicações que necessitem do entendimento completo sobre a linguagem humana e suas minúncias, como por exemplo: ferramentas de busca na Internet e assistentes pessoais, como Alexa e Siri.

Para trabalhar com dados textuais, é importante tratá-los inicialmente convertendo-os em informações numéricas, a fim de se realizar os cálculos e tratamentos necessários.

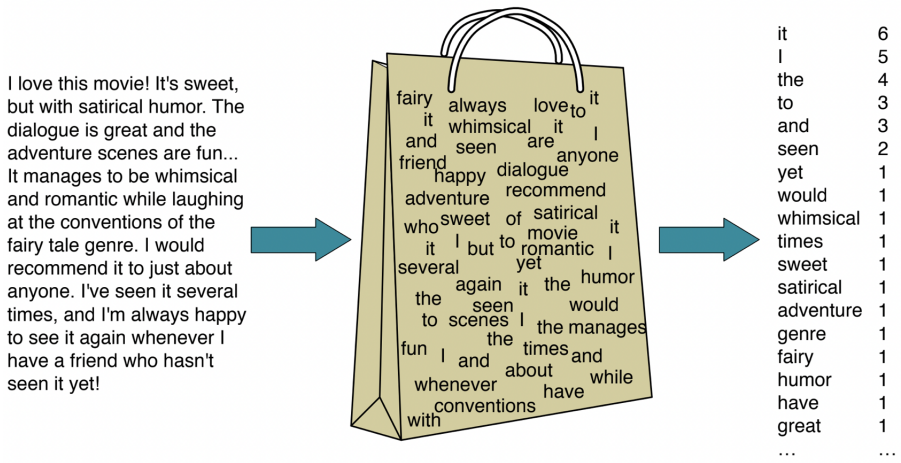
2.1.3 Word Embedding

Uma forma de realizar esse tipo de pré-processamento, de conversão entre texto e número, é por meio da transformmação de palavras em vetores. A intuição compartilhada entre modelos espaciais vetoriais de semântica é a de modelar uma palavra incorporando-a a um espaço vetorial. Por esse motivo, e por conta do termo incorporar do inglês significar *embedding*, a representação de uma palavra como vetor passou a ser denominada como *word embedding* (JURAFSKY; MARTIN, 2017).

As diferentes dimensões destes vetores permitem atribuir semântica aos termos, estabelecendo relações contextuais entre eles. É possível utilizar bibliotecas prontas para implementá-lo (como o *word2vec*), ou criar uma nova camada de processamento própria para os dados em questão.

Algumas formas de se realizar vetorização são apresentadas a seguir:

2.1.3.1 Bag of Words



Fonte: (JURAFSKY; MARTIN, 2017)

Figura 1: Exemplo visual do processo de conversão textual em bag of words

Consiste na conversão direta de uma palavra para um valor numérico. Dado o vocabulário com que se trabalhará, cada palavra deste admite um valor (inteiro), em geral o número de ocorrências da mesma dentro de um dado texto com o qual o vocabulário foi produzido; as sentenças então são convertidas para sequências dos números correspondentes a tais palavras. Esse processo pode ser sintetizado em um conjunto de palavras desordenada, ignorando suas posições no texto e mantendo apenas suas frequências. (JURAFSKY; MARTIN, 2017)

De fato, o processo de tradução de um texto para um saco de palavras (numa tradução direta do termo) pode ser melhor compreendido por meio de um exemplo visual. Um exemplo simples, da linguagem inglesa, de ser entendido é dado pela Figura 1, no qual, ao invés de se representar a ordem das palavras em cada frase como “I love this movie” e “I would recommend it”, simplesmente representa-se que a palavra ”it”ocorre 6 vezes durante todo o excerto, enaquanto ”I”aparece em 5 instâncias, ”love”, ”recommend”e ”movie”em 1 ocasião, e assim por diante.

2.1.3.2 LSI

LSI, abreviação para *Latent Semantic Indexing*, é também conhecida como LSA (*Latent Semantic Analysis*). Consiste numa técnica de redução de dimensionalidade, baseada na técnica de decomposição em valores singulares (SVD).

Isso contribui também com uma diminuição de ruído, uma vez que o *corpus* originado do conjunto de documentos sendo utilizado apresenta muitos termos, alguns pouco ou quase nunca utilizados e outros redundantes. Tal técnica permite encontrar palavras semanticamente próximas, de acordo com suas decomposições no novo espaço de menor dimensionalidade.

2.1.4 Rede neural recorrente

É uma variação da rede neural *feedforward* que permite o tratamento de dados sequenciais (relacionados entre si). Isso é possível por conta do mecanismo de realimentação que apresenta em sua camada intermediária, a qual propaga as informações não só para as camadas seguintes, mas para ela própria também, gerando uma espécie de “memória” no sistema. Para os primeiros testes em aprendizagem de máquina, foram usadas dois modelos de rede neural recorrente: ESN e LSTM.

2.1.4.1 ESN

Sigla para *Echo-State Network*, é um tipo de rede neural recorrente. Ela apresenta uma camada intermediária (conhecida como "reservatório") com pesos arbitrários, seguindo algumas regras algébricas, como certo grau de esparsidade e valor máximo dos autovetores presentes nela. É um modelo que funciona bem para dados temporais (o dado posterior está vinculado com o anterior e assim sucessivamente), pois permite uma "memória" de curto prazo.

2.1.4.2 LSTM

Sigla para *Long Short-Term Memory*, é também um tipo de rede neural recorrente. É composta por "células de memória", com parâmetros que permitem o gerenciamento dos dados e de memória. São eles o *forget gate*, que eliminam informações julgadas não ser mais relevantes no momento; *input gate*, que adicionam informações a serem consideradas no estado da célula; e *output gate*, de onde se extraem as informações de saída, baseadas no estado da célula.

2.1.5 CNN

Sigla para rede neural convolucional, é, assim como a rede neural recorrente, uma variação da rede neural *feedforward*, sendo um algoritmo de *deep learning* muito utilizado principalmente para a identificação e processamento de imagens. A utilização dele para problemas de classificação em NLP provou-se relevante e com resultados interessantes em anos recentes. Sua principal característica se dá por meio de seu *layer* convolucional, o qual utiliza de filtros (ou matrizes) incialmente de valores randomicos, com os quais operações de convolução são realizadas junto dos inputs dados, sejam eles imagens ou um conjunto de palavras em uma dada frase. Assim, a cada iteração, o algoritmo consegue identificar características mais importantes ou relevantes, com a *loss function*, e adaptar os valores utilizados pelos filtros, obtendo um entendimento maior a respeito daquilo que é analisado. Neste projeto, foi utilizada a rede neural convolucional de 1 dimensão, a qual será referida neste documento como *CNN* ou *CNN-1D*.

2.1.6 BERT

O BERT (*Bidirectional Encoder Representations from Transformers*) é um algoritmo de processamento de linguagem natural desenvolvido pela Google, capaz de entender o

contexto, a semântica e a relação das palavras em um texto, sendo utilizada atualmente na ferramenta de pesquisa da empresa. Se baseia numa arquitetura de redes neurais conhecida como *Transformer* e foi usada nos estudos de (HE et al., 2021), para a classificação de comentários como racistas ou não racistas, apresentando bons resultados.

2.1.7 Naive Bayes

É um dos modelos utilizados em *machine learning* para problemas de classificação. Se baseia no cálculo dos estimadores máximos de verossimilhança, utilizando do teorema de Bayes. Ele define a distribuição *a posteriori* de um parâmetro (dado uma variável aleatória X) a partir da sua distribuição a priori, da distribuição condicional de X dado o parâmetro e da distribuição de X, usando na premissa de que as dimensões que definem os dados de entrada são independentes entre si (JAMES et al., 2017).

No caso, partindo do cálculo da probabilidade de uma determinada amostra pertencer a uma classe c dados seus valores de entrada, obtém-se: Sendo $x = (x_1, ..., x_i)$,

$$P(Y = c|X = x) = \frac{P(Y = y)P(X = x|y)}{P(X = x)}$$

E, efetivamente, a classe a qual o dado faz parte é aquela cuja probabilidade $P(Y|X = x)$ é maior. Ou seja,

$$\hat{y} = argmax_y P(Y = y)P(X = x|y)$$

O Naive Bayes é considerado um dos modelos mais simples de se treinar e não exige um grande volume de dados para gerar resultados satisfatórios.

2.1.8 Random Forest

É um algoritmo de *machine learning* de aprendizagem supervisionada, ou seja, que aprende a identificar padrões por meio do treinamento com um conjunto de dados já rotulados. O *Random Forest* pode ser utilizado tanto para problemas de classificação quanto de regressão. A ideia por trás deste método se dá por meio do treino de diversas árvores de decisão, as quais são feitas com amostras da base de dados em estudo escolhidas de forma aletória. Assim, realizam-se predições baseadas nos resultados mais recorrentes em cada árvore.

Uma árvore de decisão é um modelo o qual se baseia em realizar agrupamentos na base de dados por meio de perguntas e *features* dadas, criando nós. Esse processo é repetido

para os subgrupos originados, até que se obtenha um resultado.

Os parâmetros mais importantes a se analisar com relação ao poder preditivo e classificatório deste modelo são: o número de árvores criadas; o número máximo de *features* consideradas ao se criar nós; e o número mínimo de elementos num nó para um novo agrupamento.

2.1.9 SVM

O SVM (*Support Vector Machine*) é um algoritmo de aprendizado supervisionado muito utilizado para problemas de classificação, regressão e deteção de *outliers*. Ele busca uma reta ou hiperplano (o primeiro para problemas lineares e o segundo para problemas com mais dimensões) que separe dois grupos distintos, em problemas de classificação, de modo que a distância para essa separação em relação a cada classe sendo analisada seja a maior possível. Desse modo, o algoritmo classifica novas entradas de acordo com a distância relativa a reta e aos dois grupos.

2.1.10 TF-IDF

É uma medida estatística que busca quantificar a importância de uma palavra considerando o conjunto de documentos (*corpus*) que está sendo analisado. Ela leva em consideração a frequência em que o termo aparece em cada documento ("TF": *term frequency*) e o inverso da frequência em que ele aparece em documentos em relação a todo o *corpus* ("IDF": *inverse document frequency*); quanto mais presente ele é no corpus, menos informativo ele tende a ser e portanto, menos importância deve ser dada a ele, invertendo o valor dessa frequência. Esta medida é importante para identificar similaridade entre documentos e classificá-los através de modelos de *machine learning*.

2.2 Funções de ativação

Funções de ativação consistem em funções que transformam os valores de saída de um nó de uma rede neural em outros, dentro de um intervalo de valores definido. São funções que, na prática, aproximam os nós a terem comportamentos similares a de neurônios, efetivamente, transmitindo determinado sinal (valor, no caso) ao receber um certo nível de energia de ativação; e outro, caso contrário.

2.2.1 ReLU

A ReLU é uma função contínua não diferenciável, cuja definição é:

$$ReLU(x) = max(0, x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{2.1}$$

É uma função fácil de calcular e derivável em quase todos os pontos. Por isso, é utilizada como função de ativação em redes neurais.

2.2.2 Sigmóide

Uma função sigmóide é uma função em forma de "S". Frequentemente, para modelos de redes neurais, utiliza-se a função logística como função de ativação. Ela é descrita como:

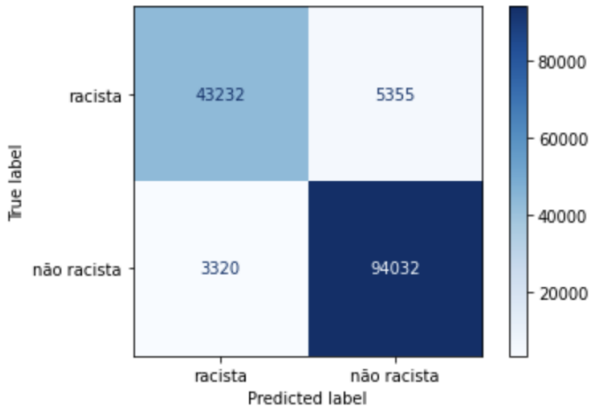
$$\sigma(x) = \frac{1}{1 + exp(-t)}$$

Ela varia, portanto de 0 a 0,5 quando x é menor que 0, e de 0,5 a 1 caso contrário.

2.3 Matriz de confusão

A matriz de confusão é uma ferramenta que auxilia na compreensão e visualização do desempenho dos modelos de *Machine Learning*. Ela consiste numa tabela que representa a quantidade de predições corretas e erradas feitas pelo modelo, sobre o *dataset* de teste, em relação às classes reais às quais as amostras pertencem.

Tomando como exemplo a matriz de confusão de um dos modelos (CNN) produzidos durante o projeto (e que será retomada em seções posteriores), o quadrante superior esquerdo da matriz indica quantas amostras foram corretamente classificadas como racistas por tal modelo. O elemento superior direito representa a quantidade de dados racistas que foram identificados como não racistas. Já o inferior esquerdo contém o número de amostras não racistas classificadas erroneamente como racistas. E por fim, o inferior direito representa a quantidade de dados não racistas do conjunto de teste classificados de fato como não racistas.



Fonte: Autores

Figura 2: Exemplo de matriz de confusão

2.4 Racismo

A discriminação racial é definida pela Convenção Internacional sobre a Eliminação de Todas as Formas de Discriminação Racial (ONU, 1965) como:

> Qualquer distinção, exclusão, restrição ou preferência baseada em raça, cor, descendência, ou origem nacional ou étnica que tem como propósito ou como consequência anular ou restringir o reconhecimento, desfrute ou exercício, em pé de igualdade, de direitos humanos e liberdades fundamentais políticas, econômicas, sociais, culturais ou de qualquer outro campo público da vida.

2.4.1 Racismo amarelo

O racismo amarelo é a discriminação racial contra asiáticos e descendentes de asiáticos da Ásia Oriental. A manifestação de violência, tanto verbal quanto física, aumentou consideravelmente com a pandemia de Covid-19. Nos Estados Unidos, o aumento de crimes de ódio foi de 339% em 2021 em relação a 2020 (NEWS, 2022), e foi possível encontrar comentários racistas vindo até mesmo de presidentes e pessoas influentes em redes sociais.

2.5 API

O significado do acrônimo é dado como *Application Programming Interface*. Desse modo, API é um conjunto de mecanismos que permite a comunicação entre componentes

de softwares por meio da utilização de uma interface. Essa interface poderia ser pensada como um contrato, o qual define como essas aplicações se comunicam com respostas e solicitações.

A visão arquitetural que se utiliza para entender o funcionamento de uma API é por meio dos termos cliente e servidor. Ou seja, a aplicação que deseja obter uma informação ou conjunto de dados seria o cliente, o qual envia requisições ao servidor. Este, por sua vez, é a representação do software que possui as informações desejadas pelo cliente, e a depender do que lhe vem como requisição e a maneira como foi implementado, responde ao cliente com mais ou menos dados em formatos diferentes.

Uma maneira muito comum de se implementar API se dá por meio da arquitetura REST (Representational State Transfer). Nela, define-se um conjunto de operações nomeadas como POST, GET, DELETE, entre outros, os quais usuários podem utilizar para acessar os dados do servidor. Cada uma dessas funções, por definição tem utilizadades pré definidas. Para o trabalho, os mais importantes são POST e GET, métodos para se publicar e buscar um dado na base do servidor, respectivamente. A comunicação entre aplicações, por sua vez, é feita usando o protocolo HTTP, o qual, não será descrito em detalhes por desviar demasiadamente do assunto do projeto. Basta entender que esse protocolo é muito utilizado em aplicações web e dentro de seu cabeçalho são enviados dados, dentre os quais a url e o serviço (POST ou GET no presente caso) requisitado.

2.6 Twitter

É uma rede social onde os usuários podem compartilhar publicações com vídeos, imagens e textos de até 280 caracteres. Atualmente, conta com cerca de 436 milhões de usuários ativos mensalmente (só no Brasil, há aproximadamente 19 milhões de usuários da plataforma), e em 2021, eram publicados 575 mil *tweets* por minuto (STATISTA, 2022). Pelo fato das publicações serem curtas e da rede disponibilizar o acesso a sua API, por onde é possível coletar dados e postar conteúdo de forma automatizada, o *Twitter* se torna também uma plataforma interessante para estudos estatísticos e linguísticos.

3 TECNOLOGIAS UTILIZADAS

O projeto envolve a criação e teste de modelos de *machine learning* para análise de sentimentos, assim como de possíveis elementos de agressão, e identificação de falas racistas contra amarelos em comentários da rede social *Twitter*, além do desenvolvimento de um *bot*. Para a realização do projeto, foram usadas as seguintes tecnologias:

3.1 Python

Tanto para a criação do modelo de *machine learning* quanto do *bot* para o *Twitter*, foi utilizada a linguagem *Python*. Isso pois ela possui diversos pacotes que contribuem para o desenvolvimento dessas aplicações. Dentre eles, podemos citar o *sci-kit learn*, um pacote próprio para a criação de modelos de aprendizagem de máquina, o *TensorFlow* e o *Keras*, com diversas funcionalidades voltadas para a construção de redes neurais.

3.2 Twitter API

Através da API do Twitter, é possível coletar *tweets* de toda a base de dados da plataforma, sob um limite mensal de coleta (500.000 *tweets* por mês), por meio de chamadas que contenham palavras chaves determinadas pelo grupo. Um exemplo de chamada para a pesquisa de tweets com termo "Twitter"poderia ser como se segue, valendo do fato que trata-se de um método do tipo GET, justamente para consulta de dados:

```
curl --request GET
 --url 'https://api.twitter.com/1.1/search/tweets.json?q=Twitter'
```

Essa funcionalidade é importante para aumentar a base de treino e teste do modelo de aprendizagem de máquina, composta inicialmente por base de dados públicas sobre temas semelhantes ao do projeto sendo proposto.

Além disso, a criação do *bot* só se tornou possível por meio do acesso à API, visto

que ela é o meio de acesso para os dados da rede social, permitindo, por exemplo a automação de ações tais como a postagem de mensagens. Esse tipo de ação, entretanto, utilizaria de uma requisição diferente da apresentada acima, do tipo *POST*. Assim, um exemplo de requisição para enviar a mensagem *"hello"* sobre o perfil autenticado, dessa vez tratando-se de um método do tipo POST, é:

```
curl -XPOST
  --url 'https://api.twitter.com/1.1/statuses/update.json?status=hello'
```

3.2.1 Versões da API

A API em questão possui 3 versões. Até o final de novembro de 2022 as versões disponíveis são: v1.1 standard, v1.1 premium e v2. As duas primeiras compartilham os mesmos métodos entre si, e o principal diferencial entre elas se dá por meio de um acesso maior ao banco de dados do *Twitter*. Enquanto a versão standard oferece apenas acesso aos dados referentes aos últimos 7 dias, a premium disponibiliza o acervo do banco de dados dos últimos 30 dias.

A v2 por sua vez, é a API que a rede social deseja tratar como a principal. Nem todas as funções já implementadas nas demais está presente nesta, mas não possui qualquer risco de ser descontinuada, ao contrário dos metodos presentes na v1.1.

Um ponto interessante que diferencia as versões 1.1 e 2, é a nomenclatura utilizada para se denominar alguns elementos presentes na rede social. O termo *status* da 1.1, por exemplo, é equivalente ao termo *tweet* da versão mais nova.

Em geral, para o projeto foram utilizadas chamadas para criação de mensagens, procura de menções a usuários e paginação.

3.2.2 Níveis de acesso

Além das versões disponíveis para uso da API, também existe outro fator que influencia na utilização com maior ou menor restrição dessa tecnologia, que se traduz por meio do tipo de acesso que um dado usuário possui.

Basicamente, existem 3 tipos de acesso: Essential, Elevated e Academic Research. Para se utilizar cada um, existe um pré-requisito diferente, envolvendo o preenchimento de formulários, a fim de informar ao *Twitter* para qual finalidade será o uso de cada acesso pelo usuário. Feito o preenchimento dos dados pedidos, a rede social pode ou não garantir

o acesso desejado.

Simplificadamente, a diferença principal entre os níveis é a quantidade de operações e chamadas permitidas dentro de um mês para coletar *tweets*, sendo a *Essential* a mais limitada, e a *Academic Research* a com menor restrições. Outro ponto relevante se dá por meio a restrição de acessos a métodos da API.

A diferença entre esses acessos pode ser melhor ilustrado por meio da Tabela 1 a seguir (apenas com as informações relevantes ao desenvolvimento do trabalho):

Níveis de acesso			
Característica analisada	Essential	Elevated	Academic Research
Pré-requisito	Registrar uma conta na rede social	Preenchimento de formulário	Preenchimento de formulário
Limite de chamadas por mês	500,000	2 milhões	10 milhões
Requisições a cada 15 minutos	25	50	100
Versões de API disponíveis	apenas chamadas de mídia da standard v1.1 e v2	standard v1.1, premium v1.1 e v2	standard v1.1, premium v1.1, v2 e operações avançadas de busca
Banco de dados	Últimos 7 dias	Últimos 30 dias	Acesso total

Fonte: Autores

Tabela 1: Níveis de acesso da Twitter API e suas permissões

3.3 Tweepy

Para facilitar na manipulação das chamadas de API, o grupo utilizou o pacote em Python *tweepy*. Esse pacote possui diversas funções que realizam chamadas de API, sem que o usuário tenha de escrever toda a requisição. Assim, elimina-se a possibilidade de erros de escrita e diminuindo qualquer tipo de poluição no código, o qual apresenta apenas nomes de funções mais intuitivas.

3.4 Bases de dados públicas

Há bases com dados tanto em português quanto em inglês, porém as primeiras são consideravelmente menores e tratam de discursos de ódio de modo geral, sem foco em racismo. Portanto, as bases desenvolvidas através do estudo em (HE et al., 2021) trouxeram um ótimo ponto de partida para o início do projeto e foram de grande importância para o desenvolvimento dele como um todo.

A base de (HE et al., 2021) é composta por dois conjuntos de dados: o primeiro contém cerca de 2290 tweets, classificados como racistas, antirracistas e neutros. Tal classificação foi feita "manualmente"pelos pesquisadores. O segundo apresenta cerca de 205 milhões de id's de tweets, porém classificados nos mesmos grupos através do modelo de BERT desenvolvido pela equipe de pesquisa. Ambas as bases foram usadas para o desenvolvimento dos modelos de aprendizagem de máquina, em conjunto com mais tweets coletados através da API do *Twitter*. Essa coleta de mais dados foi realizada, e apresentou grande importância para a realização do trabalho, uma vez que a base gerada por (HE et al., 2021) apresenta um viés muito claro para o contexto da pandemia, e ter dados diferentes acrescentou mais parâmetros para análises.

3.5 Google Cloud Platform

O *Google Cloud Platform* (GCP) é uma plataforma que reúne diversos serviços na nuvem, sendo disponibilizada e desenvolvida pela Google. Ele foi utilizado com o intuito de armazenar e realizar a coleta de novos *tweets* em um servidor na nuvem e, assim, enriquecer e melhorar as bases de dados públicas que foram encontradas pelo grupo. Além disso, possui recursos para realizar o deploy das funções referentes ao desenvolvimento do *bot*.

Dentre os motivos principais para a escolha dessa plataforma, pode-se citar a grande variedade de soluções disponiblizadas por ela, além do fornecimento de créditos gratuitos, por meio de contas criadas com vínculo à USP, que permitem a utilização deste serviço.

Os serviços mais interessantes de acordo com o que o grupo considerou mais relevante para o projeto por parte da coleta de dados, ou seja, formas eficientes e de fácil manipulação para armazenamento ou consulta de dados, foram o *Cloud Storage* e o *Compute Engine*. O primeiro é uma opção para armazenamento de dados. O segundo permite a criação de máquinas virtuais as quais poderiam rodar *scripts* para coleta de *tweets*.

Já para o deploy das funções, os serviços que apresentaram um maior valor foram as *Cloud Functions*, *Pub/Sub*, *Cloud Scheduler* e a já mencionada anteriormente *Cloud Storage*. A primeira é um meio de se colocar código na nuvem, de forma que o usuário possa configurar o serviço para acionar o processo por meio de diferentes maneiras, sendo o mais comum uma requisição HTTP. O *Pub/Sub* é um serviço de mensagens assíncrono que utiliza de uma arquitetura na qual existe um *Publisher*, que trasmite eventos, e um *Subscriber*, que recebe eles. O *Scheduler*, por sua vez, é uma ferramenta para configurar *cron jobs*, rotinas automatizadas que são acionadas periodicamente.

Durante a utilização da plataforma encontraram-se algumas dificuldades, principalmente acerca do cadastro de formas de pagamento a ela. Isso porque mesmo utilizando o período de 3 meses gratuítos, ainda foi necessário realizar o cadastro de cartões para cobranças após o fim desse tempo inicial de utilização. Não foram encontradas alternativas nas quais isso não precisaria ser feito.

Outro ponto que levantou dificuldades foi a utilização da máquina virtual do *Compute Engine* para rodar *scripts*. Sempre foi necessária a utilização da função *screen*, a qual permitia que o código rodando permanecesse funcionando mesmo com o fechamento do *shell* da VM.

3.6 Google Colaboratory e Kaggle

Para a execução dos códigos desenvolvidos, tanto para coleta de *tweets* quanto para treinamento e teste de modelos de *Machine Learning* e *Deep Learning*, foram utilizadas duas plataformas online: Google Colaboratory e Kaggle.

O Google Colaboratory é um recurso em nuvem, mantido pela Google, que permite a criação de *notebooks* em Python. Os *notebooks* são documentos que permitem a criação e execução de código (no caso, em Python), intercalados com trechos de texto em Markdown. No Google Colaboratory, é possível executar os programas desenvolvidos em CPU, GPU ou TPU, dentro de um determinado limite de tempo (12 horas, no caso da GPU). Por ser do Google, permite pegar e salvar arquivos diretamente no Google Drive, facilitando o compartilhamento de resultados.

Já o Kaggle é uma plataforma online para *Data Science*. Nela, são realizadas competições na área de Dados, com o desenvolvimento de códigos e disponibilização de *datasets* variados para estudo. É possível criar *notebooks* Python, assim como no Google Colaboratory, e executá-los em CPU, GPU ou TPU. Os pontos positivos do Kaggle é que

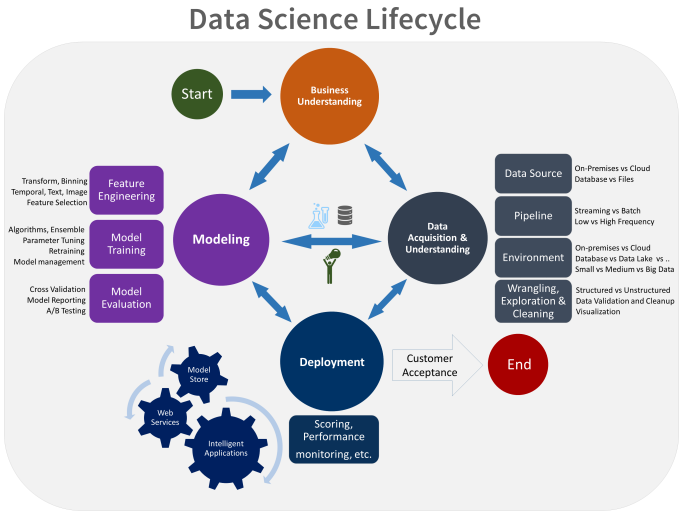
o tempo de uso da GPU é maior (30 horas por semana) e é possível deixar o código executando ao fundo, até terminar, sem necessidade de interagir com a página ou preencher *CAPTCHAs*.

4 METODOLOGIA DE TRABALHO

O trabalho segue uma metodologia a ser descrita neste capítulo. Como parte do método sendo empregado, o grupo organizou e participou de reuniões com profissionais de áreas pertinentes ao projeto, as quais também serão descritas a seguir. O capítulo encerra-se com uma breve explicação acerca dos métodos e testes em Ciência de Dados realizados pelos alunos.

4.1 Fluxo do projeto

O grupo utilizou uma metodologia baseada no ciclo de vida de um projeto de ciência de dados concebida pela *Microsoft*, o TDSP (*Team Data Science Process*) (MICROSOFT, 2022), cujo diagrama representando seus estágios e algumas de suas características encontra-se na Figura 3.

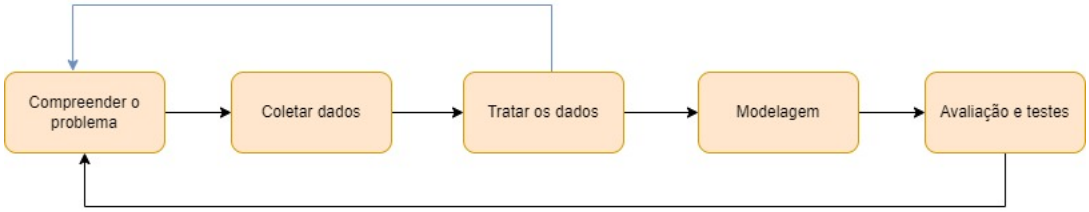


Fonte: (MICROSOFT, 2022)

Figura 3: Diagrama do ciclo de vida de um projeto de ciência de dados pela *Microsoft*

O grupo simplificou o processo e teve também como base o ciclo de vida de um projeto de ciência de dados de (STODDEN, 2020), retirando estágios que envolvam *deploy* e

validação com o cliente, adaptando-o às necessidades do trabalho desenvolvidos. Desse modo, o fluxograma da Figura 4 foi elaborado, a fim de guiar as experiências do grupo com relação ao processo de desenvolvimento do modelo de aprendizado de máquina.

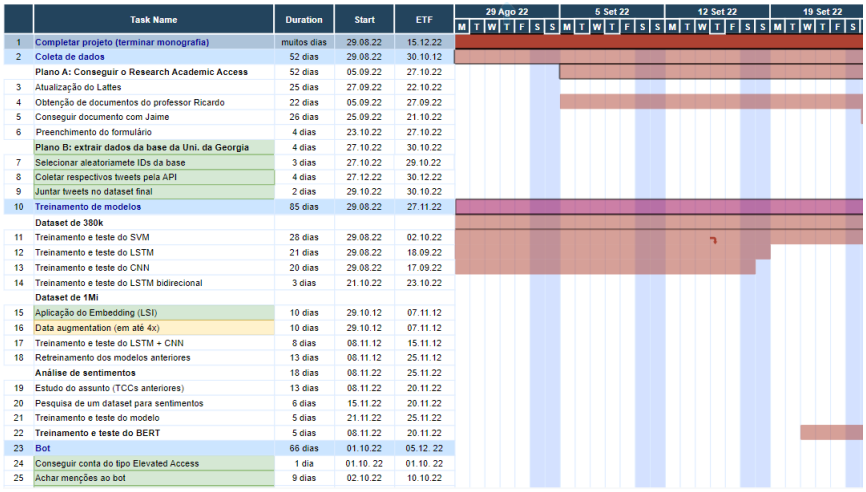


Fonte: Autores

Figura 4: Fluxograma elaborado

Primeiramente, é realizada uma análise do problema a ser solucionado. A partir disso, é feita uma pesquisa para coleta de dados, os quais são tratados e refinados no estágio seguinte. Caso o *dataset* obtido ao final dessas etapas não seja considerado satisfatório, retorna-se à fase inicial de identificação do problema para compreender aquilo que pode ou não ser viável de se desenvolver. Tendo os dados, a modelagem e os testes para avaliar sua eficácia são produzidos.

Todo o andamento do trabalho foi registrado num Diário de Bordo. Conclusões importantes de reuniões, definições de próximos passos e resultados de experimentos foram concentrados no documento para que tanto os demais membros do grupo como o professor orientador pudessem ter visibilidade sobre os avanços feitos. No decorrer do projeto, conforme os estudos e desenvolvimentos foram evoluindo e obstáculos foram surgindo, uma maior necessidade de organização se mostrou necessária. Criou-se então uma Diagrama Gantt, o qual pode ser visualizado na Figura 5 para projetar as próximas tarefas e o tempo limite de execução de cada uma, assim como organizar o que poderia ser paralelizado. O diagrama completo e em maior tamanho encontra-se no Apêndice A.



	Task Name	Duration	Start	ETF
1	Completar projeto (terminar monografia)	muitos dias	29.08.22	15.12.22
2	Coleta de dados	52 dias	29.08.22	30.10.12
	Plano A: Conseguir o Research Academic Access	52 dias	05.09.22	27.10.22
3	Atualização do Lattes	25 dias	27.09.22	22.10.22
4	Obtenção de documentos do professor Ricardo	22 dias	05.09.22	27.09.22
5	Conseguir documento com Jaime	26 dias	25.09.22	21.10.22
6	Preenchimento do formulário	4 dias	23.10.22	27.10.22
	Plano B: extrair dados da base da Uni. da Georgia	4 dias	27.10.22	30.10.22
7	Selecionar aleatoriamete IDs da base	3 dias	27.10.22	29.10.22
8	Coletar respectivos tweets pela API	4 dias	27.12.22	30.12.22
9	Juntar tweets no dataset final	2 dias	29.10.22	30.10.22
10	Treinamento de modelos	85 dias	29.08.22	27.11.22
	Dataset de 380k			
11	Treinamento e teste do SVM	28 dias	29.08.22	02.10.22
12	Treinamento e teste do LSTM	21 dias	29.08.22	18.09.22
13	Treinamento e teste do CNN	20 dias	29.08.22	17.09.22
14	Treinamento e teste do LSTM bidirecional	3 dias	21.10.22	23.10.22
	Dataset de 1Mi			
15	Aplicação do Embedding (LSI)	10 dias	29.10.12	07.11.12
16	Data augmentation (em até 4x)	10 dias	29.10.12	07.11.12
17	Treinamento e teste do LSTM + CNN	8 dias	08.11.12	15.11.12
18	Retreinamento dos modelos anteriores	13 dias	08.11.12	25.11.12
	Análise de sentimentos	18 dias	08.11.22	25.11.22
19	Estudo do assunto (TCCs anteriores)	13 dias	08.11.22	20.11.22
20	Pesquisa de um dataset para sentimentos	6 dias	15.11.22	20.11.22
21	Treinamento e teste do modelo	5 dias	21.11.22	25.11.22
22	Treinamento e teste do BERT	5 dias	08.11.22	20.11.22
23	Bot	66 dias	01.10.22	05.12.22
24	Conseguir conta do tipo Elevated Access	1 dia	01.10.22	01.10.22
25	Achar menções ao bot	9 dias	02.10.22	10.10.22

Fonte: Autores

Figura 5: Gantt do projeto

4.2 Métricas

Para avaliar a performance dos diferentes modelos testados, foram consideradas duas métricas em especial:

- Precisão: representa a proporção de *tweets* corretamente classificados como racistas em meio a todos aqueles que o modelo identificou como tal;
- Recall: representa a proporção de *tweets* corretamente classificados como racistas em meio a todos aqueles que são efetivamete racistas no *dataset*.

Tais métricas foram selecionadas considerando que o modelo seria incorporado ao bot e que há determinados erros de classificação piores de se cometer que outros. O *bot* agindo no mundo real não pode classificar como racista comentários que não o são, para não violar a regulamentação do Twitter com relação a contas automatizadas nem a privacidade de usuários reais da plataforma. Por outro lado, rotular como não racista textos que são ofensivos, apesar de não ser ideal, não gera tantos problemas burocráticos como o primeiro caso de erro mencionado. A prioridade é, portanto, garantir que os *tweets* classificados pelo modelo como racistas sejam efetivamente racistas (precisão), ao mesmo tempo que uma grande proporção de comentários racistas seja corretamente identificada (recall). A métrica F1-score, correspondente à média harmônica entre a precisão e o recall, auxilia a dar uma visão do todo da performance do problema, baseada nas duas métricas que mais importam para o grupo.

Levar em consideração a acurácia (proporção entre a quantidade de *tweets* corretamente classificados e o número total de textos no *dataset*) também é importante, para se ter uma visão geral da performance do modelo. Entretanto, é preciso cautela ao analisá-la, uma vez que pode dar uma impressão errônea da qualidade do modelo no caso de *datasets* desbalanceados e da falta de diferenciação entre as classes "racista" e "não racista".

4.3 Reuniões para melhor entendimento dos temas

Tendo em vista a análise do problema e o desejo de trazer maior valor ao projeto, o grupo procurou realizar reuniões com especialistas em áreas relevantes ao projeto, como antropologia e desenvolvimento de *bots* para o *Twitter*, com o intuito de melhor entender a problemática envolta no tema do racismo. Essas reuniões também possibilitaram discussões de possíveis soluções e aplicações que poderiam ser desenvolvidas para trazer algum valor a sociedade por meio dos estudos pretendidos.

Foram feitas conversas com dois profissionais: Lucas Lago e David Nemer, cada um a respeito de sua respectiva área de atuação, desenvolvimento de *bots* e *discursos de ódio*.

4.3.1 Sobre *bots*

Foram feitas duas consultas com o Lucas, profissional que se especializou em desenvolver *bots* para o *Twitter* e criador do Projeto 7c0, conta automatizada do Twitter cujo intuito é buscar e mostrar *tweets* deletados de figuras políticas. A recomendação pela conversa com ele foi feita pelo professor orientador Ricardo Rocha, o qual possuía o contato do mesmo.

4.3.1.1 Primeira reunião

A primeira reunião foi feita no dia 14 de fevereiro, e tinha como objetivo central tirar dúvidas e discutir possíveis aplicações que pudessem agregar valor ao projeto, tendo como foco a área de atuação do Lucas.

Dessa conversa, foram levantados e expostos as diversas regras e detalhes que deve-se prestar atenção ao se desenvolver um *bot* para o *Twitter*. Desse modo, ficou claro que deveria ser dada a devida importância a linguagem e meio de comunicação que seriam utilizados para transmissão de informação e conteúdo pela rede social. Há uma preocupação da plataforma para que casos de abuso não ocorram, ainda que, de modo geral,

a fiscalização seja subjetiva em certo ponto, visto que o *Twitter* pode revogar qualquer conta sem justificativa. Além disso, mencionar uma pessoa num *tweet* que a condena ou expõe é considerado assédio, trazendo com isso um aspecto adicional a ser considerado.

Outro ponto relacionado às regras que devem ser obedecidas se dá por meio do uso de dados adquiridos pela API da rede. Apenas análises permitidas pela plataforma podem ser feitas e se, porventura, essa limitação for desrespeitada, pode-se ter a conta revogada.

Ademais, foi sugerido pelo Lucas o desenvolvimento de um *bot* que poderia alertar seus seguidores se algum comentário de cunho racista tivesse sido feito. Com os dados obtidos por esse robô, seriam publicadas estatísticas e *dashboards* no próprio *Twitter*. Ainda que a ideia de criação de *dashboards* seja promissora, não foi levada adiante, devido ao enfoque original do projeto em classificação de mensagens.

Também foram discutidas possibilidades de auxílio que o Lucas poderia oferecer ao projeto, seja com contatos, informações e até mesmo a disponibilização de um servidor *AWS*, se o trabalho seguisse uma linha de agregação de valor à sociedade.

4.3.1.2 Segunda reunião

A segunda reunião, realizada no dia 23 de fevereiro, foi proposta a fim de se fazer uma discussão acerca de qual seria o público alvo do robô a ser desenvolvido. Concluiu-se que não faz sentido produzir uma ferramenta que alertasse apenas o usuário que evita o racismo, uma vez que ele já foi conscientizado da gravidade que são os atos racistas.

Com isso, foi proposta a ideia de se utilizar seguidores para buscar *tweets* ofensivos que poderiam ser utilizados pelo modelo, ao mesmo tempo que se possibilita aos usuários de chamar o robô para analisar discursos de ódio dentro da rede social. Desse modo, nenhuma regra de assédio seria quebrada, e a aplicação agregaria maior valor a sociedade, atingindo uma maior parcela do população em geral.

4.3.2 Sobre discursos de ódio

Por conta do tema do projeto envolver o ódio e análise de sentimentos em discursos preconceituosos, o professor orientador, Ricardo Rocha, indicou o David Nemer para explicar o assunto e direcionar o andamento do trabalho desenvolvido. O contato foi feito por meio do Lucas Lago.

David Nemer é um professor assistente no Departamento de Estudos de Mídia na *Uni-*

versity of Virginia, pesquisador, etnógrafo e antropólogo, especialista em antropologia da informática. O conhecimento compartilhado por ele durante a reunião do dia 3 de março foi fundamental para consolidação do conceito de discursos de ódio, além de limitações e desafios que o modelo poderia vir a enfrentar.

O maior desafio apontado por David é a classificação de uma mensagem como ofensiva ou não. A falha nessa distinção pode acarretar a danos morais muito grandes e por se tratar de uma parte mais delicada do trabalho deve-se levar em consideração a possibilidade de tornar esse processo manual.

Sobre discursos de ódio, explicitou-se as diferenças entre discursos de ódio voltados a diferentes etnias, sendo que o fator comum entre todos eles se dá por meio da ignorância da população. Ainda pôde-se abstrair que o *Twitter*, assim como outras redes sociais, replicam as normas comunitárias, e portanto, é complicado fazer com que a plataforma se engaje no tema.

Dentro do escopo de visibilidade de projetos foram levantados pontos interessantes. Em geral, projetos mais técnicos e que envolvam menor subjetividade, trabalho manual, possuem maior visibilidade e aceitação.

4.4 Estudos de Ciência de Dados

Foi feito um estudo inicial sobre Ciência de Dados por meio dos livros (BISHOP, 2006) e (JAMES et al., 2017). Com isso, o grupo passou a experimentar e testar o modelo de *Naive Bayes*, definido no capítulo 2, por se tratar de um modelo mais simples e que não requer uma base de dados muito extensa. Além disso, outros modelos foram testados posteriormente: *Random Forest*, *LSTM* e *ESN*. Mesmo que os resultados obtidos com eles não tenham sido tão bons, estes testes foram úteis para que o grupo pudesse se familiarizar com os algoritmos de *machine learning*.

Como descrito no capítulo 3, utilizou-se a linguagem de programação *Python* e o pacote *sci-kit learn*. O código foi desenvolvido num *Jupyter Notebook* e o *dataset* usado foi o menor encontrado no artigo (HE et al., 2021) denominado *COVID-HATE*, que conta com cerca de 2200 *tweets* classificados como racistas, antirracistas ou neutros. Nos modelos criados, 2000 *tweets* foram usados para treino e *200* para teste.

Vale ressaltar que uma dificuldade encontrada pelo grupo se deveu ao pré-processamento dos dados, uma vez que os dados a serem processados são frases e seus respectivos rótulos, enquanto os métodos do pacote *sci-kit* aceitam entradas numéricas. A fim de contornar

esse problema, utilizou-se a medida estatística definida anteriormente, *TF-IDF*.

4.4.1 Testes

O trabalho foi realizado ao longo de ciclos, segundo a metodologia *TDSP*. Inicialmente, alguns testes mais simples foram feitos e, conforme mais dados de treino foram adquiridos, assim como mais experiência e entendimento do problema por parte do grupo, novos ciclos de projeto aconteceram. Nesta subseção, são explicados como se deram estes ciclos iniciais, de entendimento de contexto e experimentação.

4.4.1.1 Testes iniciais

A fim de se familiarizar com as ferramentas e algoritmos de *machine learning*, o grupo criou alguns modelos mais simples, sem um tratamento de dados rigoroso. Foram testados o Naive Bayes (utilizado como *baseline*), o *Random Forest*, LSTM e ESN. Tais testes foram realizados com um dataset pequeno (cerca de 2200 amostras), uma vez que mais *tweets* foram sendo coletados ao longo dos meses.

O primeiro método de *Naive Bayes* testado foi o Gaussiano, por ser o mais simples dos métodos disponíveis no pacote. Sua acurácia com *stopwords* foi de 0.679, enquanto que sem as *stopwords*, esse número subiu para 0.686. Posteriormente foi feito o mesmo teste com o método multinomial, com o qual obteve-se acurácia com *stopwords* de 0.762 e 0.727 sem elas. Por fim, calculou-se a precisão do método multinomial, o qual obteve um resultado de 0.817, com 0.562 de *recall* e 0.575 de *F-beta score*.

A descrição de cada função foi retirada da documentação oficial do pacote *scikit-learn* (DEVELOPERS, 2022). Os métodos Gaussiano e multinomial foram testados por meio das funções *GaussianNB* e *MultinomialNB*. Já o pré-processamento por *TF-IDF* foi realizado com a função *TfidfVectorizer*.

Tanto o ESN quanto o LSTM não obtiveram boas acurácias: aproximadamente 61% em ambos os casos. A pouca quantidade de dados de treino e o desbalanceamento da base de dados como um todo podem, muito provavelmente, ter influenciado nos resultados.

O Random Forest obteve uma acurácia por volta de 80%. Alterando-se o número de árvores, observou-se que não há ganho em acurácia, mantendo-se em 80%. Ao se diminuir o número para um valor abaixo dos 10, contudo, a acurácia passou a atingir valores que variam entre 70 e 77%. Com respeito ao número de elementos mínimo para divisão da árvore e o número máximo de *features*, manualmente, não foi observada uma variação

notória para com o resultado obtido.

Com relação ao SVM, testou-se a utilização do conjunto de bibliotecas nltk (*Natural Language Toolkit*) para realizar o pré-processamento da base de dados que seria analisada pelo modelo. A ideia se deu pois apresentava-se como uma ferramenta bem utilizada na área de processamento de linguagem natural. O pré-processamento fez uso de funções para *tokenizer* e *lemmatizer* do conjunto nltk, junto do *TF-IDF* da biblioteca *scikit-learn*. Entretanto, a acurácia obtida não se apresentou satisfatória, com apenas 36% de acerto. Posteriormente, ainda se testou sem as funções dadas pelo nltk, tratando os dados com o *tokenizer* presente na API *Keras*, assim como o *pad-sequences*, a fim de garantir o mesmo tamanho para os vetores representando as frases e facilitar o processamento pelo modelo. Com isso, houve um aumento na acurácia, que apresentou valores próximos de 43%, ainda não sendo um valor que possa ser considerado razoável.

4.4.1.2 Modelos de Deep Learning iniciais

Treinamentos com datasets maiores foram realizados, para a construção de modelos de *deep learning*: LSTM e CNN. Inicialmente esses testes foram feitos (treinamento, validação e teste) com um dataset de 378 mil *tweets*, uma vez que foi considerado um número, ainda que baixo, aceitável para um início de testes. O aumento da base de dados no início desse processo apresentou impeditivos, tais como o número limite de *tweets* coletados, restrito a apenas uma semana anterior ao momento de coleta, e por isso, os testes foram iniciados, ainda que com um *dataset* de número longo do ideal.

Vale ressaltar que a implementação do CNN apresentou alguma dificuldades, muito por conta da demora que cada época precisava para processar os dados. Além disso, notou-se que a utilização de uma base de dados menor piorou consideravelmente o desempenho do algoritmo.

O LSTM apresentou uma acurácia de 75,8% (recall de 70% e F1-score de 66,5%) e o CNN teve acurácia de 91,7% (recall de 87,2% e F1-score de 87,8%). De modo geral, percebeu-se que aumentando o volume de dados de treino e balanceando o *dataset*, o modelo passa a performar melhor.

Testes com LSTM bidirecional também foram realizados, porém as métricas não atingiram o mesmo nível do LSTM unidirecional: com 42 épocas, a acurácia foi de 65,3%, precisão de 35,7% e recall de 2%.

4.4.1.3 Conclusões dos primeiros testes

Todos estes testes foram bastante valiosos para equipe, não necessariamente pelos resultados numéricos com eles obtidos, mas pelas aplicações e experimentações sobre pré-processamento de dados e de modelos de *Machine Learning*. Por conta das limitações de tempo para o desenvolvimento do projeto, foram selecionados apenas alguns modelos e determinadas técnicas de processamento e limpeza de dados para serem trabalhados mais a fundo.

Critérios como tempo de treinamento e presença na literatura (em especial, em trabalhos de conclusão de curso anteriores) foram os principais para tal seleção. O SVM leva um tempo muito maior que os demais modelos para ser treinado, mesmo utilizando *kernels* mais simples; enquanto o ESN ainda não foi muito explorado em Processamento de Linguagem Natural, assim como o Random Forest, e não conta com bibliotecas de auxílio para ser implementado, devendo ser construído à mão e dificultando a exploração do modelo. Por isso, foram eliminados do grupo de arquiteturas a ser testado ao longo do projeto.

4.5 Validação de resultados

Após os testes iniciais, onde foram definidos os modelos a serem testados de fato (Naive Bayes, LSTM, LSTM bidirecional, CNN, LSTM seguido de CNN, e LSTM bidirecional seguido de CNN, como detalhado em seções posteriores do documento), com os *datasets* completos e robustos, estabeleceu-se também como seriam testados. Através do *Random Permutation Cross Validation*, o conjunto de dados é dividido aleatoriamente em três seções: a primeira contendo 80% das amostras, a segunda com 10% e a terceira também com 10%, sem intersecções. Elas são usadas, respectivamente, como *datasets* de treino, validação e teste para um modelo. A cada iteração desta técnica de validação cruzada, uma nova divisão dos dados é formada e, por envolver parâmetros aleatórios, existe a possibilidade de uma mesma seção, em iterações diferentes, possuir amostras em comum. Entretanto, com um grande volume de dados no conjunto original, tais intersecções tendem a ser de menor frequência, não interferindo em conclusões finais sobre os resultados.

A ideia é gerar cinco iterações do *Random Permutation Cross Validation*, dando origem portanto a cinco versões de divisões do conjunto total de dados em *datasets* de treino, validação e teste. Cada uma das arquiteturas de *Machine Learning* propostas foi treinada, validada e testada cinco vezes, uma com cada conjunto de *datasets* gerados anteriormente.

Desta forma, foi possível criar versões dos modelos e de suas métricas de desempenho, por meio das quais verifica-se a consistência dos resultados obtidos e a legitimidade dos mesmos, comparando-os.

Dentre os cinco valores para cada métrica, é tomado o mediano. Assim, as medianas de acurácia, precisão, recall e F1-score são utilizadas para se comparar diretamente as arquiteturas e para se eleger o modelo com melhores resultados. Esperava-se que tais métricas medianas superassem às do modelo de *baseline*, o Naive Bayes Multinomial. Por isso, essas foram comparadas direta e principalmente com as dele.

Como referência para cada arquitetura, foi utilizada a versão que mais se aproxima das métricas medianas, considerando as cinco iterações realizadas. É ela a presente no site do projeto, disponível para teste livre de classificação de *tweets*.

4.6 Coleta de dados

A coleta de novos dados por meio da API do *Twitter* se fez necessária para o aumento da base de dados, composta apenas por *datasets* públicos e que podem conter algum tipo de viés. Desse modo, essa coleta poderia proporcionar novas visões e conclusões que antes poderiam ter análise dificultada com um número limitado de dados.

Para isso, foi utilizada a API do *Twitter*, como mencioando no tópico correspondente encontrado na seção de tecnologia utilizadas. A fim de se filtrar os *tweets* coletados pela API, e assim, permanecer com uma base de dados com conteúdo pertinente ao projeto, o grupo decidiu por usar *hashtags*, palavras-chaves e expressões que fizessem menção ou insinuassem um contexto evolto por racismo amarelo.

Desse modo, primeiramente foram consultadas as expressões utilizadas pelo artigo (HE et al., 2021), as quais geraram a base de dados pública em estudo no mesmo. A princípio, foram retiradas palavras que trouxessem um foco maior à pandemia, tais quais *chinese virus* e *wuhan virus*, uma vez que o projeto visa coletar visões racistas de forma geral. Se os resultados obtidos por meio dessa abordagem não tivessem sido suficientemente satisfatórios, o grupo planejava incluir os termos excluídos. Entretanto, isso não se viu como necessário ao longo do trabalho.

Em seguida, foi feita uma pesquisa sobre quais seriam as expressões em inglês que tivesesem conotação perjorativa e direcionada à população de etnia amarela. Os alunos partiram do pressuposto de que o *bot* reagiria a termos racistas voltados à população do extremo oriente num primeiro momento, visto que a maioria dos estereótipos se dá por

meio dessa parcela da população asiática. Dessa maneira, consultou-se (Hatebase Inc., 2022), o qual disponibiliza uma base de dados com termos pejorativos direcionados a diversas etnias nos mais variados idiomas. Foram selecionadas as expressões com maiores incidências na internet, em inglês, de acordo com a própria base de dados.

Ao longo do processo de coleta de dados por meio da API do *Twitter*, entretanto, o grupo encontrou problemas para conseguir um número significativo de *tweets* a fim de formar a base de dados desejada de cerca de 1 milhão de mensagens. Com isso, concluiu-se que seria interessante a coleta de dados provenientes da época em que a pandemia de COVID-19 teve seu início nos EUA, uma vez que tratou-se de um período no qual o discurso de ódio contra amarelos teve um aumento de incidências em redes sociais de modo geral. Para tanto, foi necessário requisitar o *Academic Research Access* à conta dos integrantes do grupo, com o intuito de garantir acesso aos tweets desse período específico.

Todavia, para receber esse tipo de acesso, é necessário preencher um formulário com informações pertinentes a justificar a necessidade desse tipo de conta. O preenchimento dele foi feito, analisado pelo *Twitter*, mas teve sua requisição negada, sem um feedback com maiores explicações. Com isso, o processo torna-se mais desgastante tendo de se revisar todas as respostas dadas para a empresa.

Além disso, dentre as informações pedidas, tem-se a descrição de um projeto sendo desenvolvido, o qual utilizaria desse acesso maior, e uma comprovação de que o usuário da conta possui alguma relação com ele, seja por meio de sites ou documentos online que contivessem o nome do projeto e do aluno. Por se tratar de um projeto de conclusão de curso, sentiu-se grande dificuldade em realizar essa tarefa, uma vez que ainda não há páginas ou documentos oficiais que comprovem essa ligação entre aluno, projeto e instituição de ensino.

Por recomendação dos professores responsáveis pela matéria de conclusão de curso e com o auxílio do professor Ricardo Rocha, foi elaborado um documento que declara a participação do grupo na disciplina e na concepção do projeto. Para oficializá-lo, foi preciso pedir a assinatura do professor Jaime Simão Sichman, chefe do Departamento de Engenharia de Computação e Sistemas Digitais (PCS), e que prontamente o fez, quando requisitado.

Contudo, mesmo com esses esforços, a rede social ainda rejeitou o pedido para que o acesso requisitado fosse liberado. Assim, para contornar este entrave e ainda enriquecer a base de dados, foram coleatdos 999346 *tweets* já rotulados da base dados maior (de cerca de 2 milhões de frases) encontrada em (HE et al., 2021). 333461 *tweets* coletados pela API

foram posteriormente rotulados utilizando de um modelo BERT o qual sofreu processo de *fine-tuning*. Ou seja, foi treinado novamente com os dados de treino pertinentes ao qual o grupo tinha acesso.

4.7 Conta automatizada do Twitter

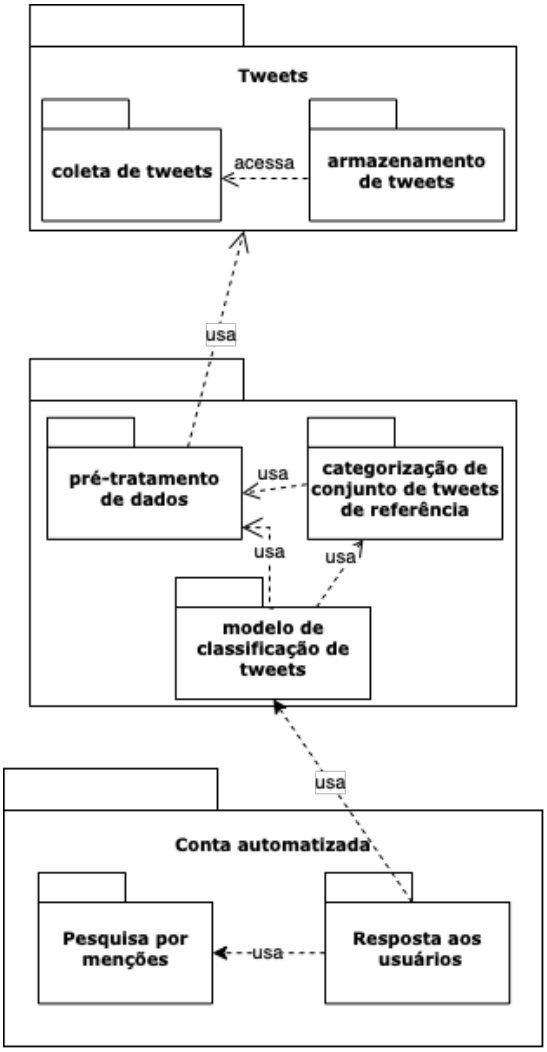
Com o intuito de promover uma aplicação prática do modelo de *Machine Learning* desenvolvido, foi criado um *bot* para o Twitter. O *bot* consiste numa conta automatizada na plataforma, capaz de postar e responder *tweets* conforme condições pré-determinadas pelo desenvolvedor, predominantemente relacionadas com identificação de padrões, palavras chaves e menções à conta em questão.

Para desenvolvê-lo e testá-lo na rede social (em pequena escala, inicialmente), foi necessário obter um novo tipo de acesso à API do Twitter (chamado *Elevated Access*). A solicitação para tal acesso foi realizada pelo grupo por meio do preenchimento de um formulário, no qual deve-se indicar para qual finalidade o novo acesso seria utilizado. Uma vez preenchido e enviado, a solicitação foi rapidamente aceita pela plataforma.

Este novo acesso atualiza as chaves e tokens dos projetos atrelados a conta, de forma que o usuário possa, além de coletar dados, escrever e alterar *tweets* de sua autoria. Ainda garante a utilização da versão 1.1 completa da API (parte dela ainda é utilizável com o acesso mais básico), que possui algumas funções importantes não presentes nas demais versões, tal como a *Cursor()*, a qual faz o processo de paginação para o método de procura por palavras-chave.

5 ETAPAS E ESPECIFICAÇÃO DE REQUISITOS DO PROJETO

Conforme avaliado pelo grupo, o projeto pode ser dividido em três grandes partes: coleta de dados, construção do modelo de *machine learning* e construção da conta automatizada no *Twitter*. O melhor cenário seria que elas ocorressem sequencialmente, dado que o treinamento de um modelo só é possível havendo uma quantidade razoável de dados disponíveis, e que o desenvolvimento do *bot* pode ser melhor estruturado tendo em vista as capacidades e qualidade de resultados do modelo construído. Todavia, o que se observou foi, devido ao tempo disponível aos integrantes, a necessidade de paralelizar essas partes. Isso foi feito por meio do treino de modelos com bases de dados menores, enquanto a coleta não era terminada, a fim de verificar qual o impacto de bases de dados com volumes diferentes. A respeito do *bot*, seu desenvolvimento e testes com funcionalidades básicas de automação, assim como seu deploy, puderam ser feitos concomitantemente às demais frentes do projeto. A Figura 6 representa simplificadamente a relação entre as partes do trabalho.



Fonte: Autores

Figura 6: Diagrama de pacotes simplificado

Para o desenvolvimento do projeto, foi necessária a coleta de *tweets*. Dado o viés da base de dados de (HE et al., 2021) para o contexto da pandemia, a obtenção de novos dados se viu como essencial para que contextos outros que não o mencionado pudessem ser analisados com sucesso. Tais dados foram coletados através da API do Twitter, utilizando "filtros" que auxiliaram na identificação desses *tweets* e podem ser encontrados na seção 5.1 do trabalho. Esses filtros foram baseados em *hashtags* e expressões específicas que fazem referência à população asiática e descendentes, conforme descrito anteriormente na Metodologia de Trabalho.

O teste de diferentes modelos de aprendizagem de máquina e comparação de suas performances foi igualmente importante para o desenvolvimento do projeto. Independentemente do modelo que seria utilizado, o pré-tratamento dos dados é necessário, considerando o grande volume de palavras e frases no *dataset*. Como não é possível ler e classificar

manualmente cada *tweet* coletado, tendo em vista o prazo de entrega do trabalho, apenas uma pequena parcela do *dataset* foi manualmente rotulada, enquanto os outros dados tiveram seus rótulos definidos com a aplicação do *BERT*, modelo pré treinado, com uma boa solidez no que se refere a resultados obtidos na comunidade científica. Esta parte do projeto consiste na experimentação de diferentes modelos de aprendizagem de máquina e combinação deles, para a comparar e entender qual tipo de modelo melhor se encaixaria na tarefa de classificar bem os *tweets* encontrados como racistas ou não.

Por fim, com relação a conta automatizada do Twitter, além das questões técnicas ligadas a seu desenvolvimento, seja em ambiente de desenvolvimento local ou na nuvem, existiu a necessidade de estudo sobre sua usabilidade. O *bot* deveria ser acessível e simples de usar, uma vez que seu intuito é auxiliar na conscientização de pessoas sobre o tema do projeto. Além disso, foi preciso refletir sobre o teor das postagens que ele realizará quando chamado (tipo de linguagem e temática das mensagens), pois é preciso respeitar a regulamentação da plataforma quanto às formas de comentário e conteúdo. O mesmo se deu em relação ao desenvolvimento do site.

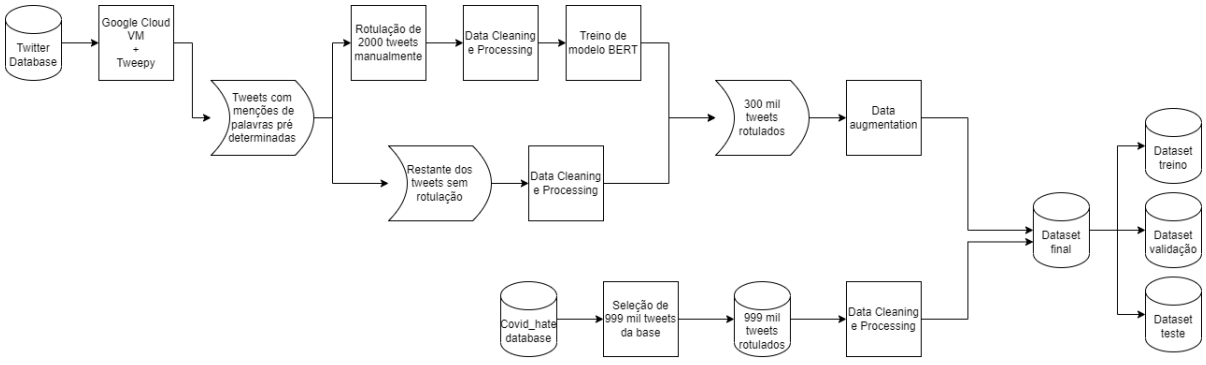
6 PROJETO E IMPLEMENTAÇÃO

O desenvolvimento do projeto depende da realização das seguintes etapas: coleta dos dados, construção e treinamento dos modelos e construção do *bot*. A seguir, estão explicitados como se deu o desenvolvimento de cada uma das partes.

6.1 Obtenção e Estruturação dos Dados

Os dados utilizados para o treinamento, teste e validação dos modelos de *Machine Learning* e *Deep Learning* criados vêm de duas fontes: base de *tweets* "covid_hate" ((HE et al., 2021)) e diretamente do *Twitter API*. Os dados da primeira fonte foram coletados durante o período da pandemia de Covid-19 e possuem termos ou *hashtags* específicos relacionados à pandemia e à comunidade asiática (chineses, principalmente). Eles já estão rotulados em "racistas", "antirracistas" e "neutros". Tal classificação foi feita manualmente pelos pesquisadores envolvidos no projeto e por modelos de *Deep Learning* auxiliares como o *BERT*. Para compor os datasets a serem utilizados no projeto de conclusão de curso, foram selecionados aleatoriamente 999.346 deles, sendo 423.848 racistas e 575.498 não racistas.

Como mencionado anteriormente, para complementar o *dataset* do projeto, foram



Fonte: Autores

Figura 7: Fluxo dos dados para obtenção do dataset utilizado

coletados através da API do *Twitter* outros 333.461 tweets. A coleta se deu através da biblioteca *Tweepy*, que abstrai requisições da API para comandos Python, e de filtros para a seleção de *tweets*. Esses filtros correspondem a termos e *hashtags* usados para referenciar asiáticos de forma depreciativa ou demonstrar apoio à luta contra o racismo. As expressões racistas foram selecionadas com auxílio do site Hatebase ((Hatebase Inc., 2022)), que registra as diferentes ofensas usadas na Internet contra as diversas etnias, nacionalidades, religiões, gêneros e sexualidade. Todos os termos utilizados, por sua vez, foram listados na Tabela 2.

Deles, 1916 foram classificados manualmente pelo grupo e por pessoas externas ao projeto. Os textos foram rotulados em "racistas", "antirracistas", "neutros", "inadequados" e "indefinido". Estas duas últimas categorias foram criadas por conta da relevante quantidade de *tweets* relacionados a conteúdo adulto ou incompreensíveis (seja pela forma em que estão escritos ou por falhas gramaticais que impedem a correta interpretação do texto), respectivamente. As amostras enquadradas nestas categorias foram descartadas. O restante dos *tweets* coletados foram rotulados com o auxílio do modelo de *Deep Learning* BERT.

Ao final, juntando-se os *tweets* obtidos por meio das diferentes fontes e tendo em vista o desbalanço entre a quantidade de comentários racistas e não racistas, foram selecionados aleatoriamente as amostras para comporem os datasets oficiais do projeto, de forma que a proporção entre *tweets* racistas e não racistas fosse 1:2. Desta forma, o base de dados final é composta por 486.461 comentários racistas e 972.932 não racistas, totalizando 1.459.393 mensagens. O fluxo lógico completo a respeito da obtenção do *dataset* final é ilustrado na Figura 7.

Termos usados no filtro de coleta de tweets	
Termos	chink, gook, gookette, mongoloid, goloid, whoriental, rice nigger, dog eater, ching chong, slant eye, chinaman, zipperhead, jap/japs, butterhead, asian, churka, niakoue
Hashtags	#ChinaDidThis, #ChinaLiedPeopleDied, #FuckChina, #MakeChinaPay, #WashTheHate, #BeCool2Asians, #StopAAPIHate, #ActToChange, #VeryAsian

Fonte: Autores

Tabela 2: Listagem de termos utilizados como filtro para pesquisa de novos *tweets*

Dos tweets coletados, apenas seus conteúdos textuais foram utilizados para compor os *datasets* de treino, teste e validação, além do rótulo a eles dados. Além disso, apesar dos textos terem sido classificados em três categorias distintas (excluindo as descartadas), ela foi reduzida para duas: racista e não racista (esta engloba tanto *tweets* antirracistas quanto neutros).

6.2 Pré-processamento dos Dados

Para inserir os *tweets* nos modelos de aprendizagem de máquina, os dados foram pré-processados. Cada modelo admite um tipo de dado de entrada distinto, entretanto em todos foram utilizados os seguintes procedimentos:

6.2.1 Letras minúsculas

Cada um dos *tweets* teve suas letras transformadas para minúscula. Isso auxilia na uniformização da grafia das palavras e consequentemente na identificação delas ao longo de todo o *dataset*.

6.2.2 Remoção de URLs e menções

Muitos *tweets* possuem menções a outros usuários da rede social (indicadas pelo caractere "@" como prefixo) ou links direcionando para páginas web, vídeos e imagens (iniciando com "http" e possuindo exatamente 27 caracteres). Esses elementos, de forma geral, não agregam valor semântico relevante para que os modelos necessitem saber diferenciá-los ou levá-los em consideração de alguma forma. Por isso são removidos antes de inserí-los no modelo.

6.2.3 Emojis

O uso de emojis em textos de redes sociais trazem valores semânticos relacionados a sentimentos ou mesmo substituem determinadas palavras no meio de uma frase. Dado que podem aparecer em sequência ou aglutinado com outras letras, sem espaçamentos, foram criadas funções responsáveis por separar devidmante um emoji do outro e de outras expressões, com o auxílio do pacote em Python *emoji*.

6.2.4 Tokenização e Padding

A tokenização consiste na tranformação das palavras em valores numéricos, que podem ser interpretados e manipulados matematicamente em matrizes. A cada palavra é associado um número distinto, de forma que os *tweets* passam a assumir a forma de um vetor de valores inteiros. Para as implementações do modelo LSTM sem uso de LSI, foi necessário também o uso do *padding*.

O *padding* consiste numa forma de padronizar os dados de entrada, de forma que todos eles possuam o mesmo tamanho. No caso, uma vez que cada palavra corresponde a um valor após a tokenização, foram adicionados "0" ao fim dos *tweets*, de forma que todos eles, ao final do processo, tivessem o mesmo tamanho de 280 elementos. O valor 280 foi escolhido dado que o número máximo de caracteres em uma mensagem no Twitter é justamente 280, portanto, no pior dos casos, a amostra teria 280 elementos semânticos distintos.

No caso de modelos com LSI, este procedimento não foi necessário. Cada sentença foi reduzida a um vetor de dimensão 200, sendo ele o dado de entrada para tais modelos.

6.3 BERT

A fim de classificar os *tweets* coletados pelo grupo, testou-se o modelo BERT para classificação. Existem diversas versões do modelo pré-treinado, que variam quanto a número de camadas, hiperparâmetros e pesos. Atualmente, existem versões para textos em inglês e mandarim, dentre outras línguas, o que compreende o escopo do projeto.

Para treinar um novo modelo baseado em uma das versões do BERT, é preciso tempo, que varia conforme seus hiperparâmetros. Quanto mais camadas e nós, maior o tempo para rodar uma época. Para os primeiros testes, foi utilizado o bert_en_uncased_L-2_H-128_A-2, com 5 épocas e *learning rate* 0,00001. Entretanto, as métricas obtidas, com relação ao dataset de teste com *tweets* de (HE et al., 2021), foram:

- Loss: 3.4969160556793213
- Accuracy: 0.42904967069625854

Esses números pouco mudaram ao se alterar o número de épocas, *learning rate* ou o modelo pré treinado. Visto que a acuidade apresentou resultados muito ruins, abaixo

dos 50%, concluiu-se que tratava-se de um erro no código ou na forma como estava sendo utilizado. Após um período de inspeção das entradas e saídas dos dados que estavam em uso, verificou-se que em um dado momento um *Tensor* (agrupamento de dados em tamanho e forma específicos) estava dentro de outro *Tensor*. Visualmente o que se encontrava era o seguinte:

```
Tensor[Tensor[n1, n2, ... ]]
```

Mas se esperava trabalhar com os dados da seguinte maneira:

```
Tensor[n1, n2, ... ]
```

Outro ponto que fez a acurácia ter um valor tão baixo se deveu aos rótulos que foram dados. Para o modelo do BERT, frases de sentimentos positivos eram aqueles com rótulo igual a 0, e aqueles de sentimento negativo eram rotulados com 1. O grupo, contudo, havia rotulado *tweets* racistas com 1 e não racistas com 0. Essa discordância no que se refere a semântica compreendida pelo modelo fez com que o seu aprendizado fosse subaproveitado.

Corrigindo esses problemas, testou-se mais algumas vezes com variação nos hiperparâmetros, para verificar qual modelo seria o utilizado. O que for apresentado na Tabela 3 são apenas os modelos que apresentaram melhor desempenho, para não poluir oconteúdo aqui apresentado. Vale ressaltar que a alteração em 2, 4, 8, 16 e 32 no número de *batch* (que ficou fixado em 4), não alterou os resultados de forma substancialmente visível, e por isso não será analisado aqui.

O modelo bert_en_cased da Tabela 3 é aquele com L-12_H-768_A-12 e o modelo small_bert com L-4_H-512_A-8.

Comparativo dos modelos						
№	Modelo pré treinado	Épocas	Learn rate	Dropout	Acurácia	Loss
1	bert_en_cased	5	2e−5	0.5	0.85348	0.89227
2	small_bert/bert_en_uncased	5	2e−5	0.1	0.85958	0.72349
3	small_bert/bert_en_uncased	5	5e−5	0.5	0.84737	0.94705
4	Albert_en_base	5	2e−5	0.5	0.85714	0.84268

Fonte: Autores

Tabela 3: Métricas dos treinamentos para o BERT

O modelo pré-treinado oficialmente utilizado para o projeto foi o *BERT en cased* base,

o primeiro apresentado na Tabela 3, por se tratar do que possui melhor acuidade.

6.4 Data Augmentation

Foram coletados através da API do Twitter 333.461 comentários. Dada a importância de um grande volume de dados de treino para modelos *Deep Learning*, buscou-se realizar um aumento da quantidade de dados para que, ao final do processo, a quantidade de documentos disponíveis para treino – além da validação e do teste – fosse mais de 1 milhão.

Das técnicas de Data Augmentation para dados textuais, foram estudadas mais a fundo três: substituição, inserção e tradução. A substituição consiste da troca de determinados termos de uma sentença por outros semelhantes. No caso de modelos computacionais, tais semelhanças se dão pela proximidade vetorial das palavras dentro de um espaço hiperdimensional definido através de um corpus pré-definido. A inserção consiste na adição de termos no meio da sentença. Por fim, a tradução consiste na conversão de um texto para uma outra língua, e na tradução de volta para a língua original. Dessa forma, o texto mantém uma carga semântica parecida com a do documento original.

Dentre as técnicas analisadas, foi escolhido trabalhar-se com a substituição somente. Isso porque, dado que as técnicas seriam aplicadas automaticamente, sem intervenção humana, elas estariam propensas a falhas e alterações inconvenientes dos textos. A técnica da tradução, apesar de ser mais sofisticada, não garante um resultado fiel ou utilizável, dado que os rótulos originais seriam apenas replicados para os textos novos, sem conferência alguma. Considerando ainda que muitos termos racistas são ambíguos ou muito específicos, o controle sobre os novos dados estaria bastante prejudicado. No caso da inserção de termos, o conteúdo das frases poderia acabar sendo alterado, levando a uma rotulação incorreta dos dados. Muitas vezes, o racismo se apresenta de forma sutil, baseado em como a frase se escreve, e a inserção ou remoção de termos modifica a semântica. Por fim, a técnica da substituição, apesar de propensa a imprecisões assim como os anteriores, busca trocar os termos de seus textos por outros matematicamente próximos (o que não necessariamente indica que são sinônimos), garantindo uma mínima coerência entre os documentos originais e os novos.

6.5 Modelos de Machine Learning

Num primeiro momento, treinos e testes com datasets pequenos foram realizados, com o principal intuito de aprender a implementar modelos de *Machine Learning* e *Deep Learning*. Foram utilizados no total 2266 *tweets*, 80% para treino, 10% para validação e 10% para teste. Tais dados vieram de duas fontes: do primeiro dataset, menor, do *Covid Hate* (HE et al., 2021), que contém 2290 *tweets* rotulados manualmente por especialistas; e das 1916 amostras classificadas consensualmente pelos alunos, com eventuais auxílios de pessoas externas ao projeto. Estes dados foram escolhidos por serem confiáveis, uma vez que foram rotuladas diretamente por humanos. Além disso, devido à proporção desequilibrada entre a quantidade de *tweets* racistas e não racistas, uniu-se os dois conjuntos de dados e selecionou-se as amostras de forma que todas as racistas fossem mantidas e as não racistas fossem escolhidas aleatoriamente, na proporção final 1:2, totalizando assim 2266 dados.

Após a coleta de mais dados conforme explicado nas seções anteriores, *datasets* maiores foram construídos para treino, validação e teste dos modelos. Puderam também ser testadas arquiteturas mais complexas com estes dados, uma vez que melhores resultados podem ser obtidos com um volume maior de dados em *Deep Learning*.

Nesta segunda etapa, os modelos foram treinados e testados cinco vezes, com os mesmos dados, porém distribuídos de modos diferentes entre os *datasets* de treino, validação e teste. Desta forma, pôde-se garantir a validade dos resultados obtidos e não dependência destes com possíveis distribuições favoráveis das amostras. Os experimentos ocorreram com: Naive Bayes, LSTM, LSTM bidirecional, Convolutional Neural Network (CNN), combinação de LSTM e CNN, e combinação LSTM bidirecional e CNN. Além disso, foi testada uma variação de cada uma dessas arquiteturas aplicando-se o LSI nos dados de entrada.

Devido ao desbalanço na proporção de dados classificados como "racistas" e "não racistas", o tipo de função de perda utilizado no treinamento dos modelos foi a *Binary Cross Entropy* com pesos. Os pesos (KING; ZENG, 2001) utilizados foram:

$$w_{\text{racista}} = \frac{n_{\text{total de dados}}}{2 \cdot n_{\text{tweets racistas}}}$$

$$w_{\text{não racista}} = \frac{n_{\text{total de dados}}}{2 \cdot n_{\text{tweets não racistas}}}$$

6.5.1 LSTM

O modelo de LSTM é composto por duas camadas de LSTM, com 50 nós cada uma, seguidas de uma camada *Dropout* (onde 30% do conteúdo que entra nele é zerado, aleatoriamente, a fim de garantir um aprendizado do modelo sem *overfitting*), uma camada linear (que, para cada amostra de entrada, retorna um único valor real) e, por fim, o resultado obtido desta passa por uma função de ativação sigmoide. Assim, o resultado fica compreendido num intervalo de valores entre 0 e 1. Quanto mais próximo de zero, mais o modelo identifica o *tweet* como racista; quanto mais próximo de um, mais ele tende a ser não racista. Para o cálculo de métricas e de matrizes de confusão, assumiu-se que se a saída do modelo fosse menor que 0,5, corresponderia a um comentário racista; caso contrário, seria não racista.

6.5.2 LSTM Bidirecional

O modelo de LSTM bidirecional, assim como o desenvolvido com a LSTM, é composto por duas camadas de LSTM bidirecional, com 50 nós cada uma; uma camada *Dropout*, seguida de uma linear aplicada a uma função sigmóide. A diferença em relação ao modelo apresentado anteriormente é a dimensão da saída das camadas de LSTM bidirecional (e consequentemente a entrada da camada seguinte), que passa a ser o dobro da dimensão do caso unidirecional.

6.5.3 CNN

O modelo utilizado de CNN, ou *1-dimension convolutional neural network*, consiste numa primeira camada *Embedding* (camada linear para representação de cada token do dado de entrada como um vetor de dimensão 100), seguida de uma camada da Conv-1D, com tamanho de janela convolucional igual a 5 e função de ativação *ReLU*, de forma que os valores de saída da camada, se forem negativos, passam a ser zero. Em seguida, os dados passam por uma camada de redução de dimensionalidade (*global max pooling*), uma linear, e por fim, pela função sigmóide.

6.5.4 Combinações de modelos

Os modelos de LSTM com CNN e de LSTM bidirecional com CNN consistem em modelos onde as primeiras duas camadas são de LSTM (ou LSTM bidirecional), seguidas

de uma camada CNN, *max pooling*, linear e a função de ativação sigmóide, nos mesmos moldes dos modelos apresentados anteriormente.

6.5.5 LSI

Nos modelos onde ocorreu a aplicação do LSI sobre os dados de entrada, os *tweets* pré-processados foram tratados da seguinte forma: primeiramente, após a tokenização, apenas os tokens relativos às 2000 palavras mais frequentes nos textos foram mantidas nos dados. Em seguida, foi realizada a transformação de dimensionalidade, com o auxílio da função "TruncatedSVD" da biblioteca scikit-learn, de forma a representar as amostras em vetores de dimensão 200. Tais vetores são então os dados de entrada dos chamados "modelos com LSI", desenvolvidos durante o projeto.

6.6 Twitter Bot

Como mencionado anteriormente, o *Twitter Bot* é o meio com o qual usuários podem interagir com o estudo e treino dos modelos desenvolvidos e especificados anteriormente. As funcionalidades básicas e que foram implementadas para o bom funcionamento do *bot* são:

- Pesquisar por menções ao *bot* junto de algum texto
- Responder ao usuário que fez a chamada ao bot
- Verificar se o usuário é seguidor do bot

Primeiramente foram desenvolvidos códigos que pudessem ser testados localmente, e posteriormente, esses códigos foram modificados para que pudessem ser colocados em ambiente de deploy. A integração com modelos de machine learning foi o último passo do projeto.

Para todos as versões desenvolvidas foi necessário realizar a autenticação por meio das funções *tweepy.OAuth1UserHandler (User Key, User Secret, Access Token, Access Token Secret)* e *tweepy.API (auth token)*, ambas da biblioteca *teeepy*. A primeira retorna um objeto autenticador com as chaves de segurança dadas como parâmetro. Posteriormente esse autenticador é utilizado como parâmetro para que a segunda função crie uma instância da API atrelada à conta do usuário, cujas chaves foram dadas inicialmente.

A fim de evitar que problemas de servidor por parte da rede social sejam confundidos com implementação errônea ou lógica falha por parte do código, foi feita uma função para realizar a autenticação. Em mais de uma situação, por exemplo, o grupo se viu com o retorno do erro de código 503, o qual indica servidores ativos mas com sobrecarga de requisições.

Com relação ao limite, por usuário, de número de requisições da API, o grupo tirou proveito do parâmetro *wait_on_rate_limit* da função *tweepy.API*. Esse parâmetro, quando atribuído o valor *true*, automaticamente espera até que o usuário possa realizar novas requisições. Tal cuidado se dá visto que, caso esse limite seja ultrapassado, nenhuma operação será bem sucedida independentemente da corretude dos parâmetros passados.

Assim, o trecho de código abaixo é a função desenvolvida para a autenticação e criação da instância da API. O uso do tratamento de erro por *try* e *except* garante que erros poderão ser identificados e tratados:

```
import tweepy

def create_api():
    auth = tweepy.OAuth1UserHandler(api_Key, api_key_secret,
                                    access_token,
                                    access_token_secret)
    api = tweepy.API(auth, wait_on_rate_limit=True)

    try:
        api.verify_credentials()
    except Exception as e:
        logger.error("Error creating API", exc_info=True)
        raise e

    logger.info("API created successfully!")

    return api
```

6.6.1 Bots desenvolvidos para teste local

Inicialmente, dois *bots* foram feitos para os primeiros testes com a conta automatizada.

6.6.1.1 Bot de procura periódica

A primeira versão se baseia em identificar *replies* que mencionem a conta automatizada e apresentem uma das palavras-chave pré-determinadas no código do *bot* (no caso, foram usadas como palavras-chave "help", "support", "calling"). As buscas por comentários nesses padrões é feita a cada minuto, dentro de um loop, como a seguir:

```
while True:
    since_id = searches_mentions(api,
                                 ["help", "support", "calling"],
                                 since_id)
    logger.info("Waiting...")
    time.sleep(60)
```

As menções, por sua vez, são coletadas por meio da utilização das funções *Cursor()* e *mentions_timeline()*. Esta última realiza requisições à API do Twitter para recuperar os 20 últimos tweets com menções à conta que instanciou o objeto chamando a função. Essa busca é filtrada por meio do parâmetro *since_id*, o qual indica que apenas tweets com *id* superior ao especificado devem ser pesquisados. Um ponto que poderia ser alterado seria o número limite de mensagens coletadas, o qual, por definição é 20. Entretando, o grupo imaginou que utilizar esse número pré-definido seria vantajoso, em termos de custo computacional, assim como para realização de testes. Ao mesmo tempo, não é esperado que sejam feitas muitas requisições ao *bot*, visto que trata-se de um projeto em início de usa vida.

Sobre o método *Cursor()*, trata-se de um paginador. Em outras palavras, é um meio de se iterar entre todos os resultados dados pela função especificada, e no caso apresentado, trata-se justamente da *mentions_timeline()*.

Desse modo, a chamada das duas funções é dada a seguir:

```
tweepy.Cursor(api.mentions_timeline, since_id=since_id).items()
```

Itera-se para cada item listado no paginador, e a cada um dos *tweets* coletados atualiza-se o maior *id* encontrado, o qual é utilizado em chamadas futuras como valor do parâmetro *since_id* especificado anteriormente.

```
for tweet in tweepy.Cursor(api.mentions_timeline,
                           since_id=since_id).items():
    new_since_id = max(tweet.id, new_since_id)
```

Em seguida são verificadas as ocorrências das palavras-chave dadas no início do pro-

grama. Caso alguma palavra-chave seja encontrada no texto, o programa responde ao usuário que o chamou com um texto arbitrário.

```
if any(keyword in tweet.text.lower() for keyword in keywords):
    api.update_status(status="Some answer",
                      in_reply_to_status_id=tweet.id,
                      auto_populate_reply_metadata=True)
```

Vale pontuar que essa versão também recupera informações das mensagens presentes na conversa do *tweet* que mencionou o *bot*. A estratégia é recursivamente encontrar a mensagem que foi respondida pela *tweet* atual. Isso pode ser concluído por meio do atributo *in_reply_to_status_id*, com o qual obtém-se o *id* do *tweet* sendo procurado.

```
original_tweet_id = tweet.in_reply_to_status_id
original_tweet = api.get_status(original_tweet_id)
```

6.6.1.2 Bot *stream*

Nesta segunda versão, o princípio é o mesmo: que o *bot* responda a um *reply* que o mencione e contenha uma palavra-chave determinada pelo desenvolvedor. A diferença neste caso é que a busca por *tweets* é contínua, e não feita de tempos em tempos como no caso anterior, utilizando para isso do método de *Streaming*.

Neste caso, uma classe é criada com os atributos desejados e tendo como argumentos principais a instância da API, assim como as chaves de acesso do projeto.

```
class SearchesMentionAndPrintOriginalTweet(tweepy.Stream):
    def __init__(self, *args, api):
        super().__init__(*args)
        self.api = api
        self.me = api.verify_credentials()
```

Dentro dessa classe, os métodos on_status e on_error definem, respectivamente, o que ocorre com cada *tweet* coletado, e o tratamento de um eventual erro que vier a ocorrer. É portanto, dento da *on_status* que é codificada toda a lógica para verificação das menções e possíveis respostas.

```
def on_status(self, tweet):
    logger.info("Processing tweet id %s", tweet.id)
    if tweet.in_reply_to_status_id is not None:
        if tweet.in_reply_to_user_id != self.me.id:
            logger.info("Tweet entities: %s", tweet.entities)
```

```
            users_mentioned_array = tweet.entities.get('user_mentions')
            users_mentioned = [users['screen_name'] for users in
                                                users_mentioned_array if
                                                 users['screen_name'] ==
                                                 self.me.screen_name]
            if users_mentioned:
                original_tweet_id = tweet.in_reply_to_status_id
                original_tweet = self.api.get_status(original_tweet_id)
                logger.info("tweet: %s", original_tweet.text)
            else:
                logger.info("I was not called here!")
```

Percebe-se que esta última versão, apesar de proporcionar uma resposta mais rápida por parte do *bot*, impõe problemas. O principal deles é o de não conseguir descobrir todos os *tweets* existentes, e, assim, deixando de responder a alguns daqueles que cumprem às pré-condições determinadas para resposta (mencionar a conta automatizada e possuir uma das palavras-chave). A primeira versão do bot não apresentou tal problema, encontrando facilmente o comentário teste adicionado à plataforma na conta pessoal de um dos membros do grupo. Por esse motivo, a versão que teve um maior enfoque a ser continuada foi a primeira. Outro obstáculo, mas este relacionado a ambos os testes, é o fato do *bot* deixar de coletar uma dada mensagem que deve ser respondida, apenas por ter um *tweet* deletado em meio à *thread* (conversa dentro da rede social) a que pertence. Refletiu-se acerca da opção de criar condições para que esse último problema não ocorresse. Porém chegou-se na conclusão de que talvez fosse mais proveitoso pesquisar soluções e implementações alternativas.

6.6.2 Testes iniciais

Para esses dois protótipos, foram feitos alguns testes referentes ao funcionamento básico do programa. Isto é, se o mesmo consegue identificar os *tweets* que mencionam o usuário, assim, como se uma resposta pode ser dada ao mesmo.

Para isso, um primeiro teste, ilustrado pela Figura 8 feito foi chamar o *bot* dentro de uma conversa feita de forma simples apenas com as contas dos integrantes do grupo. Ao final dela, foi colocada a menção a conta atrelada a automação, assim como a palavra-chave que deve ser utilizada para correto funcionamento do robô.

Fonte: Autores

Figura 8: Conversa teste reproduzida no Twitter

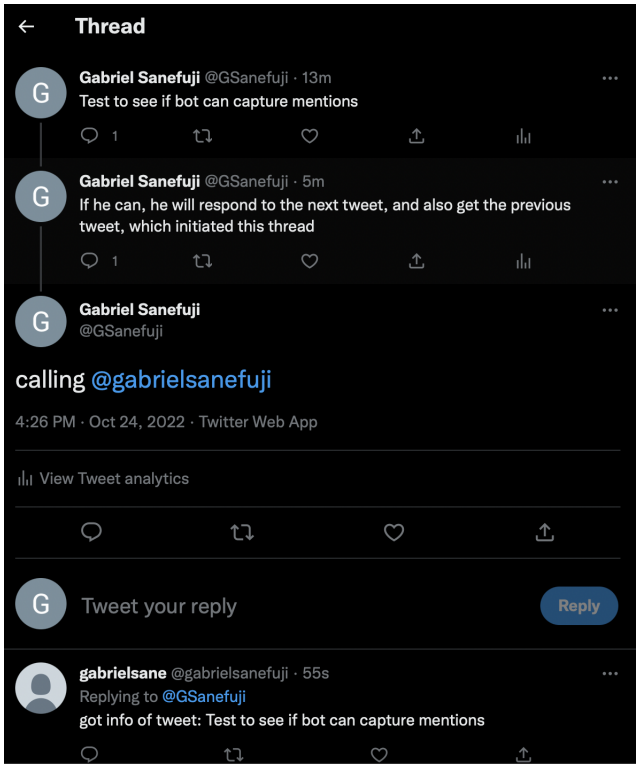
Uma vez que o programa comece a rodar, no caso do *bot* que utiliza de paginação, o *tweet* teste já é encontrado, e informações do texto presente na conversa também podem ser consultados no terminal em que se executa o código, como visto na imagem 9.

```
(.env) ➜  test_bot git:(feat/bot) ✗ python main.py
INFO:root:API created successfully!
INFO:root:Retrieving mentions
INFO:root:Answering to Gabriel Sanefuji
INFO:root:tweet: If he can, he will respond to the next tweet, and also get the previous tweet, which initiated this thread
INFO:root:tweet: Test to see if bot can capture mentions
INFO:root:Waiting...
```

Fonte: Autores

Figura 9: Terminal com programa rodando

Ao voltar para a página da rede social, observa-se na Figura 10 que o teste foi bem-sucedido, com uma resposta àquele que chamou a conta automatizada.



Fonte: Autores

Figura 10: Resposta do *bot* à conversa

Vale ressaltar que, como foi indicado anteriormente, se a conversa possuísse algum *tweet* deletado, o bot deixa de responder e não consegue coletar qualquer tipo de informação.

6.6.3 Bot para resolução de erros em ambiente local

Assim, a fim de resolver o problema a respeito da presença de *tweets* deletados, um novo código para encontrar menções ao usuário foi desenvolvido. Desta vez, descartou-se em definitivo a utilização do método de *Stream*, uma vez que em essência, não conseguir encontrar todas as ocorrências em pesquisa, além do alto custo computacional para o *deploy* dessa função, são pontos muito difíceis de serem contornados.

Inicialmente pensou-se em mudar a estratégia com que a pesquisa é feita. Ao estudar um pouco melhor a documentação do pacote *tweepy* e a API do Twitter, encontrou-se um novo atributo dos objetos retornados pela API v2: a *conversation_id*. Esse parâmetro retorna o mesmo valor para todas as mensagens de uma mesma conversa quando pesquisadas por essa versão da API. Desse jeito, não é necessário coletar as mensagens recursivamente com o atributo *in_reply_to_status_id*, o qual apresenta o número do *tweet* para quem se respondeu.

Contudo, o pedaço de código que era utilizado para chamada em *loop* da função a cada 60 segundos, tendo o parâmetro *since_id* como ponto de partida para buscas nas requisições, permaneceu intacto. Isso porque seu funcionamento já estava correto enquanto feitos os testes iniciais, e não se sentiu a necessidade de alterá-lo nesta fase do desenvolvimento.

Como esse atributo não se faz presente na versão 1.1 das chamadas da API, o método de paginação que foi utilizado é o *Paginator*, função equivalente da *Cursor* para a versão 2 da API.

```
tweepy.Paginator(client.get_users_mentions,
                 id=bot_id,
                 since_id=since_id,
                 tweet_fields=["conversation_id"]):
```

Utilizando esse método recupera-se todas as mensagens de uma mesma conversa dentro do objeto JSON de retorno. Iterando-se sobre ele, colocou-se cada um dos textos das mensagens dentro de um *array*, para facilitar a manipulação desses conteúdos em análises e consultas futuras. Desse modo, a seguinte função foi criada:

```
def searches_tweets_trough_conversation(client, original_tweet):
    responses_from_conversation = []
    for response in tweepy.Paginator(client.search_recent_tweets,
                     query = 'conversation_id:' +
                             str(original_tweet.conversation_id)):
        for tweet in response.data:
            responses_from_conversation.append(tweet.text)
    return responses_from_conversation
```

Outra alteração foi o uso da função *get_users_mentions*, em substituição da *mentions_timeline*. As duas, contudo, são equivalentes, e possuem a mesma utilidade de se pesquisar por menções ao usuário, identificado pelo parâmetro *id*, a partir de um dado *tweet*, especificado pelo número de identificação em *since_id*. Isso teve de ser feito para adequar os métodos à segunda versão da api.

Com isso, o trecho do código atualizado, responsável por recuperar todas as mensagens de uma dada conversa e responder, é como se segue:

```
tweets = response.data
try:
  for tweet in tweets:
    since_id = max(since_id, tweet.id)
```

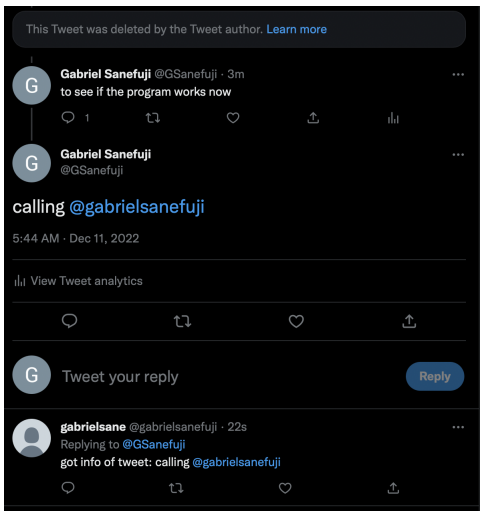
```
    if any(keyword.lower() in tweet.text.lower() for keyword in keywords
                                        ):
      responses_from_conversations = searches_on_conversation(client,
                                          tweet)
      responses_from_conversations.append(client.get_tweet(tweet.
                                          conversation_id).data.text)
      logger.info("Replying to tweet: %s", tweet.text)
      logger.info(responses_from_conversations)

      api.update_status(status="Some answer", in_reply_to_status_id=
                                          tweet.id,
                                          auto_populate_reply_metadata=
                                          True)
      responses_from_conversations = []
```

6.6.4 Testes com tweets deletados

Apesar de ser um código bastante modificado, o procedimento de testes foi o mesmo que o utilizado para as versões iniciais. Criou-se uma conversa simples entre contas dos membros do grupo, e dentro dessa conversa, era deletada uma mensagem.

Ao iniciar o programa do *bot*, verificou-se que o mesmo conseguia buscar as mensagens da *thread* e finalizar o processo com uma resposta ao usuário que o chamou. A imagem 11 ilustra e comprova o funcionamento do robô.

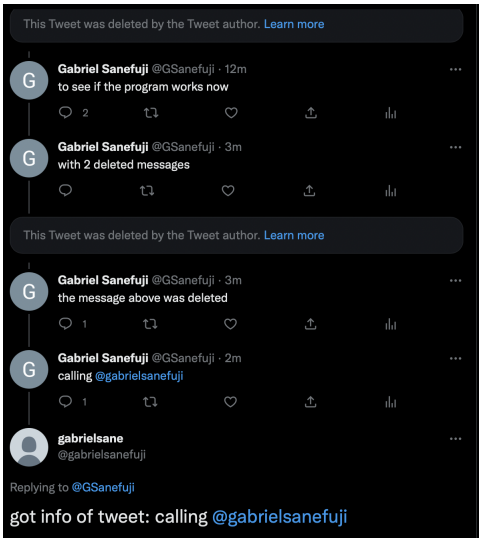


Fonte: Autores

Figura 11: Resposta do *bot* à conversa com mensagem deletada no meio

Também foram feitos testes com mais de um *tweet* deletado, para verificar se a lógica

aplicada limitava-se apenas a um caso em específico. Como era de se esperar, todavia, o programa foi executado normalmente, sem maiores complicações, como observado pela imagem 12.



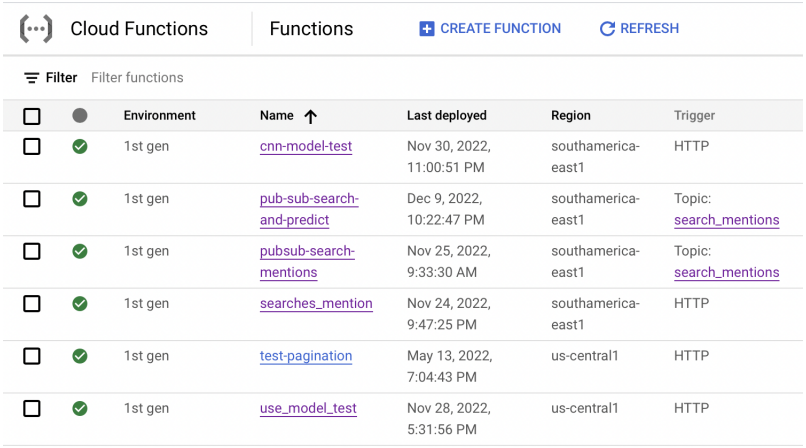
Fonte: Autores

Figura 12: Resposta do *bot* à conversa com múltiplas mensagens deletadas

Vale ressaltar que, caso o *bot* seja chamado e, no meio tempo em que seu código não é executado, o *tweet* que o chamou for deletado, nenhuma análise ou captura de mensagens será feita. Esse comportamento é esperado, pois imagina-se uma situação na qual o usuário desistiu de realizar a chamada.

6.6.5 Cloud Function

Uma vez que o comportamento do bot apresentado nos testes apresentou estabilidade no que se refere a resultados esperados, e ainda com duas das funcionalidades esperadas implementadas, busca de menções e resposta aos usuários, iniciou-se o processo de desenvolvimento das *Cloud Functions*. Como mencionado na seção de tecnologias utilizadas, elas são um meio de se colocar código na nuvem e, consequentemente, realizar o deploy do *bot*.



Fonte: Autores

Figura 13: Página inicial do serviço Google Cloud Function com o nome de todas as funções desenvolvidas

6.6.5.1 Configurações básicas

A criação de uma *cloud function* inicia-se dentro da página do serviço, representada na Figura 13 de mesmo nome. Ao se propor a fazer uma função, são necessários a definição de alguns dados que a configuram: ambiente, nome e região.

O ambiente se refere a versão do serviço que será utilizado. No momento de escrita do trabalho, há 2 versões, uma mais básica e que foi utilizada durante todo o processo de desenvolvimento, e outra voltada para operações mais complexas e que necessitam de simultaneidade, divisão de tráfego e tempos de processamento mais longos. Com relação ao nome da função que fica registrada na página inicial do serviço, ele deve ser único, iniciado com uma letra, seguido de até 61 letras, números, underlines ou hífens, e terminado com letra ou algarismo. Por fim, a região definida é justamente em qual servidor disponível ao redor do mundo a função ficará hospedada. Em geral prefere-se por escolher a região mais próxima do local em que as requisições serão feitas em sua maioria. Caso contrário, a aplicação pode correr o risco de ter um desempenho em tempo inferior ao desejado, pela demora com as requisições. Como trata-se de um trabalho de conclusão de curso numa universidade do Brasil, a região escolhida foi a de São Paulo.

6.6.5.2 Tipo de trigger

Há diferentes formas de se acionar uma *cloud function*. Por ela estar hospedada num servidor de terceiros (no caso da Google), é importante definir como e quando essa função será chamada. O simples deploy por si só não faz nada mais do que ocupar espaço no

servidor.

A *Google Cloud Function* oferece diversos meios de acionamento do código. Um dos meios mais comumente utilizados e de maior facilidade de testes é com requisições HTTP. Foi com essa opção que iniciou-se o desenvolvimento das primeiras versões da *Cloud Function*. Posteriormente, adotou-se o Pub/Sub, cujo acionamento permitiu maior facilidade em programar horários de ativação.

Para o acionamento por requisição HTTP, dentro das configurações do acionamento estão a necessidade ou não de autenticação e de protocolo HTTPS. Como o plano era apenas testar e verificar o funcionamento das funções nesse primeiro momento, o grupo optou por não requerir essas verificações, facilitando o processo de deploy.

Já para o acionamento por Pub/Sub, é necessário informar qual o tópico a ser ouvido pelo *Subscriber* (aquele que recupera a mensagem enviada pelo *Publisher*). O que é preciso entender a respeito deste protocolo, por enquanto, é que o tópico serve como um canal, pelo qual uma aplicação se comunica com outra.

6.6.5.3 Configurações adicionais

Ademais, vale pontuar algumas configurações que tiveram de ser alteradas para o correto funcionamento do programa, mas que a princípio são opcionais. Primeiramente, a quantidade de memória alocada para a função é por padrão 256 MB. A partir do momento que as operações que o robô deve realizar envolvem a utilização de modelos de *machine learning*, esse número se torna insuficiente. E, por isso, altera-se a memória alocada para pelo menos 2 GB.

Outro ponto que deve ser alterado se dá pelas variáveis de ambiente. Como o *bot* utiliza de chaves de acesso para criação de instâncias da API atreladas a conta do usuário, é importante que esses dados estejam disponíveis em locais seguros, fora do alcance de agentes externos. Por isso, são definidas variáveis de ambiente para cada uma das chaves utilizada.

6.6.5.4 Edição de código

As *Cloud Function* podem ser implementadas em diversas linguagens e cabe ao usuário selecionar a correta. Para o desenvolvimento descrito neste trabalho selecionou-se a opção de Python 3.8, que estava sendo utilizada localmente.

Outro ponto que deve ser definido é o *entry point*, ou seja, o método que é executado

dentre todos os definidos na *Cloud Function*. Para requisições HTTP o *entry point* deve possuir o parâmetro *request* para funcionar corretamente, uma vez que irá receber justamente uma requisição do protocolo. Já para a Pub/Sub, devem ser definidos parâmetros os *request* e *context*.

```
Exemplo de entry point main HTTP
    main(request):

Exemplo de entry point main Pub/Sub
    main(request, context):
```

Uma vez definido o *entry point*, é necessário especificar os pacotes (e opcionalmente suas versões), dentro do arquivo requirements.txt, para que o programa possa realizar as importações devidas de cada uma das bibliotecas utilizadas.

Na versão final do código, os seguintes pacotes foram utilizados:

```
google-cloud-storage==2.6.0
emoji==1.7.0
nltk==3.7
regex==2022.10.31
keras==2.9.0
Keras-Preprocessing==1.1.2
tensorflow==2.9.0
certifi==2022.9.24
charset-normalizer==2.1.1
idna==3.4
oauthlib==3.2.1
requests==2.28.1
requests-oauthlib==1.3.1
tweepy==4.10.1
urllib3==1.26.12
```

6.6.5.5 Alterações no código do bot

Como explicado anteriormente, o trecho de código implementado na *Google Cloud Function* só é executado quando a condição de acionamento é atingida, seja por meio de requisições HTTP ou outros métodos. Por esse motivo, o loop, que foi implementado para que a busca fosse realizada a cada 60 segundos, não possui qualquer utilidade ou explicação para ser utilizado neste contexto. Desse modo, o código teve de ser alterado para que a cada chamada da função, o *id* de maior valor encontrado seja guardado para que, numa futura chamada, a busca por novos *tweets* inicie-se por esse número.

A princípio o grupo cogitou o uso de um banco de dados do *Google Cloud* para armazenamento desse dado. Entretanto, alguns fatores pesaram contra essa ideia: prazo para entrega, familiaridade com outros serviços como o *Cloud Storage*, além do fato de que o dado a ser armazenado é um único apenas. Portanto, teve-se a ideia de realizar a leitura de um arquivo armazenado no *Storage* com o valor do *id* e que é atualizado a cada chamada.

Ainda que não seja computacionalmente vantajoso, visto que a leitura do arquivo envolve a operação de abertura do arquivo para ser bem sucedida, o grupo entendeu que seria vantajoso adotar essa estratégia por dois motivos principais. Primeiramente pelo número baixo de requisições que se espera ter num primeiro momento, e, portanto, que um pequeno atraso sofrifo com a leitura do arquivo não deve influenciar de maneira significativa sobre o desempenho do programa como um todo. Em segundo lugar, o preço para hospedagem do banco de dados seria muitas vezes superior ao da utilização do *Storage*, o qual teria um preço reduzido pelo baixo número de arquivos nele armazenado.

Pensando, por exemplo, na contratação do serviço de armazenamento mais caro, no qual os arquivos não possuem um tempo mínimo de armazenamento, o custo mensal seria de $0.035 por GB de arquivos armazenados por mês. Enquanto que para o banco de dados, a operação de armazenamento teria um custo de pelo menos $0.135 por GB armazenados em HDD. A Tabela 4 contém todos os preços para cada tipo de armazenamento.

Custos para armazenamento do Cloud SQL (por GB por mês)			
Servidor	Armazenamento SSD	Armazenamento HDD	Backup
São Paulo	$0.255	$0.135	$0.12

Tabela 4: Preços dos serviços de armazenamento do Cloud SQL no servidor em São Paulo

Vale ressaltar os demais preços do *Cloud Storage*, encontrados na Tabela 5. Os armazenamentos *Nearline*, *Coldline* e *Archive* possuem um tempo de armazenamento mínimo, no qual arquivos não podem ser alterados ou deletados, de 30, 90 e 365 dias, respectivamente.

Custos para armazenamento do Cloud Storage (por GB por mês)				
Servidor	Standard	Nearline	Coldline	Archive
São Paulo	$0.035	$0.020	$0.007	0.0030

Tabela 5: Preços dos serviços de armazenamento do Cloud Storage no servidor em São Paulo

Assim, gerou-se um arquivo texto dentro do bucket "tcc-tweet-bot", no qual é escrito o *id* encontrado. Em seguida Foram feitas rotinas para obtenção em texto do conteúdo e posterior atualização deste arquivo no bucket especificado. Esses dados são passados por meio dos parâmetros de cada uma das funções.

```
def get_since_id(bucket, filename):
    last_tweet_id = bucket.blob(filename).download_as_text()
    return last_tweet_id

def update_file(new_since_id, bucket, filename):
    blob = bucket.blob(filename)
    blob.upload_from_string(new_since_id)
```

Esses métodos são chamados ao longo da função *main*, definida como *entry point*. Nela, é instanciado o objeto API, assim como nas demais versões do código, e uma instância do bucket do *Cloud Storage* de onde podem ser retirados arquivos a serem lidos. A chamada para a rotina de busca por menções é feita normalmente. Vale pontuar também que as funções definidas como *entry point* necessariamente devem retornar um numeral ou texto, visto que, do contrário, a *Cloud Function* retorna um erro. Por isso, decidiu-se no retorno do número de maior *id* encontrado, apenas para evitar esse erro. A *main* como um todo pode ser encontrada no trecho de código abaixo.

```
def main(request):
    api = create_api()

    storage_client = storage.Client()
    bucket = storage_client.bucket("tcc-tweet-bot")

    since_id = int(get_since_id(bucket, "last_tweet_id.txt"))
    since_id = searches_mentions(api, ["help", "support", "calling"],
                                 since_id)

    update_file(str(since_id), bucket, "last_tweet_id.txt")
    return str(since_id)
```

6.6.6 Acionamento periódico

Tendo a *Cloud Function* sendo corretamente configurada, o passo seguinte foi encontrar meios pelos quais o acionamento das funções fosse automatizado. Para tanto, o serviços utilizados foram o *Google Cloud Scheduler* e o *Pub/Sub*. A ideia foi programar o *Scheduler* para que ele enviasse uma mensagem sobre um tópico criado no serviço Pub/-Sub, e a função, ao receber essa mensagem, iniciasse sua execução, como ilustrado na imagem 14.

Fonte: Autores

Figura 14: Fluxo lógico para acionamento da função

6.6.6.1 Pub/Sub

Já apresentado em seções passadas, o Pub/Sub é um padrão arquitetural disponibilizado pelo *Google Cloud PLatform* composto por editores (ou *Publishers*), assinantes (ou *Subscribers*) e canais (chamados de tópicos). A principal característica desse padrão se dá por providenciar uma comunicação assíncrona entre editores e assinantes, com a transmissão de eventos pelo canal em comum definido por cada uma das partes. É innteressante pontuar que mais de um editor ou assinante podem ouvir ao mesmo canal simultaneamente.

O fato de ser uma comunicação assíncrona torna essa solução mais interessante do que, alternativamente, utilizar requisições HTTP para acionar as funções. Primeiramente porque torna a solução escalável. Não é necessário configurar cada um dos lados da aplicação recebedora ou transmissora das mensagens, basta que o canal em que está sendo feita a comunicação de cada aplicação seja o mesmo. Assim, caso seja necessário adicionar novas funcionalidades ao *bot*, só seria preciso criar uma nova função e atribuir o canal de transmissão como o mesmo que já foi criado e está em uso.

Outro ponto importante está no fato de que mensagens assíncronas possuem um menor risco de perda por sobrecarga de mensagens recebidas. Assim, garante-se que a função, de fato, será executada.

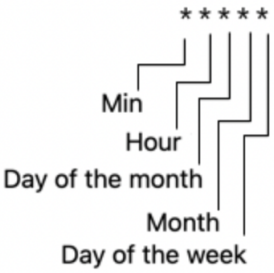
Com relação a configuração de um tópico no *Google Cloud Platform*, o processo é bem direto e intuitivo. Entra-se na página de serviço Pub/Sub e então seleciona-se a opção

de criação de tópico para abertura da janela de configuração do canal. Como a aplicação sendo feita é bem simples, não se sentiu a necessidade de alterar as configurações padrões pré-definidas. O único campo necessário de preenchimento é o nome do tópico.

6.6.6.2 Google Cloud Scheduler

Este é o serviço que realiza o acionamento das funções por meio de cron-jobs, e como o próprio nome diz, agenda horários ou intervalos de horários para ativar esse trabalho. Em sua área de configuração, é necesssário atribuir um nome ao trabalho e a região em que será utilizado. Ponto importante no caso é a definição da região e fuso horário em que será utilizado, uma vez que para zonas diferentes, horários diferentes são utilizados como base.

A definição da frequência com que o trabalho será executado é feita por meio do formato unix-cron (******), no qual cada asterisco possui um significado como apresentado na Figura 15:



Fonte: (GOOGLE, 2022)

Figura 15: Formato unix-cron

A princípio, o trabalho foi para ser acionado a cada 30 minutos e com execução Pub/Sub de tópico igual ao que havia sido definido quando configurado o serviço Pub/Sub. O mesmo tópico é utilizado para o acionamento da *Cloud Function*. Assim, ao acionar o trabalho no scheduler, uma mensagem é enviada à função, que passa a ser executada.

6.6.7 Adição de modelo de machine learning

Por fim, foi adicionado o modelo de melhor desempenho dentre todos os treinados pelo grupo no código do robô. Esse processo, apesar de simples, possui algumas etapas.

Primeiro, é feito o *upload* do arquivo contendo os pesos do modelo, treinado previamente, no *Cloud Storage*. Esse arquivo, então, não pode ser consultado e aberto de

forma direta. Deve-se primeiro realizar o download do arquivo para uma pasta local da máquina executando a função. Apenas desse modo que o modelo fica disponível para uso e o método criado para esse propósito é apresentado abaixo.

```
def download_model_file(bucket):
    data_file = "cp-0011.ckpt.data-00000-of-00001"
    index_file = "cp-0011.ckpt.index"
    tokenizer_file = "tokenizer.pickle"

    data = bucket.blob(data_file)
    index = bucket.blob(index_file)
    tokenizer = bucket.blob(tokenizer_file)

    folder = "/tmp/"
    if not os.path.exists(folder):
      os.makedirs(folder)

    data.download_to_filename(folder + "cp-0011.ckpt.data-00000-of-00001
                                        ")
    index.download_to_filename(folder + "cp-0011.ckpt.index")
    tokenizer.download_to_filename(folder + "tokenizer.pickle")
```

De modo geral, o resto da implementação segue a mesma lógica que qualquer outro desenvolvimento comum de um modelo. São definidas as camadas dele e os parâmetros de entrada e saída de cada uma delas. Ao final do método, retorna-se um objeto contendo as especificações do modelo compilado.

```
def create_model(num_words, max_length):
with tf.device("cpu:0"):
   model = Sequential()
   model.add(Embedding(num_words, 100, input_length=max_length))
   model.add(Conv1D(max_length, 5, activation='relu'))
   model.add(GlobalMaxPooling1D())
   model.add(Dropout(0.3))
   model.add(Dense(1, activation='sigmoid'))
   model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                 loss="binary_crossentropy",
                 loss_weights=[1.5, 0.75],
                 metrics=['mse', 'accuracy'])
return model
```

Tendo um array de *tweets*, resta iterar entre si, processar e tratar os dados obtidos, e realizar predições com o modelo em mãos. O processo de pré-processamento, em específico, inclui a remoção de *stopwords*, links, nomes de usuários, separação de pontuações e emojis, além da tokenização e padding.

```
for message in tweets:
    frase_processada = pre_process_tweet(message, bucket)
    prediction = model(frase_processada)[0][0]
    pred_total.append(float(prediction))
return pred_total
```

6.6.8 Ajustes finais

Com o *bot* conseguindo recuperar menções e responder apropriadamente aos usuários por meio do uso de modelos desenvolvidos, alguns pequenos detalhes ainda precisavam ser analisados. Primeiramente, uma modificação simples, porém relevante, é a definição da frase de chamada da conta automatizada. Ficou definido que a menção ao *bot* deveria ser feita em conjunto à frase *tell me*.

Em segundo lugar, como dito no início desta seção, é importante que o robô consiga identificar se o usuário que o chamou é um seguidor. Desse modo, garante-se que nenhuma regra contra direitos de expressão do usuário seja desrespeitada. Por esse motivo, foi desenvolvida a seguinte função:

```
def follows_bot(api, bot, user):
    rel = api.get_friendship(source_screen_name = user,
                             target_screen_name = bot)
    return rel[0].following
```

Ela serve justamente para identificar qual o tipo de relação entre um dado usuário e a conta aautomatizada, por meio do seguinte método:

```
api.get_friendship(source_screen_name, target_screen_name)
```

Com ela, dado o nome de um usuário da rede social origem (colocado no parâmetro *source_screen_name*), verifica-se sua relação com aquele de nome de usuário alvo (colocado no parâmetro *target_screen_name*). Se o atributo *following* retornar com o valor *true*, o *bot* deve prosseguir com análises e respostas.

Por fim, o último ponto que foi desenvolvido se tratou das respostas dadas aos usuários. A estratégia utilizada foi a de dividir mensagens racistas entre tendenciosas e, de fato,

racistas.

O *bot* atribui um valor de 0 a 1 às frases, possuindo uma conotação mais racista conforme esse número se aproxime de zero (utilizando da mesma lógica que apresentada no modelo apresentado anteriormente, LSTM, por exemplo). Assim, pôde-se classificar mensagens com valor entre 0.5 e 0.3 como tendenciosas, enquanto àquelas que apresentassem um número inferior, como realmente racistas.

Tendo a classificação de cada mensagem em uma conversa, o robô indica qual a porcentagem de *tweets* racistas ou tendenciosos. Vale ressaltar que a frase conta com uma ressalva, mostrando que o *bot* de modo algum apoia ou compactua com discursos de ódio.

```
if any(x < 0.5 for x in prediction):
    nb_racist = 100*len([x for x in prediction if x < 0.3])//len(
                                        prediction)
    nb_almost_racist = 100*len([x for x in prediction if (x < 0.5 and x
                                        >= 0.3)])//len(prediction)
    reply_message = f"Analyzing all the messages on the thread,
                    {nb_racist}% of them seems racist and
                    {nb_almost_racist}% tend to be racist\n\n
                    My goal is to make people aware of how they talk.
                    Some expressions can offend or make people
                    uncomfortable, even if not intended.
                    You can agree/disagree with me, it's just my opinion
                                                        ."
else:
    reply_message = random.choice(no_racism_responses)
```

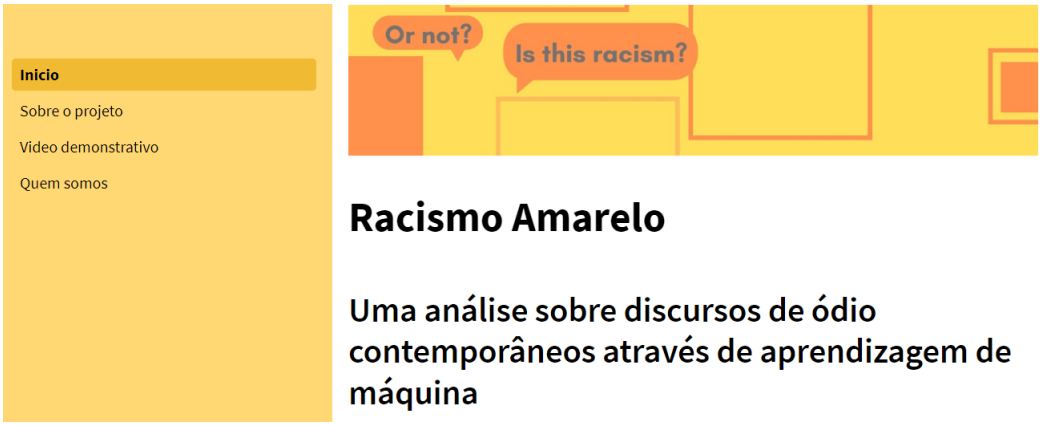
Um array com respostas aleatórias para conversas sem racismo também foi elaborado:

```
no_racism_responses =
    ["Yay! I didn't find any racist messages on the thread",
     "The HEXA VEEEEEEM, there's no racism on this thread (if you are
                                        not Brazillian, I'm sorry for
                                        the random message)",
     "On a galaxy far far away there are people making racist comments.
                                        This is not the case for this
                                        thread, good job :)",
     "I couldn't find racism connotation on this thread :D"]
```

6.7 Site

O site foi criado com o intuito de apresentar a motivação, o objetivo e os resultados do projeto para o público geral e seu *design* de forma geral pode ser observado na imagem 16. Ele foi desenvolvido com o framework *Streamlit*, que permite o compartilhamento de soluções de *Machine Learning* e Dados de forma simples e direta, sem a necessidade de recorrer a recursos externos para *deploy*. Seu código está em Python e configurações sobre o layout e o estilo da página foram definidas em arquivos *.css* e *.toml*.

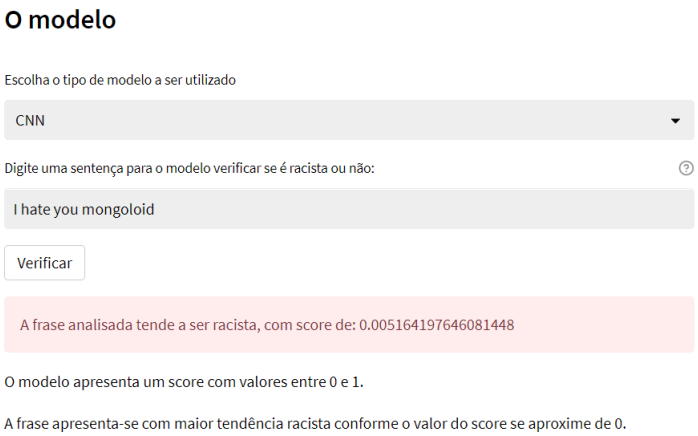
Ele possui quatro seções: Início, Sobre o projeto, Vídeo demonstrativo e Quem somos.



Fonte: Autores

Figura 16: Página inicial do site e menu lateral

No website, é possível testar cada um dos modelos desenvolvidos durante o projeto com frases submetidas pelo usuário. Essa *feature* é visualizada pela Figura 17, na qual o modelo selecionado retorna o *score* associado à predição, sendo que quanto mais próximo de zero, maiores as chances da sentença ser de cunho racista, e quanto mais próximo de 1, mais chances de ser não racista.



Fonte: Autores

Figura 17: Teste de modelos no site

A página conta ainda com um tradutor automático, que permite a submissão de um termo em português para ser traduzido para inglês. Isso, com o intuito de auxiliar o usuário da plataforma a utilizar os modelos, treinados com conteúdo em língua inglesa.

7 TESTES E RESULTADOS

Nesta seção, são mostrados os resultados obtidos com os modelos treinados e testados. São feitas também análises comparativas entre as iterações (validações, utilizando diferentes *datasets* de treino, validação e teste) realizadas sobre uma mesma arquitetura de *Machine Learning* e entre modelos diferentes. Por fim, também é apresentado como o *bot* desenvolvido se comporta em seu estágio final.

7.1 Testes preliminares reduzidos

Os primeiros resultados obtidos foram com os modelos treinados e testados com os *datasets* pequenos. Para cada modelo, foram realizados cinco treinamentos, com os dados (2266 *tweets*) distribuídos de forma aleatória entre os datasets de treino, teste e validação. Modelos muito complexos não foram utilizados nestes testes preliminares, uma vez que a quantidade limitada de dados influencia na performance dos modelos, podendo levar a *overfitting*, por exemplo, uma vez que a quantidade de parâmetros neles é notadamente maior que a de amostras na base de dados.

7.1.1 Bayes

O modelo de Bayes Multinomial foi treinado cinco vezes, com conjuntos de treino variados. As métricas obtidas foram compiladas e organizadas na Tabela 6:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	**0,7456**	0,7257	0,7832	0,7080	0,7522	0,7456	2,5e−2
Precisão	0,8889	0,9545	**0,8788**	0,6190	0,8182	0,8788	1,1e−1
Recall	0,2222	**0,2561**	0,3919	0,1831	0,2571	0,2561	7,0e−2
F1-score	0,3556	0,4038	0,5421	0,2826	**0,3913**	0,3913	8,5e−2

Fonte: Autores

Tabela 6: Métricas dos modelos de Bayes para o dataset pequeno

É possível perceber que os resultados mudam bastante de uma iteração para outra, uma vez que cada um dos dados do pequeno *dataset* representa uma relevante porcentagem do conjunto de dados como um todo. Assim, vê-se também que os desvios padrões são elevados, ultrapassando uma margem de 5% na maioria das métricas.

7.1.2 LSTM

Assim como o modelo de Bayes, o LSTM foi treinado e testado cinco vezes, uma com cada versão dos *datasets* de treino, validação e teste gerados pelo *Random Permutation Cross Validation*. As métricas obtidas sobre o dataset de teste seguem na Tabela 7 abaixo:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,5658	0,5310	0,6593	**0,6018**	0,6062	0,6018	4,3e−2
Precisão	0,3714	0,3750	0,4754	0,4021	**0,3932**	0,3932	3,8e−2
Recall	0,5417	0,4390	0,3919	0,5493	**0,5000**	0,5000	6,1e−2
F1-score	0,4407	0,4045	0,4296	0,4643	**0,4403**	0,4403	1,9e−2

Fonte: Autores

Tabela 7: Métricas dos modelos de LSTM para o dataset pequeno

É possível observar que não apenas a acurácia é mais baixa, mas também a precisão na LSTM é bem menor que a obtida pelo Naive Bayes. Na prática, isso indica que para os modelos de LSTM, de todos as amostras que predisseram ser racistas, apenas uma pequena parcela (menos de 40%) realmente era. Já o Naive Bayes não indica erroneamente, de forma tão frequente, que um *tweet* é racista quando não o é.

Por outro lado, o recall é notavelmente mais alto na segunda arquitetura testada, indicando assim que, de todas as amostras racistas que existem no *dataset* de teste, o LSTM consegue encontrar metade delas, enquanto o Naive Bayes identifica apenas um quarto do total. O F1-score, correspondente à média harmônica entre a precisão e o recall, se encontra um pouco menor neste último modelo.

7.1.3 CNN

Treinando e testando o modelo de rede convolucional de uma dimensão, em cinco iterações, obteve-se os seguintes resultados, listados na Tabela 8:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,7895	**0,7788**	0,7655	0,7832	0,7566	0,7788	1,2e−2
Precisão	0,7308	0,8077	0,6438	**0,6897**	0,6056	0,6897	7,0e−2
Recall	0,5278	0,5122	0,6351	**0,5634**	0,6143	0,5634	4,8e−2
F1-score	0,6129	0,6269	0,6395	**0,6202**	0,6099	0,6202	1,1e−2

Fonte: Autores

Tabela 8: Métricas dos modelos de CNN para o dataset pequeno

É interessante perceber que todas as métricas do CNN superam 0,5. Isso mostra que, em geral, não comete erros tão frequentes quanto os modelos anteriores no que diz respeito a indicar como não racistas sentenças que são ou como racista aquelas que não são (ainda que a precisão mediana neste caso esteja um abaixo do Bayes).

Assim, como nos modelos de Bayes, pode-se perceber consideráveis variações nas métricas, de iteração para iteração. Os desvios padrões são menores que no caso do Naive Bayes, entretanto continua-se observando o impacto do baixo volume de dados no treinamento de um algoritmo de *Machine Learning*. Isso, pois para se aprender padrões, é importante que o *dataset* de treino contenha diversas amostras com sua aplicação, o que um conjunto de dados pequeno não permite.

7.2 Baseline

O modelo utilizado como baseline foi o Naive Bayes Multinomial. Ele foi treinado cinco vezes, com bases de treino variadas, obtidas através da seleção aleatória de 80% do

dataset maior, de 1459393 *tweets* rotulados. As métricas de desempenho foram calculadas, para cada versão.

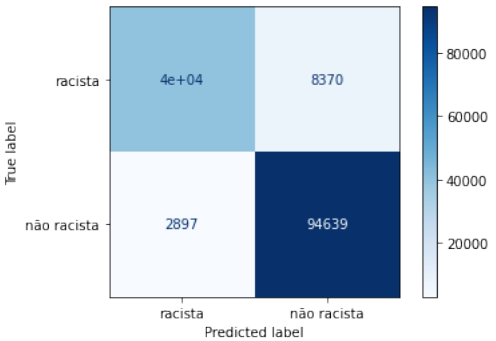
Tais métricas são utilizadas como referência, para a seleção dos melhores modelos testados ao longo do projeto e listadas na Tabela 9. São elas:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,9226	0,9229	0,9226	0,9231	**0,9228**	0,9228	1,9e−4
Precisão	0,9325	0,9361	0,9343	**0,9332**	0,9325	0,9332	1,4e−3
Recall	0,8276	0,8246	0,8240	0,8284	**0,8271**	0,8271	1,7e−3
F1-score	0,8769	**0,8768**	0,8757	0,8776	0,8766	0,8768	6,1e−4

Fonte: Autores

Tabela 9: Métricas dos modelos de Bayes

É possível perceber que as métricas obtidas nos diferentes testes foram bastante próximas entre si, demonstrando a consistência dos modelos e a ausência de eventuais "favorecimentos" que as distribuições aleatórias de amostras nos datasets de treino poderiam fornecer. Escolhendo o modelo da iteração 5 como versão representativa dos cinco testados (sendo o mais próximo da mediana de todas as métricas calculadas), obtém-se com ele a matriz de confusão para o conjunto de teste apresentada na Figura 18:



Fonte: Autores

Figura 18: Matriz de confusão do modelo Bayes

7.3 Modelos sem LSI

Comparando-se as métricas de desempenho medianas dos modelos testados sem a aplicação do *Latent Semantic Indexing*, obtém-se a Tabela 10 abaixo:

Comparativo dos modelos						
Métricas	Baseline	LSTM	LSTM Bi	CNN-1D	LSTM e CNN-1D	LSTM Bi e CNN-1D
Acurácia	0,9228	0,7763	0,669*	0,9406	0,8256	0,8407
Precisão	0,9332	0,6404	0	0,9287	0,7123	0,7370
Recall	0,8271	0,7481	0	0,8898	0,7988	0,8058
F1-score	0,8768	0,6901	-	0,9087	0,7498	0,7727

Fonte: Autores

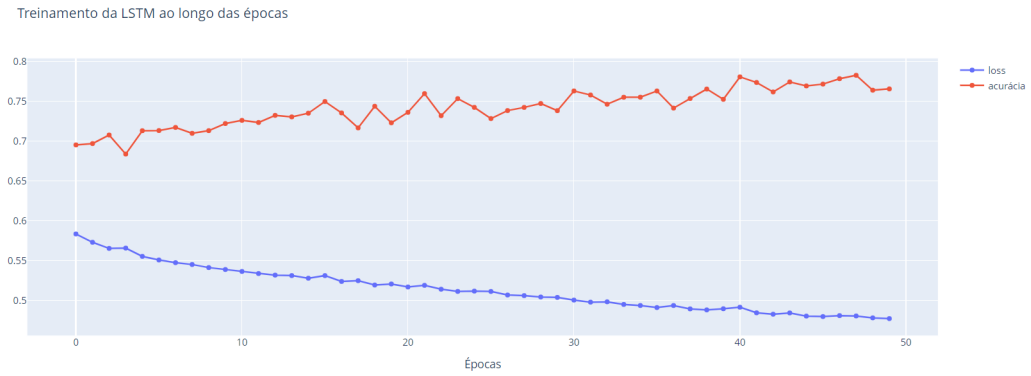
Tabela 10: Métricas dos modelos de Bayes

É possível concluir que, de todos os modelos treinados, apenas o CNN-1D superou o Naive Bayes em acurácia, recall e F1-score. Em precisão, apesar de ter obtido um valor menor, segue bastante próximo do modelo de *baseline*. No caso do LSTM bidirecional, os modelos criados não são capazes de identificar *tweets* racistas ou não, retornando sempre a mesma classe como resultado para qualquer dado de entrada. Desta forma, a acurácia representa apenas a proporção de amostras não racistas de seu *dataset* de teste, e por consequência, não há dados racistas classificados corretamente, levando as métricas precisão e recall a zero.

Todos os modelos foram treinados com 50 épocas e *batch* de tamanho 256. Em alguns casos, como no modelo LSTM bidirecional, tamanhos de *batch* menores, de 16 a 256 foram testados, assim como valores de *learning rate* de 0,00001 a 0,0001, a fim de se construir um modelo com resultados melhores que um modelo de saída constante. Nas seções a seguir, as análises individuais dos treinos de cada modelo são explicitadas.

7.3.1 LSTM

O treinamento, com cada um dos *datasets* de treino, foi realizado em 50 épocas. Foram gerados *checkpoints*, salvando os pesos dos modelos a cada época e, ao fim, o melhor deles foi escolhido com base nos valores de *loss* e acurácia sobre o *dataset* de validação (idealmente, o modelo deve ter a menor loss e a maior acurácia, atentando-se a eventuais *overfittings*). Os cinco treinamentos dos modelos de LSTM seguiram padrões similares ao indicado no gráfico apresentado na Figura 19, que corresponde ao primeiro treinamento realizado.



Fonte: Autores

Figura 19: Curva de treinamento do modelo LSTM

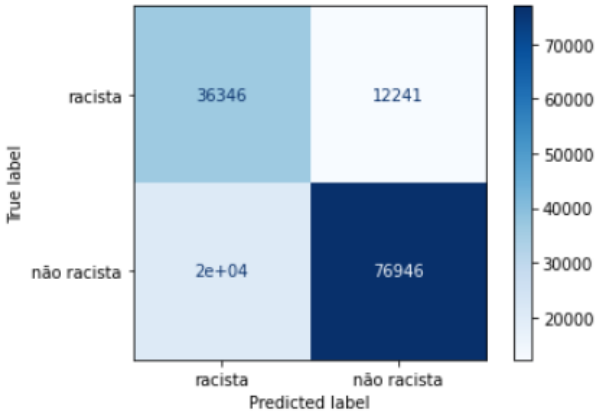
Checkpoints são os parâmetros, os pesos dos modelo em dado estado do treinamento. Para cada final de época, caso o valor da função de perda sobre o *dataset* de validação diminuísse, um checkpoint era criado. Com base na acurácia e na função de perda, para cada uma das cinco iterações, *checkpoints* foram escolhidos e utilizados para se predizer as classificações do *dataset* de teste. Analisando individualmente a performance dos cinco testes com os modelos LSTM, pode-se chegar às métricas medianas conforme indicado na Tabela 11:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,7729	0,7816	0,7750	**0,7763**	0,7833	0,7763	4,0e−3
Precisão	0,6341	0,6507	0,6307	**0,6404**	0,6568	0,6404	9,9e−3
Recall	0,7524	0,7418	0,7712	**0,7481**	0,7261	0,7481	1,5e−2
F1-score	0,6883	0,6933	0,6939	**0,6901**	0,6897	0,6901	2,2e−3

Fonte: Autores

Tabela 11: Métricas dos modelos de LSTM

É possível notar que as métricas estão bastante próximas, com o desvio padrão populacional calculado menor que 0,02 para todas as métricas. Em geral, os resultados se mostram melhores que os obtidos com o *dataset* menor, sendo superiores a 60% em todos as iterações. Tomando o modelo gerado na iteração 4 como referência, uma vez que foi o modelo mediano em relação a todas as categorias de comparação, sua matriz de confusão segue na Figura 20 abaixo.



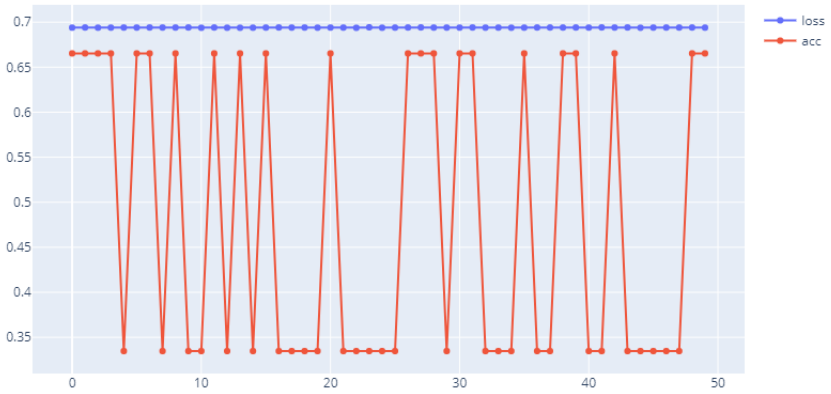
Fonte: Autores

Figura 20: Matriz de confusão do modelo LSTM

7.3.2 LSTM bidirecional

Os modelos de LSTM bidirecional foram os mais demorados para se treinar, levando mais de 12 horas para uma única iteração de 50 épocas. Seus resultados não foram promissores e, mesmo variando o tamanho do *batch* – foram testados valores de 16 a 256 – e da taxa de aprendizagem (*learning rate*), de 1e−5 a 1e−4, o modelo mostrou não conseguir aprender os padrões para classificação das mensagens.

Por conta do tempo de treinamento, não foi possível concluí-lo em muitas das iterações, uma vez que o tempo limite para uso da GPU do Kaggle em uma sessão é de 12 horas. O gráfico da Figura 21 corresponde justamente ao único treinamento completo.



Fonte: Autores

Figura 21: Curva de treinamento do modelo LSTM bidirecional

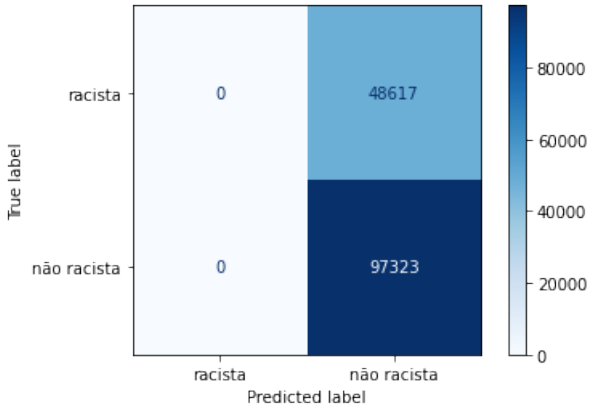
Nos testes realizados, em todos os casos, o modelo não aprendeu de fato a classificar as mensagens, predizendo sempre tudo como “racista” ou “não racista”. Desta forma, a comparação de métricas entre as diferentes iterações não agrega valor, uma vez que variações nos resultados são apenas fruto de diferenças na proporção de dados de cada categoria nos *datasets* de teste.

Tomando como referência o teste realizado na primeira iteração, tem-se como métricas o que é apresentado pela Tabela 12:

Métricas do LSTM bidirecional	
Métricas	Iter. 1
Acurácia	0,6669
Precisão	0
Recall	0
F1-score	–

Tabela 12: Métricas do modelo de LSTM bidirecional

Uma vez que a precisão e o recall foram zero, não é possível calcular o F1-score, que consiste na média harmônica dos dois valores. A matriz de confusão obtida para esta mesma primeira iteração segue na Figura 22 abaixo:



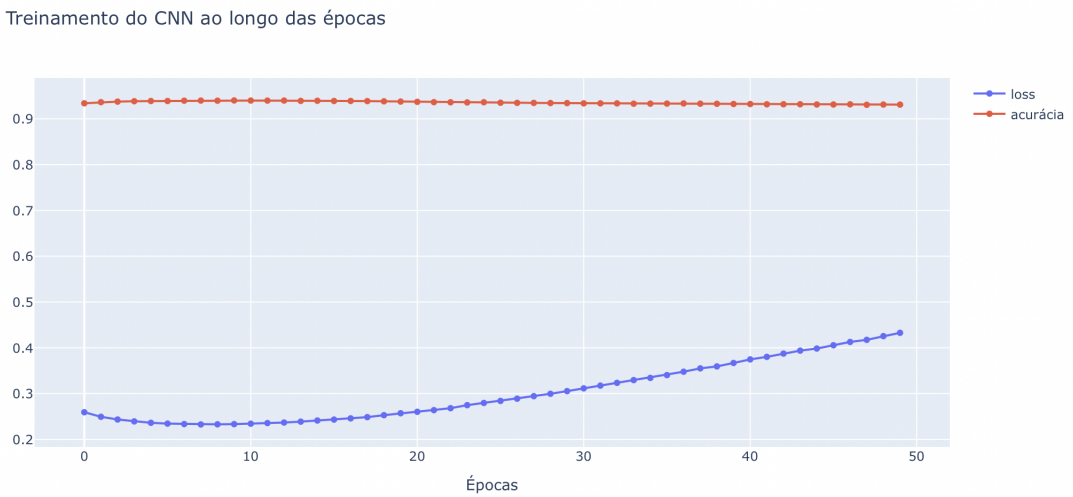
Fonte: Autores

Figura 22: Matriz de confusão do modelo LSTM bidirecional

7.3.3 CNN

O treinamento dos modelos CNN seguiu o padrão apresentado na Figura 23. Logo nas primeiras épocas foram obtidos os melhores pesos de modelos, que levam às mais altas

acurácias e menores perdas:



Fonte: Autores

Figura 23: Curva de treinamento do modelo CNN

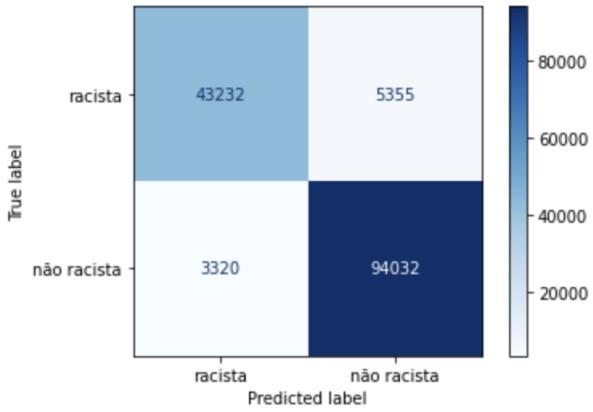
Observando-se as métricas para cada iteração na Tabela 13, pode-se concluir que apresenta os menores desvios padrões, dentre os modelos analisados, superando apenas o Naive Bayes. Isso indica a consistência dos resultados e da aprendizagem, sem *overfitting* decorrente da aprendizagem de padrões muito específicos do *dataset* de treino.

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,9390	0,9406	0,9408	**0,9406**	0,9398	0,9406	6,7e−4
Precisão	0,9214	0,9292	0,9255	**0,9287**	0,9342	0,9287	4,2e−3
Recall	0,8931	0,8891	0,8931	**0,8898**	0,8806	0,8898	4,6e−3
F1-score	0,9070	**0,9087**	0,9090	0,9088	0,9066	0,9087	1,0e−3

Tabela 13: Métricas dos modelos de CNN

Os resultados obtidos foram os maiores dentre os modelos sem LSI. O fato do F1-score ultrapassar 90% mostra a capacidade do modelo em classificar os *tweets* da maneira correta. A precisão e o recall estarem próximos indica que não há necessariamente uma tendência sobre o tipo de erro do modelo (de tender a classificar sentenças como racistas, ou como não racistas).

A matriz de confusão do modelo gerado na iteração 4 é a da Figura 24:

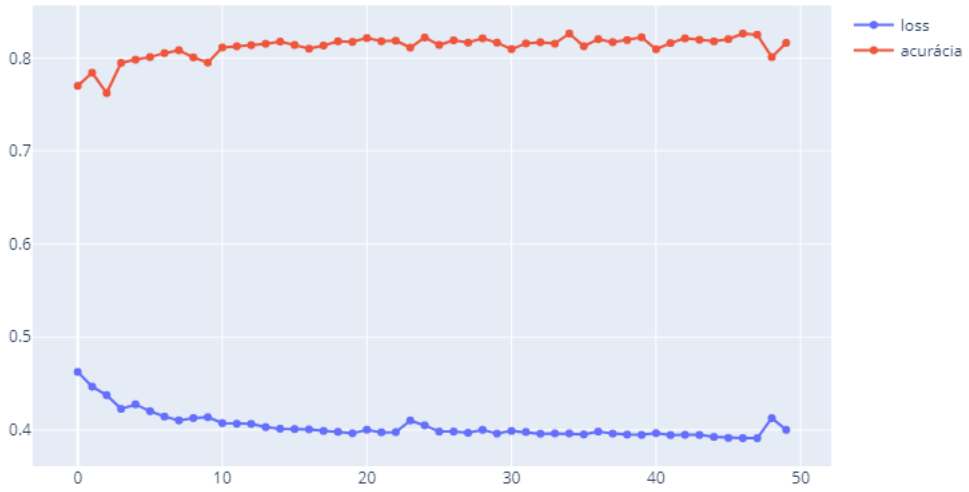


Fonte: Autores

Figura 24: Matriz de confusão do modelo CNN

7.3.4 LSTM com CNN

O treinamento dos modelos seguiu o padrão da Figura 25, com o valor da função de perda diminuindo e o de acurácia, crescendo, lentamente, ao longo das épocas:



Fonte: Autores

Figura 25: Curva de treinamento do modelo LSTM com CNN

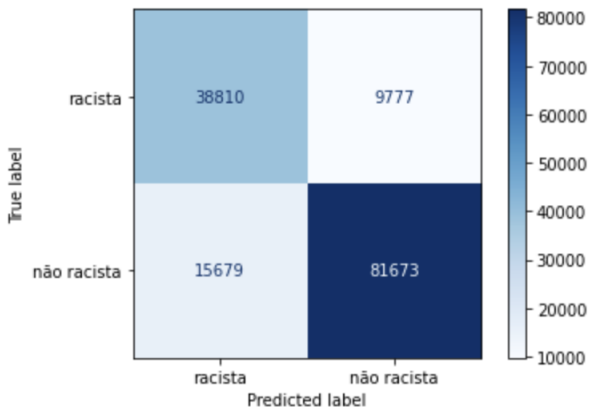
Comparando as métricas dos modelos das cinco iterações, tem-se a Tabela 14:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8257	0,8244	0,8297	**0,8256**	0,8229	0,8256	2,3e−3
Precisão	0,7214	0,7077	0,7298	**0,7123**	0,7053	0,7123	9,1e−3
Recall	0,7769	0,8044	0,7702	**0,7988**	0,8004	0,7988	1,4e−2
F1-score	0,7481	0,7530	0,7494	0,7530	**0,7498**	0,7498	2,0e−3

Fonte: Autores

Tabela 14: Métricas dos modelos de LSTM com CNN

Os desvios padrões são próximos aos obtidos nos modelos LSTM, porém as métricas em si são um pouco melhores. Isso mostra o impacto positivo que a adição de uma camada convolucional tem sobre o modelo de LSTM. Por outro lado, os *tweets* pré-processados e tokenizados, passados diretamente como entrada para o modelo de CNN, trazem mais informação relevante para ele do que receber os dados de saída da LSTM. Utilizando a iteração 4 como referência para os modelos desta arquitetura, obtém-se como matriz de confusão para o *dataset* de teste a Figura 26:

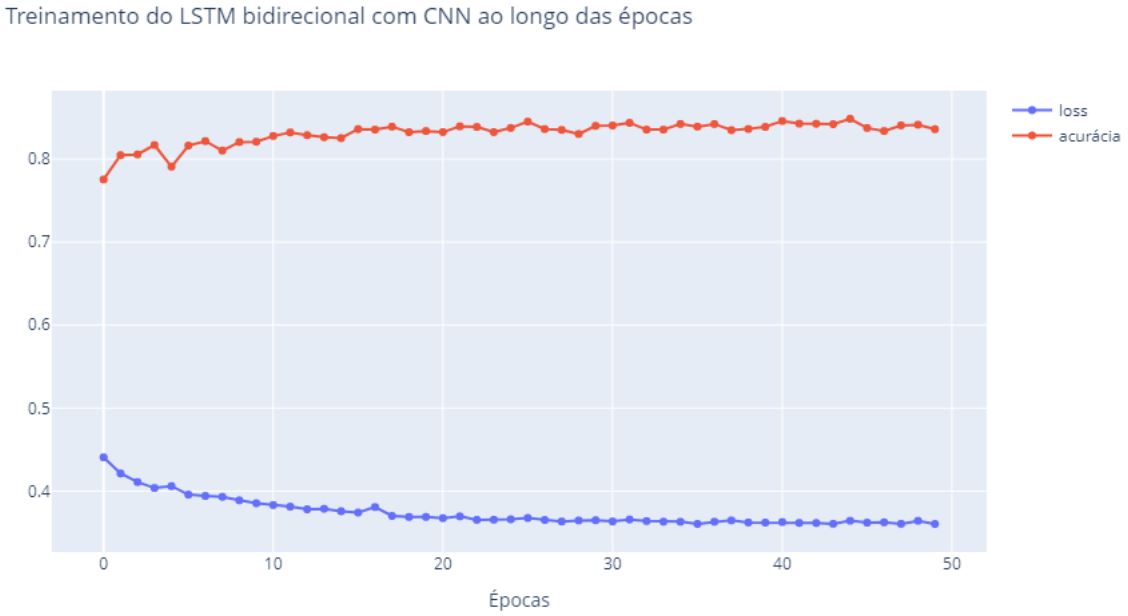


Fonte: Autores

Figura 26: Matriz de confusão do modelo LSTM com CNN

7.3.5 LSTM bidirecional com CNN

O treinamento dos modelos seguiu o padrão apresentado na Figura 27, com o valor da função de perda diminuindo e o de acurácia, crescendo lentamente e estabilizando, ao longo das épocas, semelhante à arquitetura mostrada anteriormente:



Fonte: Autores

Figura 27: Curva de treinamento do modelo LSTM bidirecional com CNN

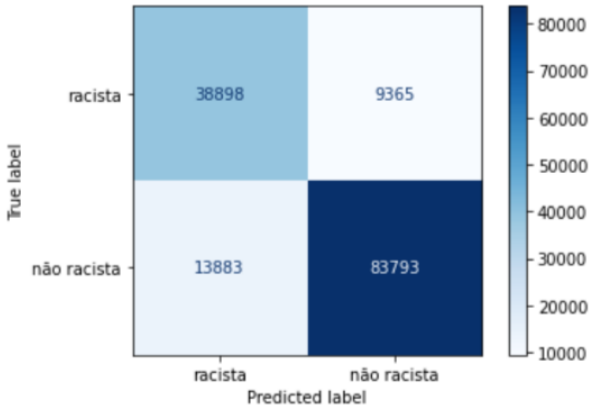
Comparando-se os resultados das diferentes iterações para a validação do modelo, tem-se a Tabela 15:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8427	0,8487	**0,8407**	0,8401	0,8357	0,8407	4,2e−3
Precisão	0,7448	0,7561	**0,7370**	0,7333	0,7279	0,7370	9,8e−3
Recall	0,8028	0,8048	0,8060	0,8166	**0,8058**	0,8058	4,8e−3
F1-score	**0,7727**	0,7797	0,7699	0,7727	0,7649	0,7727	4,8e−3

Fonte: Autores

Tabela 15: Métricas dos modelos de LSTM bidirecional com CNN

Em geral, percebe-se que os resultados são próximos mas, ainda assim, superiores ao do LSTM com CNN. Mais do que isso, as métricas obtidas são notadamente melhores que as do LSTM bidirecional puro. De forma geral, a aplicação de uma camada convolucional nos modelos gerou um impacto muito positivo a eles. O modelo 3 é o que mais se aproxima da mediaa dos resultados. Logo, utilizando-o como referência, obtém-se a matriz de confusão da Figura 28:



Fonte: Autores

Figura 28: Matriz de confusão do modelo LSTM bidirecional com CNN

7.4 Modelos com LSI

A camada de Embedding baseada em *Latent Semantic Indexing* é aplicada aos dados antes de passarem para os modelos, conforme explicado em seções anteriores. Uma síntese das métricas medianas obtidas em cada uma das arquiteturas de modelo utilizadas segue na Tabela 16 abaixo:

Comparativo dos modelos						
Métricas	Baseline	LSTM	LSTM Bi	CNN-1D	LSTM e CNN-1D	LSTM Bi e CNN-1D
Acurácia	0,9228	0,8828	0,8686	0,7992	0,8776	0,8968
Precisão	0,9332	0,8035	0,7975	0,8271	0,7935	0,8240
Recall	0,8271	0,8547	0,7941	0,5049	0,8529	0,8724
F1-score	0,8768	0,8295	0,8026	0,6282	0,8237	0,8481

Fonte: Autores

Tabela 16: Métricas dos modelos de Bayes

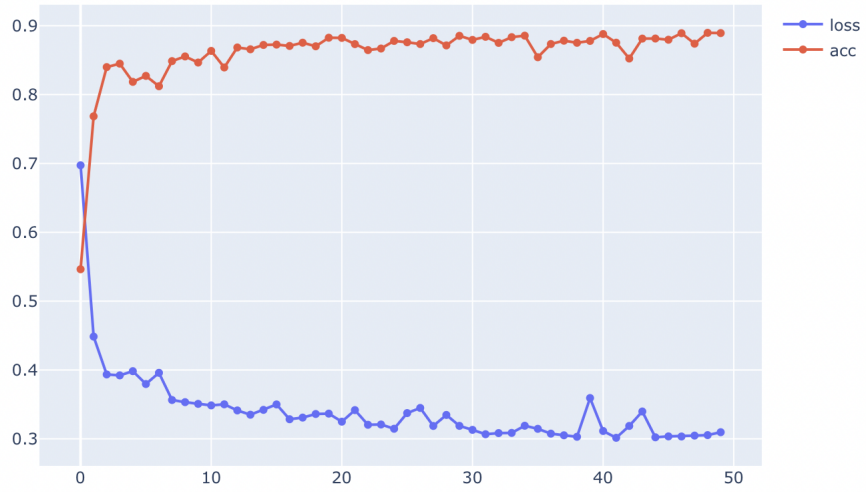
Apesar de nenhum dos modelos ter efetivamente superado o modelo de *baseline*, as métricas obtidas com o *LSI* são, em geral, melhores que sem ele, para todos os modelos, com exceção do CNN-1D. Isso mostra não apenas a importância da redução de dimensionalidade dos dados de entrada e da consequente diminuição de informação redundante ou desnecessária, mas o potencial dos modelos. Testar diferentes hiperparâmetros e pré-processamentos pode levar a resultados ainda mais interessantes.

É importante mencionar ainda que a inclusão de uma camada convolucional no modelo de LSTM bidirecional continua contribuindo para melhores performances do mesmo. É capaz de fazê-lo superar as métricas do LSTM, o que não acontecia sem a presença da camada em questão.

A seguir, são analisados os resultados dos treinamentos de cada um dos modelos, individualmente.

7.4.1 LSTM

Apesar das curvas de aprendizado terem variado um pouco de iteração para iteração, em geral elas seguiram o mesmo padrão indicado na Figura 29, iniciando com valores altos de perda e baixos de acurácia nas primeiras épocas, e rapidamente estabilizando nas seguintes:



Fonte: Autores

Figura 29: Curva de treinamento do modelo LSTM com LSI

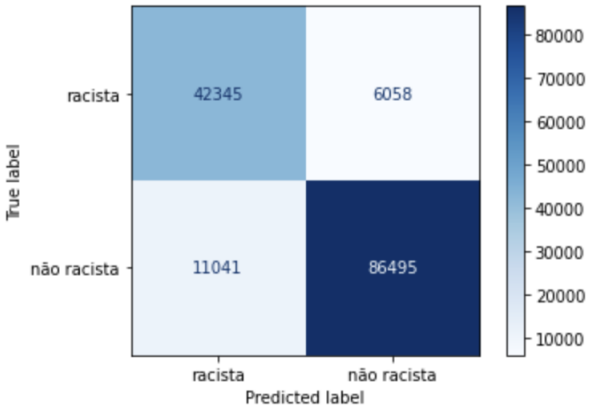
Novamente, cinco versões do modelo foram treinadas, a partir dos *datasets* gerados a partir da *Random Permutation Cross Validation*. Seus resultados seguem na Tabela 17:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8953	0,8856	0,8760	0,8802	**0,8828**	0,8828	6,5e−3
Precisão	0,8350	0,8230	0,7844	**0,8035**	0,7932	0,8035	1,8e−2
Recall	**0,8547**	0,8361	0,8620	0,8473	0,8748	0,8547	1,3e−2
F1-score	0,8447	**0,8295**	0,8213	0,8248	0,8320	0,8295	8,0e−3

Fonte: Autores

Tabela 17: Métricas dos modelos LSTM com LSI

Comparando diretamente com o LSTM sem LSI, percebe-se que as métricas melhoram consideravelmente com o *Word Embedding* em questão. Na primeira arquitetura testada, a acurácia e o recall não superam o valor de 80%, e a precisão e o F1-score não chegam a 70%. Entretanto, nesta segunda, todas superam 80%, com um desvio padrão relativamente baixo, mostrando uma certa consistência nos modelos treinados. Tomando o modelo da iteração 5 como referência, por ter as métricas como um todo mais próximas das medianas, chega-se à matriz de confusão da Figura 30:

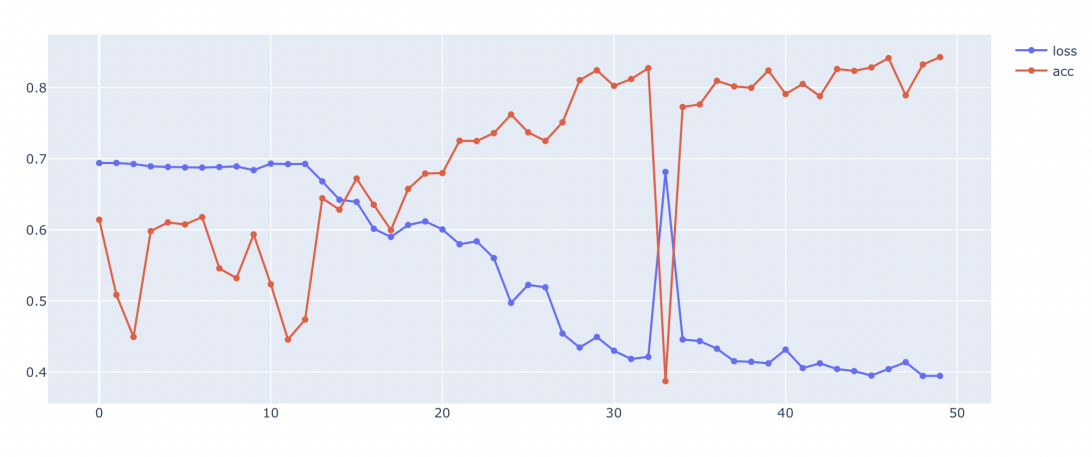


Fonte: Autores

Figura 30: Matriz de confusão do modelo LSTM com LSI

7.4.2 LSTM bidirecional

Apesar das curvas de acurácia e perda ao longo das épocas se mostrarem mais irregulares que a das outras arquiteturas desenvolvidas no projeto, como é possível observar na Figura 31, é possível identificar um progresso de aprendizagem no decorrer do tempo. A acurácia aumenta com o passar das épocas, assim como o valor da perda diminui, de forma mais acentuada que os outros modelos observados no trabalho.



Fonte: Autores

Figura 31: Curva de treinamento do modelo LSTM bidirecional com LSI

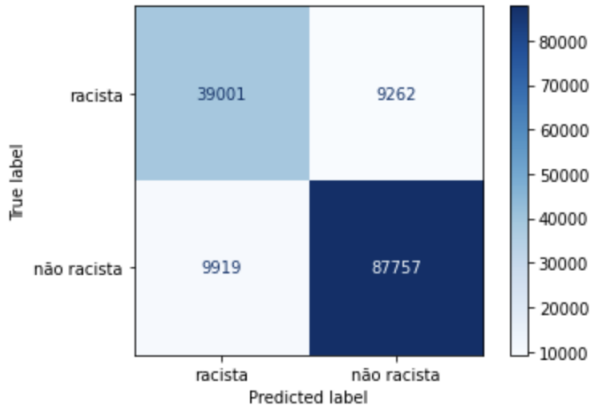
As métricas dos cinco treinamentos seguem na Tabela 18:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8409	0,8774	**0,8686**	0,8356	0,8829	0,8686	1,9e−2
Precisão	0,7677	0,8195	**0,7972**	0,7571	0,8437	0,7972	3,2e−2
Recall	0,7492	0,8099	0,8081	0,7452	**0,7941**	0,7941	2,8e−2
F1-score	0,7583	0,8147	**0,8026**	0,7511	0,8181	0,8026	2,9e−2

Fonte: Autores

Tabela 18: Métricas dos modelos LSTM bidirecional com LSI

Neste caso, os desvios padrões se mantiveram pequenos, apesar das métricas variarem um pouco mais que os modelos sem LSI. Os resultados obtidos não foram tão altos quanto do LSTM com LSI. Contudo, é interessante notar que aplicando-se esta técnica de *Word Embedding* antes da entrada dos dados no modelo, o LSTM bidirecional passou a efetivamente aprender durante o treinamento, chegando a atingir boas métricas sobre o *dataset* de teste. O modelo da iteração 3 foi o que mais se aproximou de um modelo "mediano", e sua matriz de confusão sobre o *dataset* de teste é a da Figura 32:

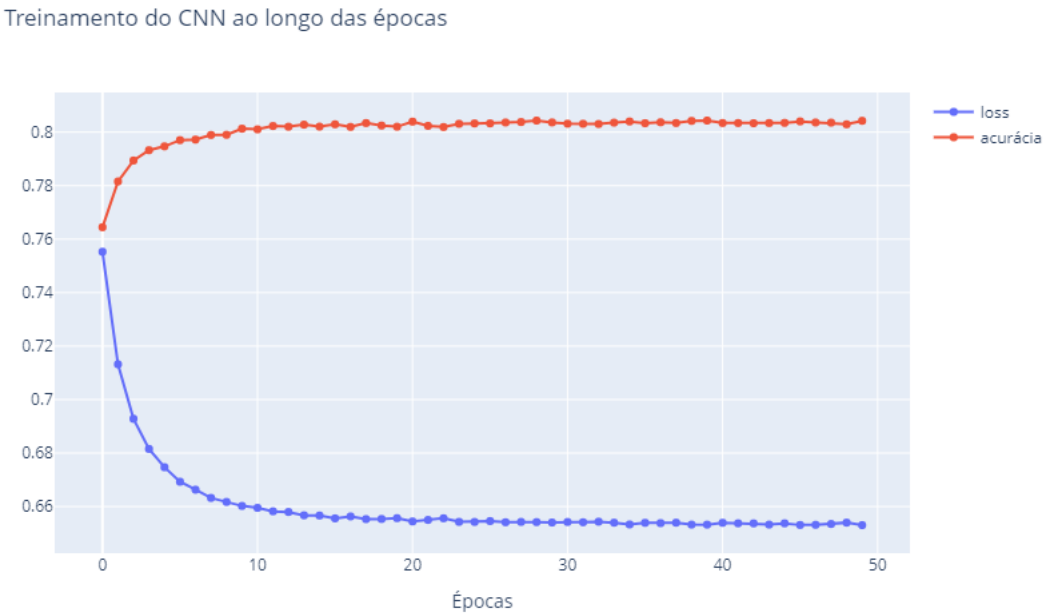


Fonte: Autores

Figura 32: Matriz de confusão do modelo LSTM bidirecional com LSI

7.4.3 CNN

A curva de aprendizado, mostrada na Figura 33 se mostrou bastante regular para o modelo de CNN com LSI, com a acurácia sobre o *dataset* de validação aumentando e o valor da função de perda, diminuindo, gradativamente, e estabilizando após a vigésima época.



Fonte: Autores

Figura 33: Curva de treinamento do modelo CNN com LSI

As métricas obtidas com os modelos gerados em cada uma das iterações foram apre-

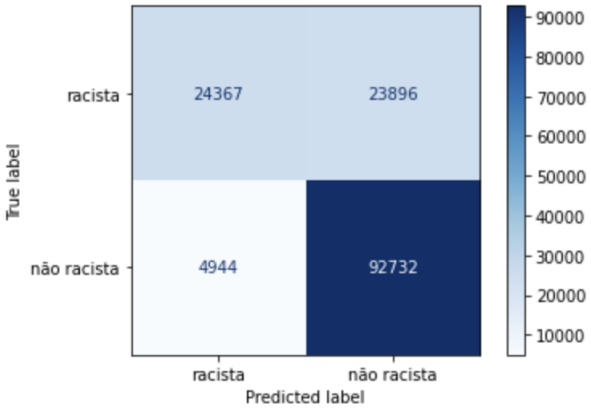
sentadas na Tabela 19:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8064	0,7953	0,8024	0,7977	**0,7992**	0,7992	3,9e−3
Precisão	0,8248	**0,8271**	0,8313	0,8067	0,8284	0,8271	8,7e−3
Recall	0,5318	0,4864	**0,5049**	0,5160	0,4976	0,5049	1,6e−2
F1-score	0,6467	0,6126	**0,6282**	0,6294	0,6217	0,6282	1,1e−2

Fonte: Autores

Tabela 19: Métricas dos modelos CNN com LSI

Frente aos outros modelos com LSI, este apresentou as piores métricas, com exceção da precisão, que foi a maior obtida dentre os experimentos desta categoria. Sua performance é bastante inferior a do CNN sem LSI e, de todas as arquiteturas testadas ao longo do projeto, supera apenas o LSTM sem LSI. O recall é o mais baixo até então, aproximando-se do obtido durante os testes preliminares, com o *dataset* pequeno. O modelo da iteração 3 foi o que mais se aproximou das métricas medianas de forma geral, e sua matriz de confusão é a da Figura 34:

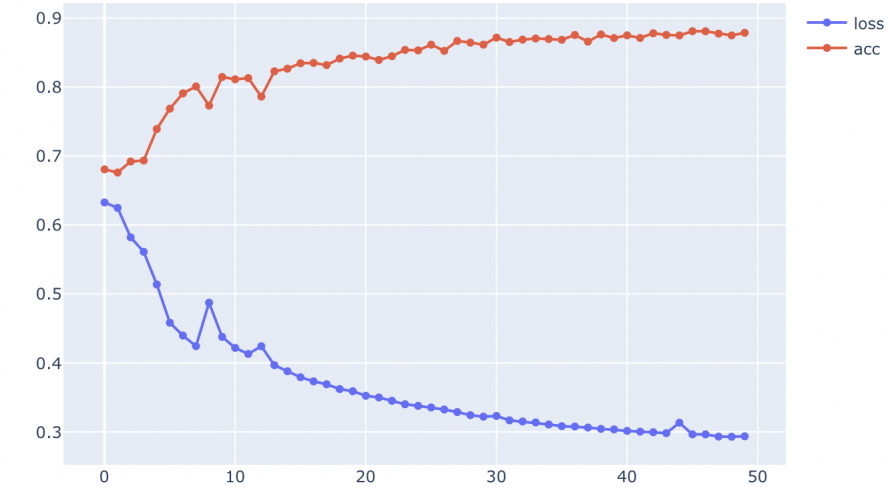


Fonte: Autores

Figura 34: Matriz de confusão do modelo CNN com Embedding

7.4.4 LSTM com CNN

Nos treinamentos destes modelos, pôde-se ver pela Figura 35, claramente, a evolução dos valores de perda e de acurácia ao passar das épocas, seguindo um progresso mais acentuado nos primeiros ciclos, e tendendo à estabilidade nos últimos.



Fonte: Autores

Figura 35: Curva de treinamento do modelo LSTM e CNN com LSI

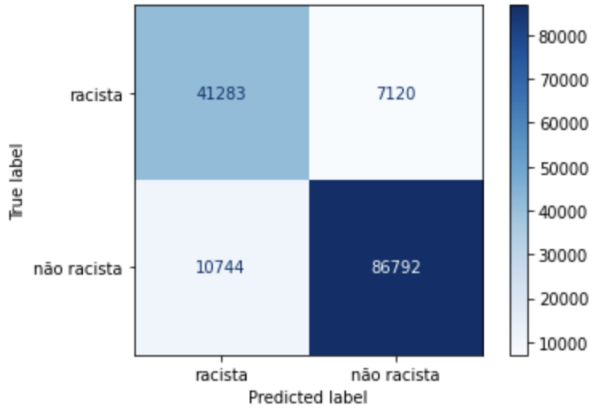
Assim como a maioria dos outros modelos, as métricas giraram em torno de 80%, e os desvios padrões se mantiveram na ordem de 1e−2. Como se pode ver na Tabela 20:

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad
Acurácia	0,8601	0,8762	0,8847	0,8800	**0,8776**	0,8776	8,3e−3
Precisão	0,7654	0,7827	0,8115	0,7979	**0,7935**	0,7935	1,5e−2
Recall	0,8362	0,8692	0,8486	0,8566	**0,8529**	0,8529	1,1e−2
F1-score	0,7993	**0,8237**	0,8296	0,8262	0,8221	0,8237	1,1e−2

Fonte: Autores

Tabela 20: Métricas dos modelos de LSTM e CNN com LSI

Neste caso, a adição da camada convolucional ao LSTM não contribuiu para uma melhora de desempenho do modelo, tornando suas métricas ainda ligeiramente menores que as do LSTM com LSI apenas. Porém, a presença do *Word Embedding* em questão contribuiu para que os resultados superassem os da versão sem ele. O modelo obtido da iteração 5 foi o que mais se aproximou das medianas. Calculando a matriz de confusão para ele, com relação ao *dataset* de teste, obtém-se a Figura 36:

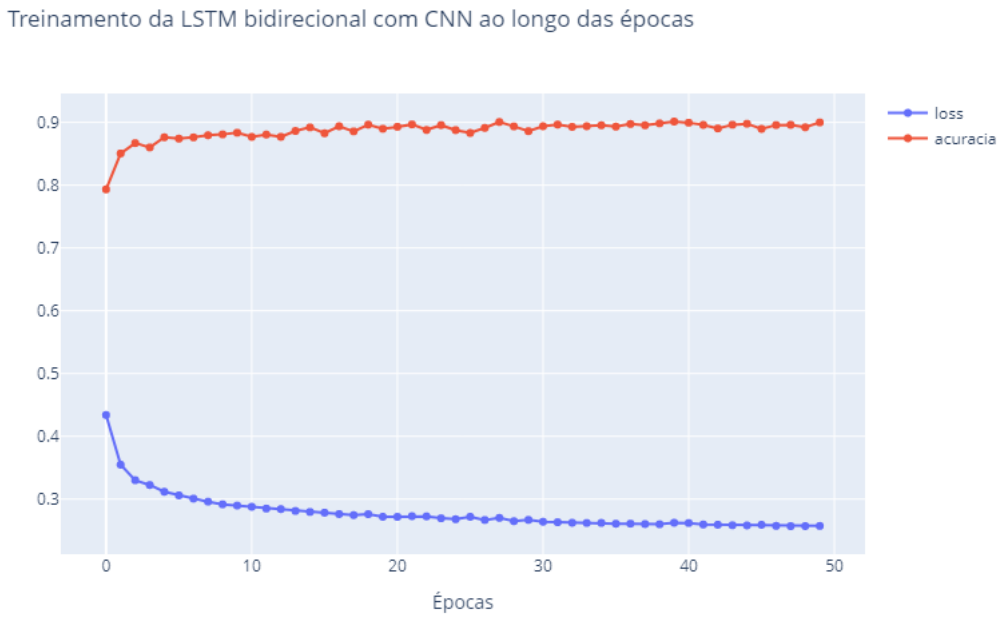


Fonte: Autores

Figura 36: Matriz de confusão do modelo LSTM com CNN e Embedding

7.4.5 LSTM bidirecional com CNN

A convergência dos pesos do modelo se deu logo nas primeiras épocas de treinamento do modelo, com a acurácia variando pouco e a perda diminuindo de forma pouco mais perceptível, como visto na Figura 37.



Fonte: Autores

Figura 37: Curva de treinamento do modelo LSTM bidirecional e CNN com LSI

Conforme é possível visualizar na Tabela 21 de métricas das diferentes iterações a seguir, os desvios padrões entre os diferentes modelos foram bastante pequenos, e todas

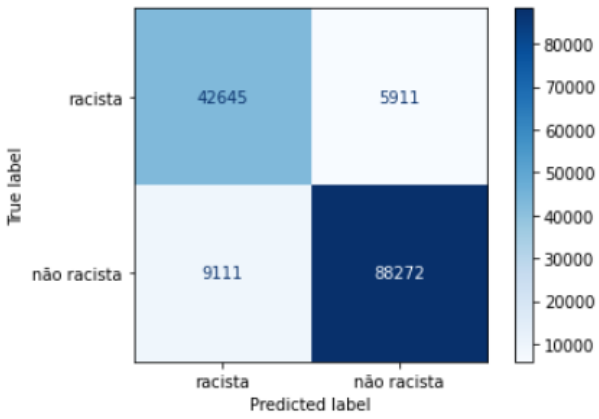
as estatísticas foram superiores a 81%.

Comparativo dos modelos							
Métricas	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Mediana	Desv. Pad.
Acurácia	0,8985	0,8971	0,8937	**0,8968**	0,8954	0,8968	1,6e−3
Precisão	0,8341	**0,8240**	0,8141	0,8316	0,8228	0,8240	7,1e−3
Recall	0,8677	0,8783	0,8794	0,8653	**0,8724**	0,8724	5,6e−3
F1-score	0,8506	0,8502	0,8455	**0,8481**	0,8469	0,8481	1,9e−3

Fonte: Autores

Tabela 21: Métricas dos modelos de LSTM bidirecional e CNN com LSI

Vale ressaltar o impacto positivo da junção da camada convolucional ao modelo de LSTM bidirecional. Dentre os modelos com LSI, este é o que apresenta as melhores métricas (com exceção da precisão que é um pouco menor que a do modelo CNN com LSI). O modelo extraído da segunda iteração foi o que mais se aproximou das métricas medianas obtidas a partir dos cinco testes. Sua matriz de confusão com relação ao *dataset* de teste é a da Figura 38:



Fonte: Autores

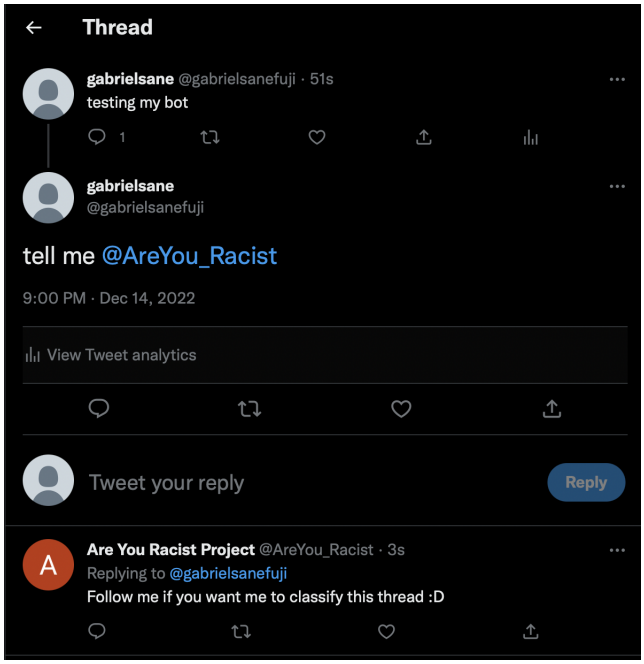
Figura 38: Matriz de confusão do modelo LSTM bidirecional com CNN e Embedding

7.5 Testes finais com o *bot*

Foram feitos alguns testes com o *bot* finalizado (cuja página encontra-se no link https://twitter.com/AreYou_Racist), para verificar se suas funcionalidades básicas estavam de acordo com o que se tinha planejado. Ou seja, ele deveria conseguir recuperar

menções, responder aos usuários, mas apenas aqueles que seguem a página da conta automatizada. Isso, tendo em vista que suas funções seriam ativadas dentro do período de tempo definido pelo *Scheduler*.

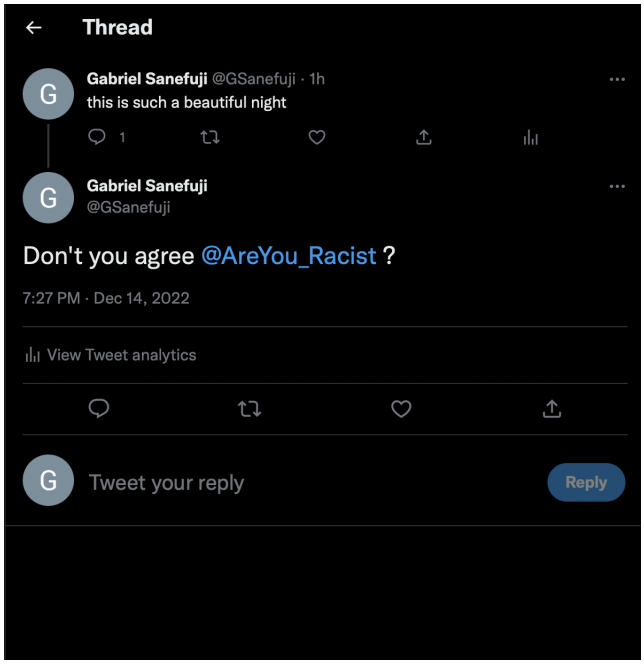
Para validar que o *bot* só responderia a chamadas de seus seguidores, foram feitas chamadas com uma conta que não o seguia. Ao encontrar a mensagem deste usuário o robô responde alertando que consegue analisar *tweets* por pedido de seguidores de sua página. Esse comportamento é ilustrado na Figura 39.



Fonte: Autores

Figura 39: *Bot* respondendo com mensagem padrão ao usuário que ainda não o segue

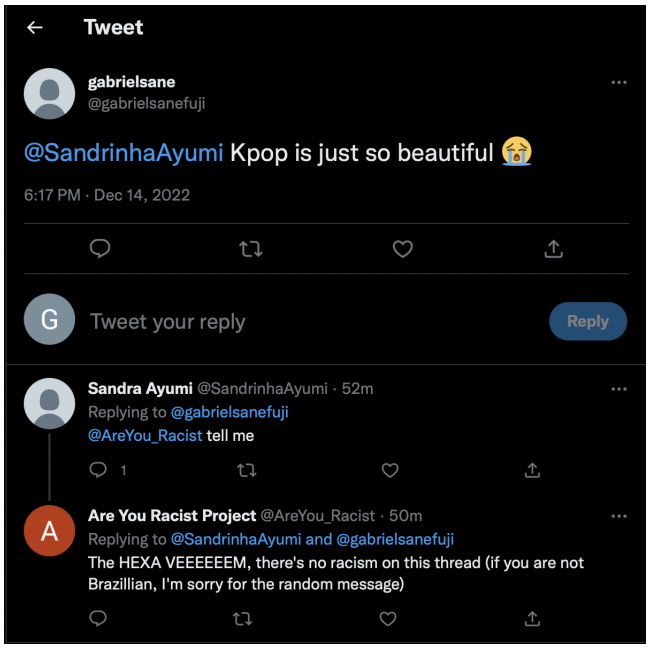
Outro ponto importante é a utilização correta da frase de chamada. Isso porque, se a conta respondesse a todas as suas menções, não seria viável ao programa e nem conveniente aos usuários. Para isso, foram feitas chamadas sem a frase correta (que foi definida anteriormente como *tell me*. O resultado, como esperado, é que nenhuma resposta seja dada, e pode ser visto na Figura 40



Fonte: Autores

Figura 40: Usuário sem resposta

Além disso, verifica-se que para uma conversa sem contexto de injúria racial, com a chamada correta, o *bot*, de fato, apresenta uma mensagem não violenta e que informa a ausência de termos com conotação negativa, como visto na Figura 41.

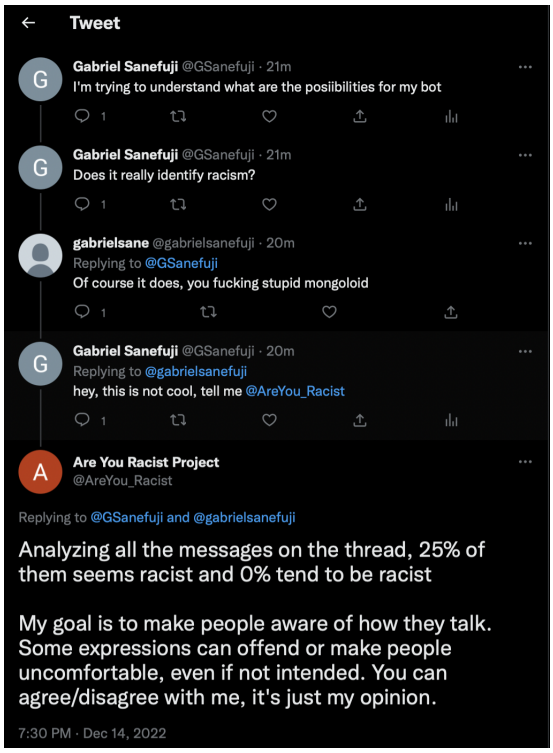


Fonte: Autores

Figura 41: Resposta do *bot* frente a uma conversa sem racismo

Em contrapartida, para *threads* com mensagens de teor negativo, o robô averigua

qual a porcentagem de frases racistas dentro da conversa, sempre ressaltando que em nenhum momento impõe opiniões. Esse comportamente é visto na Figura 42, no qual de quatro *tweets*, um em específico configura-se como racista, e, portanto, 25% desse diálogo apresenta-se como tal.

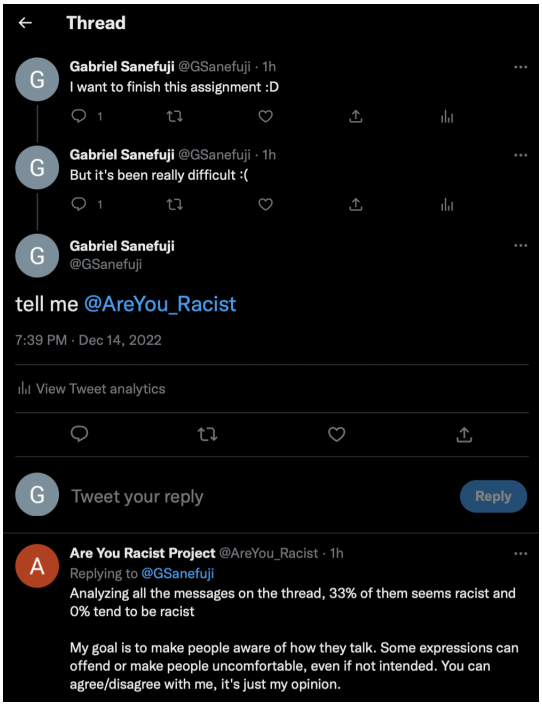


Fonte: Autores

Figura 42: Resposta do *bot* frente a uma conversa com racismo

Ainda que suas funções principais estejam funcionais, o *bot* ainda apresenta algumas falhas com relação a sua classificação de mensagens. Isso pode ser verificado na Figura 43. No qual nenhuma das frases (*I want to finish this assignment*, *But it's been really difficult* e *tell me*) possuem conotação particularmente racista. Contudo, o robô ainda assim classificou uma delas como de teor negativo.

Ao se explorar o *log* da *Cloud Function*, percebeu-se que a frase *I want to finsh this assignment* foi classificada como racista. Uma explicação para isso poderia ser a recorrência de algum dos termos dentro do treinamento em frases com conotação negativa. A palavra *finish*, que do inglês significa acabar ou terminar, pode sim ser utilizada em frases depreciativas. Um exemplo é a frase *This player is finished* que, em tradução direta, poderia ser lida como "este jogador está acabado". Mesmo que não seja propriamente uma mensagem racista, ainda é um exemplo de como a palavra pode ter pertencido a frases cujo significado semântico não era positivo.



Fonte: Autores

Figura 43: Resposta incorreta dada pelo *bot*

Outra explicação se dá pelo fato de que nenhum modelo obteve uma avaliação de 100% em qualquer uma das métricas utilizadas. Ou seja, o *bot* está sujeito a erros, assim como qualquer outro ser humano, dentro de uma interpretação da semântica de mensagens.

8 CONCLUSÕES

Neste capítulo, são abordadas as conclusões do grupo acerca do projeto e possibilidades de trabalhos futuros para seu aprimoramento.

8.1 Conclusões finais

Primeiramente, através do desenvolvimento do trabalho, é possível concluir que a área de Processamento de Linguagem Natural – e de Aprendizagem de Máquina como um todo – é bastante exploratória: inúmeras arquiteturas de modelos e formas de se pré-processar dados existem e podem ser juntadas ou adaptadas. Para cada problema, uma determinada configuração se encaixa melhor, porém encontrá-la depende de reconhecimento de experiências similares anteriores e de exploração.

Além disso, a importância dos dados para os modelos ficou cada vez mais evidente ao longo do projeto. Garantir a qualidade deles assim como um alto volume de amostras é crucial para o desenvolvimento de um projeto confiável, sem viés, mais abrangente e com melhores métricas. Por meio dos resultados observados nos testes preliminares reduzidos e nos testes com *dataset* completo, percebeu-se a maior consistência das métricas nas diferentes iterações com diferentes dados de treino, teste e validação. A qualidade pode ser garantida pelas técnicas de pré-processamento de dados empregadas no início do ciclo do TDSP, como pela verificação manual dos textos sendo utilizados, por exemplo.

A exploração de diferentes parâmetros e hiperparâmetros dos modelos criados auxilia na obtenção de melhores resultados finais. Em contrapartida, demanda bastante tempo e recursos para os testes, como GPU, memória RAM e computadores especiais para tais processamentos.

Foi também possível validar que é possível classificar *tweets* em racistas ou não racistas. O racismo implícito, aquele que não se utiliza de palavras de conotação negativa, ainda é um desafio ,visto que os modelos procuram principalmente a utilização de algumas

palavras em específico para sua classificação. Ainda assim, os resultados obtidos se mostraram bastante promissores, principalmente no que se refere ao racismo mais explícito, mesmo com as limitações impostas pela fonte de dados de treino e pelos recursos operacionais disponíveis da equipe.

8.2 Trabalhos futuros

Existem melhorias e estudos que não puderam ser feitos, devido a restrições de tempo e recursos. Mesmo com os bons resultados obtidos no projeto, existem outros experimentos que poderiam melhorar substancialmente o desempenho dos modelos e do *bot*, ou ao menos trazer outras conclusões que podem vir a ser interessantes ao meio acadêmico.

Desse modo, o primeiro ponto que poderia ser destacado e talvez o que tenha sido aquele com o qual o grupo enfrentou maiores problemas se dá pela obtenção de uma quantidade maior de dados relevantes ao tema de racismo contra asiáticos. Esses dados poderiam ser obtidos pela própria API do *Twitter*, mas com o acesso de pesquisa acadêmica (Academic Research Access), possibilitando a busca por mensagens enviadas nos períodos de início da pandemia da Covid-19, quando discursos de ódio se fizeram mais presentes na rede social. A contribuição desses dados com os modelos já produzidos poderia trazer efeitos muito positivos ao agregar novas perspectivas sobre o tema através das mensagens disseminadas na época.

Outro ponto que poderia ser destacado se dá pela maior exploração dos efeitos da variação no valor de hiperparâmetros. Por conta do tempo disponível à equipe, não foi possível realizar maiores testes com uma variação maior dos valores de *learning rate*, *batch size*, *dropout rate*, dentre outros.

O desenvolvimento dos modelos para a classificação de *tweets* em português (ou mesmo outras línguas) é uma possibilidade de trabalho futuro. Sendo o Brasil um dos países com maior número de descendentes asiáticos no mundo, e dada a alta presença dos brasileiros nas redes sociais, essa iniciativa agregaria valor à comunidade local, onde o projeto está efetivamente sendo criado.

Com relação ao *bot*, diversas melhorias podem ser feitas. Uma melhor implementação em *Cloud Function* pode ser feita, com a separação entre funcionalidades de pesquisa e de predição em diferentes funções que comunicam entre si, estruturando melhor o código. Outro ponto que poderia ser levantado é no desenvolvimento de *features* para a análise automatizada acerca dos dados que foram coletados durante o funcionamento

do robô, de forma que o mesmo pudesse levantar uma diminuição ou aumento do número de ocorrências de racismo em sua página do *Twitter*, por exemplo, tornando-o uma conta mais completa sobre a disseminação de informações do tema.

REFERÊNCIAS

JEUNG, R.; HORSE, A. J. Y.; CAYANAN, C. *STOP AAPI HATE NATIONAL REPORT*. 2021. Disponível em: ⟨https://stopaapihate.org/wp-content/uploads/2021/05/Stop-AAPI-Hate-Report-National-210506.pdf⟩.

HE, B. et al. Racism is a virus: anti-asian hate and counterspeech in social media during the covid-19 crisis. *ASONAM '21: Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, n. 7, p. 90–94, nov. 2021.

VIDGEN, B. et al. Detecting east asian prejudice on social media. *Proceedings of the Fourth Workshop on Online Abuse and Harms*, p. 162—-172, nov. 2020.

MICHIE, D.; SPIEGELHALTER, D.; TAYLOR, C. *Machine Learning, Neural and Statistical Classification*. 1999. Disponível em: ⟨hhttp://www1.maths.leeds.ac.uk/charles/statlogi⟩.

JURAFSKY, D.; MARTIN, J. *Speech and Language Processing*: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2017. Disponível em: ⟨https://web.stanford.edu/~jurafsky/slp3/⟩.

JAMES, G. et al. *An Introduction to Statistical Learning*. [S.l.]: Springer, 2017. ISBN 9781461471370.

ONU. *International Convention of the Elimination of All Forms of Racial Discrimination*. 1965. Acesso em 2022-03-15. Disponível em: ⟨https://www.un.org/en/development/desa/population/migration/generalassembly/docs/globalcompact/A_RES_2106(XX).pdf⟩.

NEWS, N. *Anti-Asian hate crimes increased 339 percent nationwide last year, report says*. 2022. Acesso em 2022-03-15. Disponível em: ⟨https://www.nbcnews.com/news/asian-america/anti-asian-hate-crimes-increased-339-percent-nationwide-last-year-repo-rcna14282⟩.

STATISTA. *Twitter – Statistics & Facts*. 2022. Disponível em: ⟨https://www.statista.com/topics/737/twitter/\#dossierContents__outerWrapper⟩.

MICROSOFT. *The Team Data Science Process lifecycle*. 2022. Acesso em 2022-02-16. Disponível em: ⟨https://docs.microsoft.com/en-us/azure/architecture/data-science-process/lifecycle⟩.

STODDEN, v. The Data Science Life Cycle: A Disciplined Approach to Advancing Data Science as a Science. *Communications of the ACM*, n. 7, p. 58–66, jul. 2020.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. ISBN 9780387310732.

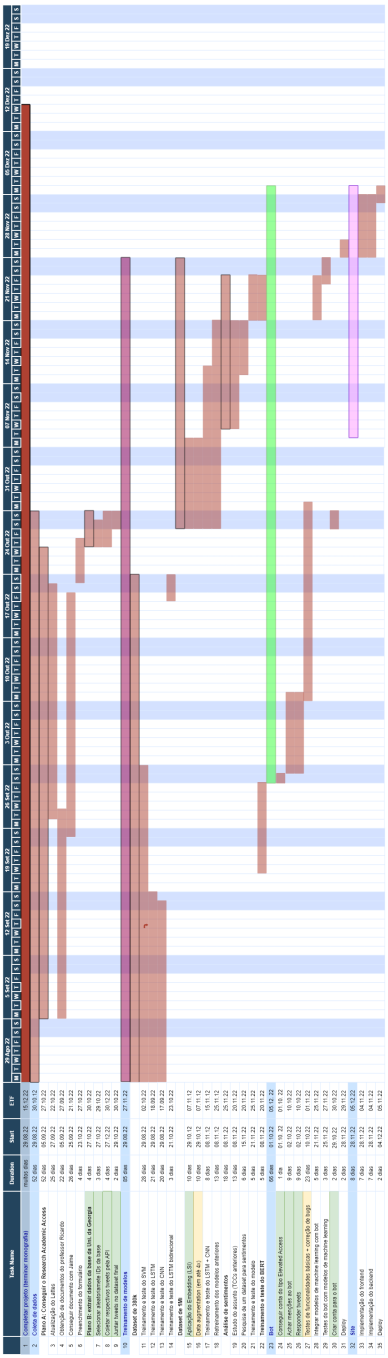
DEVELOPERS scikit-learn. *sci-kit learn API reference*. 2022. Acesso em 2022-03-16. Disponível em: ⟨https://scikit-learn.org/stable/modules/classes.html⟩.

Hatebase Inc. *Hatebase*. 2022. Acesso em 2022-04-20. Disponível em: ⟨https://hatebase.org/⟩.

KING, G.; ZENG, L. Logistic regression in rare events data. *Political Analysis*, v. 9, p. 137 – 163, 2001.

GOOGLE. *Configure cron job schedules*. 2022. Acesso em 2022-12-10. Disponível em: ⟨https://cloud.google.com/scheduler/docs/configuring/cron-job-schedules⟩.

APÊNDICE A – GANTT DO PROJETO



Fonte: Autores

Figura 44: Gráfico de Gantt do Projeto de Formatura (Fonte: Autores)