UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

INSTITUTO DE COMPUTAÇÃO

INTELIGÊNCIA ARTIFICIAL PARA CLASSIFICAÇÃO DE SENTIMENTOS EM POSTAGENS DO TWITTER: UMA ANÁLISE COMPARATIVA DO DESEMPENHO DE ALGORITMOS DE PROCESSAMENTO DE LINGUAGEM NATURAL E APRENDIZADO DE MÁQUINA

GABRIEL BRUM TAVARES

SEROPÉDICA / RIO DE JANEIRO

2023

Gabriel Brum Tavares

INTELIGÊNCIA ARTIFICIAL PARA CLASSIFICAÇÃO DE SENTIMENTOS EM POSTAGENS DO TWITTER: UMA ANÁLISE COMPARATIVA DO DESEMPENHO DE ALGORITMOS DE PROCESSAMENTO DE LINGUAGEM NATURAL E APRENDIZADO DE MÁQUINA

Trabalho de Final de Curso apresentado à Coordenação do Curso de Graduação da Universidade Federal Rural do Rio de Janeiro, como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em MÊS de ANO.

BANCA EXAMINADORA

_____

Prof. Dr. Raimundo José Macário Costa - Orientador

UFRRJ

_____

Prof. Dr. Jorge Juan Zavaleta Gavidia - Coorientador

UFRJ

_____

Prof. Dr. Tiago Cruz de França

UFRRJ

_____

Prof. Dr. Nilton Jose Rizzo

UFRRJ

AGRADECIMENTOS

Agradeço especialmente à minha mãe, que me apoiou e me deu todo o suporte para que eu pudesse passar por essa longa e difícil jornada. Agradeço também aos professores Raimundo Macário e Jorge Zavaleta, pela paciência e apoio para que esse trabalho pudesse ser realizado.

RESUMO

A análise de sentimentos utilizando inteligência artificial é uma abordagem cujo objetivo é identificar e compreender as emoções expressas em textos. A rede social *Twitter* (X) é conhecida por permitir que seus usuários se comuniquem de forma curta e objetiva através de postagens chamadas *tweets*, já tendo mostrado relevância em grandes eventos ao redor do mundo, se mostrando como uma interessante fonte de dados para realizar estas análises. Entretanto, devido ao grande volume de informação, uma análise manual se torna inviável. Para tornar esse processo mais rápido e eficiente, uma alternativa é utilizar técnicas como processamento de linguagem natural (PLN), ferramentas específicas para lidar com textos, e algoritmos supervisionados de aprendizado de máquina. Neste trabalho serão utilizadas as bibliotecas e ferramentas de PLN NLTK (*Natural Language Toolkit*), Vader (*Valence Aware Dictionary and sEntiment Reasoner*) e *TextBlob*, e também os algoritmos de Regressão Logística, *Multi-layer Perceptron* (MLP) e *K-Nearest Neighbors* (KNN), para processar e analisar os textos destas postagens. O objetivo é classificar e comparar objetivamente os resultados obtidos, utilizando as métricas fornecidas pela ferramenta *scikit-learn*, para avaliar e comparar a eficácia e o desempenho das bibliotecas e algoritmos propostos.

Palavras–chave: Análise de Sentimentos, Inteligência Artificial, Processamento de Linguagem Natural, Machine Learning, Twitter.

ABSTRACT

Sentiment analysis using artificial intelligence is an approach whose objective is to identify and understand emotions expressed in texts. The social network Twitter (X) is known for allowing its users to communicate in a short and objective way through posts called tweets, having already shown relevance in major events around the world, proving itself as an interesting source of data to carry out these analyses. However, due to the large volume of information, manual analysis becomes unfeasible. To make this process faster and more efficient, an alternative is to use techniques such as natural language processing (NLP), specific tools for dealing with texts, and supervised machine learning algorithms. In this work, the PLN libraries and tools NLTK (Natural Language Toolkit), Vader (Valence Aware Dictionary and sEntiment Reasoner) and TextBlob will be used, as well as the Logistic Regression, Multi-layer Perceptron (MLP) and K-Nearest Neighbors algorithms (KNN), to process and analyze the texts of these posts. The objective is to objectively classify and compare the results obtained, using the metrics provided by the scikit-learn tool, to evaluate and compare the effectiveness and performance of the proposed libraries and algorithms.

Keywords: Sentiment Analysis, Artificial Intelligence, Natural Language Processing, Machine Learning, Twitter.

SUMÁRIO

1. Introdução

Atualmente, o volume de dados produzidos e compartilhados na Internet está aumentando exponencialmente. Segundo Galdino (2016):

> "A quantidade de dados gerados pela humanidade nos últimos anos aumentou exponencialmente. Segundo uma pesquisa recente, no ano 2000, 25% (vinte e cinco por cento) dos dados foram digitalizados, no ano de 2007, esse número saltou para 93% (noventa e três por cento) e no ano de 2013, foi para 98% (noventa e oito por cento). Este crescimento, devido principalmente a fatores como aumento do acesso a dispositivos eletrônicos e a popularização da internet, está gerando uma revolução no tratamento de dados." (p.2)

Este cenário cria um ambiente fértil para a extração de informações valiosas que podem ser utilizadas na construção de aplicações para utilidades públicas ou privadas, como por exemplo, serviços de atendimento ao cliente, ensino, sistemas de recomendação ou simplesmente *chatbots*. No entanto, também ajudam a entender, dentre outras coisas, o comportamento humano, mas também apresenta um desafio significativo: como processar e analisar essa quantidade avassaladora de informações?

Neste contexto, a Inteligência Artificial (IA) tem sido fundamental para ajudar a solucionar esse problema. A IA se refere a sistemas capazes de realizar tarefas que normalmente exigem inteligência humana, como reconhecimento de voz, tomada de decisão e compreensão de linguagem natural (Russell & Norvig, 2016).

Dentre as várias subáreas da IA, o Processamento de Linguagem Natural (PLN) destaca-se. O PLN é um campo de estudo focado no desenvolvimento de algoritmos e modelos capazes de compreender e manipular a linguagem humana, permitindo a interação entre humanos e máquinas de maneira natural (Jurafsky & Martin, 2019). Uma aplicação importante do PLN é a análise de sentimentos, que visa identificar e extrair informações subjetivas, como emoções e opiniões, de textos (Pang, Lee & Vaithyanathan, 2002).

Além disso, para lidar com o vasto volume de dados e ampliar a capacidade de análise, técnicas de Machine Learning (aprendizado de máquina) também têm sido empregadas. O Machine Learning envolve o desenvolvimento de algoritmos e modelos que permitem que

sistemas computacionais aprendam padrões e tomem decisões com base em dados, sem a necessidade de programação explícita (Alpaydin, 2014).

O uso dessas tecnologias tem implicações significativas em muitas áreas, incluindo análise de mídia social, atendimento ao cliente e inteligência de mercado. As redes sociais são a principal ferramenta que as pessoas utilizam para expressarem suas opiniões, gerando uma abundância de dados concentrados na web: texto, som, vídeos, imagens e misturas em formatos estruturados e não-estruturados. Quando tratados adequadamente, esses dados podem gerar ativos valiosos (Stephens-Davidowitz, 2017).

O *Twitter* (agora chamado de X), é uma rede social multiplataforma onde os usuários compartilham textos curtos chamados tweets. A plataforma também permite visualizar em tempo real os chamados *trending topics*, que são os assuntos que estão em alta no momento (O'Reilly & Milstein, 2012), se mostrando assim, uma boa fonte de dados para algoritmos de análise de sentimentos que serão utilizados neste trabalho.

1.2. Justificativa

Em uma época em que milhões de pessoas compartilham seus sentimentos diariamente na internet (Kumar & Sebastian, 2012), evidencia-se um notório crescimento da utilização das redes sociais, onde a análise de sentimentos se mostra como uma alternativa de ferramenta poderosa para entender as emoções e opiniões dos usuários em tempo real (Hutto & Gilbert, 2014). Segundo Elbagir e Yang (2019) ferramentas de PLN como *Vader* e *NLTK* (*Natural Language Toolkit*) conseguem alcançar um desempenho satisfatório na análise de sentimentos de textos oriundos de redes sociais, embora os autores ressaltem que um modelo treinado possivelmente obteria uma melhor acurácia. O mesmo foi observado por Kevin Lee (2021), que demonstrou a capacidade do *Vader* de produzir resultados de polaridade relativamente precisos, embora demonstre resultados inferiores a modelos treinados com aprendizado de máquina, uma vez que o *Vader* possui o léxico *rule-based* (léxico baseado em regras), ou seja, o algoritmo possui uma espécie de dicionário pré-definido que classifica os sentimentos dos textos de entrada de acordo com as palavras que o compõem, não podendo ser adaptado através de treinamento para diferentes cenários. Desta forma, este trabalho propõe-se a comparar a acurácia das técnicas de PLN e Aprendizado de Máquina na análise de sentimentos em textos obtidos da rede social *Twitter*. A análise comparativa dos resultados obtidos pelas abordagens permitirá identificar suas respectivas vantagens e limitações, contribuindo para a compreensão da melhor estratégia a ser adotada em diferentes contextos.

1.3. Objetivos

1.3.1. Objetivo Geral

O objetivo deste estudo é comparar as técnicas de Processamento de Linguagem Natural (PLN) e *Machine Learning* na análise de sentimentos de textos obtidos da plataforma *Twitter*. Dessa forma, esperamos contribuir para o avanço do conhecimento na área de análise de sentimentos, investigando as possibilidades de aplicação dessas técnicas em um cenário real de análise de dados provenientes de redes sociais.

1.3.2. Objetivos Específicos

- Entender o que é inteligência artificial e fornecer o seu contexto histórico;
- Estudar e entender a análise de sentimentos;
- Entender o que é Aprendizado de Máquina;
- Apontar como a inteligência artificial pode auxiliar com as tecnologias a realizar e classificar esses sentimentos;
- Apresentar as duas tecnologias de PLN (Vader e TextBlob);
- Apresentar os algoritmos de Aprendizado de Máquina supervisionado Regressão Logística, MLP (*Multilayer Perceptron*) e KNN (*K-Nearest Neighbors*);
- Mostrar os seus desempenhos e comparar os seus resultados com o *scikit-learn*, uma ferramenta que fornece métricas objetivas para algoritmos de aprendizado de máquina.

2. Pressupostos Teóricos

2.1. Inteligência Artificial

Segundo Russell e Norvig (2016), Inteligência Artificial (IA) é um campo multidisciplinar que busca simular ou imitar a inteligência humana em máquinas, para que elas sejam capazes de "aprender" a partir da experiência, ajustar-se a novas entradas e realizar tarefas que normalmente requerem inteligência humana. Em seu livro, os autores trazem oito definições de IA, organizadas em quatro categorias:

Tabela 1 – Definição de IA de acordo com os autores Russell e Norvig (2016).

Pensar de forma humana	**Pensar de forma racional**
"O esforço empolgante de fazer computadores pensarem... máquinas com mentes, no sentido completo e literal." (Haugeland, 1985) "A automação de atividades que associamos ao pensamento humano, como tomada de decisão, resolução de problemas, aprendizado..." (Bellman, 1978)	"O estudo das faculdades mentais por meio de modelos computacionais." (Charniak e McDermott, 1985) "O estudo das computações que possibilitam a percepção, o raciocínio e a ação." (Winston, 1992)
Agir de forma humana	**Agir de forma racional**
"A arte de criar máquinas que executam funções que requerem inteligência quando realizadas por pessoas." (Kurzweil, 1990) "O estudo de como fazer com que os computadores realizem tarefas nas quais, atualmente, as pessoas são melhores." (Rich e Knight, 1991)	"A Inteligência Computacional é o estudo do design de agentes inteligentes." (Poole et al., 1998) "A IA... está preocupada com o comportamento inteligente em artefatos." (Nilsson, 1998)

As definições na parte superior estão relacionadas aos processos de pensamento e raciocínio, enquanto as da parte inferior abordam o comportamento. As definições à esquerda medem o sucesso em termos de fidelidade ao desempenho humano, enquanto as da direita medem em relação a uma medida de desempenho ideal, chamada racionalidade. Um sistema é racional se fizer a "coisa certa", com base no que sabe.

A IA remonta a antiguidade, com histórias mitológicas de artefatos mecânicos e automáticos dotados de inteligência e consciência por artesãos deuses. A palavra "robô" apareceu pela primeira vez em uma peça de teatro tcheca em 1921 (Kaplan, 2016). Na década

de 1940, com o advento da máquina digital de computação baseada na obra de Alan Turing, a ideia de criar uma máquina que simulasse o cérebro humano começou a ganhar força (Copeland, 2019).

No verão de 1956, um grupo de cientistas jovens e otimistas, incluindo Marvin Minsky e John McCarthy, organizaram uma conferência no Dartmouth College, que é agora amplamente considerada o evento de nascimento da IA (McCarthy et al., 2006). Na conferência de Dartmouth, McCarthy definiu IA como "a ciência e engenharia de fazer máquinas inteligentes, especialmente programas de computador inteligentes" (McCarthy, 2007).

A IA teve altos e baixos ao longo de sua história, passando por períodos de otimismo e investimento intenso, conhecidos como "verões de IA", seguidos por períodos de desilusão e falta de financiamento, os "invernos da IA". Apesar desses desafios, a IA continuou a evoluir e expandir, adotando uma variedade de abordagens, desde as que buscam replicar a inteligência humana até as que tomam a forma de solução de problemas pragmáticos.

Atualmente existe um cenário de pesquisa bem estabelecido em inteligência artificial, e embora ainda haja muito a ser explorado, a inteligência artificial já é reconhecida como uma tecnologia capaz de replicar certas habilidades antes possuídas apenas pelos humanos (Silva & Mairink, 2019). A IA é uma parte integral da tecnologia moderna, impulsionando avanços em áreas tão diversas quanto reconhecimento de voz, processamento de linguagem natural, visão de computador e robótica. Com o desenvolvimento contínuo da capacidade de processamento e da disponibilidade de dados, a IA está preparada para continuar a desempenhar um papel cada vez mais importante em nossa sociedade e vida cotidiana (Nilsson, 2009). No entanto, após anos de pesquisa, ainda existem limitações na aplicação dessa abordagem, como a falta de técnicos especializados que traduzam o conhecimento e o comportamento humano em linguagem computacional.

A IA está revolucionando uma ampla gama de indústrias e setores. Desde saúde até serviços financeiros, está ajudando a melhorar a eficiência, a precisão e a eficácia das operações (Kapoor et al., 2017). Uma das características mais notáveis da IA é a sua capacidade de lidar com grandes volumes de dados e realizar tarefas repetitivas de forma mais eficiente do que os seres humanos (Davenport & Ronanki, 2018). Isso permite que as organizações economizem tempo e recursos valiosos, aumentando a produtividade e a eficiência. Além disso, a IA consegue operar 24 horas por dia, sete dias por semana, sem sofrer de fadiga ou perda de foco. Também se destaca a sua capacidade de análise e previsão. Os algoritmos de IA podem analisar grandes conjuntos de dados para identificar padrões e tendências que seriam quase impossíveis para os seres humanos detectarem (Agrawal et al., 2018). Isso pode ser útil em uma variedade

de contextos, como previsão do mercado de ações, detecção de fraudes, diagnóstico médico, dentre outras aplicações.

A IA também tem um impacto profundo no campo da medicina. Ela pode auxiliar os médicos a diagnosticarem doenças com maior precisão, personalizar os tratamentos para os pacientes, e até mesmo prever o surgimento de doenças antes mesmo que os sintomas apareçam (Jiang et al., 2017). Isto pode levar a melhores resultados de saúde e a um tratamento mais personalizado e eficiente.

Também demonstrando ser útil na educação, os sistemas de IA podem oferecer um ensino personalizado, adaptando-se às necessidades e habilidades e capacidades individuais dos estudantes, o que pode resultar em um melhor engajamento e resultados de aprendizagem (Luckin et al., 2016).

A IA também pode desempenhar um papel importante no combate às mudanças climáticas. Os algoritmos de IA podem ser utilizados para otimizar os sistemas de energia, prever a demanda energética, e facilitar a transição para fontes de energia renováveis (Rolnick et al., 2019).

No entanto, é importante notar que, embora a IA tenha muitos benefícios, também existem desafios e preocupações, incluindo questões de privacidade, segurança e ética. Assim, é essencial que a implementação da IA seja feita de forma consciente e responsável.

Um dos principais pontos de preocupação é o impacto da IA no mercado de trabalho. Muitos temem que a automação alimentada por IA possa resultar na perda de empregos em grande escala, especialmente nos setores de manufatura e de serviços (Arntz et al., 2016). Ainda que a IA possa criar novos empregos ao longo do tempo, a transição pode ser dolorosa para aqueles cujos empregos são deslocados.

Levantando preocupações acerca da privacidade, a coleta e análise de dados em grande escala, que é parte essencial de muitos sistemas de IA, pode levar a invasões e quebras de privacidade sem precedentes (Borgesius et al., 2015). Isso é particularmente problemático quando os sistemas de IA são utilizados em contextos sensíveis, como na saúde ou em serviços financeiros.

Outro desafio significativo é o viés na IA. Como os algoritmos de IA são treinados em dados históricos, eles podem perpetuar e amplificar os preconceitos existentes nesses dados (Barocas & Selbst, 2016). Isso pode levar a resultados injustos em áreas como contratação, crédito, e aplicação da lei.

Além disso, a IA apresenta desafios únicos de segurança. À medida que a IA se torna mais integrada em sistemas essenciais, como infraestruturas e redes de energia, o potencial de ataques cibernéticos maliciosos aumenta (Brundage et al., 2018). A IA também pode ser usada para criar *deepfakes* convincentes, o que pode ter implicações perturbadoras para a desinformação e a manipulação política.

Finalmente, a tomada de decisões por sistemas de IA pode ser enigmática, levantando questões de transparência e responsabilidade (Castelvecchi, 2016). Isso pode ser particularmente problemático quando a IA é usada para tomar decisões que afetam a vida das pessoas, como na medicina ou no direito.

Em suma, enquanto a IA oferece um enorme potencial, também apresenta sérios desafios. É crucial que se continue a discutir e abordar esses desafios à medida que a IA é integrada na vida das pessoas e nas sociedades.

2.2. Processamento de Linguagem Natural

O PLN é uma ferramenta poderosa para análise de sentimentos, pois pode processar grandes quantidades de dados de texto e identificar sentimentos positivos, negativos e neutros (Turney, 2002). Isso é alcançado através do uso de várias técnicas, como análise sintática (que analisa a estrutura gramatical do texto), análise semântica (que interpreta o significado do texto) e análise pragmática (que leva em consideração o contexto no qual o texto é usado) (Manning et al., 2014).

Um desafio na análise de sentimentos usando PLN é o entendimento do sarcasmo e da ironia (Tsur et al., 2010), que são formas comuns de expressão usando textos informais nas redes sociais e outras plataformas online, o que pode dificultar a identificação do sentimento. Isso é um problema porque o sarcasmo e a ironia podem inverter o significado pretendido de uma declaração, tornando-a o oposto do que parece à primeira vista. Além disso, a análise de sentimentos pode ser desafiada pela presença de erros gramaticais, gírias, emoticons e outros elementos comuns em textos informais online (Kouloumpis et al., 2011).

Apesar desses desafios, a análise de sentimentos com o uso de PLN é uma área de pesquisa ativa e tem um grande potencial para aplicações práticas. Por exemplo, pode ser usada para rastrear a opinião pública sobre questões políticas ou sociais, para análise de mercado em empresas, para desenvolvimento de produtos, entre outras possibilidades (Liu, 2012; Cambria et al., 2016).

Serão utilizadas neste trabalho as bibliotecas de processamento de linguagem natural Vader e TextBlob, ambas implementadas na linguagem Python, que realizarão a análise de sentimentos e serão apresentadas nas subseções seguintes.

2.2.1. Vader

O VADER (Valence Aware Dictionary and sEntiment Reasoner), em sua versão 3.3.2, é uma biblioteca de análise de sentimentos amplamente utilizada no campo do processamento de linguagem natural (NLP). Foi desenvolvida por Hutto e Gilbert em 2014 e é especialmente conhecida por sua capacidade de avaliar o sentimento em texto de maneira rápida e eficaz.

O VADER funciona atribuindo pontuações de sentimento a palavras e frases com base em um léxico incorporado que contém uma lista de palavras com pontuações de sentimento associadas. Cada palavra recebe uma pontuação de sentimento que pode ser positiva, negativa ou neutra. Além disso, o VADER considera intensificadores, negações e outras nuances linguísticas para calcular uma pontuação geral de sentimento para um pedaço de texto.

A pontuação de sentimento resultante é composta por quatro valores: positivo, negativo, neutro e uma pontuação composta que resume o sentimento geral do texto. Essas pontuações são úteis para determinar o sentimento predominante em um texto e podem ser valiosas em aplicações de análise de sentimentos, mineração de opiniões e muito mais (Hutto & Gilbert, 2014).

2.2.2. TextBlob

O TextBlob, em sua versão 0.17.1, é outra poderosa biblioteca de processamento de linguagem natural (PLN) utilizada em Python. O TextBlob se destaca por sua simplicidade e acessibilidade, tornando-o uma ferramenta excelente para iniciantes na área de PLN. Seu design é fortemente influenciado pelo NLTK e outro pacote chamado *Pattern* (Loria, 2020), resultando em uma interface elegante e fácil de usar para tarefas comuns de PLN).

A biblioteca TextBlob oferece uma API (*Application Interface*) simples para mergulhar em tarefas comuns de processamento de texto, tais como extração de substantivos, análise de sentimentos, tradução e mais (Loria, 2020). Além disso, o TextBlob também fornece ferramentas para criação e treinamento de modelos de aprendizado de máquina, facilitando a construção de sistemas de classificação de texto personalizados.

A análise de sentimentos é uma das principais características do TextBlob. Essa funcionalidade permite determinar a atitude ou o sentimento do escritor em relação a algum tópico ou a totalidade do texto subjacente. O TextBlob realiza essa tarefa atribuindo uma

polaridade e um valor de subjetividade a uma determinada parte do texto. A polaridade varia de -1 a 1, onde -1 indica um sentimento negativo e 1 indica um sentimento positivo. A subjetividade, por outro lado, varia de 0 a 1, com 0 sendo muito objetivo e 1 sendo muito subjetivo (Bose, Rajesh et al., 2020).

Apesar de todas as suas vantagens, o TextBlob não está livre de desafios. A detecção de sarcasmo, por exemplo, permanece um problema para a análise de sentimentos, independentemente da ferramenta utilizada. Além disso, embora o TextBlob seja poderoso e fácil de usar, ele pode não ser suficiente para tarefas de PLN mais complexas e personalizadas, que podem exigir o uso de ferramentas mais avançadas ou especializadas (Bose, Rajesh et al., 2020).

2.3. Análise de Sentimentos

A análise de sentimentos, também conhecida como mineração de opinião, é um campo de estudo que analisa as emoções das pessoas e as opiniões expressas em um texto (Liu, 2012), que se beneficia de uma disciplina chamada PLN, que é uma subárea da inteligência artificial que se concentra em como os computadores podem entender e manipular a linguagem humana (Jurafsky e Martin, 2008). A análise de sentimentos é uma das tarefas comuns em PLN, que também inclui tradução automática, reconhecimento de fala, geração de texto, dentre outros (Liddy, 2001).

2.3.1. Valoração

Como saída dos algoritmos que realizam a análise de sentimentos, obtém-se valores numéricos que indicam a classificação do texto de acordo com certos parâmetros. O Vader fornece uma pontuação de polaridade para o texto. Esta pontuação é um número entre -1 (negativo) e +1 (positivo), indicando a polaridade geral dos dados analisados. Essa pontuação de polaridade é decomposta em quatro componentes: *Positive*, *Negative*, *Neutral* e *Compound*, todas variando entre o mesmo intervalo numérico e que sinalizam, respectivamente, o quanto o texto analisado foi considerado positivo, negativo, neutro, e a pontuação geral agregada, que combina todas as anteriores.

No caso do TextBlob, obtém-se os indicadores *Polarity* e *Subjectivity*, correspondendo, respectivamente, à polaridade dos sentimentos contidos no texto analisado. A Polaridade reflete a inclinação dos sentimentos presentes no texto em análise, variando de -1, que indica um extremo negativo, a +1, que denota um extremo positivo. Já a Subjetividade vai de 0, que

representa um nível mínimo de objetividade, a 1, que aponta um elevado grau de objetividade. Em outras palavras, a polaridade informa sobre a carga emocional do texto, enquanto a subjetividade oferece uma medida da presença de elementos subjetivos ou objetivos na composição textual. Esses indicadores fornecem uma avaliação abrangente das características sentimentais e objetivas de um determinado texto, facilitando a compreensão das nuances presentes na expressão linguística.

2.4. Aprendizado de Máquina

O Aprendizado de Máquina (ou *Machine Learning*, ML) é um subcampo da ciência da computação que envolve um conjunto de algoritmos capazes de aprender e fazer previsões a partir de dados. Ele tem suas raízes na inteligência artificial (IA), e tem desempenhado um papel cada vez mais importante na era da informação devido à crescente disponibilidade e complexidade dos dados.

Segundo Alpaydin (2014):

> "O aprendizado de máquina deve ser um dos campos de mais rápido crescimento na ciência da computação. Não apenas os dados estão continuamente aumentando de tamanho, mas também a teoria para processá-los e transformá-los em conhecimento. Em diversos campos da ciência, desde astronomia até biologia, mas também na vida cotidiana, à medida que a tecnologia digital cada vez mais se infiltra em nossa existência diária e nossa pegada digital se aprofunda, mais dados são continuamente gerados e coletados. Sejam dados científicos ou pessoais, informações que ficam inativas e passivas não têm utilidade, e pessoas inteligentes têm encontrado cada vez mais novas maneiras de utilizar esses dados e transformá-los em produtos ou serviços úteis. Nessa transformação, o aprendizado de máquina desempenha um papel cada vez maior." (prefácio)

O conceito de aprendizado de máquina foi proposto pela primeira vez pelo cientista da computação e pioneiro da IA, Arthur Samuel, em 1959. Samuel descreveu o aprendizado de máquina como a "capacidade de aprender sem ser explicitamente programado" (Samuel, 1959). Em um de seus experimentos mais famosos, ele desenvolveu um programa de computador para

jogar damas que era capaz de aprender com seus próprios erros e melhorar seu desempenho ao longo do tempo.

A partir da década de 1980, a aprendizagem de máquina começou a florescer como um campo de pesquisa distinto da IA. Durante este período, os pesquisadores começaram a explorar novos tipos de algoritmos de aprendizado, tais como redes neurais, árvores de decisão e máquinas de vetores de suporte (Cortes & Vapnik, 1995). Estes algoritmos foram capazes de lidar com tipos de dados mais complexos e tarefas de aprendizagem mais difíceis do que os métodos de aprendizagem anteriores.

Na última década, o campo da aprendizagem de máquina tem experimentado um crescimento explosivo, impulsionado na maioria pelos avanços em hardware e pela disponibilidade de grandes volumes de dados. Os algoritmos de aprendizagem profunda, uma subclasse de aprendizagem de máquina que envolve a construção de redes neurais com muitas camadas, têm alcançado desempenho de ponta em uma variedade de tarefas, desde o reconhecimento de imagem (Krizhevsky et al., 2012) até a tradução automática de idiomas (Vaswani et al., 2017).

A aprendizagem de máquina é uma área da inteligência artificial que tem ganhado cada vez mais importância na sociedade moderna. Ela tem aplicações em diversos setores da economia, desde a saúde, onde é usada para prever doenças e personalizar tratamentos, até o setor financeiro, onde é usada para detectar fraudes e fazer previsões de mercado. Além disso, a aprendizagem de máquina é fundamental para muitos dos serviços que usamos todos os dias, como motores de busca, sistemas de recomendação e assistentes de voz. É notável que a aprendizagem de máquina tem um papel crucial na melhoria da qualidade de vida das pessoas e no desenvolvimento econômico.

Em resumo, o aprendizado de máquina é um campo empolgante e em rápido crescimento que está transformando a maneira como as pessoas resolvem os problemas cotidianos. Com o contínuo avanço da tecnologia e a disponibilidade cada vez maior de dados, espera-se que o impacto da aprendizagem de máquina na sociedade continue a crescer no futuro.

Neste trabalho, será usada a biblioteca de aprendizado de máquina *scikit-learn* (também conhecida como *sklearn*), implementada na linguagem *Python*, para treinar modelos para posteriormente realizar a análise de sentimentos com o uso dos algoritmos de aprendizado de máquina supervisionados que serão apresentados nas subseções seguintes.

2.4.1. Regressão Logística

A regressão logística é um algoritmo de Aprendizagem supervisionada, que é um tipo de modelo linear utilizado para classificação binária, tendo como objetivo classificar objetos de uma classe a partir de valores. (Grossi et al., 2013; Oliveira, 2016).

Desta forma, a análise de regressão tem como atribuição não somente detectar quais dados independentes manipulam o resultado do dado dependente, mas também como estas o fazem (Armstrong, 2012). Em aprendizado de máquina, a regressão logística é frequentemente usada como um algoritmo de base devido à sua simplicidade e eficiência. Embora possa não ser capaz de capturar relacionamentos complexos não lineares entre as características e o resultado como alguns outros algoritmos de aprendizado de máquina, ela ainda pode ser muito eficaz, especialmente quando o espaço de características é de alta dimensão.

No contexto do *scikit-learn*, uma biblioteca popular de aprendizado de máquina em Python, a regressão logística é implementada pela classe "*LogisticRegression*"". Ela utiliza o conceito de função logística (também conhecida como função sigmóide) para modelar a probabilidade de uma instância pertencer a uma classe específica. A função sigmóide mapeia qualquer valor real em um intervalo entre 0 e 1, o que a torna adequada para estimar probabilidades, uma vez que essa delimitação garante que o valor estimado pelo modelo permaneça no intervalo, permitindo a interpretação do valor como figura probabilística (Oliveira, 2016).

2.4.2. MLP

O *Multilayer Perceptron* (MLP) é um algoritmo supervisionado cuja arquitetura está baseada numa rede neural artificial que consiste em múltiplas camadas de unidades de processamento, incluindo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada unidade em uma camada está conectada a todas as unidades na camada seguinte por meio de conexões ponderadas. Essa arquitetura permite que o MLP aprenda representações complexas de dados e realize tarefas de classificação ou regressão. (Bento & Carolina, 2021)

O treinamento de um MLP envolve alimentar os dados de entrada através da rede, calculando as saídas das unidades em cada camada e ajustando os pesos das conexões para minimizar o erro entre as saídas da rede e os valores desejados. Isso geralmente é feito usando algoritmos de otimização, como o gradiente descendente, para ajustar os pesos de forma iterativa.

Na biblioteca *sklearn*, a implementação do MLP é fornecida pela classe "*MLPClassifier*" para tarefas de classificação e "*MLPRegressor*" para tarefas de regressão.

Essas classes permitem que se ajuste e treine um modelo personalizado com base em dados e requisitos específicos.

2.4.3. KNN

O *K-Nearest Neighbors* (KNN) é um algoritmo de aprendizado de máquina supervisionado que pertence à categoria de classificação e regressão baseada em instâncias. Ele é usado para fazer previsões com base na similaridade entre os dados de entrada e os dados de treinamento. O KNN atribui uma classe ou valor a uma nova instância com base nas classes ou valores das K instâncias mais próximas no conjunto de treinamento (Hastie, Tibshirani & Friedman, 2016).

O funcionamento básico do KNN envolve a seguinte lógica: dado um novo ponto de entrada, o algoritmo calcula a distância entre esse ponto e todos os pontos de treinamento. Ele então seleciona os K pontos de treinamento mais próximos e toma uma decisão com base nas classes ou valores desses vizinhos, como a classe mais comum ou a média dos valores.

O valor de K é um hiperparâmetro que afeta a sensibilidade do modelo a pontos atípicos e ruído. Um valor pequeno de K pode levar a uma classificação mais sensível a variações locais nos dados, enquanto um valor grande pode suavizar a decisão, considerando mais vizinhos.

No contexto da biblioteca *sklearn* do Python, o KNN é implementado pela classe "*KNeighborsClassifier*" para tarefas de classificação e pela classe *"KNeighborsRegressor*" para tarefas de regressão.

2.4.4. Scikit-learn

O *scikit-learn* é uma biblioteca de aprendizado de máquina em Python amplamente utilizada, que oferece uma variedade de ferramentas para análise de dados e modelagem preditiva. Desenvolvido sobre o *NumPy*, *SciPy* e *Matplotlib*, o *scikit-learn* fornece uma implementação eficiente de algoritmos de aprendizado supervisionado e não supervisionado (Pedregosa et al., 2011). Dentre as suas funcionalidades, destacam-se algoritmos para classificação, regressão, *clustering*, redução de dimensionalidade e pré-processamento de dados.

A biblioteca é conhecida por sua abordagem consistente e fácil de usar, fornecendo uma interface consistente para diversos algoritmos, facilitando assim o processo de experimentação e implementação de modelos de *machine learning*. Além disso, o *scikit-learn* inclui ferramentas para avaliação de modelos, seleção de modelos e pré-processamento de dados. Neste trabalho será utilizada para todo o processamento e

preparação dos dados e algoritmos de aprendizado de máquina supervisionados, bem como para obter as métricas utilizadas nas comparações.

2.5. *Twitter*

O *Twitter* (agora chamado de X) é uma das principais redes sociais do mundo, permitindo que os usuários compartilhem mensagens curtas, conhecidas como "*tweets*", sendo lançado oficialmente em julho de 2006 (Govan, 2021). Inicialmente a plataforma tinha um limite de 140 caracteres por tweet, esse limite foi posteriormente aumentado para 280 caracteres, ainda que esse aumento tenha tido pouco impacto no tamanho dos textos publicados pelos usuários na rede, que parecem ter se acostumado a utilizar a rede para se comunicar de forma breve e direta (Perez, 2018).

Em poucos anos a plataforma demonstrou ser mais do que uma simples rede social, tendo um impacto significativo na opinião pública e na forma como as notícias são disseminadas (Kwak et al., 2010), desempenhando um papel importante em eventos políticos, como as Eleições Presidenciais dos EUA (Bode et al., 2013) e a Primavera Árabe (Lotan et al., 2011). Desta forma, o *twitter* se mostra como um ambiente fértil para o emprego de ferramentas de análise de sentimentos, que ajudam a compreender o sentimento e opiniões públicas em relação a eventos, produtos ou políticas (Pak & Paroubek, 2010).

O *sentiment140* é um conjunto de dados (*dataset*) composto por 1.60,000 *tweets* classificados como neutros, positivos e negativos, coletados originalmente em 2009. Esse conjunto de dados tornou-se uma referência fundamental na comunidade de processamento de linguagem natural e análise de sentimentos, e tem sido amplamente utilizado como um recurso de treinamento e avaliação para algoritmos de aprendizado de máquina, contribuindo para o desenvolvimento de abordagens mais eficazes na compreensão das emoções e opiniões expressas em plataformas de mídia social. Embora o *sentiment140* tenha sido descontinuado, seu impacto perdura, continuando a influenciar a pesquisa e a inovação na área.

3. Metodologia

3.1. Finalidade

O estudo tem por finalidade comparar a acurácia e performance das técnicas de Processamento de Linguagem Natural e Aprendizado de Máquina na classificação dos sentimentos contidos em postagens da rede social *Twitter*. Para realizar o estudo, serão

utilizadas as bibliotecas *Vader* e *TextBlob* para aplicar as técnicas tradicionais de PLN, e também os algoritmos de Aprendizado de Máquina Regressão Logística, KNN e MLP, implementados com a linguagem de programação *Python*. Os resultados do estudo serão analisados e discutidos, visando identificar as vantagens e desvantagens de cada técnica, bem como as possibilidades de aplicação em cenários reais. O estudo também visa contribuir para o desenvolvimento de novas técnicas de análise de sentimentos.

3.2. Abordagem

O trabalho contará com uma análise quali-quanti das informações. A análise qualitativa será feita através da classificação dos sentimentos entre positivo e negativo, enquanto a quantitativa se dará através das pontuações obtidas com o uso da biblioteca *sklean*, sendo a principal delas a acurácia. A acurácia mede a proporção de exemplos classificados corretamente em relação ao total de exemplos no conjunto de dados de teste, quando este existe, ou, no caso das bibliotecas de PLN, a acurácia mede a proporção das classificações corretas em relação à classificação original de todo o conjunto de dados.

3.3. Método

O método aplicado no trabalho é o hipotético-dedutivo, se pautando na hipótese de que os algoritmos PLN alcançam uma acurácia menor do que os algoritmos de Aprendizado de Máquina, e sendo experimental no contexto da implementação dos algoritmos de PLN e ML para a análise dos textos obtidos do *dataset*.

3.4. Procedimento

Para o desenvolvimento do trabalho foi realizado um levantamento bibliográfico com teses, artigos e livros sobre o impacto que as mídias sociais e a inteligência artificial exercem na sociedade. Também foram levantados estudos que demonstram as diferentes técnicas utilizadas na automação da análise de sentimentos utilizando diferentes algoritmos, onde foi levantada a hipótese. Os dados necessários para realizar a parte experimental do trabalho foram obtidos do conjunto de dados *sentiment140*, um *dataset* contendo mais de 1 milhão de *tweets* com sentimentos previamente classificados. A metodologia desenvolvida foi estabelecida a partir dos seguintes passos:

1. O *dataset* original utilizado neste trabalho foi adquirido no website *kaggle* (https://www.kaggle.com/datasets/kazanova/sentiment140), com extensão .csv, com tamanho de 233,207 KB, 6 colunas e 1.600.000 linhas.

2. Usando a biblioteca *Pandas*, foram extraídos do *dataset* 10 mil registros correspondentes a *tweets* com sentimentos positivos, e 10 mil *tweets* com sentimentos negativos.

3. Também com o *Pandas*, foram extraídas apenas as colunas necessárias, sendo mantidas as colunas referentes ao texto do *tweet* e os seus sentimentos predominantes.

4. Foram removidos *links*, caracteres não-alfanuméricos, e menções a usuários (feitas com @ no *twitter*) utilizando *Regex* (*regular expression*) .

5. Os textos foram pré-processados com a biblioteca NLTK (*Natural Language Toolkit*), fazendo a tokenização e removendo as *stopwords*, que são palavras que não são úteis para a análise de sentimentos.

6. Foi usada a função *train_test_split* de ML para particionar o conjunto de dados entre teste e treino, sendo, respectivamente, 30 e 70% do total de dados.

7. Também para os algoritmos de *Machine Learning*, foi utilizada a técnica de *Grid SearchCV*, que otimiza o conjunto dos parâmetros utilizados pelos modelos, de modo a maximizar o seu desempenho.

8. A análise dos sentimentos foi feita com cada um dos algoritmos, tendo seus resultados armazenados em novas colunas no *dataset*.

9. Foi utilizado o conjunto de funções e classes de avaliação *metrics* da biblioteca *sklean*, para obter as métricas desejadas, por exemplo, a acurácia dos algoritmos.

10. Foram utilizadas as bibliotecas *matplotlib* e *seaborn* para exibir os resultados das análises e desempenho dos algoritmos em forma de gráficos, de modo a facilitar a comparação.

3.5. Arquitetura e Ambiente de Desenvolvimento

Para realizar o desenvolvimento do algoritmo e processamento dos dados foi utilizado um computador com um processador core i5 9400f de 64 *bits* com 6 núcleos e 6 *threads*, 24 GB de memória RAM (*Random Access Memory*) e um SSD (*Solid State Drive*) de 480 GB modelo A400 da marca *Kingston*. O sistema operacional utilizado foi o *Windows* 10, e a versão do *Python* foi a 3.9. A IDE (*Integrated Development Environment*) escolhida foi a *Visual Studio Code* na sua versão 1.73.1. Todas as bibliotecas utilizadas estão em sua última versão na data atual (Outubro de 2023).

4. Aplicação dos Algoritmos e Apresentação dos Resultados

A etapa experimental do estudo foi realizada através da aplicação dos algoritmos de PLN e *Machine Learning* no ambiente descrito na subseção anterior. A seguir, será mostrado o trecho do código responsável por importar as bibliotecas e coleções no projeto, divididas entre as bibliotecas padrão, e bibliotecas de terceiros.

Figura 1 – Importação das bibliotecas e coleções a serem utilizadas no projeto.

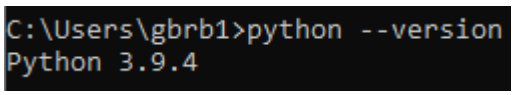
```
# Bibliotecas Padrão
import os
import re
import numpy as np
import time

# Bibliotecas de Terceiros
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
```

Tabela 2 – Descrição das bibliotecas importadas na Figura 1.

Bibliotecas	
os	Linha 2. Esta biblioteca fornece funções para interagir com o sistema operacional, como manipular caminhos de arquivo, criar pastas, listar diretórios, etc.
re	Linha 3. Permite trabalhar com expressões regulares, sendo útil para buscar e manipular padrões de texto em *strings*.
numpy	Linha 4. Biblioteca necessária para a manipulação de vetores em operações matemáticas, lógicas, manipulativas, organizativas e seletivas.
time	Linha 5. O módulo *time* em *Python* é usado para implementar diversas funcionalidades relacionadas ao tempo. Neste trabalho, será usado como temporizador para contar o tempo decorrido durante o processamento de cada algoritmo. Também será utilizado para que a hora em que o programa é executado seja utilizada como parâmetro de aleatoriedade em algumas funções.
matplotlib	Linha 8. *Matplotlib* é uma biblioteca de visualização de dados em *Python*, oferecendo diversas ferramentas para criação de gráficos e visualização de dados.
seaborn	Linha 9. *Seaborn* é uma biblioteca de visualização de dados baseada no *Matplotlib*. Ela fornece uma interface de alto nível para criar gráficos estatísticos atraentes.
pandas	Linha 10. Biblioteca usada para manipulação e análise de dados em *Python*, fornecendo estruturas de dados como o *DataFrame* para armazenar e trabalhar com dados tabulares.
	Linha 11. Biblioteca para processamento de linguagem natural em *Python*. Ela fornece ferramentas para trabalhar

nltk	com texto, como tokenização, análise de sentimento e *stopwords*.
textblob	Linha 15. Outra biblioteca de PLN que será usada no trabalho para realizar a análise de sentimentos.
sklearn	Linha 16. Biblioteca *scikit-learn*, que utiliza inteligência artificial e permite implementar os algoritmos de RL, MLP, KNN, *Grid Search*, métricas, dentre outros.

Figura 2 – Verificação da versão do *Python* no sistema operacional *Windows* 10.

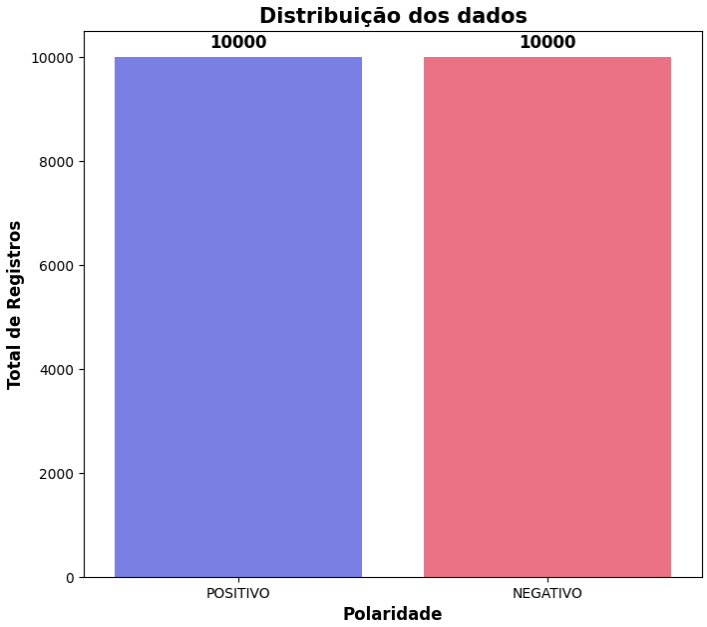


Para reduzir o tamanho *dataset sentiment140* de modo a diminuir o tempo de processamento, foi realizada a partição do conjunto original em 10 mil registros com *tweets* com sentimentos classificados como positivos, e 10 mil com sentimentos classificados como negativos. Também foram utilizadas apenas duas colunas: *target* – que é um valor numérico correspondente ao sentimento predominante, sendo 0 para sentimentos negativos e 4 para sentimentos positivos –, e *text*, correspondente ao texto do *tweet*.

Figura 3 – Algoritmo particionando o *dataset* original.

```python
import pandas as pd
import os

# Definindo as colunas e a codificação
DATASET_COLUMNS = ["target", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"

# Lendo o arquivo CSV
dataset_filename = os.listdir("input")[0]
dataset_path = os.path.join("input", dataset_filename)
print("Abrindo arquivo:", dataset_path)
df = pd.read_csv(dataset_path, encoding=DATASET_ENCODING)

# Pegando as linhas de 0 a 9999 e de 805000 a 814999
parte_1 = df.iloc[0:10000]
parte_2 = df.iloc[805000:815000]

# Concatenando as duas partes
df_final = pd.concat([parte_1, parte_2])

# Obtendo apenas as colunas necessárias
df_final = df_final[["target", "text"]]

# Escrevendo o resultado em um novo arquivo CSV
df_final.to_csv('dataset_particionado.csv', index=False)
```

Figura 4 – Distribuição dos dados após o *Dataset* original ser particionado.



Após abrir o arquivo e usar o pacote *pandas* para criar um *DataFrame* de forma similar à Figura 3, a coluna *"target"* foi mapeada, de modo a transformar os seus rótulos numéricos em "positivo" e "negativo".

Figura 5 – Mapeando a coluna *"target"* do *dataset*.

```
 98  # Mapeando a coluna label do dataset
 99  decode_map = {0: "NEGATIVO", 4: "POSITIVO"}
100
101  # Definindo uma função chamada "decode_sentiment" que
102  # recebe um rótulo como entrada e retorna um rótulo legível
103
104
105  def decode_sentiment(label):
106      return decode_map[int(label)]
107
108  # Aplica a função "decode_sentiment" a cada elemento da coluna
109  # "target" do DataFrame "df".
110  # Isso é feito usando a função "apply" do pandas com uma função
111  # lambda que passa cada valor da coluna "target" para "decode_sentiment".
112  df.target = df.target.apply(lambda x: decode_sentiment(x))
```

Uma vez particionado e mapeado, o conjunto de dados final está pronto para ser novamente pré-processado. O trecho de código a seguir faz o *download* dos pacotes necessários para que o NLTK faça a sua parte no pré-processamento dos dados.

Figura 6 – Baixando os pacotes para que o NLTK faça o pré-processamento dos dados.

```
27  # Baixando pacotes do NLTK
28  nltk.download('punkt')
29  nltk.download('stopwords')
30  nltk.download('vader_lexicon')
```

Essas linhas são executadas apenas uma vez durante o ciclo de vida do programa. A seguir, uma tabela explicando a funcionalidade de cada pacote que foi baixado no trecho de código anterior.

Tabela 3 – Descrição dos pacotes baixados com o NLTK.

punkt	Linha 28. O pacote punkt do NLTK é um "tokenizador" de palavras e sentenças pré-treinado, que divide o texto em palavras e sentenças individuais, segmentando o texto e facilitando a sua manipulação .
stopwords	Linha 29. Este pacote será utilizado para remover as palavras classificadas como stopwords do texto a ser analisado por todos os algoritmos. Stopwords são palavras que são comuns em uma linguagem porém não influenciam significativamente na análise dos sentimentos.
vader_lexicon	Linha 30. Este trecho está fazendo download do léxico Vader, para que a ferramenta possa realizar a análise de sentimentos.

Após ser feito o *download* dos pacotes necessários, foi realizada a definição da lista de *stopwords*, para posteriormente serem utilizadas na função responsável pelo processamento do texto. O idioma selecionado foi a língua inglesa, uma vez que os dados do *dataset* estão predominantemente em inglês. (Ver figura 7)

Figura 7 – Definindo a lista de *stopwords*.

```
32  # Definindo lista de stopwords
33  stop_words = set(stopwords.words('english'))
```

Uma vez definida, a lista está pronta para ser usada na função responsável pelo processamento do texto. Dentro da função, foi feito o uso de expressões regulares para limpar o texto, removendo *links*, menções a outros usuários e caracteres não-alfanuméricos. Em seguida, foi realizada a "tokenização" do texto, fazendo a divisão do mesmo em unidades menores, a fim de facilitar a interação na lista de *tokens* para remoção das *stopwords*.

Figura 8 – Função responsável por fazer o pré-processamento do texto.

```
39  # Função para limpar e processar o texto
40  def process_text(text):
41      # Limpa o texto, removendo menções a usuários,
42      # URLs e caracteres não-alfanuméricos.
43      text = re.sub(r'@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+', ' ', text)
44
45      # Tokeniza o texto em palavras
46      tokenized = word_tokenize(text)
47
48      # Retorna uma lista de palavras, removendo aquelas que
49      # estão na lista de palavras de parada (stopwords).
50      return [word for word in tokenized if word.casefold() not in stop_words]
```

Uma vez que o texto foi processado, foram feitas as análises com os algoritmos de PLN.

Figura 9 – Instanciando o analisador de sentimentos *Vader* e implementando função que realiza a análise.

```
53  # Instanciando o analisador de sentimentos do Vader
54  sia = SentimentIntensityAnalyzer()
55
56  # Classificando os sentimentos usando Vader
57  def nltk_sentiment(text):
58      text = process_text(text)
59      text = ' '.join(text)
60      sentiment = sia.polarity_scores(text)
61      if sentiment['compound'] > 0 and sentiment['pos'] > 0:
62          return "POSITIVO"
63      else:
64          return "NEGATIVO"
```

Para classificar os sentimentos entre positivo e negativo com o *Vader*, foram considerados como positivos os textos que obtiveram as pontuações de *compound* e *positive* maiores que zero.

Figura 10 – Aplicando a função de análise de sentimentos do *Vader* no *DataFrame* e contabilizando o tempo decorrido.

```
121 # Aplicando a função de análise de sentimentos com o Vader
122 # no DataFrame
123 start_time = time.time()
124 df['nltk_pred'] = df['text'].apply(nltk_sentiment)
125 end_time = time.time()
126 process_times["Vader"] = end_time - start_time
```

Na figura 10, a variável *start time* foi atribuída com a hora no início da execução da análise do *Vader*, para no final ser utilizada na computação do tempo decorrido durante o processamento e utilizada posteriormente para exibir o gráfico de tempo de processamento. Também foi criada uma nova coluna no *DataFrame* chamada *nltk_pred*, que recebe o retorno da função *ntlk_sentiment*, passando o conteúdo da coluna *text* do *DataFrame* como parâmetro. Esta atribuição é feita com o método *apply* da biblioteca *pandas*, que permite aplicar a função a cada elemento da série.

Figura 11 – Implementação da função que realiza a análise de sentimentos com o *TextBlob*.

```
67  # Função para classificar os sentimentos
68  # usando o TextBlob
69  def textblob_sentiment(text):
70      text = process_text(text)
71      text = ' '.join(text)
72      sentiment = TextBlob(text).sentiment.polarity
73      if sentiment > 0:
74          return "POSITIVO"
75      else:
76          return "NEGATIVO"
```

Para classificar os sentimentos entre positivo e negativo com o *TextBlob*, foram considerados como positivos os textos que obtiveram a pontuação de *sentiment* maior que zero.

Figura 12 – Realizando o mesmo processo demonstrado na Figura 10, porém desta vez com o TextBlob.

```
128 # Aplicando a função de análise de sentimentos com o TextBlob
129 # no DataFrame
130 start_time = time.time()
131 df['textblob_pred'] = df['text'].apply(textblob_sentiment)
132 end_time = time.time()
133 process_times["TextBlob"] = end_time - start_time
```

A acurácia das predições do *Vader* e do *TextBlob* são obtidas com o método *accuracy_score* da biblioteca *sklearn*. O método recebe como parâmetro a coluna *target* do *DataFrame* e também a coluna criada para armazenar as predições dos algoritmos. O cálculo da acurácia é obtido pela divisão do número de previsões corretas pelo número total de previsões, sendo o seu resultado um número entre 0 e 1. A acurácia é uma métrica útil quando as classes estão balanceadas (ou seja, a quantidade de exemplos em cada classe é aproximadamente a mesma).

Figura 13 – Trecho de código responsável por obter e imprimir no terminal a acurácia das predições do *Vader* e *TextBlob*.

```
135 # Obtendo e imprimindo a acurácia do Vader e TextBlob
136 print('Acurácia do Vader:',
137       accuracy_score(df['target'], df['nltk_pred']))
138 print('Acurácia do TextBlob:',
139       accuracy_score(df['target'], df['textblob_pred']))
```

Figura 14 – Saída no terminal com as acurácias.

```
Acurácia do Vader: 0.64245
Acurácia do TextBlob: 0.6255
```

Para realizar o treinamento dos algoritmos de *Machine Learning*, os dados foram particionados entre um conjunto de testes e um conjunto de treinos. O conjunto de treinos – 70% do total dos dados – foi utilizado para que os algoritmos "aprendessem" a realizar as análises de acordo com o texto do *tweet* e o seu sentimento pré-classificado, enquanto o conjunto de testes foi utilizado para que as análises de sentimentos fossem de fato realizadas.

Figura 15 – Declaração das constantes responsáveis por definir as proporções do conjunto de teste e treino.

```
112 # Definindo a proporção do conjunto de teste em relação
113 # ao conjunto de dados completo como 0.3 (30%).
114 TEST_SIZE = 0.3
115
116 # Definindo o tamanho do conjunto de treinamento como o
117 # complemento do tamanho do conjunto de teste (70%).
118 TRAIN_SIZE = 1 - TEST_SIZE
```

Figura 16 – Representação gráfica das proporções dos conjuntos de treino e testes e do número total de registros em cada um.

Foi criada uma coluna no *DataFrame* chamada *processed_text*, essa coluna recebeu o retorno da função *process_text* – que é uma lista de *tokens* – passando como parâmetro os valores da coluna *text* por meio da função *apply*, e posteriormente reunificando as palavras que compõem cada frase.

Figura 17 – Aplicando a função de processamento de texto e inserindo o seu retorno na coluna *processed_text*.

```
146 # Processando os textos e transformando-os em listas de strings
147 # que serão armazenadas na coluna processed_text
148 df['processed_text'] = df['text'].apply(lambda x: ' '
149                                          .join(process_text(x)))
```

Após a coluna *processed_text* ter os seus valores atribuídos, foi iniciado o processo de vetorização. A vetorização consiste em converter o texto em dados numéricos que podem ser usados como entrada para algoritmos de *Machine Learning*.

Figura 18 – Instanciando o vetorizador e vetorizando o texto da coluna *processed_text*.

```
152 # Instanciando o vetorizador
153 vectorizer = TfidfVectorizer()
154
155 # Vetorizando o texto
156 X = vectorizer.fit_transform(df['processed_text'])
157 y = df['target']
```

A função *fit_transform* ajusta o modelo e transforma os dados de entrada – neste caso, os textos contidos na coluna *processed_text* do *DataFrame* –. A etapa de transformação aplica os ajustes aprendidos na etapa anterior. No caso do *TfidfVectorizer*, a etapa de transformação converte os textos em representações vetoriais usando o vocabulário e as frequências de termos aprendidos durante o ajuste. "X" recebe uma matriz com os dados transformados, enquanto "Y" recebe os respectivos rótulos de sentimentos (POSITIVO e NEGATIVO).

Em seguida, foi feita a divisão do conjunto de dados em um subconjunto de treinos e um subconjunto de testes, utilizando a função *train_test_split*.

Figura 19 – Utilização do método *train_test_split*.

```
159 # Utilizando o temporizador para obter um número inteiro
160 # para ser utilizado como seed no parametro random_state
161 current_time = int(time.time())
162
163 # Dividindo os dados em treinamento e teste
164 X_train, X_test, y_train, y_test = train_test_split(
165     X, y, test_size=TEST_SIZE, random_state=current_time)
```

O código da figura 19 cria uma divisão estratificada dos dados em conjuntos de treinamento e teste, usando o valor do temporizador atual como uma semente pseudo aleatória para garantir que a divisão seja um pouco diferente a cada execução do programa. *X_train* é utilizado para armazenar as amostras de características que os modelos usam para aprender, ou seja, é o conjunto fornecido ao algoritmo de treinamento para que estes ajustes os parâmetros com base nestes dados. *Y_train* é o conjunto de rótulos de classificação (POSITIVO e NEGATIVO) correspondentes às amostras contidas em *X_train*. *X_test* contém as amostras de características que não foram utilizadas no treinamento, para posteriormente serem utilizadas para testar a acurácia dos algoritmos de ML. Assim como com os conjuntos de treino, *Y_test*

são os rótulos correspondente às amostras contidas em *X_test*. A proporção do conjunto de testes é determinada pela constante *TEST_SIZE*, definida neste projeto como 30% do conjunto total (Figura 14).

Com os dados organizados, as análises com os algoritmos de *Machine Learning* podem ser realizadas. O *Grid SearchCV* é uma técnica usada para encontrar os melhores hiperparâmetros para um modelo. Os hiperparâmetros são configurações específicas do modelo que não são aprendidas durante o treinamento, e que geralmente não são ajustadas de forma automática pelos modelos. Na prática, o que o *Grid SearchCV* faz é encontrar os melhores parâmetros a serem escolhidos em uma lista – aqui chamada de grade de parâmetros – para as funções dos modelos quando estes são instanciados, sem que haja necessidade de uma edição manual destes parâmetros de forma especulativa. A utilização desta técnica tende a melhorar a acurácia dos modelos quando bem aplicada, porém, aumenta consideravelmente o tempo de processamento.

Figura 20 – Definindo a grade de parâmetros para o algoritmo de Regressão Logística.

```
163  # Definindo a grade de parâmetros para a regressão logística
164  param_grid = {
165      'C': [1, 10, 20, 30, 40, 50, 100, 150],
166      'penalty': ['l2'],
167      'solver': ['lbfgs'],
168      'max_iter': [250, 500, 1000]
169  }
```

Figura 21 – Instanciando o objeto da Regressão Logística e iniciando o *GridSearchCV*.

```
175  # Criando uma instância do modelo de regressão logística
176  start_time = time.time()
177  log_reg = LogisticRegression()
178
179  # Criando o objeto GridSearchCV com o modelo, os parâmetros e a validação cruzada
180  grid_search = GridSearchCV(
181      estimator=log_reg, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

O parâmetro *cv=5* é referente ao número de dobras (*folds*) na validação cruzada. Significa que o *Grid SearchCV* utilizará validação cruzada com 5 dobras para avaliar o desempenho do modelo com cada conjunto de hiperparâmetros. Isso ajuda a estimar o desempenho do modelo de forma mais robusta. *n_jobs* diz respeito à quantidade de núcleos do

processador que serão utilizados, o valor de -1 significa que todos os núcleos disponíveis serão usados.

Figura 22 – Aplicando o *Grid Search* com os subconjuntos de treinamento.

```
183  # Executando a pesquisa em grade nos dados de treinamento
184  grid_search.fit(X_train, y_train)
```

Figura 23 – Obtendo os melhores parâmetros com o *Grid Search* e os utilizando para realizar as predições com o algoritmo de Regressão Logística.

```
186  # Obtendo os melhores hiperparâmetros encontrados
187  best_params = grid_search.best_params_
188
189  # Usando o modelo com os melhores hiperparâmetros para fazer previsões
190  log_reg_best = grid_search.best_estimator_
191  log_reg_preds = log_reg_best.predict(X_test)
192  df['log_reg_pred'] = log_reg_best.predict(X)
```

Figura 24 – Grade de parâmetros a ser utilizada no algoritmo MLP.

```
196  param_grid = {
197      'hidden_layer_sizes': [(5, 5), (10, 10), (25, 25)],
198      'max_iter': [50, 100, 150],
199      'alpha': [0.001],
200      'learning_rate_init': [0.01],
201      'solver': ['lbfgs'],
202      'activation': ['relu'],
203      'random_state': [current_time],
204  }
```

Figura 25 – Instanciando o MLP e iniciando o *Grid Search*.

```
205  # Criando uma instância do modelo MLP
206  mlp = MLPClassifier()
207
208  # Criando um objeto GridSearchCV
209  grid_search = GridSearchCV(
210      estimator=mlp, param_grid=param_grid, cv=5, n_jobs=-1)
```

Figura 26 – Aplicando o *Grid Search* com os subconjuntos de treinamento.

```
212  # Realizando a busca em grade no conjunto de treinamento
213  grid_search.fit(X_train, y_train)
```

Figura 27 – Obtendo os melhores parâmetros e modelo com o *Grid Search*.

```
215  # Obtendo os melhores parâmetros encontrados pelo grid search
216  best_params = grid_search.best_params_
217
218  # Obtendo o melhor modelo treinado
219  best_mlp_model = grid_search.best_estimator_
```

Figura 28 – Fazendo as predições com o MLP utilizando o melhor modelo e salvando no *DataFrame*.

```
221  # Fazendo previsões com o melhor modelo
222  mlp_preds = best_mlp_model.predict(X_test)
223  df['mlp_pred'] = best_mlp_model.predict(X)
```

Figura 29 – Grade de parâmetros a ser utilizada no algoritmo KNN.

```
230  param_grid = {
231      'n_neighbors': [100, 151, 200],
232      'weights': ['uniform', 'distance'],
233      'algorithm': ['auto', 'brute']
234  }
```

Figura 30 – Instanciando o KNN e iniciando o *Grid Search*.

```
236  # Criando uma instância do KNeighborsClassifier
237  knn = KNeighborsClassifier()
238
239  # Criando uma instância do GridSearchCV
240  grid_search = GridSearchCV(
241      estimator=knn, param_grid=param_grid, cv=5, n_jobs=-1)
```

Figura 31 – Aplicando o *Grid SearchCV* com os subconjuntos de treinamento.

```
243  # Executando a pesquisa em grade nos dados de treinamento
244  grid_search.fit(X_train, y_train)
```

Figura 32 – Obtendo os melhores parâmetros e modelo com o *Grid SearchCV* e os utilizando para realizar as predições com o KNN.

```
246  # Obtendo o melhor modelo treinado com os melhores hiperparâmetros
247  best_knn_model = grid_search.best_estimator_
248
249  # Obtendo os melhores hiperparâmetros encontrados pela pesquisa em grade
250  best_params = grid_search.best_params_
251
252  # Fazer previsões usando o melhor modelo
253  knn_preds = best_knn_model.predict(X_test)
254  df['knn_pred'] = best_knn_model.predict(X)
```

Após realizadas as análises com os algoritmos de *Machine Learning*, foi exibido o *Classification Report*. Implementado pela função *classification_report*, o relatório fornece uma visão detalhada do desempenho dos modelos, sendo exibido de forma tabular.

Tabela 4 – Métricas do relatório de classificação.

Métricas separadas por classe	
Precision	É a proporção de verdadeiros positivos (instâncias corretamente classificadas como pertencentes a determinada classe) em relação ao total de instâncias classificadas (verdadeiros positivos + falsos positivos).
Recall	É a proporção de verdadeiros positivos em relação ao total de instâncias reais pertencentes à classe (verdadeiros positivos + falsos negativos).
F1-Score	É a média harmônica entre a precisão e o recall. É calculado como: 2 * (precisão * recall) / (precisão + recall).
Support	Contagem total dos elementos pertencentes à respectiva classe.
Outras métricas	
Accuracy	Proporção de todas as previsões corretas em relação ao total de previsões.
Macro avg	É a média das métricas (precision, recall, F1-Score e support).
Weighted avg	Média ponderada das métricas. Como as classes estão balanceadas, apresentará os mesmos valores de Macro avg.

Figura 33 – Imprimindo os relatórios de classificação dos algoritmos de ML no terminal.

```
266  print('Relatório de classificação da regressão logística:\n',
267        classification_report(y_test, log_reg_preds, digits=4))
268
269  print('Relatório de classificação do MLP:\n',
270        classification_report(y_test, mlp_preds, digits=4))
271
272  print('Relatório de classificação do KNN:\n',
273        classification_report(y_test, knn_preds, digits=4))
```

Figura 34 – Relatório de classificação do algoritmo Regressão Logística.

```
Relatório de classificação da regressão logística:
              precision    recall  f1-score   support

    NEGATIVO     0.7505    0.7191    0.7345      3008
    POSITIVO     0.7290    0.7597    0.7440      2992

    accuracy                         0.7393      6000
   macro avg     0.7398    0.7394    0.7392      6000
weighted avg     0.7398    0.7393    0.7392      6000
```

Figura 35 – Relatório de classificação do algoritmo MLP.

```
Relatório de classificação do MLP:
              precision    recall  f1-score   support

    NEGATIVO     0.7456    0.7045    0.7244      3008
    POSITIVO     0.7185    0.7584    0.7379      2992

    accuracy                         0.7313      6000
   macro avg     0.7320    0.7314    0.7312      6000
weighted avg     0.7321    0.7313    0.7311      6000
```

Figura 36 – Relatório de classificação do algoritmo KNN.

Relatório de classificação do KNN:

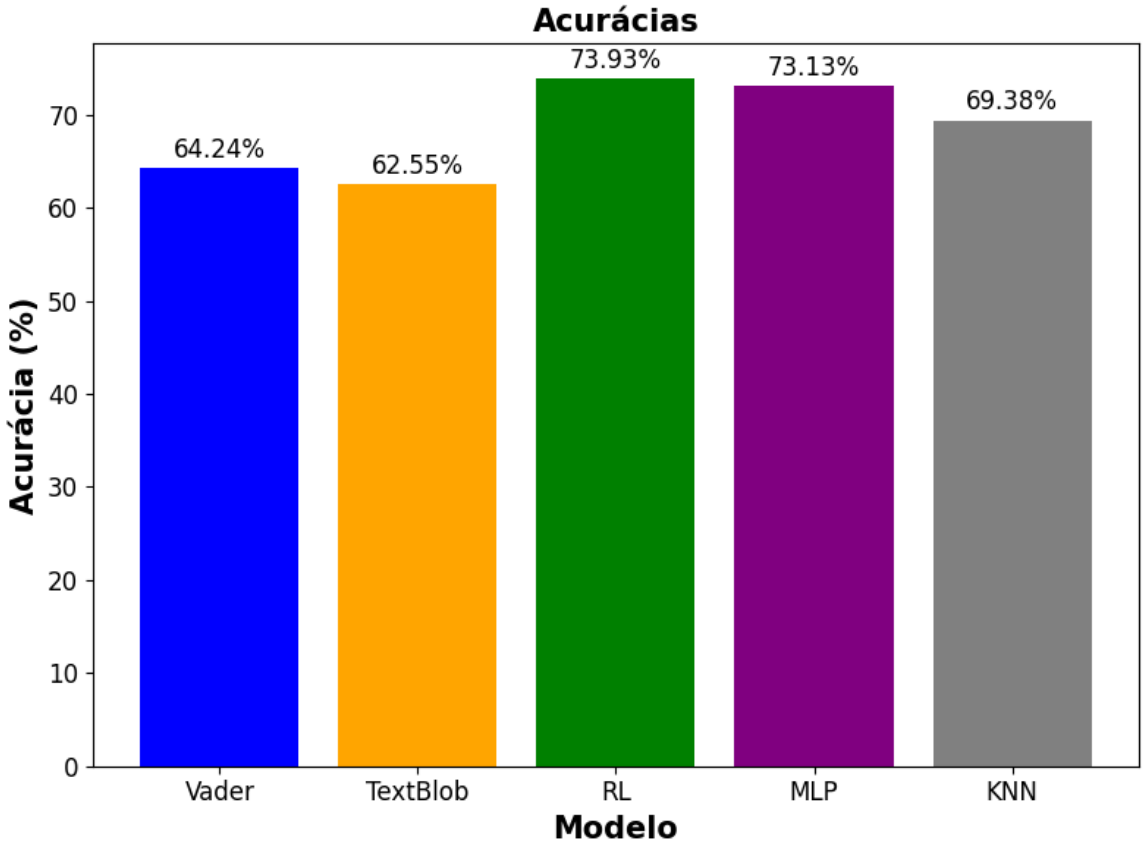
	precision	recall	f1-score	support
NEGATIVO	0.6600	0.8029	0.7245	3008
POSITIVO	0.7467	0.5842	0.6555	2992
accuracy			0.6938	6000
macro avg	0.7034	0.6935	0.6900	6000
weighted avg	0.7032	0.6938	0.6901	6000

Tabela 5 – Média dos melhores resultados encontrados pela validação cruzada.

	Acurácia	**Precisão**	**Recall**	**F1-Score**
RL	0.7393	0.7398	0.7394	0.7392
MLP	0.7313	0.7320	0.7314	0.7312
KNN	0.6938	0.7034	0.6935	0.6900

Como pode ser observado na figura 37, os algoritmos de PLN alcançam menor acurácia do que os modelos de *Machine Learning* treinados com uma parte dos dados. Como observado por Elbagir e Yang (2019), essa diferença na acurácia pode ser atribuída à natureza dos métodos de PLN, que geralmente dependem de léxicos e regras pré-definidas, limitando sua capacidade de adaptação a diferentes contextos e nuances da linguagem natural. Em contraste, os modelos de *Machine Learning* têm a capacidade de aprender a partir dos dados, identificando padrões complexos e ajustando-se às particularidades dos textos.

Figura 37 – Gráfico exibindo a porcentagem das acurácias dos algoritmos implementados.



É importante destacar que essa maior acurácia dos modelos de *Machine Learning* muitas vezes vem acompanhada de um tempo de processamento significativamente maior. O treinamento desses modelos requer grandes quantidades de dados e cálculos intensivos, o que pode ser demorado, especialmente em tarefas que envolvem processamento de texto em grande escala. Além disso, ao otimizar esses modelos por meio do *Grid SearchCV*, o tempo de busca pela melhor combinação de hiperparâmetros pode se tornar ainda mais demorado, pois o algoritmo avalia diversas configurações iniciais e otimiza para uma configuração ideal.

Figura 38 – Tempo de processamento por algoritmo (em segundos).

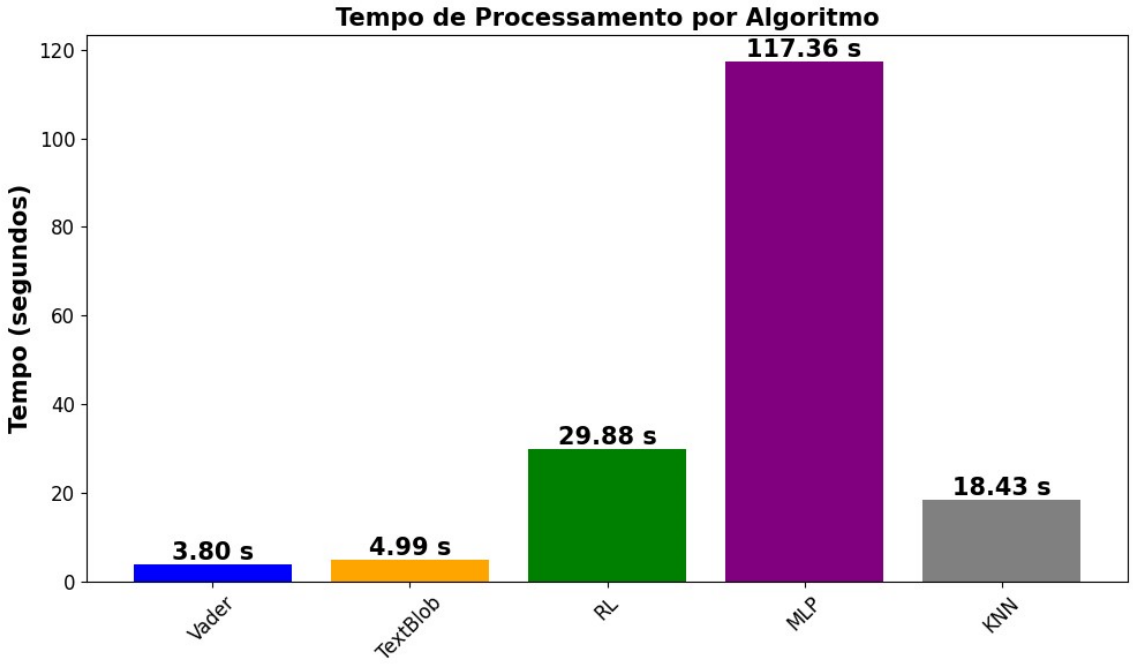


Figura 39 – Distribuição das previsões feitas pelo *Vader*.

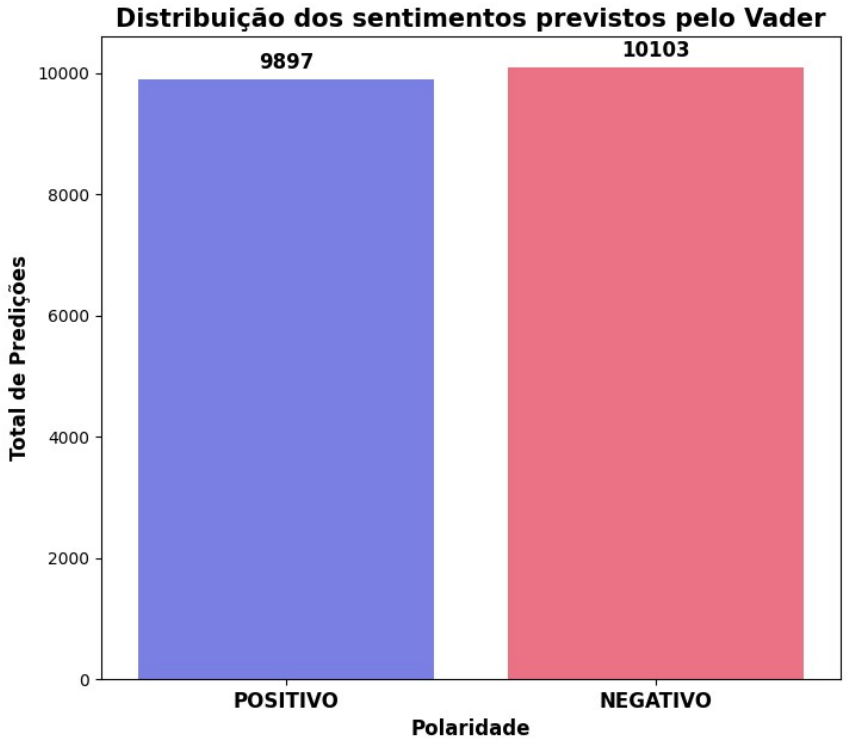


Figura 40 – Distribuição das previsões feitas pelo *TextBlob*.

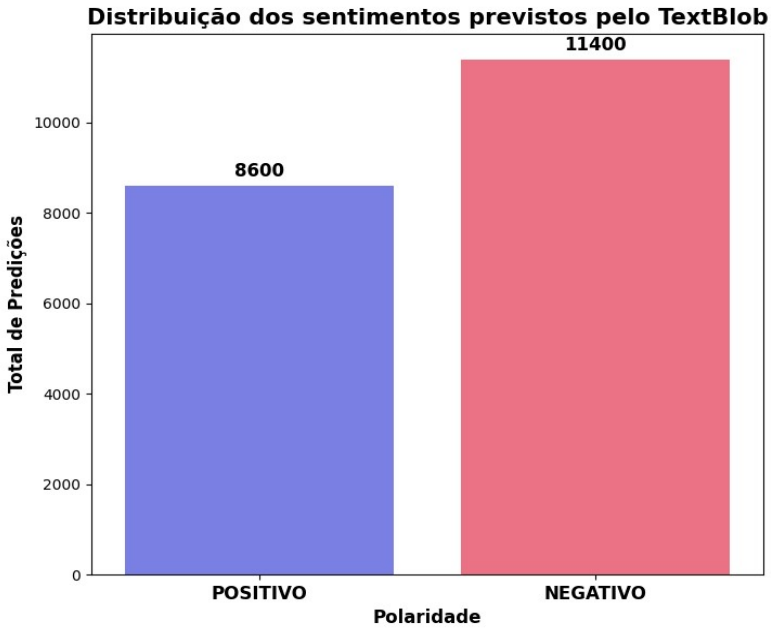


Figura 41 – Distribuição das previsões feitas pelo algoritmo Regressão Logística no conjunto de testes.

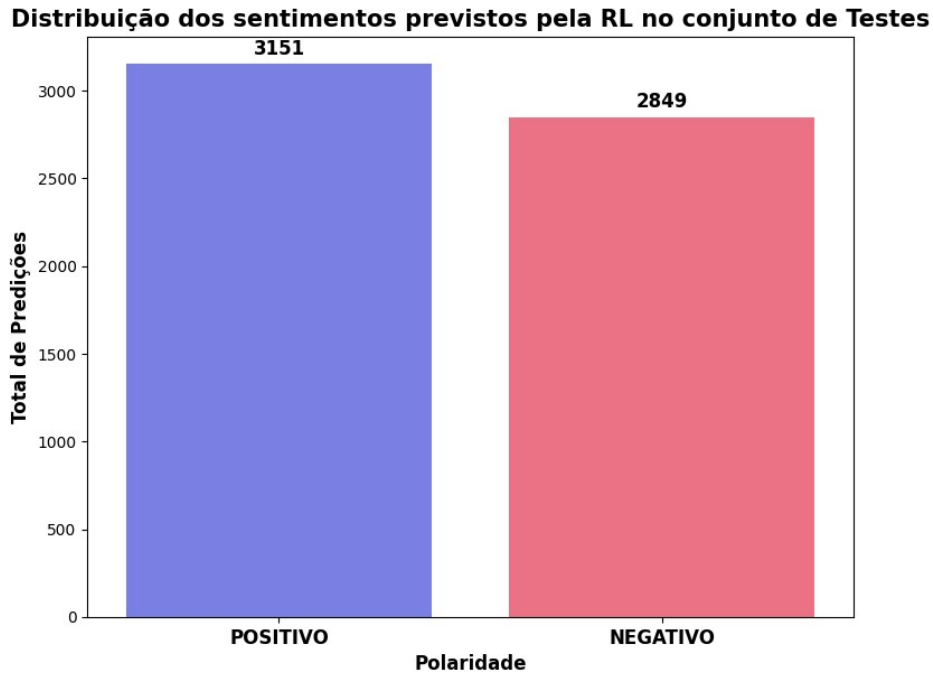


Figura 42 – Distribuição das previsões feitas pelo algoritmo MLP no conjunto de testes.

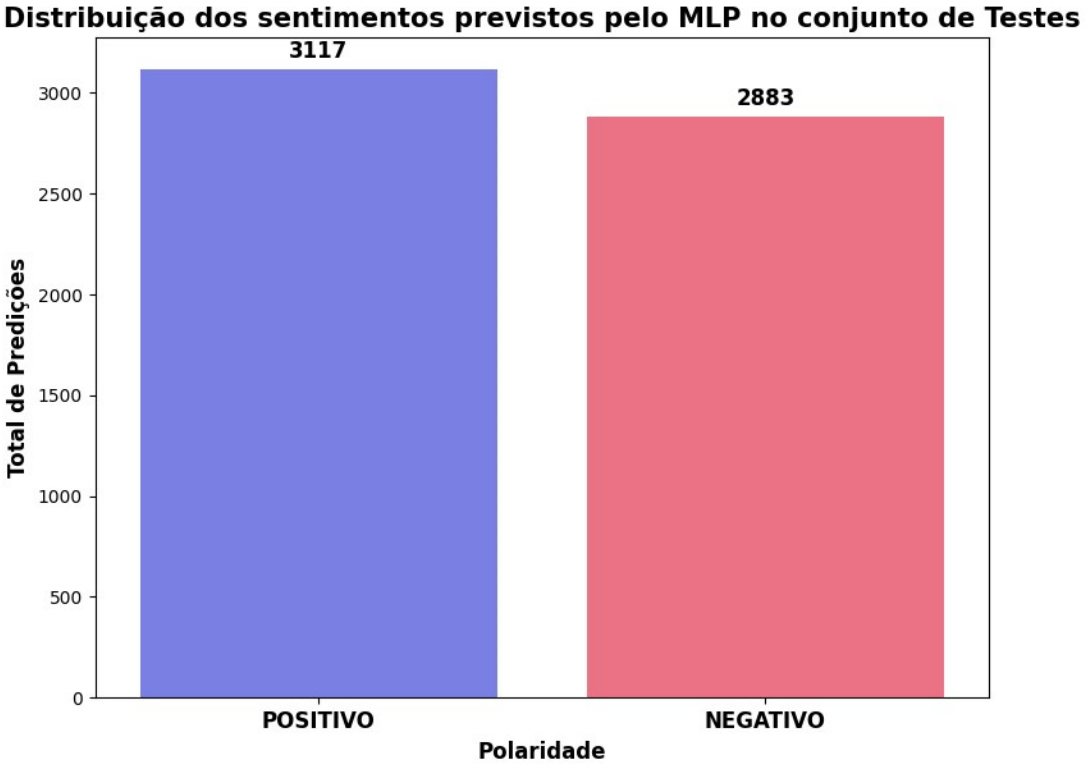


Figura 43 – Distribuição das previsões feitas pelo algoritmo KNN no conjunto de testes.

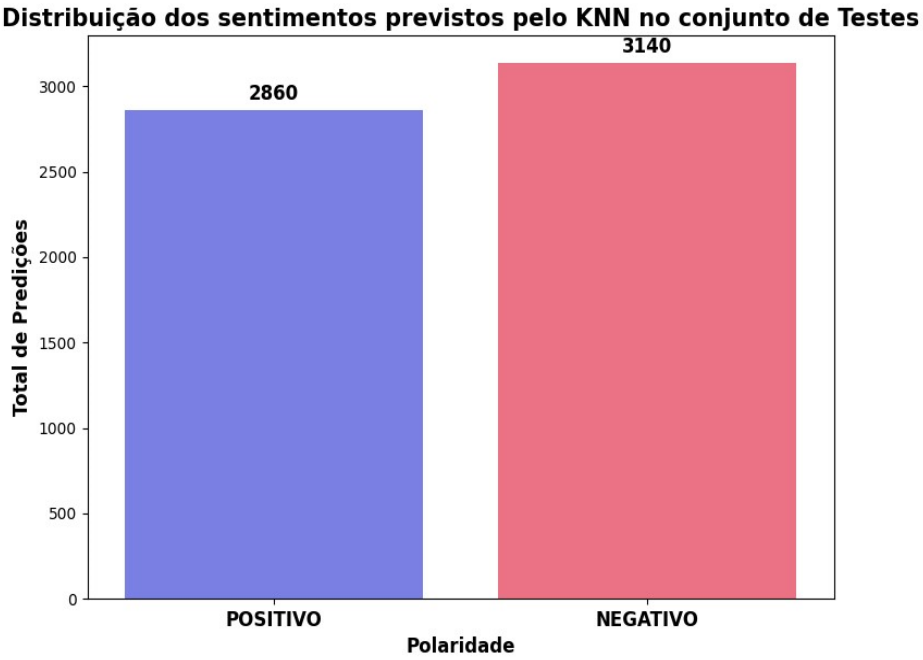


Figura 44 – *F1-Score* por Modelo de *Machine Learning*, dividido por classe.

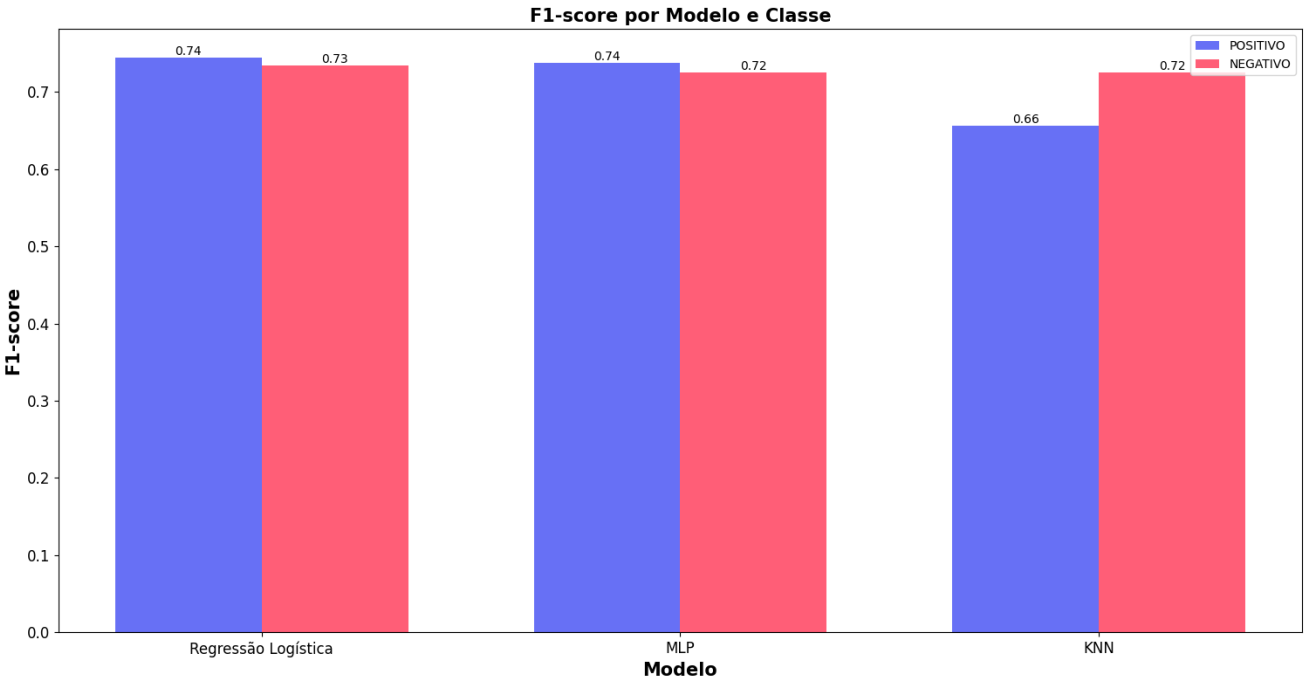
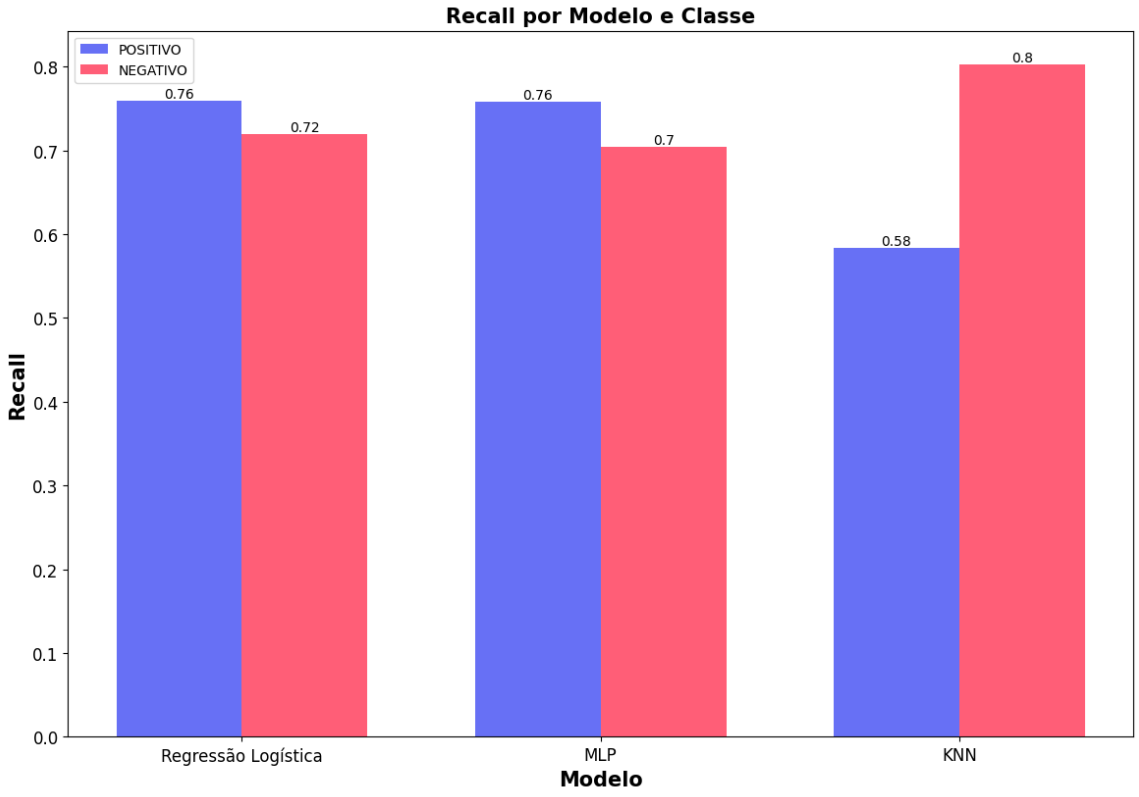


Figura 45 – *Recall* por Modelo de *Machine Learning*, dividido por classe.



A seguir serão apresentadas as matrizes de confusão de cada algoritmo. A matriz de confusão permite visualizar o desempenho dos algoritmos comparando as previsões feitas pelo modelo com os valores reais dos dados. Exibindo os verdadeiros e falsos positivos, e também os verdadeiros e falsos negativos. Embora neste trabalho esse processo tenha sido automatizado, os dados da matriz de confusão podem ser usados para calcular as métricas dos modelos, como o *recall*, *F1-Score*, etc.

Figura 46 – Matriz de confusão do *Vader*.

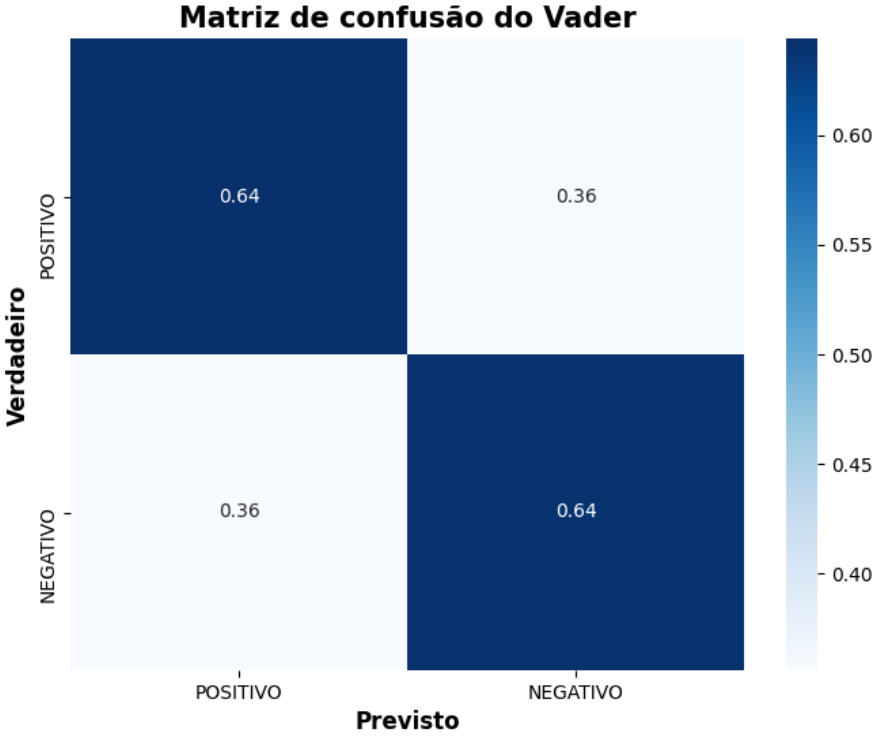


Figura 47 – Matriz de confusão do *TextBlob*.

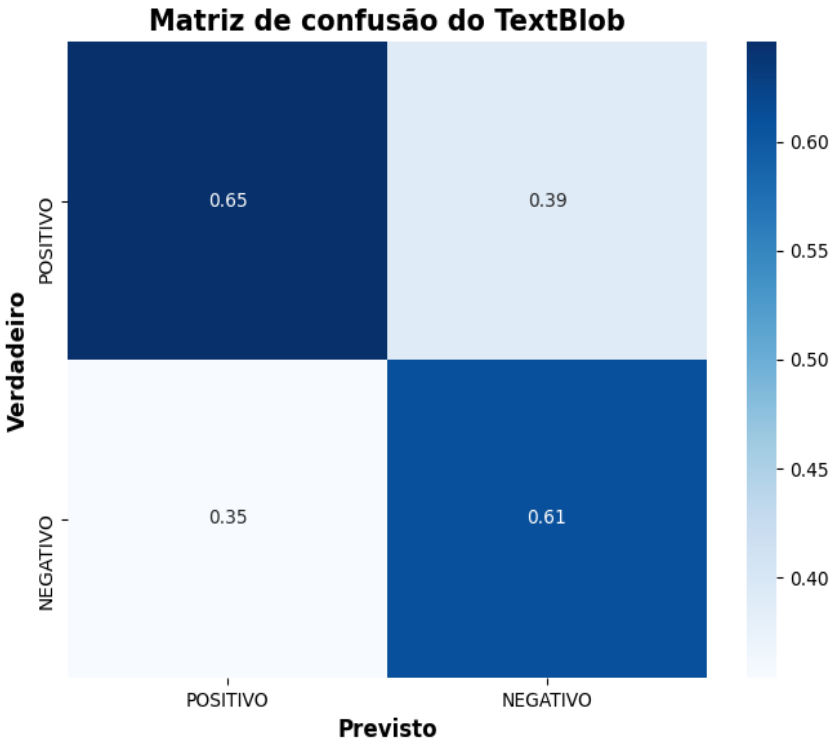


Figura 48 – Matriz de confusão do algoritmo Regressão Logística.

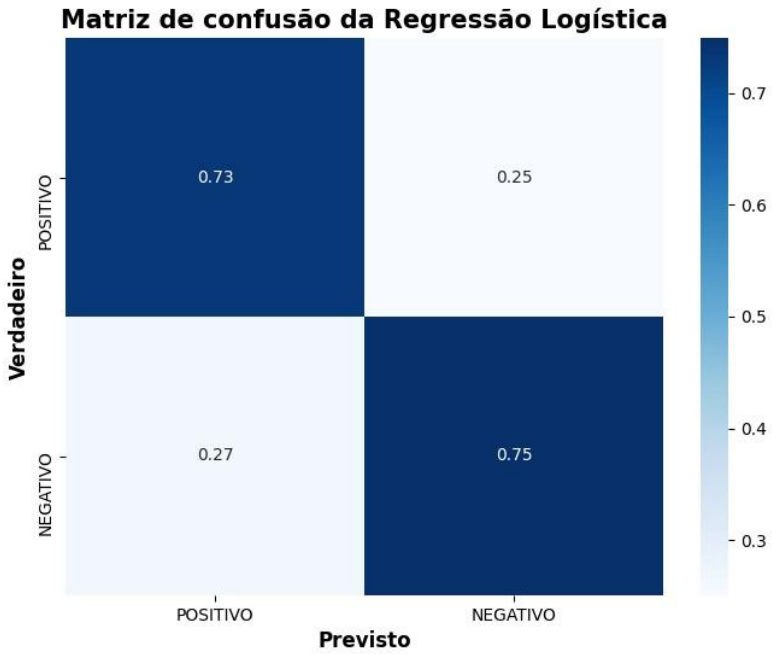


Figura 49 – Matriz de confusão do algoritmo MLP.

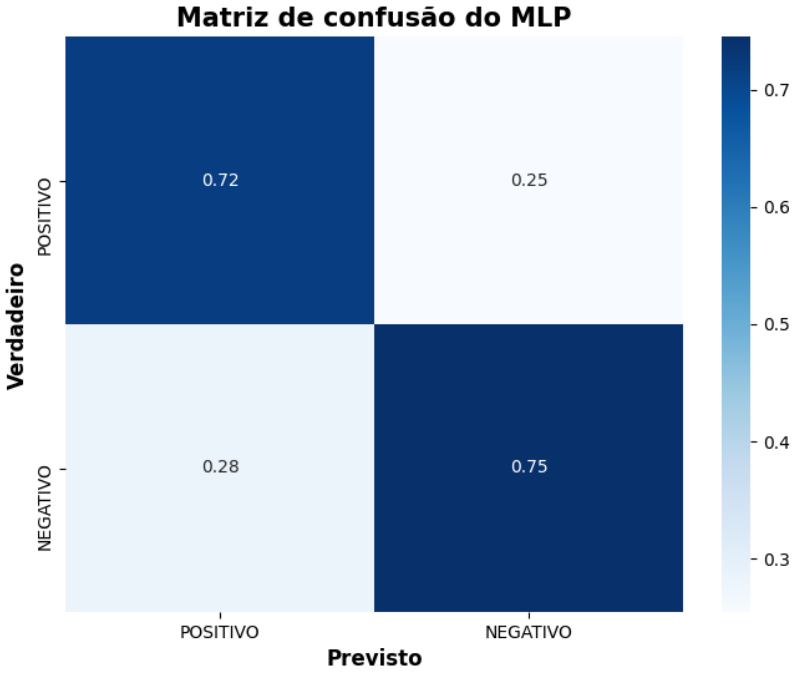
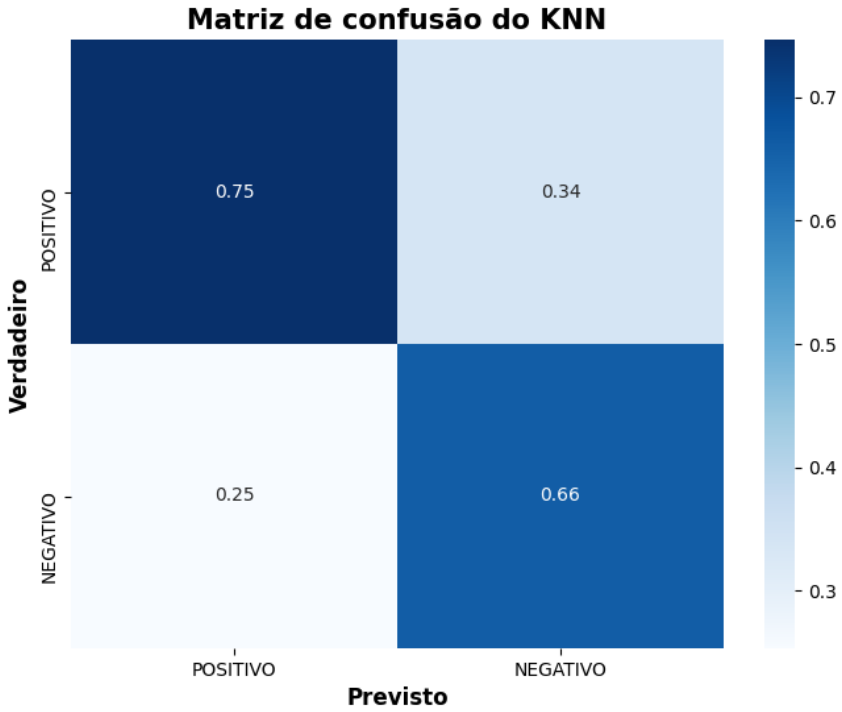


Figura 50 – Matriz de confusão do algoritmo KNN.



5. Considerações Finais

Este trabalho teve como objetivo comparar as técnicas de Processamento de Linguagem Natural (PLN) e *Machine Learning* na análise de sentimentos de textos obtidos da plataforma Twitter. Esse objetivo geral foi alcançado, tendo sido verificados a acurácia e desempenho dos algoritmos de processamento de linguagem natural em relação aos algoritmos de aprendizado de máquina na análise de sentimentos, através de uma implementação dos algoritmos em um código escrito na linguagem de programação *Python*, utilizando métricas de avaliação objetivas para realizar as comparações.

Ao avaliar o desempenho dos algoritmos de PLN *Vader* e *TextBlob*, em comparação com os modelos de *Machine Learning* Regressão Logística, MLP e KNN, os resultados indicaram que os modelos de aprendizado de máquina superaram os algoritmos de processamento de linguagem natural na tarefa de classificação de textos em relação à acurácia. Esta descoberta é relevante, pois sugere que algoritmos de aprendizado de máquina podem ser preferíveis em cenários em que a precisão na classificação de sentimentos é fundamental.

Entretanto, é importante destacar que essa vantagem em acurácia muitas vezes vem acompanhada de um tempo de processamento significativamente maior. Os modelos de *Machine Learning* exigem mais recursos computacionais e tempo para treinamento, tornando-se uma escolha crítica em situações em que a eficiência de tempo e outros recursos é um fator determinante. Portanto, a decisão de adotar uma abordagem baseada em PLN ou *Machine Learning* deve ser cuidadosamente ponderada, levando em consideração os objetivos específicos do projeto e as restrições de recursos disponíveis.

Ao longo da pesquisa foram explorados os conceitos fundamentais de inteligência artificial, processamento de linguagem natural, análise de sentimentos e aprendizado de máquina, proporcionando uma compreensão profunda do contexto do trabalho. Além disso, os algoritmos de PLN e Aprendizado de Máquina foram detalhadamente apresentados, bem como seus desempenhos e acurácias foram demonstrados e comparados. Desta forma, o objetivo geral proposto foi alcançado de forma plena através da consecução dos objetivos específicos delineados. Esse estudo visa contribuir para o avanço do conhecimento na área de análise de sentimentos, investigando as possibilidades de aplicação dessas técnicas em um cenário real de análise de dados provenientes de redes sociais.

5.1. Limitações

Durante o desenvolvimento do trabalho, a forma como os dados seriam obtidos mostrou-se uma questão relevante. A ideia original era utilizar a API (*Application Interface*) do *Twitter* para obter as postagens e posteriormente classificá-las, entretanto, devido às recentes alterações nas políticas de acesso da API, essa abordagem mostrou-se inviável, sendo decidido então utilizar um *dataset* com os sentimentos dos textos já classificados. Outra questão foi o tempo de processamento e consumo de recursos computacionais das ferramentas de Aprendizado de Máquina. Foi observado que embora essas ferramentas consigam, de forma geral, acurácia maior que as ferramentas de PLN, após certo ponto a razão acurácia por tempo de processamento se torna muito pequena, o que é especialmente relevante, uma vez que todos o desenvolvimento do trabalho foi feito com o uso de um computador pessoal.

5.2. Trabalhos Futuros

Para trabalhos futuros, faz-se necessária a investigação de técnicas de pré-processamento de texto mais avançadas, utilização de outros modelos e uma configuração mais ampla dos hiperparâmetros utilizados no *Grid SearchCV*. Sugere-se também que sejam explorados conjuntos de dados maiores e que se faça uso de recursos computacionais mais robustos.

6. Referências

Agrawal, A., Gans, J. & Goldfarb, A. Prediction Machines: The Simple Economics of Artificial Intelligence. Harvard Business Press, 2018.

Alpaydin, Ethem. Introduction to Machine Learning, 4ª Edição. MIT Press, 2014.

Armstrong, J. S. Illusions in regression analysis, 2012. Disponível em: https://www.researchgate.net/profile/J-Armstrong/publication/228194929_Illusions_in_Regression_Analysis/links/5b61ed410f7e9bc79a74e57e/Illusions-in-Regression-Analysis.pdf Acesso em: 9 jun. 2023.

Arntz, M., Gregory, T., & Zierahn, U. The Risk of Automation for Jobs in OECD Countries: A Comparative Analysis. OECD Social, Employment and Migration Working Papers, No. 189, OECD Publishing, Paris, 2016. Disponível em: https://www.oecd-ilibrary.org/docserver/5jlz9h56dvq7-en.pdf Acesso em: 1 jun. 2023.

Barocas, S., & Selbst, A. D. Big Data's Disparate Impact. California Law Review, 2016. Disponível em: https://www.jstor.org/stable/24758720 Acesso em: 12 jun. 2023. Acesso em: 1 jun. 2023.

Bento, Carolina. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis, 2021. Disponível em: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. Acesso em: 01 Jul. 2023.

Bode, L., Vraga, E. K., Borah, P., Shah, D. V. A New Space for Political Behavior: Political Social Networking and its Democratic Consequences, 2013. Disponível em: https://onlinelibrary.wiley.com/doi/full/10.1111/jcc4.12048 Acesso em: 28 ago. 2023.

Borgesius, F. J. Z., Gray, J., & Eechoud, M. V. Open Data, Privacy, and Fair Information Principles: Towards a Balancing Framework. Berkeley Technology Law Journal, 2015. Disponível em: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2695005 Acesso em: 1 jun. 2023.

Bose, Rajesh et al. Sentiment Analysis on the Basis of Tweeter Comments of Application of Drugs by Customary Language Toolkit and TextBlob Opinions of Distinct Countries, 2020. Disponível em: https://www.openacessjournal.com/article-file/20210112411902612527 80sen.pdf Acesso em: 9 jun. 2023.

Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B. & Anderson, H. The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018. Disponível em: https://arxiv.org/ftp/arxiv/papers/1802/1802.07228.pdf Acesso em: 12 jun. 2023.

Cambria, E., Das, D., Bandyopadhyay, S., & Feraco, A. Affective Computing and Sentiment Analysis. In IEEE Intelligent Systems, 2016. Disponível em: https://sentic.net/affective-computing-and-sentiment-analysis.pdf Acesso em: 12 jul. 2023.

Castelvecchi, D. Can We Open the Black Box of AI?. Nature News, 2016. Disponível em: https://www.nature.com/news/polopoly_fs/1.20731!/menu/main/topColumns/topLeftColumn/pdf/538020a.pdf Acesso em: 12 jun. 2023.

Copeland, J. The Modern History of Computing. In Zalta E. N. (Ed.), The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), 2019.

Cortes, C.; Vapnik, V. Support-Vector Networks, 1995. Disponível em: https://link.springer.com/article/10.1007/bf00994018 Acesso em: 15 jul. 2023.

Davenport, T. & Ronanki, R. Artificial Intelligence for the Real World. Harvard Business Review, 2018.

Elbagir, Shihab; Yang, Jing. Twitter Sentiment Analysis Using Natural Language Toolkit and VADER Sentiment, 2019. Disponível em: https://www.iaeng.org/publication/IMECS2019/IMECS2019_pp12-16.pdf Acesso em: 2 out. 2023.

Galdino, N. Big data: Ferramentas e Aplicabilidade, 2016. Disponível em: https://www.aedb.br/seget/arquivos/artigos16/472427.pdf. Acesso em: 2 out. 2023.

Govan, Jennifer. Today In History: Twitter Launches, 2021. Disponível em: https://library.tc.columbia.edu/blog/content/2021/july/24603---today-in-history-twitter-launches.php Acesso em: 28 ago. 2023.

Grossi, A. A. D. et al. Comparação e avaliação de técnicas de aprendizado de máquina para

Hastie, T., Tibshirani, R., & Friedman, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2ª Edição, 2016.

Hutto, C. J, & Gilbert, E. Vader: A parsimonious rule-based model for sentiment analysis of social media text, 2014. Disponível em: https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399 Acesso em: 1 jun. 2023.

indicação de biópsia para o câncer de próstata, 2013. Disponível em: http://www.uel.br/cce/dc/wp-content/uploads/PlanoTCC-Andr%C3%A9-Del-Grossi-Revisado.pdf Acesso em: 9 jun. 2023.

Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S. & Wang, Y. Artificial intelligence in healthcare: past, present and future. Stroke and Vascular Neurology, 2017. Disponível em: https://svn.bmj.com/content/svnbmj/2/4/230.full.pdf Acesso em: 1 jun. 2023.

Jurafsky, D. & Martin, J. H. Speech and Language Processing (2ª Edição), 2008.

Jurafsky, D. & Martin, J.H. Speech and Language Processing, 2019. Disponível em: https://web.stanford.edu/~jurafsky/slp3/old_oct19/17.pdf Acesso em: 9 jun. 2023.

Kaplan, F. A Short History of Robotic Art. In 22nd International Symposium on Electronic Art (ISEA2016), Hong Kong, 2016.

Kapoor, R., Lee, J. M., & Venkataraman, N. Artificial Intelligence: Its Implications for Income Distribution and Unemployment, 2017. Disponível em: https://www.nber.org/system/files/chapters/c14018/revisions/c14018.rev1.pdf Acesso em: 15 jun. 2023.

Kouloumpis, E., Wilson, T., & Moore, J. Twitter Sentiment Analysis: The Good the Bad and the OMG!, 2011. Disponível em: https://ojs.aaai.org/index.php/ICWSM/article/view/14185/14034 Acesso em: 12 jul. 2023.

Krizhevsky, A. et al. ImageNet Classification with Deep Convolutional Neural Networks, 2012. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf Acesso em: 15 jul. 2023.

Kumar, V.; Sebastian, T. M. Sentiment Analysis on Twitter, 2012. Disponível em: https://www.researchgate.net/publication/304579281_Sentiment_Analysis_on_Twitter Acesso em: 1 jun. 2023.

Kwak, H., Lee, C., Park, H., & Moon, S. What is Twitter, a social network or a news media? 2010. Disponível em: https://dl.acm.org/doi/10.1145/1772690.1772751 Acesso em: 28 ago. 2023.

Lee, Kevin C. Sentiment Analysis — Comparing 3 Common Approaches: Naive Bayes, LSTM, and VADER, 2021. Disponível em: https://towardsdatascience.com/sentiment-analysis-comparing-3-common-approaches-naive-bayes-lstm-and-vader-ab561f834f89 Acesso em: 2 out. 2023.

Liddy, E. D. Natural Language Processing, 2001. Disponível em: https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub Acesso em: 11 jul. 2023.

Liu, B. Sentiment Analysis and Opinion Mining, 2012. Disponível em: https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf Acesso em: 11 jul. 2023.

Loria, Steven. textblob Documentation Release 0.16.0, 2020. Disponível em: https://buildmedia.readthedocs.org/media/pdf/textblob/latest/textblob.pdf Acesso em: 9 jun. 2023.

Lotan, G., Graeff, E., Ananny, M., Gaffney, D., Pearce, I., & boyd, d. The revolutions were tweeted: Information flows during the 2011 Tunisian and Egyptian revolutions, 2011. Disponível em: https://ijoc.org/index.php/ijoc/article/view/1246 Acesso em: 28 ago. 2023.

Luckin, R., Holmes, W., Griffiths, M., & Forcier, L. B. Intelligence Unleashed: An Argument for AI in Education. Pearson, 2016. Disponível em: https://www.researchgate.net/profile/Rosemary-Luckin/publication/299561597_Intelligence_Unleashed_An_argument_for_AI_in_Education/links/5812782608ae1f5510c2aa4d/Intelligence-Unleashed-An-argument-for-AI-in-Education.pdf Acesso em: 12 jun. 2023.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. The Stanford CoreNLP Natural Language Processing Toolkit, 2014. Disponível em: https://aclanthology.org/P14-5010.pdf Acesso em: 12 jul. 2023.

McCarthy, J. What is Artificial Intelligence? Stanford University, 2007. Disponível em: http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html. Acesso em 15 de Junho de 2023.

McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. A proposal for the Dartmouth summer research project on artificial intelligence, August 31, 1955. AI Magazine, 2006.

Disponível em: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1904/1802 Acesso em: 1 jun. 2023.

Nilsson, N. The Quest for Artificial Intelligence: A History of Ideas and Achievements, Cambridge University Press, 2009.

Oliveira, A. R. Comparação de algoritmos de aprendizagem de máquina para construção de modelos preditivos de diabetes não diagnosticado, 2016. Disponível em: https://lume.ufrgs.br/handle/10183/140847 Acesso em: 9 jun. 2023.

O'Reilly, Tim; Milstein, Sarah. The Twitter Book. 2ª Edição. Sebastopol: O'Reilly Media, 2012.

Pak, A., & Paroubek, P. Twitter as a Corpus for Sentiment Analysis and Opinion Mining, 2010. Disponível em: https://aclanthology.org/L10-1263/ Acesso em: 29 ago. 2023.

Pang, B., Lee, L. & Vaithyanathan, S. Thumbs up? sentiment classification using machine learning techniques, 2002 Disponível em: https://aclanthology.org/W02-1011.pdf Acesso em: 9 jun. 2023.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. Scikit-learn: Machine Learning in Python, 2011. Journal of Machine Learning Research.

Perez, Sarah. Twitter's doubling of character count from 140 to 280 had little impact on length of tweets, 2018. Disponível em: https://techcrunch.com/2018/10/30/twitters-doubling-of-character-count-from-140-to-280-had-little-impact-on-length-of-tweets Acesso em: 28 ago. 2023.

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., ... & Bengio, Y. Tackling Climate Change with Machine Learning, 2019. Disponível em: https://arxiv.org/pdf/1906.05433 Acesso em: 1 jun. 2023.

Russell, S. & Norvig, P. Artificial Intelligence: A Modern Approach, 3ª Edição. Pearson, 2016.

Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers, IBM Journal of Research and Development, 1959. Disponível em: http://people.csail.mit.edu/brooks/idocs/Samuel.pdf Acesso em: 15 jul. 2023.

Silva, J. A. S.; Mairink, C. H. P. Inteligência artificial: aliada ou inimiga. LIBERTAS: Rev. Ciênci. Soc. Apl., Belo Horizonte, 2019.

Stephens-Davidowitz, Seth. Everybody lies - Big Data, New Data and what the internet can tell us about who we really are, 2017.

Tsur, O., Davidov, D., & Rappoport, A. ICWSM - A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews, 2010. Disponível em: https://ojs.aaai.org/index.php/ICWSM/article/view/14018/13867 Acesso em: 12 jul. 2023.

Turney, P. D. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews, 2002. Disponível em: https://arxiv.org/ftp/cs/papers/0212/0212032.pdf Acesso em: 11 jul. 2023.

Vaswani, A. et al. Attention Is All You Need, 2017. Disponível em: https://arxiv.org/pdf/1706.03762.pdf Acesso em: 9 jun. 2023.

7. Anexos

Anexo 1 – *Link* para o repositório contendo o dataset particionado, todos os códigos-fonte, e as instruções para a utilização dos algoritmos.

https://github.com/gbrb1/datasets_tcc

Anexo 2 – Código do programa responsável pela extração dos dados do *dataset* original.

```
import pandas as pd
import os

# Definindo as colunas e a codificação
DATASET_COLUMNS = ["target", "ids", "date", "flag", "user", "text"]
DATASET_ENCODING = "ISO-8859-1"

# Lendo o arquivo CSV
dataset_filename = os.listdir("input")[0]
dataset_path = os.path.join("input", dataset_filename)
print("Abrindo arquivo:", dataset_path)
df = pd.read_csv(dataset_path, encoding=DATASET_ENCODING)

# Pegando as linhas de 0 a 9999 e de 805000 a 814999
parte_1 = df.iloc[0:10000]
parte_2 = df.iloc[805000:815000]

# Concatenando as duas partes
df_final = pd.concat([parte_1, parte_2])

# Obtendo apenas as colunas necessárias
df_final = df_final[["target", "text"]]

# Escrevendo o resultado em um novo arquivo CSV
df_final.to_csv('dataset_particionado.csv', index=False)
```

Anexo 3 – Código do programa principal.

```
# Bibliotecas Padrão
import os
import re
import numpy as np
import time

# Bibliotecas de Terceiros
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV

# Iniciando o objeto do temporizador
process_times = {}

# Baixando pacotes do NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
```

```
# Definindo lista de stopwords
stop_words = set(stopwords.words('english'))

# Definindo as cores utilizadas nos gráficos
cor_positivo = '#6770f5'
cor_negativo = '#ff5e77'

# Função para limpar e processar o texto


def process_text(text):
    # Limpa o texto, removendo menções a usuários,
    # URLs e caracteres não-alfanuméricos.
    text = re.sub(r'@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+', ' ', text)

    # Tokeniza o texto em palavras
    tokenized = word_tokenize(text)

    # Retorna uma lista de palavras, removendo aquelas que
    # estão na lista de palavras de parada (stopwords).
    return [word for word in tokenized if word.casefold() not in stop_words]


# Instanciando o analisador de sentimentos do Vader
sia = SentimentIntensityAnalyzer()

# Classificando os sentimentos usando Vader


def nltk_sentiment(text):
    text = process_text(text)
    text = ' '.join(text)
    sentiment = sia.polarity_scores(text)
    if sentiment['compound'] > 0 and sentiment['pos'] > 0:
```

```
        return "POSITIVO"
    else:
        return "NEGATIVO"

# Função para classificar os sentimentos
# usando o TextBlob


def textblob_sentiment(text):
    text = process_text(text)
    text = ' '.join(text)
    sentiment = TextBlob(text).sentiment.polarity
    if sentiment > 0:
        return "POSITIVO"
    else:
        return "NEGATIVO"


# Definindo parâmetro para assegurar que apenas
# as colunas necessárias serão utilizadas
DATASET_COLUMNS = ["target", "text"]

# Definindo a codificação do conjunto de dados como "ISO-8859-1".
DATASET_ENCODING = "ISO-8859-1"

# Lendo o arquivo CSV
dataset_filename = os.listdir("input")[0]
dataset_path = os.path.join("input", dataset_filename)
print("Abrindo arquivo:", dataset_path)
df = pd.read_csv(dataset_path, encoding=DATASET_ENCODING,
                 names=DATASET_COLUMNS)

# Mapeando a coluna label do dataset
decode_map = {0: "NEGATIVO", 4: "POSITIVO"}
```

```
# Definindo uma função chamada "decode_sentiment" que
# recebe um rótulo como entrada e retorna um rótulo legível


def decode_sentiment(label):
    return decode_map[int(label)]


# Aplica a função "decode_sentiment" a cada elemento da coluna
# "target" do DataFrame "df".
# Isso é feito usando a função "apply" do pandas com uma função
# lambda que passa cada valor da coluna "target" para "decode_sentiment".
df.target = df.target.apply(lambda x: decode_sentiment(x))

# Definindo a proporção do conjunto de teste em relação
# ao conjunto de dados completo como 0.3 (30%).
TEST_SIZE = 0.3

# Definindo o tamanho do conjunto de treinamento como o
# complemento do tamanho do conjunto de teste (70%).
TRAIN_SIZE = 1 - TEST_SIZE

# Aplicando a função de análise de sentimentos com o Vader
# no DataFrame
start_time = time.time()
df['nltk_pred'] = df['text'].apply(nltk_sentiment)
end_time = time.time()
process_times["Vader"] = end_time - start_time

# Aplicando a função de análise de sentimentos com o TextBlob
# no DataFrame
start_time = time.time()
df['textblob_pred'] = df['text'].apply(textblob_sentiment)
```

```
end_time = time.time()
process_times["TextBlob"] = end_time - start_time

# Obtendo e imprimindo a acurácia do Vader e TextBlob
print('Acurácia do Vader:',
      accuracy_score(df['target'], df['nltk_pred']))
print('Acurácia do TextBlob:',
      accuracy_score(df['target'], df['textblob_pred']))

# Processando os textos e transformando-os em listas de strings
# que serão armazenadas na coluna processed_text
df['processed_text'] = df['text'].apply(lambda x: ' '
                                        .join(process_text(x)))

# Instanciando o vetorizador
vectorizer = TfidfVectorizer()

# Vetorizando o texto
X = vectorizer.fit_transform(df['processed_text'])
y = df['target']

# Utilizando o temporizador para obter um número inteiro
# para ser utilizado como seed no parametro random_state
current_time = int(time.time())

# Dividindo os dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, random_state=current_time)

# Definindo a grade de parâmetros para a regressão logística
param_grid = {
    'C': [1, 10, 20, 30, 40, 50, 100, 150],
    'penalty': ['l2'],
    'solver': ['lbfgs'],
```

```
    'max_iter': [250, 500, 1000]
}

# Criando uma instância do modelo de regressão logística
start_time = time.time()
log_reg = LogisticRegression()

# Criando o objeto GridSearchCV com o modelo, os parâmetros e a validação cruzada
grid_search = GridSearchCV(
    estimator=log_reg, param_grid=param_grid, cv=5, n_jobs=-1)

# Executando a pesquisa em grade nos dados de treinamento
grid_search.fit(X_train, y_train)

# Obtendo os melhores hiperparâmetros encontrados
best_params = grid_search.best_params_

# Usando o modelo com os melhores hiperparâmetros para fazer previsões
log_reg_best = grid_search.best_estimator_
log_reg_preds = log_reg_best.predict(X_test)
df['log_reg_pred'] = log_reg_best.predict(X)

end_time = time.time()
process_times["RL"] = end_time - start_time

param_grid = {
    'hidden_layer_sizes': [(5, 5), (10, 10), (25, 25)],
    'max_iter': [50, 100, 150],
    'alpha': [0.001],
    'learning_rate_init': [0.01],
    'solver': ['lbfgs'],
    'activation': ['relu'],
    'random_state': [current_time],
}
```

```
# Instanciando temporizador para calcular o tempo de processamento do MLP
start_time = time.time()
# Criando uma instância do modelo MLP
mlp = MLPClassifier()

# Criando um objeto GridSearchCV
grid_search = GridSearchCV(
    estimator=mlp, param_grid=param_grid, cv=5, n_jobs=-1)

# Realizando a busca em grade no conjunto de treinamento
grid_search.fit(X_train, y_train)

# Obtendo os melhores parâmetros encontrados pelo grid search
best_params = grid_search.best_params_

# Obtendo o melhor modelo treinado
best_mlp_model = grid_search.best_estimator_

# Fazendo previsões com o melhor modelo
mlp_preds = best_mlp_model.predict(X_test)
df['mlp_pred'] = best_mlp_model.predict(X)

end_time = time.time()
process_times["MLP"] = end_time - start_time

start_time = time.time()

param_grid = {
    'n_neighbors': [100, 151, 200],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'brute']
}
```

```
# Criando uma instância do KNeighborsClassifier
knn = KNeighborsClassifier()

# Criando uma instância do GridSearchCV
grid_search = GridSearchCV(
    estimator=knn, param_grid=param_grid, cv=5, n_jobs=-1)

# Executando a pesquisa em grade nos dados de treinamento
grid_search.fit(X_train, y_train)

# Obtendo o melhor modelo treinado com os melhores hiperparâmetros
best_knn_model = grid_search.best_estimator_

# Obtendo os melhores hiperparâmetros encontrados pela pesquisa em grade
best_params = grid_search.best_params_

# Fazer previsões usando o melhor modelo
knn_preds = best_knn_model.predict(X_test)
df['knn_pred'] = best_knn_model.predict(X)

# Calculando o tempo de processamento
end_time = time.time()
process_times["KNN"] = end_time - start_time

# Calculando acurácia e processando relatórios das ferramentas de ML
print('Acurácia da regressão logística:',
      accuracy_score(y_test, log_reg_preds))
print('Acurácia do MLP:', accuracy_score(y_test, mlp_preds))
print('Acurácia do KNN:', accuracy_score(y_test, knn_preds))

print('Relatório de classificação da regressão logística:\n',
      classification_report(y_test, log_reg_preds, digits=4))

print('Relatório de classificação do MLP:\n',
```

```
      classification_report(y_test, mlp_preds, digits=4))

print('Relatório de classificação do KNN:\n',
      classification_report(y_test, knn_preds, digits=4))

# Calculando o F1-score e Recall para cada classe (positivo e negativo) usando o classification_report
report_nltk = classification_report(
    df['target'], df['nltk_pred'], output_dict=True)



report_textblob = classification_report(
    df['target'], df['textblob_pred'], output_dict=True)

report_log_reg = classification_report(y_test, log_reg_preds, output_dict=True)
log_reg_f1_pos = report_log_reg['POSITIVO']['f1-score']
log_reg_recall_pos = report_log_reg['POSITIVO']['recall']
log_reg_f1_neg = report_log_reg['NEGATIVO']['f1-score']
log_reg_recall_neg = report_log_reg['NEGATIVO']['recall']

report_mlp = classification_report(y_test, mlp_preds, output_dict=True)
mlp_f1_pos = report_mlp['POSITIVO']['f1-score']
mlp_recall_pos = report_mlp['POSITIVO']['recall']
mlp_f1_neg = report_mlp['NEGATIVO']['f1-score']
mlp_recall_neg = report_mlp['NEGATIVO']['recall']

report_knn = classification_report(y_test, knn_preds, output_dict=True)
knn_f1_pos = report_knn['POSITIVO']['f1-score']
knn_recall_pos = report_knn['POSITIVO']['recall']
knn_f1_neg = report_knn['NEGATIVO']['f1-score']
knn_recall_neg = report_knn['NEGATIVO']['recall']



# GRÁFICOS
```

```
# Plotando um gráfico de barras com os tempos de processamento
plt.figure(figsize=(10, 6))
times = list(process_times.values())
algorithms = list(process_times.keys())
plt.bar(algorithms, times, color=['blue', 'orange', 'green', 'purple', 'grey'])
plt.title('Tempo de Processamento por Algoritmo',
          fontsize=15, fontweight='bold')
plt.ylabel('Tempo (segundos)', fontsize=15, fontweight='bold')
plt.xticks(fontsize=12, rotation=45)
plt.yticks(fontsize=12)

# Adicionar os valores numéricos acima das barras
for i, time in enumerate(times):
    plt.text(i, time, f'{time:.2f} s',
             ha='center', va='bottom', fontsize=15, fontweight='bold')

plt.tight_layout()
plt.show()

# Plotando o gráfico da distribuição da polaridade dos textos
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='target', data=df, order=['POSITIVO', 'NEGATIVO'], palette={
    'POSITIVO': cor_positivo, 'NEGATIVO': cor_negativo})
plt.title('Distribuição dos dados', fontsize=15,
          fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Registros', fontsize=12, fontweight='bold')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
                textcoords='offset points')
plt.show()
```

```
# Plotando o gráfico dos conjuntos de testes e treino

sizes = [X_train.shape[0], X_test.shape[0]]
labels = ['Treino', 'Teste']
colors = [cor_positivo, cor_negativo]
fig, ax = plt.subplots()

# Usando uma função lambda para formatar o rótulo


def format_label(p): return f'{p:.1f}%\n({int(p * sum(sizes) / 100)})'


pie = ax.pie(sizes, colors=colors, labels=labels,
             autopct=format_label, startangle=140)
ax.legend(labels, loc='upper right')
plt.title('Distribuição dos Dados de Treino e Teste',
          fontsize=15, fontweight='bold')

# Personalizando as fontes dos rótulos
for label in pie[1]:
    label.set_fontsize(12)
    label.set_fontweight('bold')

plt.show()

# Função para plotar a matriz de confusão normalizada


def plot_confusion_matrix(cm, classes, title):
    # Normalização das matrizes
    cm = cm.astype('float') / cm.sum(axis=0)[np.newaxis, :]
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='.2f', xticklabels=classes,
```

```
            yticklabels=classes, cmap='Blues')
    plt.title(title, fontsize=15, fontweight='bold')
    plt.ylabel('Verdadeiro', fontsize=12, fontweight='bold')
    plt.xlabel('Previsto', fontsize=12, fontweight='bold')
    plt.show()


# Plotando as matrizes de confusão...
nltk_cm = confusion_matrix(df['target'], df['nltk_pred'], labels=[
                    'POSITIVO', 'NEGATIVO'])
plot_confusion_matrix(
    nltk_cm, classes=['POSITIVO', 'NEGATIVO'], title='Matriz de confusão do Vader')

textblob_cm = confusion_matrix(df['target'], df['textblob_pred'], labels=[
                        'POSITIVO', 'NEGATIVO'])
plot_confusion_matrix(textblob_cm, classes=[
                'POSITIVO', 'NEGATIVO'], title='Matriz de confusão do TextBlob')

log_reg_cm = confusion_matrix(y_test, log_reg_preds, labels=[
    'POSITIVO', 'NEGATIVO'])
plot_confusion_matrix(log_reg_cm, classes=[
                'POSITIVO', 'NEGATIVO'], title='Matriz de confusão da Regressão Logística')

mlp_cm = confusion_matrix(y_test, mlp_preds, labels=[
    'POSITIVO', 'NEGATIVO'])
plot_confusion_matrix(mlp_cm, classes=[
                'POSITIVO', 'NEGATIVO'], title='Matriz de confusão do MLP')


knn_cm = confusion_matrix(y_test, knn_preds, labels=[
    'POSITIVO', 'NEGATIVO'])
plot_confusion_matrix(knn_cm, classes=[
                'POSITIVO', 'NEGATIVO'], title='Matriz de confusão do KNN')
```

```
# Plotando o gráfico de acurácias
accuracies = [accuracy_score(df['target'], df['nltk_pred']), accuracy_score(
    df['target'], df['textblob_pred']), accuracy_score(y_test, log_reg_preds),
accuracy_score(y_test, mlp_preds), accuracy_score(y_test, knn_preds)]

# Multiplicando as acurácias por 100 para obter porcentagens
accuracies_percent = [accuracy * 100 for accuracy in accuracies]

plt.figure(figsize=(8, 6))

bars = plt.bar(['Vader', 'TextBlob', 'RL', 'MLP', 'KNN'],
               accuracies_percent, color=['blue', 'orange', 'green', 'purple', 'grey'])
plt.title('Acurácias', fontsize=15, fontweight='bold')
plt.ylabel('Acurácia (%)', fontsize=15, fontweight='bold')
plt.xlabel('Modelo', fontsize=15, fontweight='bold')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Adicionando a acurácia em porcentagem acima das barras
for bar in bars:
    yval = round(bar.get_height(), 2)
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.5,
             f'{yval}%', ha='center', va='bottom', fontsize=12)

plt.tight_layout()
plt.show()

# Plotando o gráfico de barras do F1-score para cada classe
f1_scores_pos = [log_reg_f1_pos, mlp_f1_pos, knn_f1_pos]
f1_scores_neg = [log_reg_f1_neg, mlp_f1_neg, knn_f1_neg]

plt.figure(figsize=(8, 6))
bar_width = 0.35
```

```
index = np.arange(len(f1_scores_pos))

plt.bar(index, f1_scores_pos, bar_width, color=cor_positivo, label='POSITIVO')
plt.bar(index + bar_width, f1_scores_neg,
        bar_width, color=cor_negativo, label='NEGATIVO')
plt.title('F1-score por Modelo e Classe', fontsize=15, fontweight='bold')
plt.ylabel('F1-score', fontsize=15, fontweight='bold')
plt.xlabel('Modelo', fontsize=15, fontweight='bold')
plt.xticks(index + bar_width/2,
           ['Regressão Logística', 'MLP', 'KNN'], fontsize=12)
plt.yticks(fontsize=12)
plt.legend()

# Adicionando os valores numéricos acima das barras
for i, score in enumerate(f1_scores_pos):
    plt.text(i, score, str(round(score, 2)),
             ha='center', va='bottom', fontsize=10)
for i, score in enumerate(f1_scores_neg):
    plt.text(i + bar_width, score, str(round(score, 2)),
             ha='center', va='bottom', fontsize=10)
plt.tight_layout()
plt.show()

# Plotando o gráfico de barras do Recall para cada classe
recall_scores_pos = [log_reg_recall_pos, mlp_recall_pos, knn_recall_pos]
recall_scores_neg = [log_reg_recall_neg, mlp_recall_neg, knn_recall_neg]

plt.figure(figsize=(8, 6))
bar_width = 0.35
index = np.arange(len(recall_scores_pos))

plt.bar(index, recall_scores_pos, bar_width,
        color=cor_positivo, label='POSITIVO')
plt.bar(index + bar_width, recall_scores_neg,
```

```
        bar_width, color=cor_negativo, label='NEGATIVO')
plt.title('Recall por Modelo e Classe', fontsize=15, fontweight='bold')
plt.ylabel('Recall', fontsize=15, fontweight='bold')
plt.xlabel('Modelo', fontsize=15, fontweight='bold')
plt.xticks(index + bar_width/2,
           ['Regressão Logística', 'MLP', 'KNN'], fontsize=12)
plt.yticks(fontsize=12)
plt.legend()

# Adicionando os valores numéricos acima das barras
for i, score in enumerate(recall_scores_pos):
    plt.text(i, score, str(round(score, 2)),
             ha='center', va='bottom', fontsize=10)
for i, score in enumerate(recall_scores_neg):
    plt.text(i + bar_width, score, str(round(score, 2)),
             ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()

# Gráficos de distribuição de sentimentos...

plt.figure(figsize=(8, 6))
ax = sns.countplot(x='nltk_pred', data=df, order=['POSITIVO', 'NEGATIVO'], palette={
    'POSITIVO': cor_positivo, 'NEGATIVO': cor_negativo})
plt.title('Distribuição dos sentimentos previstos pelo Vader',
          fontsize=15, fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Predições', fontsize=12, fontweight='bold')
plt.xticks(fontsize=12, fontweight='bold')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
                textcoords='offset points')
```

```
plt.show()

plt.figure(figsize=(8, 6))
ax = sns.countplot(x='textblob_pred', data=df, order=[
    'POSITIVO', 'NEGATIVO'], palette={'POSITIVO': cor_positivo, 'NEGATIVO':
cor_negativo})
plt.title('Distribuição dos sentimentos previstos pelo TextBlob',
          fontsize=15, fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Predições', fontsize=12, fontweight='bold')
plt.xticks(fontsize=12, fontweight='bold')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
                textcoords='offset points')
plt.show()

# Filtrar o DataFrame para incluir apenas as linhas do conjunto de teste
df_test = df.iloc[y_test.index]
print(df_test['target'].value_counts())

plt.figure(figsize=(8, 6))
ax = sns.countplot(x='log_reg_pred', data=df_test, order=['POSITIVO', 'NEGATIVO'],
palette={
    'POSITIVO': cor_positivo, 'NEGATIVO': cor_negativo})
plt.title(
    'Distribuição dos sentimentos previstos pela RL no conjunto de Testes', fontsize=15,
fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Predições', fontsize=12, fontweight='bold')
plt.xticks(fontsize=12, fontweight='bold')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
```

```
            textcoords='offset points')
plt.show()

plt.figure(figsize=(8, 6))
ax = sns.countplot(x='mlp_pred', data=df_test, order=['POSITIVO', 'NEGATIVO'], palette={
    'POSITIVO': cor_positivo, 'NEGATIVO': cor_negativo})
plt.title(
    'Distribuição dos sentimentos previstos pelo MLP no conjunto de Testes', fontsize=15,
fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Predições', fontsize=12, fontweight='bold')
plt.xticks(fontsize=12, fontweight='bold')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
            textcoords='offset points')
plt.show()

plt.figure(figsize=(8, 6))
ax = sns.countplot(x='knn_pred', data=df_test, order=['POSITIVO', 'NEGATIVO'], palette={
    'POSITIVO': cor_positivo, 'NEGATIVO': cor_negativo})
plt.title(
    'Distribuição dos sentimentos previstos pelo KNN no conjunto de Testes', fontsize=15,
fontweight='bold')
plt.xlabel('Polaridade', fontsize=12, fontweight='bold')
plt.ylabel('Total de Predições', fontsize=12, fontweight='bold')
plt.xticks(fontsize=12, fontweight='bold')

# Adicionando os números acima das barras
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center', fontsize=12, fontweight='bold', color='black', xytext=(0, 10),
            textcoords='offset points')
plt.show()
```