MINERAÇÃO DE TEXTOS NA REDE SOCIAL X:
Uma análise comparativa de algoritmos de classificação e técnicas de processamento de linguagem natural

Rodrigo Costa de Souza Melo[1*]
Sandro Corrêa Rocha Júnior[2**]
Maicon Vinícius Ribeiro[3***]

RESUMO

Este artigo apresenta uma análise comparativa de algoritmos de classificação e técnicas de processamento de linguagem natural (PLN) aplicadas à mineração de textos na rede social X (anteriormente conhecida como Twitter). A pesquisa foca em identificar a combinação mais eficaz de algoritmo de classificação e técnicas de PLN, utilizando métodos como *Bag of Words*, TF-IDF, lematização e stemização, junto com algoritmos *Logistic Regression*, *Naive Bayes* e *Support Vector Machine* (SVM). Através de uma metodologia rigorosa, o estudo compara as eficácias dessas técnicas e algoritmos, fornecendo insights práticos aplicáveis em diversas disciplinas, desde marketing digital até análise de sentimentos e opinião pública. Os resultados destacam a superioridade do modelo *Logistic Regression*, especialmente quando combinado com a técnica de *Bag of Words* e a abordagem de lematização, demonstrando ser uma prática robusta para classificação de sentimentos em *tweets* em múltiplos idiomas.

Palavras-chave: Mineração de Textos. Processamento de Linguagem Natural. Análise de Sentimentos. Algoritmos. *Tweets*. Eficácia. Comparação.

ABSTRACT

This article conducts a comparative analysis of classification algorithms and natural language processing (NLP) techniques for text mining on social network X (formerly Twitter). The study aims to identify the most effective combination of classification algorithm and NLP techniques, employing methods like Bag of Words, TF-IDF, lemmatization, and stemming, in conjunction with Logistic Regression, Naive Bayes, and Support Vector Machine (SVM) algorithms. Through rigorous methodology, the research compares the efficacies of these techniques and algorithms, offering practical insights applicable in various fields ranging from digital marketing to sentiment analysis and public opinion research. The findings highlight the superiority of the Logistic Regression model, particularly when combined with the Bag of Words technique and lemmatization approach, proving to be a robust strategy for sentiment classification in tweets across multiple languages.

---

[1]* Rede de Ensino Doctum – Unidade Caratinga – aluno.rodrigo.melo@doctum.edu.br – Graduando em Ciência da Computação
[2]** Rede de Ensino Doctum – Unidade Caratinga – aluno.sandro.junior@doctum.edu.br – Graduando em Ciência da Computação
[3]*** Rede de Ensino Doctum – Unidade Caratinga – maicon.ribeiro@doctum.edu.br – Professor Especialista em Ciência da Computação – Orientador(a) do trabalho

Keywords: Text Mining. Natural Language Processing. Sentiment Analysis. Algorithms. Tweets. Efficacy. Comparison.

1 – Introdução

A era digital transformou radicalmente a maneira como comunicamos e compartilhamos informações. Redes sociais, particularmente a rede social X (anteriormente conhecida como Twitter), tornaram-se plataformas fundamentais para expressar opiniões e sentimentos (DUMKA, 2022). Neste cenário, a mineração de textos, que envolve a extração de informações valiosas de grandes conjuntos de dados textuais, assume um papel crucial. O processamento de linguagem natural (PLN) é uma ferramenta essencial nesse processo, habilitando a compreensão e análise automatizada de dados textuais (LESKOVEC; RAJARAMAN; ULLMAN, 2020).

Diante dos inúmeros algoritmos de classificação e técnicas de PLN (Processamento de língua natural) disponíveis, realizar uma análise comparativa entre diferentes combinações destas ferramentas pode trazer contribuições significativas tanto para a comunidade científica quanto para o setor profissional, oferecendo insights práticos aplicáveis em várias disciplinas, desde o marketing digital até a análise de sentimentos e opinião pública. Nesse sentido, o objetivo principal deste trabalho é identificar a combinação mais eficaz de algoritmo de classificação e técnicas de PLN para a mineração de textos na rede social X.

Para isso, foram aplicadas e comparadas técnicas como *Bag of Words* e TF-IDF (*Term Frequency-Inverse Document Frequency*), fundamentais para a representação de textos em um formato que pode ser efetivamente processado por algoritmos de aprendizado de máquina (THANAKI, 2017), juntamente com métodos de normalização de texto, lematização e stemização, essenciais para reduzir a complexidade linguística dos dados (THANAKI, 2017).

No que se refere a algoritmos de classificação, este estudo comparou a eficácia do *Logistic Regression*, *Naive Bayes* e *Support Vector Machine* (SVM). A escolha destes algoritmos foi fundamentada em uma revisão bibliográfica, que levou em conta uma série de aspectos. Cada um desses algoritmos possui características únicas que podem influenciar os resultados da análise de dados textuais em redes sociais (MÜLLER, 2017).

Através de uma metodologia rigorosa, espera-se que os resultados deste estudo possam contribuir significativamente para o campo da mineração de textos, oferecendo uma compreensão mais profunda acerca da eficácia de diferentes algoritmos de classificação e técnicas de PLN. Assim, este trabalho visa não apenas avançar no conhecimento teórico existente, mas também servir como um guia prático para profissionais e pesquisadores interessados na aplicação eficiente de técnicas de mineração de textos na rede social X.

2 – Referencial Teórico

Nesta seção, serão abordados os conceitos fundamentais para a compreensão deste trabalho.

2.1 Mineração de Textos

Mineração de textos, ou *text mining* em inglês, consiste em extrair as características mais relevantes em uma quantidade de texto não estruturada (WEISS, 2010, apud GIROTO, 2020). Envolve a aplicação de algoritmos computacionais que processam textos com o objetivo de identificar informações úteis e implícitas, contidas nos dados armazenados em formato não estruturado.

A internet tornou-se o maior repositório de informações já reunido pela humanidade. De acordo com Amaral (2016), cerca de 80% desses dados estão em formato não estruturado, ou seja, não possuem um modelo de dados predefinidos ou organizados, no qual uma parte significativa são textos.

Nesse processo de desenvolvimento, envolvendo a extração de textos na internet, a informação tornou-se um recurso fundamental para as organizações, sendo considerada uma fonte de sucesso ou de fracasso, por trazer consigo um grande desafio: saber como lidar com o grande volume de informações produzidas nas organizações e redes sociais (CHISTOL, 2020).

Por mais que a organização inteligente dessas coleções textuais seja bastante codificada e complicada de se obter, elas não podem deixar de ser realizadas. Os benefícios da mineração de textos podem se estender a qualquer domínio que os utilizem, nesse contexto, a mineração de textos permite a transformação desse grande volume de dados textuais não estruturados em

conhecimento útil, muitas vezes inovador para as organizações (SANTOS et al., 2014), (CAVALCANTI, 2020). Nesse mesmo sentido, Dias (2021) afirma:

> A evolução tecnológica fez da Internet o principal agente de mudança econômica e uma fonte inesgotável de dados – que é a unidade básica da informação. Esses dados são coletados, armazenados, analisados e transformados em informações pelas plataformas online. O crescimento do volume de coleta e análise de dados veio acompanhado do crescente uso de algoritmos para possibilitar o tratamento de dados e a tomada de decisões pela plataforma online (DIAS, 2021, p. 3).

Pezzini (2016) complementa:

> A mineração de textos é uma extensão da mineração de dados, e pode ser definida como um processo de extração de informações desconhecidas e úteis de documentos textuais escritos em linguagem natural. Como a maioria das informações são armazenadas em forma de texto, a mineração de textos possui alto valor comercial, e pode ser aplicada em áreas como medicina e atendimento ao cliente (PEZZINI, 2016, p.58).

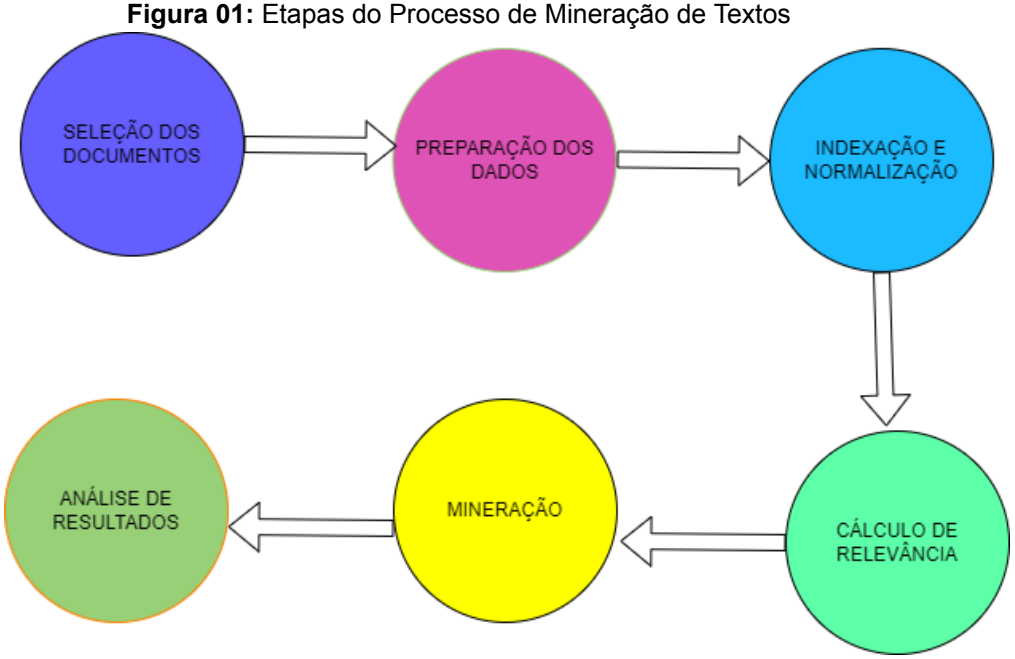
Morais e Ambrósio (2007) reforça:

> A Mineração de textos (*text mining*) é um Processo de Descoberta de Conhecimento, que utiliza técnicas de análise e extração de dados a partir de textos, frases ou apenas palavras. Envolve a aplicação de algoritmos computacionais que processam textos e identificam informações úteis e implícitas, que normalmente não poderiam ser recuperadas utilizando métodos tradicionais de consulta, pois a informação contida nestes textos não pode ser obtida de forma direta, uma vez que, em geral, estão armazenadas em formato não estruturado (Morais e Ambrósio, 2007, p.1).

Conforme as referências supracitadas, a mineração de texto é uma técnica amplamente empregada por pesquisadores, caracterizada pela exploração e identificação de termos relevantes em conjuntos de textos ou documentos. Esta prática assemelha-se à mineração de dados, mas difere significativamente no tipo de dados manipulados. A mineração de dados trata de informações estruturadas, oriundas de sistemas como bancos de dados e planilhas, enquanto a mineração de texto foca em dados não estruturados presentes em documentos, e-mails, redes sociais e na internet. Portanto, a distinção crucial entre as duas está na fonte dos padrões extraídos: na mineração de texto, estes provêm de textos em linguagem natural, ao invés de bases de dados estruturadas (HASSANI et al., 2020).

De acordo com Morais e Ambrósio (2007), o processo de mineração de texto envolve diversas etapas, tais como a seleção de documentos, a preparação dos dados, a indexação e normalização, o cálculo da relevância dos termos, a mineração propriamente dita e a análise dos resultados, conforme ilustrado na Figura 01.

Figura 01: Etapas do Processo de Mineração de Textos



Fonte: Adaptado de Morais Ambrósio (2007)

A Figura 1 apresenta um diagrama que ilustra as etapas principais do processo de mineração de textos. A primeira etapa consiste na seleção de textos, que são posteriormente analisados conforme a sua semântica ou estatística. Em seguida é feita a preparação dos dados, em que é selecionado o núcleo que melhor expressa o conteúdo do texto. A partir deste texto, é possível indexar e normalizar, o que facilita identificar a semelhança de significado entre suas palavras, tendo em vista as variações morfológicas. Na etapa de cálculo da relevância dos termos, é feita uma análise das palavras, com um significado mais relevante. A mineração de termos é feita com base no termo, podendo ser baseada no seu peso e posição sintática em relação ao texto. E, por fim, na etapa de pós-processamento ou análise de resultados, é possível ter acesso apenas ao padrão que lhe interessa.

2.2 Rede Social X

A rede social X, chamada anteriormente de Twitter, é tanto um serviço de *microblogging*, quanto uma rede social de informações composta por mensagens

curtas (incluindo fotos, vídeos e links) provenientes do mundo todo (X, 2023), com uma das maiores bases de usuários pelo mundo. Conforme o relatório publicado em parceria entre *We Are Social e Hootsuite*, em 2022, a rede social X ocupou a 15ª posição entre as redes sociais mais utilizadas no mundo, enquanto no Brasil, a rede ficou na 9ª posição (WE ARE SOCIAL, 2022).

ADWAN et al. (2020) afirmam que o modelo de interação da rede social X induz os usuários a compartilhar e expressar, de forma contínua, suas opiniões e sentimentos de forma rápida e objetiva, postando novas mensagens chamadas de *tweets* ou compartilhando mensagens postadas por outros usuários através de *retweets*. Em ambos os casos, o conteúdo de cada mensagem é limitado a 280 caracteres.

Em 2023 a rede social X já possuía cerca de 564 milhões de usuários ativos (STATISTA, 2023) que geram cerca de 350.000 *tweets* sendo enviados por minuto (BLOGGERSPASSION, 2023). Segundo Oliveira e Mergen (2018), o serviço possui algumas características peculiares que facilitam a difusão de conteúdo na rede. A seguir, estão descritas algumas das seguintes:

- Menções: No X, um usuário pode se referir a outro colocando "@" antes do nome de usuário desejado no *tweet*. Este recurso, denominado menção, tem como objetivo atrair a atenção do usuário mencionado para a mensagem específica e fomentar interações na plataforma (X, 2023).
- *Retweet*: Quando um *tweet* de outro usuário é particularmente atraente, há uma opção para compartilhá-lo, conhecida como *retweet*. O *retweet* permite que o *tweet* original seja distribuído para os seguidores do usuário que está compartilhando, mantendo a autoria original (X, 2023).
- *Hashtags*: Para destacar um tópico específico em um *tweet*, o usuário pode adicionar uma *hashtag*, o qual é um termo precedido por um símbolo "#". Estas etiquetas, ou *hashtags*, possibilitam categorizar o conteúdo do *tweet*. Cada *hashtag* torna-se um link que, quando clicado, direciona o usuário para uma busca mostrando os *tweets* mais recentes contendo essa mesma *hashtag* (X, 2023).
- Assuntos do Momento: No início da página de cada usuário, é mostrada uma lista com os 10 tópicos mais discutidos, que podem ser globais ou regionais,

de acordo com a preferência do usuário. Esses tópicos, chamados de “Tópicos em Alta”, podem ser representados por *hashtags* ou frases em maiúsculas. Cada tópico listado leva a uma busca que exibe os *tweets* mais recentes que mencionam esse tópico específico (X, 2023).

2.3 Processamento de Linguagem Natural

Santos et al. (2022) definem linguagem natural como aquela que é utilizada em comunicações no dia a dia dos seres humanos. Processamento de linguagem natural (PLN) é uma área da computação que tem como objetivo o estudo da linguagem natural humana e que reúne diversas técnicas que permitem a análise, compreensão e geração de textos pelo computador (CHOWDHARY, 2020).

O pré-processamento de textos é uma etapa importante no processamento de linguagem natural, ela consiste em uma série de técnicas que são aplicadas para otimizar a seleção de termos que serão utilizados na aprendizagem. A seguir, serão descritas algumas técnicas de pré-processamento que colaboram para atingir esse objetivo.

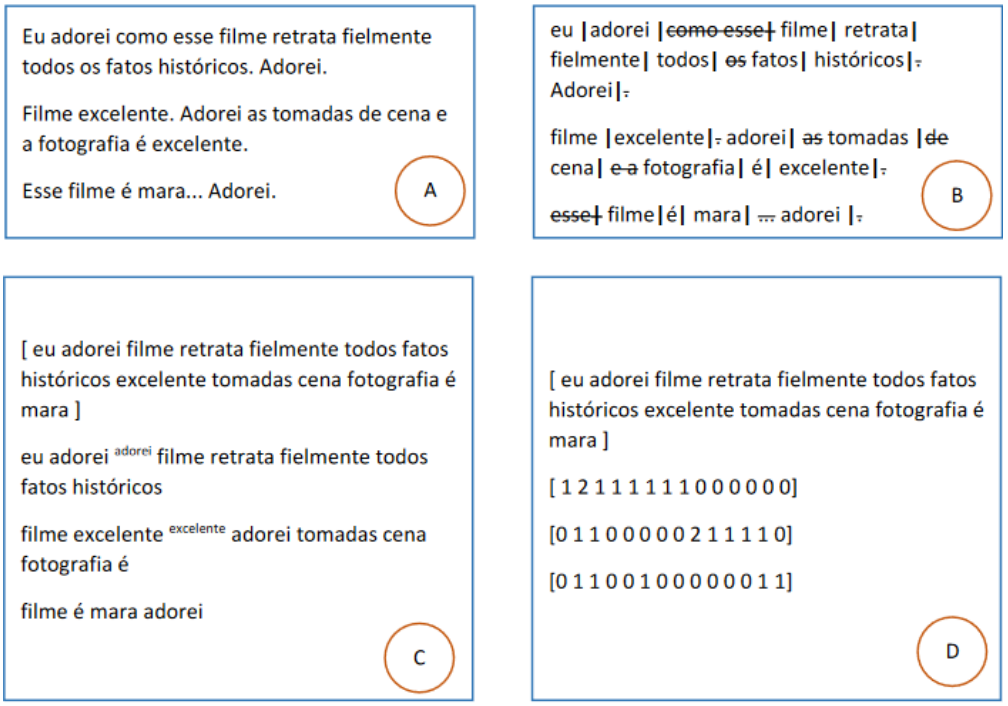
- Normalização: é a fase onde, após a coleta de dados, se normatiza e padroniza as *strings* (cadeias de caracteres) com o intuito de diminuir *tokens* únicos e assim melhorar o desempenho, custo computacional, facilitar análises descritivas e melhorar a acurácia de modelos preditivos (CHANDAK; MANISHA, 2019).
- Tokenização: também conhecida como segmentação de palavras, quebra a sequência de caracteres em um texto localizando o limite de cada palavra, ou seja, os pontos onde uma palavra termina e outra começa. Para fins de linguística computacional, as palavras assim identificadas são frequentemente chamadas de *tokens* (PALMER, 2010 apud LEANDRO et al., 2020).
- Remoção de *stopwords*: consiste em remover palavras sem significado, palavras vazias, que costumam ser as palavras mais comuns do idioma estudado. Podendo ser realizado durante ou depois da etapa de coleta, esse tratamento tem o intuito de melhorar o desempenho, a qualidade da busca e a análise feita nos textos coletados (CHOWDHARY, 2020).

- Lematização: A lematização é utilizada de diferentes formas, as quais dependem da tarefa a ser realizada pelo sistema de processamento de linguagem natural. Por exemplo, usando o verbo ENTREGAR: ENTREGAR + {3ª pessoa do singular, presente}. Dessa forma, todas as palavras são formadas a partir do lemma, retirando o sufixo de conjugação (HIPPISLEY, 2010 apud LEANDRO et al., 2020).
- Stemização: consiste em uma normalização linguística, na qual as formas variantes de um termo são reduzidas a uma forma comum denominada *stem* (que significa tronco, base), isto é, um exemplo: “no caso da palavra “aprendendo”, a stemização da palavra gera como resultado “aprend””. A aplicação de algoritmos de *stemming* consiste na remoção de prefixos ou sufixos de um termo, ou mesmo na transformação de um verbo para sua forma no infinitivo. Desse modo, algoritmos de *stemming* podem ser utilizados para reduzir a dimensão dos dados (MATSUBARA et al., 2003 apud FERREIRA; HONDA, 2019).

Outra tarefa em PLN é a extração de características. Ela consiste em transformar um texto em um vetor de características, que são os termos que serão utilizados para a classificação.

Uma técnica bastante usada na PLN é o *Bag of Words* (BoW), devido a sua capacidade de extrair recursos de documentos de textos, é uma das técnicas mais simples e populares desse tipo, ela consiste em transformar um texto em um vetor de características, onde cada termo é uma dimensão do vetor (MARTINS, 2020). O valor de cada dimensão é a frequência do termo no texto. A Figura 02 ilustra as etapas para a transformação do corpus em um vetor de frequência das palavras mais comuns.

Figura 02: Construção dos atributos para o método *Bag of Words*



Fonte: Adaptado de Bonadia e Barreto (2019)

Na Figura 2, por exemplo, há uma avaliação de um filme A. Cada avaliação, demonstrada em A, é um parágrafo e passa pelo processo de tokenização com a retirada das *stopwords*, fazendo com que as avaliações se transformem em um conjunto de palavras soltas B. Cada palavra que aparece no corpus é associada uma posição em um vetor C, que é preenchida com a quantidade de vezes que aparece na avaliação D (BONADIA; BARRETO, 2019).

Para capturar o contexto das palavras, é possível utilizar técnicas de representação de palavras, como *Term Frequency - Inverse Document Frequency* (TF-IDF) (CHRISTIAN; AGUS; SUHARTONO, 2016). Essa técnica consiste em calcular a frequência de um termo no texto e, ao mesmo tempo, a frequência do termo em todos os textos. O valor de cada dimensão é a multiplicação da frequência do termo no texto pelo inverso da frequência do termo em todos os textos. Segue como exemplo a Figura 03.

Figura 03: Conceito de (TF-IDF)

Fonte: Adaptado de Terenteva (2019)

Conforme ilustrado na Figura 3, a técnica de (TF-IDF), mostra quantas vezes uma palavra pode ser encontrada nos textos, reduzindo o peso de palavras com pouca importância. Assim, baseado na frequência e total de artigos, os termos com maior número de repetições têm menos peso e os que possuem um menor número de repetições têm um ganho de peso maior (TERENTEVA, 2019).

2.4 Aprendizado de Máquina

Samuel (2000) e NG (2023), definem aprendizado de máquina como “o estudo de algoritmos computacionais que podem melhorar seu desempenho em uma tarefa específica sem serem explicitamente programados para isso”.

Outros autores definem o aprendizado de máquina de acordo com “a experiência E, em relação a um tipo de tarefa T e uma medida de desempenho P, se o seu desempenho na execução da tarefa T for melhorado com a experiência” (MITCHELL, 1997, apud VINÍCIUS; CRISTINA, 2021).

Um exemplo de aprendizado pode ser dado por: suponhamos que um programa preveja o acumulado de precipitação na próxima hora a partir de dados anteriores. Nesse caso, a tarefa T seria estimar o acumulado de precipitação na próxima hora, a medida de desempenho P poderia ser alguma métrica de erro, como a diferença entre o valor previsto e o observado, já a experiência E seria as várias tentativas de realizar a previsão. O programa aprende à medida que sua previsão se aproxima do valor observado durante suas experiências. A forma com que o

programa aprende, está associada a um conjunto de configurações previamente definidas, denominadas de hiperparâmetros.

2.4.1 Principais Abordagens em Aprendizado de Máquina

Há três abordagens principais em aprendizado de máquina, a saber: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço (DRIDI, 2021).

O aprendizado supervisionado é o processo de treinamento de um modelo de aprendizado de máquina com dados de entrada para os quais as respostas corretas são conhecidas. O modelo é treinado com um conjunto de exemplos de treinamento e ajusta os seus parâmetros de forma a minimizar o erro na previsão dos exemplos. Esse tipo de aprendizado é usado quando temos dados rotulados, e os exemplos típicos incluem a classificação e a regressão. Algoritmos comuns de aprendizado supervisionado incluem árvores de decisão, Naive Bayes, *Support Vector Machine* (SVM) e Redes Neurais (GOODFELLOW; BENGIO; COURVILLE, 2016).

O aprendizado não supervisionado ocorre quando o modelo é treinado com dados de entrada para os quais as respostas corretas não são conhecidas. O modelo busca estruturas ou padrões ocultos nos dados. Isso é frequentemente usado quando temos muitos dados, mas não sabemos o que estamos procurando. Exemplos comuns incluem *clustering*, detecção de anomalias e regras de associação. Algoritmos típicos incluem *K-means*, *DBSCAN* e *autoencoders* (GOODFELLOW; BENGIO; COURVILLE, 2016).

Por fim, o aprendizado por reforço é um tipo de aprendizado no qual o modelo, ou “agente”, aprende a tomar decisões ao interagir com um ambiente. O agente recebe recompensas ou punições (*feedback*) com base na qualidade de suas ações e visa maximizar sua recompensa total ao longo do tempo. Essa técnica é frequentemente usada em situações onde o modelo precisa aprender a controlar um sistema, como jogar um jogo ou dirigir um carro. O algoritmo Q-Learning é um exemplo popular de aprendizado por reforço (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.4.2 Classificação em Aprendizado Supervisionado

A classificação é uma das principais tarefas do aprendizado supervisionado, e envolve a predição de uma variável alvo categórica a partir de um conjunto de variáveis de entrada ou características (MUHAMMAD; YAN, 2015, apud DRIDI, 2021). Essencialmente, o objetivo da classificação é construir um modelo que possa atribuir uma classe ou categoria a um novo exemplo com base em seu conhecimento de exemplos anteriores. Comumente, a classificação é empregada em problemas binários, isto é, quando há duas possíveis categorias ou classes, mas também pode ser aplicada em problemas com mais de uma classe ou categoria.

Um exemplo de classificação pode ser dado por: categorizamos cada registro de um conjunto de dados contendo as informações sobre os colaboradores de uma empresa: Perfil Técnico, Perfil Negocial e Perfil Gerencial. O modelo analisa os registros e então é capaz de dizer em qual categoria um novo colaborador se encaixa.

Existem vários algoritmos de classificação em aprendizado supervisionado, cada um com seus próprios pontos fortes e fracos. A árvore de decisão, por exemplo, é um algoritmo de classificação que aprende a classificar instâncias baseando-se em uma série de decisões lógicas, formando uma estrutura de árvore. Este tipo de algoritmo é especialmente útil para conjuntos de dados com características categóricas e é facilmente interpretável, uma vez que o modelo resultante pode ser visualizado e compreendido como uma série de regras de decisão (MURPHY, 2012).

Outro algoritmo comum é o *Naive Bayes*, que é fundamentado no teorema de Bayes e assume que as características são independentes entre si, uma suposição que é “naive” (ingênua), mas que, na prática, muitas vezes resulta em modelos de classificação bastante eficazes (MURPHY, 2012).

As *Support Vector Machines* (SVM) são outro exemplo de algoritmo de classificação. Elas trabalham encontrando o hiperplano que maximiza a margem entre as classes no espaço de características. SVMs são particularmente úteis para problemas de classificação binária e podem ser adaptadas para problemas com mais de uma forma de classificação (MURPHY, 2012).

O modelo de regressão logística, segundo (SANTOS, 2013, apud REGINA, 2022) é um algoritmo de classificação frequentemente usado por muitos anos na área de estatística, mas que começou a ser utilizado na área de aprendizado de

máquina recentemente, devido à proximidade com o SVM. Com esse modelo, é possível analisar a relação entre uma variável dependente categórica e diversas variáveis independentes, estimando a probabilidade de ocorrência de um evento ajustando os dados a uma curva logística, usada para prever um resultado binário.

O Modelo de Regressão Logística Binário é um caso particular dos modelos lineares generalizados, em que o componente aleatório tem distribuição de Bernoulli e a função de ligação é o logit (KANASHIRO, 2022).

2.4.3 Avaliação de Modelos

A avaliação de modelos de aprendizado de máquina é fundamental para medir sua eficácia e desempenho. Diversas métricas e técnicas de avaliação são utilizadas com base nas características do problema e nas metas do modelo.

As métricas de avaliação comuns incluem acurácia, precisão, *recall* e *F1-score*, que fornecem uma medida da qualidade das previsões realizadas pelo modelo. Segundo Japkowicz e Shah (2011), essas métricas “permitem avaliar a capacidade do modelo em classificar corretamente as instâncias e lidar com possíveis desequilíbrios nas classes”.

A validação cruzada é uma técnica amplamente utilizada para estimar o desempenho do modelo em dados não vistos. De acordo com Hastie et al. (2009), a validação cruzada envolve a divisão do conjunto de dados em partes menores, chamadas de *folds*, e a realização de múltiplos treinamentos e testes em diferentes combinações de *folds*. Essa abordagem permite uma avaliação mais robusta do modelo e reduz o impacto de variações aleatórias nos resultados.

Outra consideração importante na avaliação de modelos de aprendizado de máquina é o problema de *overfitting* e *underfitting*. Bishop e Nasrabadi (2006) explicam que o *overfitting* ocorre quando o modelo se ajusta muito bem aos dados de treinamento, mas falha em generalizar para novos dados, resultando em baixo desempenho. Por outro lado, o *underfitting* ocorre quando o modelo não é capaz de capturar os padrões presentes nos dados de treinamento, também levando a um desempenho insatisfatório.

Além disso, a seleção adequada de métricas de avaliação e a escolha de técnicas de otimização de modelos são essenciais para o desenvolvimento de modelos de aprendizado de máquina eficazes. Hastie et al. (2009) ressaltam que a

avaliação rigorosa e a análise cuidadosa dos resultados são fundamentais para garantir a confiabilidade e a utilidade prática dos modelos construídos.

3 – Metodologia

Nesta seção, são apresentados os métodos e técnicas utilizados para o desenvolvimento deste trabalho. Isso inclui a descrição do ambiente de desenvolvimento, a coleta e pré-processamento dos dados, a definição dos algoritmos de classificação, o estabelecimento dos critérios de comparação, a implementação dos modelos e a avaliação em diferentes cenários. As principais etapas desse processo podem ser visualizadas na Figura 04.

Figura 04: Fluxograma do processo de classificação de *tweets*

Fonte: Autoria própria (2023)

A Figura 04 apresenta um fluxograma que ilustra as principais etapas do processo de classificação de *tweets* adotado neste trabalho. A primeira etapa consiste na coleta dos *tweets*, os quais são posteriormente pré-processados para remover informações irrelevantes e realizar a normalização dos dados textuais. Em seguida, são extraídas as características relevantes dos *tweets* por meio de técnicas de representação textual. A partir dessas características, são construídos e avaliados diferentes modelos de classificação utilizando algoritmos de aprendizado de máquina. Todo o procedimento descrito foi conduzido utilizando um *notebook* Python, cujos detalhes e código-fonte são disponibilizados no Apêndice A, facilitando a reprodução e a verificação dos resultados apresentados.

3.1 Ambiente de Desenvolvimento

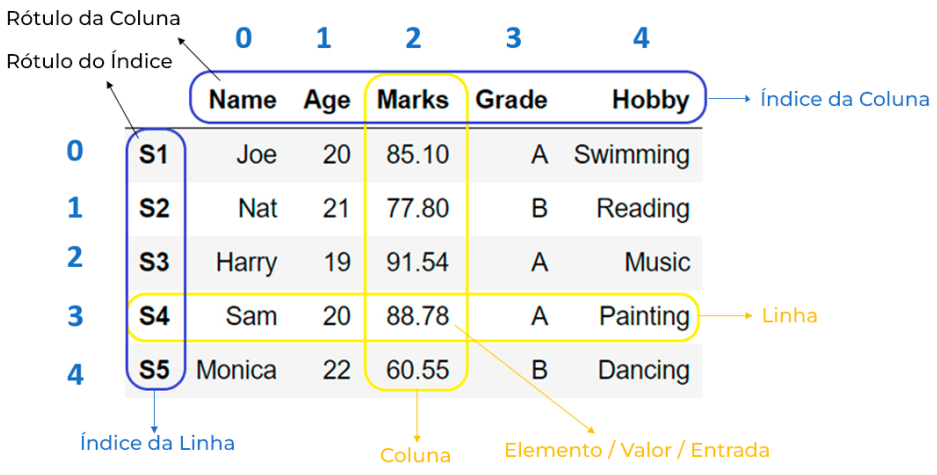
O ambiente de desenvolvimento utilizado para a realização deste trabalho foi o Jupyter Notebook, uma aplicação web que permite a criação e compartilhamento de documentos contendo código-fonte, equações, visualizações e texto explicativo

(JUPYTER, 2023). O Jupyter Notebook é amplamente utilizado na área de ciência de dados, ao permitir a execução de código-fonte em tempo real, facilitando a visualização dos resultados e a análise dos dados. Embora tenha sido desenvolvido em Python, é possível utilizar outras linguagens de programação, como R, Julia, Scala, entre outras (JUPYTER, 2023).

Para este trabalho, optou-se pelo uso da linguagem de programação Python 3.6.5, uma linguagem de alto nível, interpretada, orientada a objetos e de tipagem dinâmica (PYTHON, 2023). A escolha se deve à grande comunidade de desenvolvedores e às diversas bibliotecas disponíveis que facilitam o desenvolvimento de aplicações de aprendizado de máquina.

Uma das bibliotecas utilizadas foi a Pandas, uma biblioteca de código-fonte aberto que fornece estruturas de dados e ferramentas de análise de dados para a linguagem de programação Python (PANDAS, 2023). A biblioteca Pandas é amplamente utilizada em ciência de dados por suas diversas funções que permitem a manipulação de dados, como a leitura de arquivos, a criação de *DataFrames*, a remoção de linhas e colunas, a ordenação de dados, entre outras (PANDAS, 2023). A Figura 05 ilustra a estrutura de um *DataFrame*.

Figura 05: Estrutura de um *DataFrame*



Fonte: Adaptado de PYnative (2021)

O *DataFrame* é uma das principais estruturas de dados fornecidas pela biblioteca Pandas. É uma estrutura bidimensional que possui colunas com nomes e tipos de dados e linhas com índices e valores (PANDAS, 2023).

Outra biblioteca utilizada neste trabalho foi a NLTK, uma biblioteca de código-fonte aberto que oferece ferramentas para o processamento de linguagem natural (NLTK, 2023). Com diversas funções como remoção de *stopwords*, tokenização, stemização e lematização (BIRD et al., 2009), a biblioteca NLTK facilita o processamento do texto.

Além disso, foi utilizada a biblioteca Scikit-learn, uma biblioteca de código-fonte aberto que fornece ferramentas para o desenvolvimento de aplicações de aprendizado de máquina na linguagem Python. Essas ferramentas incluem funções para separação dos dados em conjuntos de treinamento e teste, normalização dos dados e criação e avaliação dos modelos (SCIKIT-LEARN, 2023). A Scikit-learn é amplamente utilizada em projetos científicos para construção e avaliação desses modelos.

3.2 Definição dos Algoritmos

Neste trabalho, foram utilizados três algoritmos de classificação: *Naive Bayes*, *Support Vector Machines* (SVM) e *Logistic Regression*. A escolha destes algoritmos foi fundamentada em uma revisão bibliográfica sobre técnicas de aprendizado de máquina voltadas para classificação textual, levando em consideração aspectos como popularidade, robustez, eficiência e capacidade de gerenciamento de grandes conjuntos de dados.

O primeiro algoritmo escolhido, *Naive Bayes*, é um método probabilístico que opera sob o pressuposto de independência condicional entre as variáveis preditoras (MÜLLER, 2017). Para o escopo deste trabalho, as variáveis são representadas pelas palavras contidas em um *tweet*. Este algoritmo busca estimar a probabilidade de pertencimento a uma classe específica, seja ela positiva ou negativa, por exemplo. Devido à sua notável simplicidade, velocidade de processamento e eficácia, o Naive Bayes tem sido vastamente empregado em tarefas de classificação textual, com destaque em aplicações de análise de sentimentos (BAGHERI e SARAEE, 2014).

Em sequência, temos o SVM, uma técnica de aprendizado supervisionado cuja essência reside em identificar um hiperplano que otimize a distância entre as classes. O SVM se destaca pela habilidade de processar dados de alta dimensionalidade e, ao mesmo tempo, manter uma elevada capacidade de

generalização, tornando-o uma escolha proeminente para problemas de classificação textual (MÜLLER, 2017).

Por último, foi adotado o *Logistic Regression*, um modelo estatístico utilizado para prever a probabilidade de ocorrência de um evento. Este modelo é valorizado por sua simplicidade e eficiência, sendo uma escolha comum em análises preditivas e em aplicações de processamento de linguagem natural, onde as relações entre as palavras e os sentimentos que expressam precisam ser compreendidas e quantificadas (HOSMER JR, LEMESHOW e STURDIVANT, 2013).

A opção por esses algoritmos, decorrente da revisão literária, visa primordialmente comparar o desempenho de cada um sob diferentes cenários. Tal abordagem possibilitará discernir qual método se mostra mais promissor para a classificação de *tweets*. Para a implementação dos mesmos, optou-se pela biblioteca Scikit-learn, reconhecida por proporcionar uma interface eficaz e de fácil manipulação para tais métodos (SCIKIT-LEARN, 2023).

3.3 Definição dos Critérios de Comparação

Para comparar o desempenho dos algoritmos de classificação e das técnicas de processamento de linguagem natural empregadas neste trabalho, estabeleceram-se critérios de comparação claros e objetivos. Estes critérios baseiam-se em métricas amplamente aceitas na literatura.

Uma métrica primordial é a precisão (*precision*). Esta é definida como a proporção de verdadeiros positivos (*tweets* corretamente classificados como positivos ou negativos) pelo somatório de verdadeiros positivos e falsos positivos (*tweets* erroneamente classificados como positivos ou negativos) (POWERS, 2020). Assim, a precisão oferece um olhar sobre a capacidade do modelo em minimizar classificações incorretas.

A revocação (*recall*) também foi utilizada. Esta métrica expressa a proporção de verdadeiros positivos em relação à soma dos verdadeiros positivos e falsos negativos (*tweets* positivos ou negativos que foram incorretamente classificados como negativos ou positivos, respectivamente) (POWERS, 2020). Tal métrica indica a habilidade do algoritmo em reconhecer acertadamente os exemplos positivos.

Outro indicador relevante é o F1-score, métrica que representa a média harmônica entre precisão e revocação. Esta métrica assume particular importância

em contextos em que há desequilíbrio entre as classes, conferindo um balanço entre as duas métricas anteriores (POWERS, 2020).

A comparação entre as diferentes técnicas de processamento de linguagem natural e extração de características tomou como base o desempenho dos modelos de classificação associados a cada técnica. Dessa forma, foi possível identificar quais técnicas potencializam positivamente o desempenho dos modelos.

Além disso, o tempo de execução estabeleceu um critério adicional, primordial ao avaliar a eficiência dos algoritmos e técnicas, sobretudo em aplicações que demandam respostas em tempo real, onde o tempo de processamento pode ser determinante (ATEFEH e KHREICH, 2015).

Estes critérios, ao serem aplicados em conjunto, fornecem uma avaliação holística e criteriosa do desempenho dos distintos métodos e técnicas empregados neste trabalho, viabilizando assim a definição do modelo mais eficaz para classificação de sentimentos em *tweets*.

3.4 Definição e Obtenção dos *Datasets*

Os conjuntos de dados utilizados neste trabalho foram obtidos através da plataforma Kaggle, uma comunidade online voltada para cientistas de dados que disponibiliza uma vasta gama de conjuntos de dados para aplicação em projetos de aprendizado de máquina e análise de dados (KAGGLE, 2023).

O primeiro *dataset* escolhido foi o "*Portuguese Tweets for Sentiment Analysis*", especificamente a versão com três classes, composta por 100.000 tweets em português, sem tema específico, categorizados em três classes: positivos, negativos ou neutros. Este conjunto de dados inclui informações como o identificador e o texto do *tweet*, a data de publicação, a classificação do sentimento expresso e a consulta utilizada para sua coleta. Os dados foram coletados entre agosto e outubro de 2018, e a categorização foi realizada por meio de um processo manual de anotação por avaliadores humanos (KAGGLE, 2023).

Já o segundo *dataset*, "*Coronavirus tweets NLP - Text Classification*", abrange 55.000 *tweets* em inglês, coletados em março de 2020 e relacionados à pandemia do Coronavírus. Estes são categorizados em cinco classes: extremamente positivos, positivos, neutros, negativos e extremamente negativos. Além de incluir o texto do *tweet*, a data de publicação, e a classificação do mesmo, este *dataset* também inclui

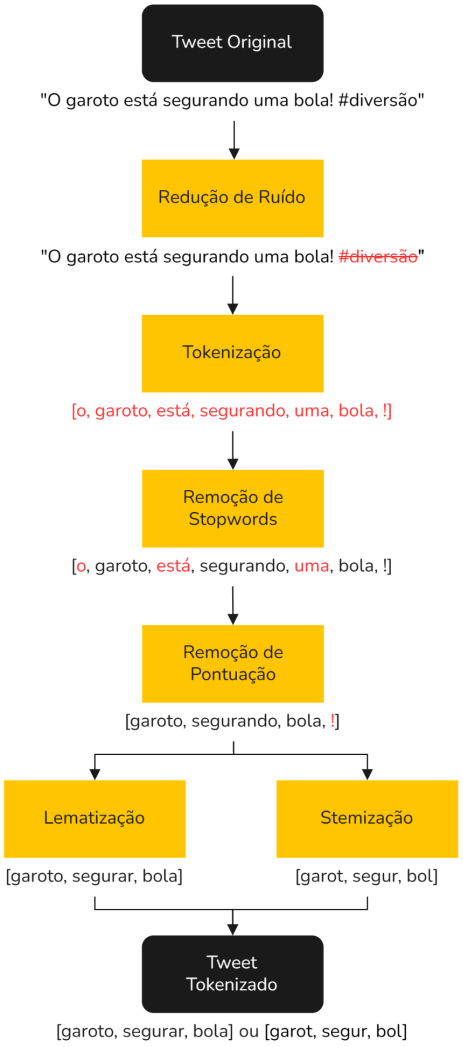
a localização geográfica de cada *tweet*. A categorização desses *tweets* também foi efetuada manualmente por avaliadores humanos (KAGGLE, 2023).

As diferenças intrínsecas entre os *datasets*, particularmente no que se refere à quantidade de *tweets* e à variedade de categorias de sentimentos, apresentam vantagens consideráveis para a pesquisa. Esta diversidade nos *datasets* proporciona um terreno fértil para uma análise comparativa mais robusta, analisando os algoritmos e técnicas empregados sob condições variadas. Tal abordagem permite avaliar a adaptabilidade e eficácia dos mesmos em cenários distintos, oferecendo *insights* valiosos sobre seu desempenho geral e especificidades em diferentes contextos.

3.5 Pré-processamento

O pré-processamento de dados textuais é uma etapa crucial para garantir a qualidade e relevância das informações. No contexto deste trabalho, os *tweets* foram submetidos a uma série de transformações para limpar, normalizar, tokenizar e, posteriormente, extrair características relevantes para o treinamento dos modelos.

Figura 06: Etapas do pré-processamento de um *tweet*



Fonte: Elaborado pelos autores (2023)

A Figura 06 ilustra a transformação de um *tweet* durante o processo de pré-processamento. Inicialmente, a etapa de redução de ruído é aplicada, onde os *tweets* são submetidos a uma limpeza profunda. Durante essa fase, elementos como URLs, menções a usuários e *hashtags* são eliminados utilizando expressões regulares (*regex*). Posteriormente, ocorre a *tokenização* dos *tweets*, realizada pela função `TweetTokenizer` da biblioteca NLTK. Esta etapa fragmenta o conteúdo em unidades menores, comumente palavras, facilitando sua análise posterior.

As *stopwords*, termos comuns que geralmente não agregam significado relevante ao contexto, são removidas para refinar a representação textual. A seleção das *stopwords* é feita com base na linguagem do *dataset*, utilizando recursos da

biblioteca NLTK. Além disso, pontuações, que não contribuem para a classificação, também são descartadas.

Por fim, conforme o contexto específico de cada cenário analisado, foram aplicadas técnicas de redução de palavras, optando-se entre lematização ou stemização. Estas técnicas buscam unificar diversas formas de uma mesma palavra em uma representação singular, com a lematização empregando dicionários e análise morfológica, e a stemização focando na simplificação das palavras à sua forma base, ou raiz.

Esse conjunto de processos foi eficientemente integrado em uma função singular, denominada `_tokenizer`. Essa função é encarregada de processar o texto original do *tweet*, realizando sua *tokenização* e pré-processamento, de modo a prepará-lo para ser utilizado como entrada nos subsequentes algoritmos de representação textual.

3.6 Extração de Características

A etapa de extração de características consistiu na transformação dos *tweets*, já pré-processados, em vetores numéricos. Essa etapa crucial foi realizada empregando duas técnicas de representação textual distintas, selecionadas conforme o contexto específico de cada análise: *Bag of Words* (BoW) e *Term Frequency-Inverse Document Frequency* (TF-IDF). Enquanto o BoW representa os textos com base na contagem de palavras, o TF-IDF pondera essa contagem, levando em consideração a importância relativa da palavra no corpus.

Para implementar o BoW, foi utilizado o `CountVectorizer`, e para o TF-IDF, o `TfidfVectorizer`, ambos da biblioteca Scikit-learn. Estes métodos foram configurados para receber como parâmetro a função `_tokenizer`, criando assim vetorizadores que, além de converterem o texto em vetores numéricos, também realizam o processo de *tokenização*.

3.7 Implementação dos Modelos

A implementação dos modelos de classificação foi realizada utilizando a abordagem de pipelines, uma funcionalidade da biblioteca Scikit-learn. Pipelines são estruturas de alto nível que permitem a sequência lógica e eficiente de várias etapas de processamento e modelagem. Essa abordagem garante não apenas a

simplificação do processo de codificação, mas também a consistência e a reprodução dos resultados.

Em cada execução do *notebook*, um único pipeline foi construído, dedicado exclusivamente a um dos algoritmos de classificação selecionados: *Naive Bayes*, SVM ou *Logistic Regression*. Cada pipeline compreendeu as etapas de pré-processamento, extração de características e o próprio modelo de aprendizado de máquina. Essa metodologia permitiu uma avaliação uniforme e comparável entre os diferentes modelos e técnicas aplicadas.

No início do pipeline, era empregado um vetorizador específico - `CountVectorizer` para a técnica BoW e `TfidfVectorizer` para TF-IDF. Esses vetorizadores, previamente configurados com a função `_tokenizer`, transformavam os *tweets* em vetores numéricos. Em seguida, os dados vetorizados eram processados pelo modelo de classificação selecionado - `MultinomialNB` para *Naive Bayes*, `SVC` para SVM ou `LogisticRegression` para *Logistic Regression* -, onde eram treinados e posteriormente avaliados.

3.8 Avaliação em Diferentes Cenários

Para assegurar uma avaliação abrangente da robustez e eficácia dos modelos de classificação de sentimentos, estes foram submetidos a testes em variados cenários. Esses cenários foram definidos por combinações específicas de técnicas de redução de palavras e representação textual, resultando em seis cenários distintos: sem redução de palavras com BoW (1), lematização com BoW (2), stemização com BoW (3), sem redução de palavras com TF-IDF (4), lematização com TF-IDF (5) e stemização com TF-IDF (6).

Utilizando o *notebook* desenvolvido (Apêndice A), os três modelos selecionados - *Naive Bayes*, SVM e *Logistic Regression* - foram treinados e avaliados em cada um dos seis cenários para cada conjunto de dados, somando dezoito execuções por *dataset*. Ao término de cada execução, as métricas de desempenho, coletadas através do método `classification_report` da Scikit-learn, foram armazenadas em arquivos CSV e catalogadas em um arquivo *Markdown*. Adicionalmente, o tempo de treinamento e de predição de cada modelo foi registrado, servindo como critério adicional.

Os dados coletados foram processados e organizados em um *DataFrame* (Apêndice C), seguido por uma análise exploratória detalhada dos dados. Para facilitar a visualização e interpretação das métricas, foram gerados gráficos ilustrativos. Os Gráficos 01 e 05 exibem as médias de precisão, *recall* e *F1-score* para cada modelo. Os Gráficos 02 e 06 apresentam as médias dessas métricas para cada técnica de redução de palavras, enquanto os Gráficos 03 e 07 ilustram as médias por técnica de representação textual. Por fim, os Gráficos 04 e 08 comparam o *F1-score* para cada combinação de modelo, técnica de redução de palavras e técnica de representação textual.

Após a compilação e análise dos dados, realizou-se uma avaliação comparativa do desempenho dos diferentes modelos, técnicas e suas combinações. Finalmente, baseando-se nessa análise e nas pesquisas bibliográficas realizadas, foram feitas inferências a fim de identificar a combinação mais eficaz dentre as avaliadas.

4 – Resultados

Esta seção apresenta os resultados da análise comparativa realizada para avaliar a eficácia de distintos algoritmos de classificação quando aliados a técnicas de redução de palavras e representação textual. A análise abrangeu a classificação de *tweets* em português e inglês. Uma discussão detalhada dos resultados e suas implicações é apresentada a seguir.

4.1 Classificação de *Tweets* em Português

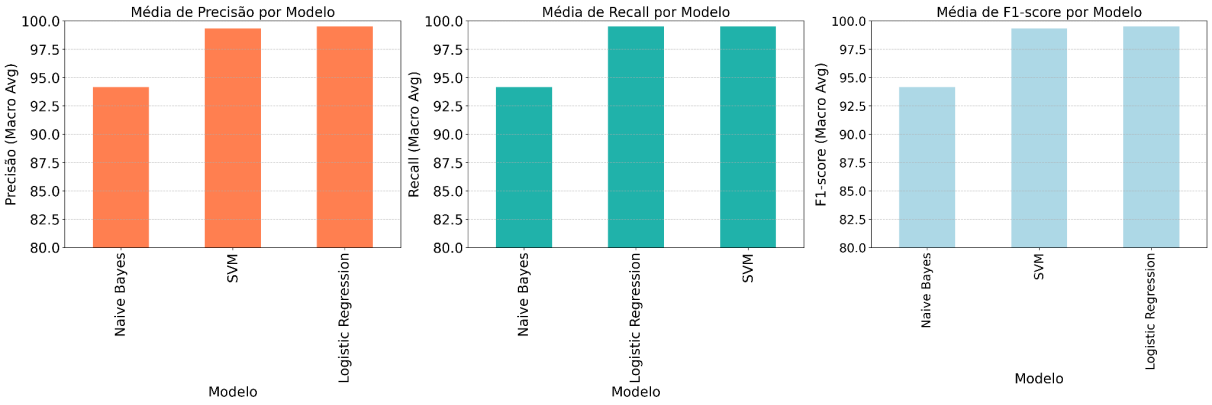
Baseando-se nos critérios e métricas estabelecidos na metodologia, bem como nos dados oriundos da aplicação dos algoritmos de classificação, em conjunto com técnicas de redução de palavras e representação textual, esta subseção visa elucidar e interpretar os resultados obtidos para o *dataset* “*Portuguese Tweets for Sentiment Analysis*”.

4.1.1 Avaliação dos Modelos para o *Dataset* em Português

A análise da média global das métricas (precisão, recall e *F1-score*) revela que os modelos *Logistic Regression* e SVM foram superiores, com médias de 99,5%

e 99,3%, respectivamente. Enquanto o *Naive Bayes* obteve uma média de 94,1%, que, apesar de ser inferior aos outros dois, ainda é um desempenho considerável.

Gráfico 01: Média de Precisão, *Recall* e *F1-score* por modelo



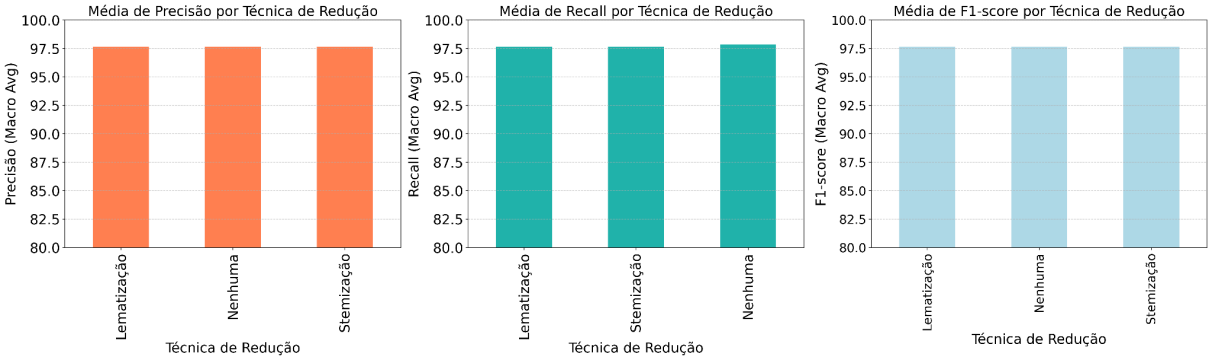
Fonte: Elaborado pelos autores (2023)

O Gráfico 01 ilustra claramente a superioridade dos modelos *Logistic Regression* e SVM em todas as métricas avaliadas quando comparados ao *Naive Bayes*. A diferença entre o *Logistic Regression* e o SVM é mínima, indicando uma performance quase equivalente entre eles. O *Logistic Regression* apresentou uma média uniforme de 99,5% em todas as métricas, enquanto o SVM alcançou 99,5% em *recall*, mas registrou 99,3% tanto em precisão quanto em *F1-score*, o *Naive Bayes*, por sua vez, obteve uma média constante de 94,1% em todas as métricas. Portanto, o *Logistic Regression* emerge como o modelo mais eficaz, apesar de uma margem estreita, e o *Naive Bayes*, embora menos eficiente, ainda mostra um bom desempenho.

4.1.2 Avaliação das Técnicas de Redução de Palavras para o *Dataset* em Português

Ao avaliar as técnicas de redução de palavras, constatou-se um desempenho uniforme em todas as métricas (precisão, *recall* e *F1-score*). A exceção é uma vantagem marginal de 0,2% observada quando nenhuma técnica de redução é aplicada.

Gráfico 02: Média de Precisão, *Recall* e *F1-score* por técnica de redução de palavras



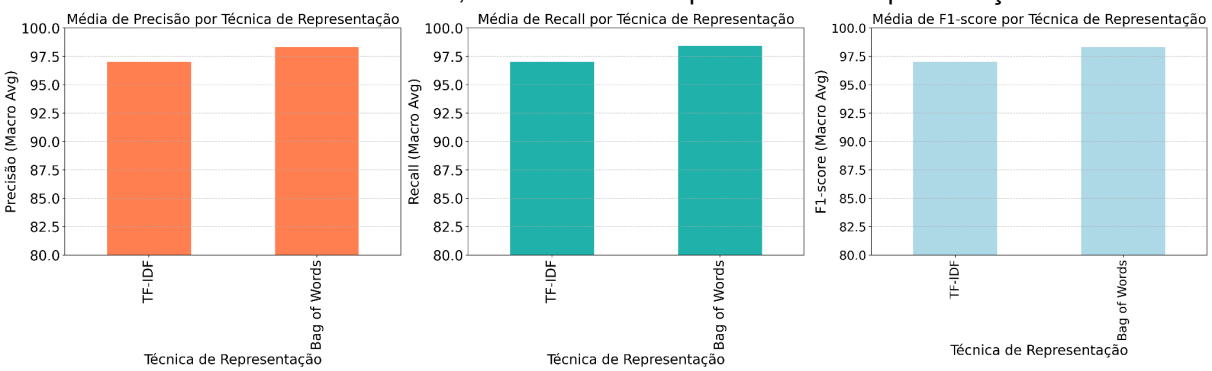
Fonte: Elaborado pelos autores (2023)

A consistência apresentada no Gráfico 02 reflete a similaridade de desempenho entre as técnicas avaliadas. As técnicas lematização e stemização alcançaram uma média uniforme de 97,6% em todas as métricas, enquanto a opção por não aplicar nenhuma técnica de redução resultou em 97,6% em precisão e *F1-score*, mas registrou 97,8% em *recall*. Essa regularidade pode ser influenciada por características específicas do *dataset* utilizado. Contudo, com base apenas nos dados disponíveis, não é possível determinar com precisão a razão desse comportamento.

4.1.3 Avaliação das Técnicas de Representação Textual para o *Dataset* em Português

Diferentemente das técnicas de redução de palavras, a representação textual mostrou variações significativas em desempenho. A técnica *Bag of Words* (BoW) se destacou, superando a *Term Frequency-Inverse Document Frequency* (TF-IDF) em todas as métricas avaliadas (precisão, *recall* e *F1-score*).

Gráfico 03: Média de Precisão, *Recall* e *F1-score* por técnica de representação textual



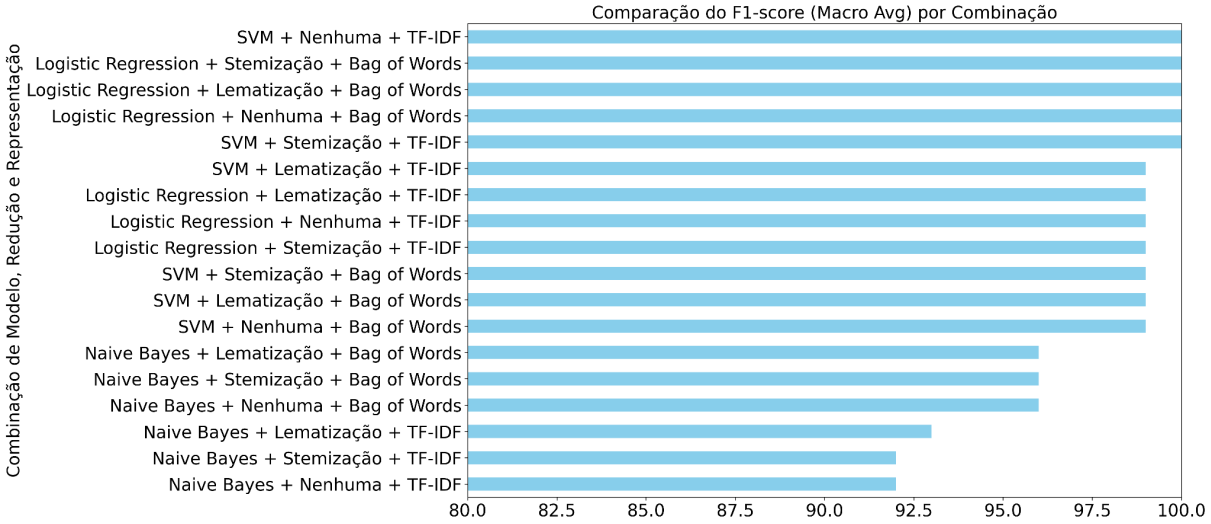
Fonte: Elaborado pelos autores (2023)

O Gráfico 03 reforça essa constatação, demonstrando que, apesar de sua simplicidade conceitual, a BoW apresentou uma vantagem média de 1,3% em todas as métricas quando comparada à TF-IDF. A BoW alcançou uma média de 98,3% em precisão e *F1-score*, e 98,4% em *recall*, enquanto a TF-IDF registrou uma média uniforme de 97,0% em todas as métricas. Assim, sob as condições analisadas, a BoW demonstra-se a técnica de representação textual mais eficiente.

4.1.4 Avaliação das Combinações para o *Dataset* em Português

No que diz respeito à combinação, conforme demonstrado no Gráfico 04, cinco combinações atingiram o desempenho máximo, enquanto duas apresentaram os menores resultados, baseando-se na média do *F1-score* macro. Para determinar a combinação mais eficaz, foram considerados dois critérios adicionais: tempo de treinamento e tempo de predição.

Gráfico 04: Comparação do F1-score (*Macro Avg*) por combinação



Fonte: Elaborado pelos autores (2023)

Nesse contexto, identificou-se que a combinação mais eficiente foi a que utilizou o modelo *Logistic Regression*, sem aplicação de técnicas de redução de palavras e empregando BoW para representação textual. Esta configuração não só alcançou um *F1-score* de 100%, mas também se destacou por sua rapidez, com apenas 10,3 segundos de treinamento e 1 segundo de predição. Por outro lado, a combinação que se mostrou menos eficiente utilizou o modelo *Naive Bayes*, combinado com stemização para redução de palavras e TF-IDF para representação

textual. Esta abordagem resultou em um *F1-score* de 92,0%, demandando 15,1 segundos para treinamento e 3,0 segundos para predição.

A análise do Gráfico 04 revela que, em linha com as descobertas das subseções anteriores, o modelo *Logistic Regression* apresentou desempenhos superiores, alcançando 100% de *F1-score* em três combinações distintas. Em contraste, as seis combinações com menor desempenho envolveram o uso do *Naive Bayes*.

Quanto às técnicas de representação textual, constata-se que, embora o BoW tenha geralmente apresentado os melhores resultados, as combinações com o modelo SVM tiveram um melhor desempenho quando associadas à técnica TF-IDF. Isso reforça a ideia de que a eficácia de uma técnica pode variar dependendo do modelo de classificação com o qual é combinada.

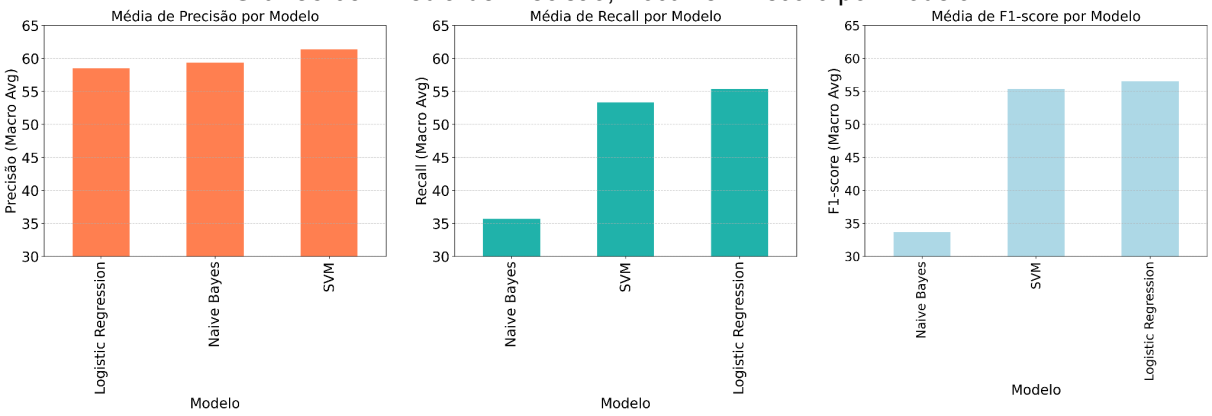
4.2 Classificação de *Tweets* em Inglês

Esta subseção tem como foco elucidar e interpretar os resultados obtidos para o *dataset* "*Coronavirus tweets NLP - Text Classification*", empregando os mesmos critérios, métricas, algoritmos e técnicas utilizados para o dataset em português. O objetivo é compreender como as características do dataset podem influenciar o desempenho dos modelos e técnicas aplicadas.

4.2.1 Avaliação dos Modelos para o *Dataset* em Inglês

A análise da média global das métricas (precisão, *recall* e *F1-score*), corrobora a tendência observada nas seções anteriores, destacando a performance superior dos modelos *Logistic Regression* e SVM, com médias de 56,7% e 56,6%, respectivamente. Enquanto o *Naive Bayes* obteve uma média de 42,8%, sendo um desempenho consideravelmente inferior aos outros dois.

Gráfico 05: Média de Precisão, *Recall* e *F1-score* por modelo



Fonte: Elaborado pelos autores (2023)

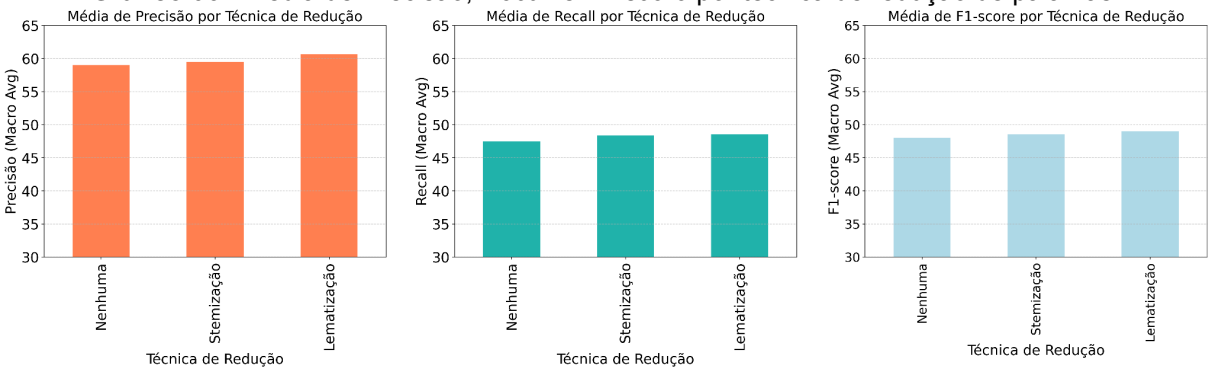
Ao analisar o Gráfico 05, nuances interessantes são revelados. Apesar da superioridade na média global, o modelo *Logistic Regression*, mesmo que por uma estreita margem, registrou a menor média de precisão, alcançando 58,5%, seguido pelo *Naive Bayes* e SVM, com médias de 59,3% e 61,3%, respectivamente. No que diz respeito ao *recall, Logistic Regression* e SVM demonstraram uma notável superioridade, com médias de 55,3% e 53,3% respectivamente, em contraste com o modesto 35,6% alcançado pelo *Naive Bayes*. Um padrão semelhante é observado no *F1-score*, onde, *Logistic Regression* se destaca com a melhor média de 56,5%, seguido de perto pelo SVM com 53,3%, e distanciando-se significativamente do *Naive Bayes*, que registrou 35,6%.

Esses resultados indicam que, embora o modelo *Logistic Regression* mantenha uma liderança geral, as diferenças nas métricas individuais sugerem uma complexidade na eficácia dos modelos que merece uma análise mais aprofundada para entender plenamente suas implicações.

4.2.2 Avaliação das Técnicas de Redução de Palavras para o *Dataset* em Inglês

A avaliação das técnicas de redução de palavras aplicadas ao *dataset* em inglês revelou um cenário contrastante em comparação com a uniformidade observada no *dataset* em português. As métricas apontam variações sutis, porém significativas, entre as diferentes técnicas. A lematização emergiu como a técnica mais eficaz, alcançando uma média de 52,7%. Em seguida, vem a stemização com uma média ligeiramente inferior de 52,1%, enquanto a abstenção de qualquer técnica de redução registrou uma média de 51,5%.

Gráfico 06: Média de Precisão, *Recall* e *F1-score* por técnica de redução de palavras



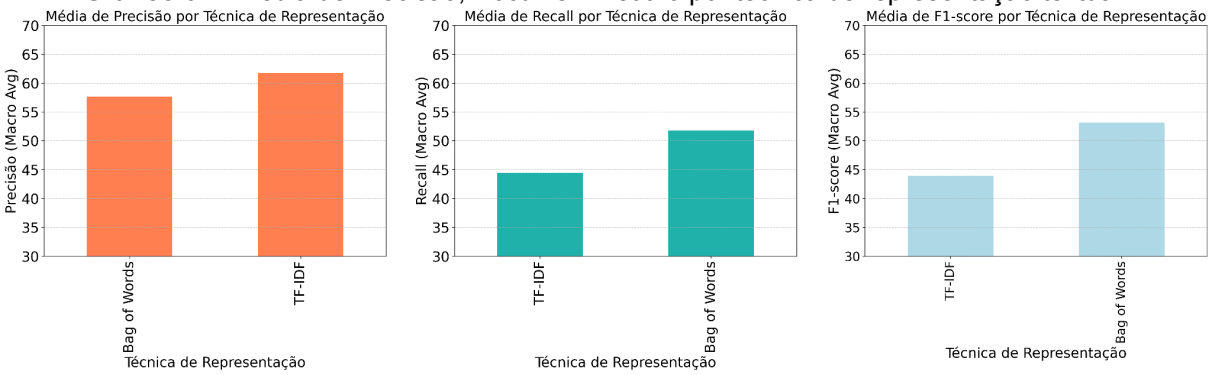
Fonte: Elaborado pelos autores (2023)

Conforme evidenciado pelo Gráfico 06, a implementação de técnicas de redução de palavras no *dataset* inglês demonstrou uma melhoria discreta no desempenho, divergindo dos resultados obtidos com o *dataset* em português, onde tais técnicas não apresentaram efeito significativo. A lematização mostrou-se mais eficiente em todas as métricas avaliadas, alcançando médias de 60,6% em precisão, 48,4% em *recall* e 49,0% em *F1-score*. Em sequência, a stemização registrou 59,5% em precisão, 48,3% em *recall* e 48,5% em *F1-score*. Por fim, a opção por não aplicar nenhuma técnica de redução resultou em 59,0% em precisão, 47,5% em *recall* e 48,0% em *F1-score*. Esta diferença pode estar atrelada a linguagem de cada *dataset*, uma vez que métodos distintos de redução foram empregados, considerando a especificidade de cada idioma. Entretanto, com base nos dados disponíveis, não é possível obter uma confirmação definitiva dessa hipótese.

4.2.3 Avaliação das Técnicas de Representação Textual para o *Dataset* em Inglês

Na avaliação das técnicas de representação textual para o *dataset* em inglês, observou-se a predominância do método BoW, que alcançou uma média de 54,1%, superando a técnica TF-IDF, que registrou 50,0%. Esses resultados estão alinhados com as observações feitas para o *dataset* em português, confirmando a superioridade da BoW sobre a TF-IDF nos contextos examinados.

Gráfico 07: Média de Precisão, *Recall* e *F1-score* por técnica de representação textual



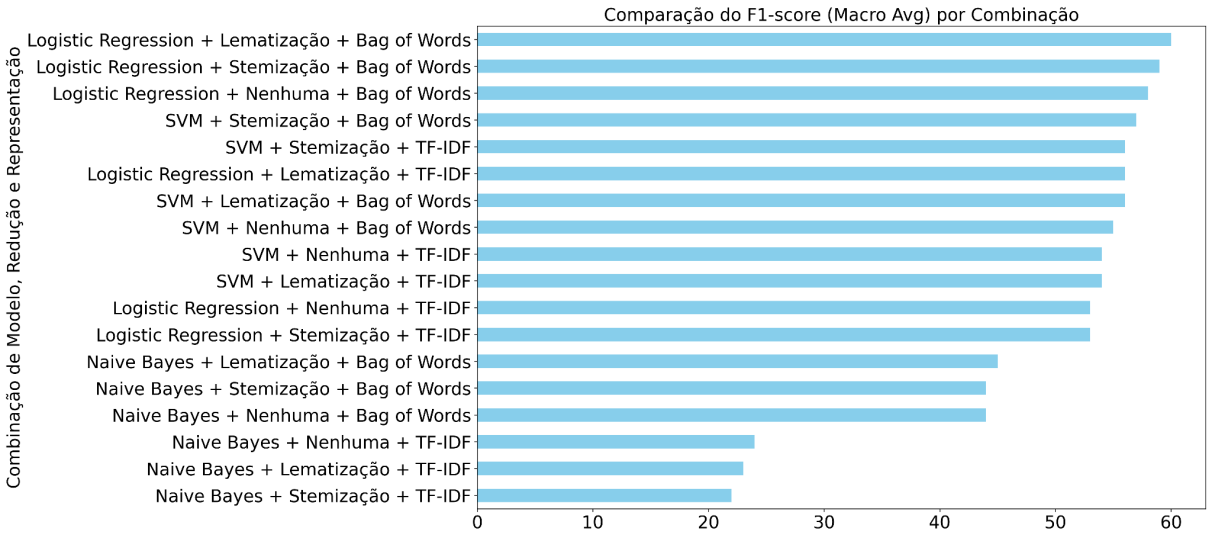
Fonte: Elaborado pelos autores (2023)

O Gráfico 07 destaca uma diferença interessante em comparação com os resultados do *dataset* em português. Embora a BoW tenha mantido uma performance geral superior, não foi a técnica dominante em todas as métricas. Especificamente em termos de precisão, a TF-IDF se sobressaiu, com uma média de 61,7%, enquanto a BoW obteve 57,6%. No entanto, em *recall*, a BoW retomou a liderança com 51,7%, superando os 44,4% da TF-IDF. Já no *F1-score*, a BoW reafirmou sua eficácia com uma média de 51,3%, contra 43,8% da TF-IDF. Assim, nos cenários analisados, a BoW demonstrou consistentemente um desempenho superior ao da TF-IDF, consolidando-se como a técnica de representação textual mais efetiva para os *datasets* em questão.

4.2.4 Avaliação das Combinações para o *Dataset* em Inglês

Na análise das combinações de modelos, técnicas de redução de palavras e representação textual para o *dataset* em inglês, observamos um panorama distinto. Conforme ilustrado no Gráfico 08, uma combinação destacou-se isoladamente com um desempenho superior aos demais, enquanto outra apresentou isoladamente o pior desempenho, baseando-se na média do *F1-score* macro. Neste contexto, optou-se por não considerar os critérios adicionais de eficiência temporal.

Gráfico 08: Comparação do F1-score (*Macro Avg*) por combinação



Fonte: Elaborado pelos autores (2023)

A combinação mais eficaz empregou o modelo Logistic Regression, complementado pela lematização na redução de palavras e pela técnica BoW na representação textual. Essa configuração atingiu um *F1-score* de 60,0%. Também vale ressaltar que essa mesma combinação também figurou entre as mais eficientes no *dataset* em português, onde alcançou um *F1-score* máximo de 100%. Por outro lado, a combinação menos eficiente utilizou o modelo *Naive Bayes* com stemização e TF-IDF, resultando em um *F1-score* de apenas 22,0%. Essa abordagem foi igualmente a menos eficaz no *dataset* em português, com um *F1-score* de 92,0%.

Esta análise reitera as tendências observadas nas subseções anteriores, destacando a superioridade do modelo *Logistic Regression*, que figurou nas três combinações de melhor desempenho. Em contraste, tal como nos resultados para o *dataset* em português, as seis combinações menos eficazes envolveram o modelo *Naive Bayes*. Notavelmente, ao contrário do observado com o *dataset* em português, onde o modelo SVM teve melhor desempenho quando associado à técnica TF-IDF, neste caso, todas as combinações obtiveram melhores resultados ao empregar a técnica BoW, independentemente do modelo utilizado.

4.3 Discussão dos Resultados dos *Datasets* em Português e Inglês

Nesta subseção, objetiva-se consolidar as descobertas mais significativas dos *datasets* em português e inglês para estabelecer uma compreensão mais clara das práticas mais eficientes em classificação de sentimentos em *tweets*.

4.3.1 Superioridade do Modelo *Logistic Regression*

Os resultados obtidos com os dois *datasets* indicam uma tendência clara: o modelo *Logistic Regression* demonstrou ser o mais eficiente em ambas as línguas. Este modelo se destacou não apenas em termos de métricas, mas também na consistência de desempenho quando combinado a diferentes técnicas de processamento de texto. A sua capacidade de adaptar-se a variáveis linguísticas e contextuais ressalta a sua superioridade entre os algoritmos analisados para a análise de sentimentos em *tweets*.

4.3.2 *Bag of Words* como Técnica de Representação Textual Preferencial

Outra constatação relevante é a eficácia da técnica *Bag of Words* (BoW) na representação textual. Em ambas as línguas e na maioria das combinações testadas, a BoW demonstrou superioridade sobre a TF-IDF, demonstrando melhores métricas e maior consistência. Quando aliada ao modelo *Logistic Regression*, a BoW se mostrou particularmente poderosa, proporcionando os melhores resultados em termos de precisão e eficácia.

4.3.3 Avaliação das Técnicas de Redução de Palavras

Em relação às técnicas de redução de palavras, os dados não forneceram evidências conclusivas para estabelecer a supremacia de uma técnica específica em ambos os *datasets*. Entretanto, ao analisar as combinações mais eficazes, a lematização demonstrou maior consistência. No *dataset* em português, uma combinação com lematização atingiu um *F1-score* perfeito de 100%. Já no *dataset* em inglês, a lematização se fez presente na combinação mais eficaz.

4.3.4 Efetividade das Combinações de Modelo e Técnicas

Com base nos resultados compilados, a combinação de Logistic Regression, lematização e BoW emerge como uma estratégia robusta para a classificação de

tweets. Esta combinação alcançou os melhores resultados em ambos os *datasets* analisados, estabelecendo-se como uma estratégia ótima para a tarefa de classificação de sentimentos em *tweets*. Esta configuração pode servir como uma recomendação prática para pesquisadores e profissionais que buscam eficácia e precisão na análise de sentimentos em múltiplos idiomas.

5 – Conclusão

Este trabalho representou um passo significativo na área da mineração de textos na rede social X, oferecendo uma visão abrangente sobre a eficácia de diferentes algoritmos de classificação e técnicas de processamento de linguagem natural. As descobertas deste estudo evidenciaram que a escolha das técnicas de redução de palavras, representação textual e algoritmos de classificação impactam significativamente a eficácia na mineração de textos. Especificamente, os resultados mostraram que o modelo *Logistic Regression*, empregado em conjunto com a técnica *Bag of Words* e a abordagem de lematização, sobressaiu significativamente na classificação de sentimentos em *tweets*, tanto em português quanto em inglês.

Os resultados alcançados destacaram o modelo *Logistic Regression* e a técnica de representação textual *Bag of Words*, que se sobressaíram em todas as métricas avaliadas. Por outro lado, as técnicas de redução de palavras demonstraram um desempenho mais uniforme, indicando uma importância relativamente menor destas técnicas nos cenários analisados. Além disso, a influência das características específicas de cada *dataset* foi marcantemente evidente nos resultados, reforçando a necessidade de uma seleção cuidadosa e adaptada de ferramentas para a mineração de textos, conforme as particularidades de cada conjunto de dados.

Adicionalmente, as descobertas deste trabalho sublinham a importância de uma análise cuidadosa das técnicas de pré-processamento em PLN, cruciais para a preparação de dados textuais para análise. A escolha correta dessas técnicas pode impactar significativamente a eficácia dos modelos de classificação.

Este artigo contribui não apenas para a academia, mas também para profissionais de marketing, cientistas de dados e desenvolvedores, fornecendo *insights* valiosos que podem ser aplicados em estratégias de marketing digital,

monitoramento de redes sociais e outras aplicações onde a compreensão do sentimento e da opinião pública é crucial.

Em futuros trabalhos, seria interessante explorar a aplicação dessas técnicas e algoritmos em outras plataformas de mídia social, bem como incorporar métodos de aprendizado profundo que têm ganhado destaque na área de PLN. Além disso, estender a pesquisa para incluir análises multilíngues mais amplas e maiores volumes de dados, poderia revelar *insights* adicionais sobre a universalidade e a adaptabilidade desses métodos em contextos variados.

Em resumo, este trabalho não apenas enriquece o entendimento atual de mineração de textos em redes sociais, mas também pavimenta o caminho para futuras investigações e aplicações práticas nesta área em rápida evolução.

Referências

ADWAN, Omar Y. et al. Twitter sentiment analysis approaches: A survey. International Journal of Emerging Technologies in Learning, [s. l.], v. 15, n. 15, p. 79–93, 2020. Disponível em: <https://doi.org/10.3991/ijet.v15i15.14467>. Acesso em 01 out. 2023.

AMARAL, Fernando. Introdução à Ciência de Dados: Mineração de Dados e Big Data. Rio de Janeiro: Alta Books, 2016. 320 p.

ATEFEH, Farzindar; KHREICH, Wael. A survey of techniques for event detection in twitter. Computational Intelligence, v. 31, n. 1, p. 132-164, 2015. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12017>. Acesso em: 16 jun. 2023.

BAGHERI, Ayoub; SARAEE, Mohamad. Persian Sentiment Analyzer: A Framework based on a Novel Feature Selection Method. arXiv (Cornell University), 27 dez. 2014. Disponível em: <https://arxiv.org/abs/1412.8079>. Acesso em: 01 nov. 2023.

BIRD, Steven; KLEIN, Ewan; LOPER, Edward. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009.

BISHOP, Christopher M.; NASRABADI, Nasser M. Pattern recognition and machine learning. Vol. 4, no. 4. New York: Springer, 2006.

BLOGGERSPASSION. 30+ Latest Twitter Statistics, Facts, and Trends for 2023. 2023. Disponível em: <https://bloggerspassion.com/twitter-statistics/>. Acesso em: 22 nov. 2023.

BONADIA, Graziella C.; BARRETO, Gilmar. Análise de Sentimentos em comentários na língua portuguesa: uma comparação de métodos. Disponível em: <https://d1wqtxts1xzle7.cloudfront.net/84411252/fulltext_file2-libre.pdf?1650306854=

&response-content-disposition=inline%3B+filename%3DAnalise_de_Sentimentos_em_comentarios_na.pdf&Expires=1687460089&Signature=I~1eTHTHXtW3gm-S9QKygjAxsq0UiQ~rVXdAMxIUVwR8Qyo9ZBJJSC52Bc1CeUbRxmaIxSML-PK3zOAp85v3ybAD1oa28YeOhswmAny5NlkUulmLOIIFjnEb4pjq1tOCXw3DHXuv0b7BZlTUlvAj3SdPc~3sX3bBT5~s9YzuwrhM0N24fv-Ut1Ohse1EWpqLfuS7VzdI7e3mHv86QScJTfXSn-EahBWUYVwligUZXXJhdGh5mYID6ciuJr2Es-WOYAXFtjeX0CunAtoxI780kiGV7vnlPptCJE4-wDu8LOFJwMb54cyEXcyK3~qLHjto34kKzlnJq4Ycy4k5zxWECQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>. Acesso em 20 jun. 2023.

CAVALCANTI, Marcelo. A Internet como ferramenta na produção e gestão do conhecimento organizacional. RECIMA21-Revista Científica Multidisciplinar-ISSN 2675-6218, [s. l.], v. 1, n. 2, p. 8–20, 2020. Acesso em: 11 set. 2023.

CHISTOL, Mihaela. A Comparative study of parametric versus non-parametric text classification algorithms. In: , 2020, Suceava, Romania. 15th International Conference on Development and Application Systems. Suceava, Romania: [s. n.], 2020. p. 208–213. Acesso em: 01 out. 2023.

CHOWDHARY, K. R. Fundamentals of artificial intelligence. New Delhi: Springer India, 2020.

CHRISTIAN, Hans; AGUS, Mikhael Pramodana; SUHARTONO, Derwin. Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). ComTech: Computer, Mathematics and Engineering Applications, v. 7, n. 4, p. 285-294, 2016. Disponível em: <https://doi.org/10.21512/comtech.v7i4.3746>. Acesso em: 20 jun. 2023.

DIAS, Júlia de Figueiredo Pinheiro. Big Data e direito da concorrência: a concorrência no mercado de dados pessoais à luz do RGPD. 2021. - Universidade de Lisboa, [s. l.], 2021. Acesso em: 11 set. 2023.

DRIDI, Salim. Supervised Learning - A Systematic Literature Review. 2021. Disponível em: <https://files.osf.io/v1/resources/tysr4/providers/osfstorage/624a442a7a7d9e04500608ce?action=download&direct&version=3>. Acesso em: 06 jun. 2023.

DUMKA, A. et al. A Novel Deep Learning Based Healthcare Model for COVID-19 Pandemic Stress Analysis. Computers, Materials & Continua, [s.l.], v. 70, n. 3, p. 5867–5882, 2022. Disponível em: <https://doi.org/10.32604/cmc.2022.024698>. Acesso em: 01 nov. 2023.

FERREIRA, Hugo Honda. Processamento de linguagem natural e classificação de textos em sistemas modulares. 2019. x, 61 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação)—Universidade de Brasília, Brasília, 2019. Disponível em: <https://bdm.unb.br/handle/10483/25114>. Acesso em: 01 nov. 2023.

GIROTO. Thiago Milani. Aprendizado de máquina para detecção de spam: um estudo comparativo de algoritmos de mineração de texto e classificadores. 2020. Disponível em: <http://hdl.handle.net/11449/204207>. Acesso em: 20 ago. 2023.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep learning. MIT press, 2016. Disponível em: <https://www.deeplearningbook.org/>. Acesso em: 06 jun. 2023.

HASSANI, Hossein et al. Text mining in big data analytics. Big Data and Cognitive Computing, [s. l.], v. 4, n. 1, p. 1–34, 2020. Disponível em: <https://doi.org/10.3390/bdcc4010001>. Acesso em: 21 nov. 2023.

HASTIE, Trevor et al. The elements of statistical learning: data mining, inference, and prediction. Vol. 2. New York: Springer, 2009.

HOSMER JR, David W.; LEMESHOW, Stanley; STURDIVANT, Rodney X. Applied logistic regression. Vol. 398. John Wiley & Sons, 2013.

JAPKOWICZ, Nathalie; SHAH, Mohak. Evaluating learning algorithms: a classification perspective. Cambridge University Press, 2011.

JUPYTER. Project Jupyter. Disponível em: <https://jupyter.org/>. Acesso em: 09 de jun. 2023.

KAGGLE. Kaggle: Your Home for Data Science. Disponível em: <https://www.kaggle.com/>. Acesso em: 09 de jun. 2023.

KANASHIRO, L. H. Aplicação do modelo de regressão logística na identificação das causas que levam a acidentes com vítimas fatais nas rodovias br-101 e br-116. Universidade Federal de São Paulo, 2022. 14 p. Acesso em: 30 de out. 2023.

LEANDRO, J. N. B. et al. Introdução ao Processamento de Linguagem Natural usando Python. 2017. Disponível em: <https://www.facom.ufu.br/~wendelmelo/terceiros/tutorial_nltk.pdf>. Acesso em: 03 out. 2023.

LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. Mining of massive data sets. Cambridge: Cambridge University Press, 2020.

MARTINS, Julio Serafim; et al. Processamentos de Linguagem Natural. Porto Alegre: SAGAH, 2020. 21 p.

MORAIS, Edison Andrade Martins; AMBRÓSIO, Ana Paula L. Mineração de textos. Relatório Técnico–Instituto de Informática (UFG), Goiânia, n. RT-INF 005/07, 2007. Disponível em: <https://ww2.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_005-07.pdf>. Acesso em: 06 abr. 2023.

MÜLLER, A. C.; GUIDO, S. Introduction to machine learning with Python : a guide for data scientists. Beijing: O'reilly, 2017.

MURPHY, K. P. Machine learning : a probabilistic perspective. Cambridge (Ma): Mit Press, 2012.

NG, Andrew. Machine Learning — Stanford University. 2023. Disponível em: <http://mlclass.stanford.edu/>. Acesso em: 07 jun. 2023.

NLTK. Natural Language Toolkit — NLTK 3.4.4 documentation. Disponível em: <https://www.nltk.org/>. Acesso em: 09 de jun. 2023.

OLIVEIRA, Lucas L. de; MERGEN, Sérgio L. S. Análise da Popularidade de Tuítes com Base em Características Extra ídas de seu Conteúdo. In: ESCOLA REGIONAL DE BANCO DE DADOS (ERBD), 14. , 2018. Disponível em: <https://sol.sbc.org.br/index.php/erbd/article/view/2834/2796/>. Acesso em: 18 jun. 2023.

PANDAS. Python Data Analysis Library — pandas: Python Data Analysis Library. Disponível em: <https://pandas.pydata.org/>. Acesso em: 09 de jun. 2023.

STATISTA. Number of X (formerly Twitter) users worldwide from 2019 to 2024. Disponível em: <https://www.statista.com/statistics/303681/twitter-users-worldwide/?ref=definicao.marketing/>. Acesso em 20 out. 2023.

PEZZINI, Anderson. Mineração de textos: conceito, processo e aplicações. Revista Eletrônica do Alto Vale do itajaí, [s. l.], v. 5, n. 8, p. 58–61, 2016. Disponível em: <https://doi.org/10.5965/2316419005082016058>. Acesso em: 20 de ago. 2023

MANISHA, Pooja Rahate; CHANDAK M. B. Text Normalization and Its Role in Speech Synthesis. 2019. Disponível em: <https://www.ijeat.org/wpcontent/uploads/papers/v8i5S3/E10290785S319.pdf./>. Acesso em: 03 nov. 2023.

POWERS, David MW. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061, 2020. Disponível em: <https://arxiv.org/abs/2010.16061>. Acesso em: 16 de jun. 2023.

PYnative. Pandas DataFrame. Disponível em: <https://pynative.com/python-pandas-dataframe/>. Acesso em: 09 de jun. 2023.

PYTHON. Python. Disponível em: <https://www.python.org/>. Acesso em: 09 de jun. 2023.

REGINA, Elayne Lima Silva.Classificação automática de questões de concursos: um estudo comparativo utilizando algoritmos de mineração de textos, 2022. Disponível em: <https://repositorio.ifpb.edu.br/handle/177683/2479/>. Acesso em: 20 ago. 2023.

SAMUEL, Arthur L. Some studies in machine learning using the game of checkers. IBM Journal of research and development, v. 44, n. 1.2, p. 206-226, 2000. Disponível em: <https://ieeexplore.ieee.org/document/5389202> Acesso em: 07 abr. 2023

SANTOS, Frances A. et al. Processamento de Linguagem Natural em Textos de Mídias Sociais: Fundamentos, Ferramentas e Aplicações. Sociedade Brasileira de Computação, 2022. Disponível em:

<https://sol.sbc.org.br/livros/index.php/sbc/catalog/download/106/473/746-1>. Acesso em: 20 jun 2023.

SANTOS, Ronnie E S et al. Técnicas de processamento de linguagem natural aplicadas ao processo de mineração de textos: resultados preliminares de um mapeamento sistemático. Revista de Sistemas e Computação, [s. l.], v. 4, p. 116–125, 2014. Acesso em: 11 set. 2023.

SCIKIT-LEARN. scikit-learn: machine learning in Python. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 09 de jun. 2023.

TERENTEVA,Elena. Guia Semrush: Melhores Práticas de Marketing de Conteúdo 2019: TF-IDF. Semrush, 2019. Disponível em: <https://pt.semrush.com/blog/guia-semrush-melhores-praticas-de-marketing-de-conteudo-2018/>. Acesso em: 18 jun. 2023.

THANAKI, J. Python natural language processing. Birmingham: Packt Publishing Ltd, 2017.

VINÍCIUS, Marcos Zivolo; CRISTINA Sandra Costa Prado. Estudo comparativo dos algoritmos de aprendizado de máquina naive bayes, árvore de decisão e redes neurais artificiais para análise de sentimento – implementação com pandas e scikit-learn. 2021. Disponível em: <https://congresso.fatecmococa.edu.br/index.php/congresso/article/view/323/99>. Acesso em: 15 nov. 2023.

WE ARE SOCIAL. Digital 2022: Another Year of Bumper Growth. Disponível em: <https://wearesocial.com/uk/blog/2022/01/digital-2022-another-year-of-bumper-growth-2/>. Acesso em: 07 abr. 2023.

X. Glossário. 2023. Disponível em: <https://help.twitter.com/pt/resources/glossary>. Acesso em: 21 de ago. 2023.

Apêndices

Apêndice A — *Notebook* Python utilizado para a execução da analise comparativa

```
# From python
from argparse import Namespace

import os
import random
import re
import string
```

```
import time

# From pypi
from nltk.corpus import stopwords
from nltk.stem import RSLPStemmer
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

import pickle
import nltk
import pandas as pd
import spacy
```

```
nltk.download('stopwords')
nltk.download('rslp')
```

```
LANGUAGE = 'english' # english or portuguese

PORTUGUESE_TRAIN_DATASET_PATH =
'../datasets/portuguese/TrainingDatasets/Train3Classes.csv'
PORTUGUESE_TEST_DATASET_PATH =
'../datasets/portuguese/TestDatasets/Test3classes.csv'

ENGLISH_TRAIN_DATASET_PATH =
'../datasets/english/Corona_NLP_train.csv'
ENGLISH_TEST_DATASET_PATH =
'../datasets/english/Corona_NLP_test.csv'

STEMMING = True # True or False
```

```
LEMMATIZATION = False # True or False

VECTORIZER = 'tfidf' # bow or tfidf
CLASSIFIER = 'logistic_regression' # bayes, svm or
logistic_regression
```

```
# Set dataset path and columns
train_dataset_path = PORTUGUESE_TRAIN_DATASET_PATH if LANGUAGE ==
'portuguese' else ENGLISH_TRAIN_DATASET_PATH
test_dataset_path = PORTUGUESE_TEST_DATASET_PATH if LANGUAGE ==
'portuguese' else ENGLISH_TEST_DATASET_PATH

sepator = ';' if LANGUAGE == 'portuguese' else ','
encoding = 'utf-8' if LANGUAGE == 'portuguese' else 'latin-1'

X_col = 'tweet_text' if LANGUAGE == 'portuguese' else
'OriginalTweet'
y_col = 'sentiment' if LANGUAGE == 'portuguese' else 'Sentiment'
```

```
# Import dataset
raw_trainDF = pd.read_csv(train_dataset_path, sep=sepator,
encoding=encoding)
raw_testDF = pd.read_csv(test_dataset_path, sep=sepator,
encoding=encoding)
raw_trainDF.head()
```

```
# Copy the values of the data for further uses
trainDF = raw_trainDF
testDF = raw_testDF
```

```
TRAINING_SIZE = 0.8
SEED = 20200724

# Concat the two datasets and split them
allDF = pd.concat((trainDF, testDF), ignore_index=True)

# Sample dataset due to the large size
allDF = allDF.sample(frac=0.5).reset_index(drop=True)

# Split the train, test, validation set
```

```
trainDF, testDF = train_test_split(allDF, train_size=TRAINING_SIZE,
random_state=SEED)
testDF, validDF = train_test_split(testDF,
train_size=TRAINING_SIZE, random_state=SEED)

# Print values
print("Train:",len(trainDF), "Test:", len(testDF),"Valid:",
len(validDF))
```

```
X_all = allDF[X_col]
y_all = allDF[y_col]

print(f"Tweets: {len(X_all):,}")
print(f"Labels: {len(y_all.unique()):,}")
```

```
random.seed(SEED)
```

```
encoder = LabelEncoder()
trainDF.encoded_sentiment = encoder.fit_transform(trainDF[y_col])
trainDF.encoded_sentiment

encoder = LabelEncoder()
testDF.encoded_sentiment = encoder.fit_transform(testDF[y_col])
testDF.encoded_sentiment
```

```
def _remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub('', text)
```

```
def _remove_mentions(text):
    mention_pattern = re.compile(r'@\w+')
    return mention_pattern.sub('', text)
```

```
def _remove_hashtags(text):
    hashtag_pattern = re.compile(r'#\w+')
    return hashtag_pattern.sub('', text)
```

```
def _noise_reduction(text):
    text = _remove_urls(text)
```

```
    text = _remove_mentions(text)
    text = _remove_hashtags(text)

    return text
```

```
THE_CHOSEN = X_all[(X_all.str.contains("http")) &
               (X_all.str.contains("#")) &
               (X_all.str.contains("@"))].iloc[4]

print(THE_CHOSEN)
```

```
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
reduce_len=True)
```

```
stopwords = stopwords.words('portuguese') if LANGUAGE ==
'portuguese' else stopwords.words('english')
punctuations = string.punctuation

print(stopwords)
print(punctuations)
```

```
stemmers = Namespace(
    portuguese = RSLPStemmer(),
    english = PorterStemmer(),
)
```

```
if LANGUAGE == 'portuguese':
    stemmer = stemmers.portuguese
elif LANGUAGE == 'english':
    stemmer = stemmers.english
else:
    raise ValueError('Invalid language')
```

```
lemmatizers = Namespace(
    portuguese = spacy.load('pt_core_news_sm',
disable=['parser','ner']),
    english = spacy.load('en_core_web_sm',
disable=['parser','ner']),
)
```

```
if LANGUAGE == 'portuguese':
    lemmatizer = lemmatizers.portuguese
elif LANGUAGE == 'english':
    lemmatizer = lemmatizers.english
else:
    raise ValueError('Invalid language')
```

```
def _tokenizer(text):
    # Remove URLs, Mentions and Hashtags
    text = _noise_reduction(text)

    # Tokenize
    tokens = tokenizer.tokenize(text)

    # Remove stopwords and punctuations
    tokens = [word for word in tokens if (word not in stopwords and
word not in punctuations)]

    # Stemming
    if STEMMING:
        tokens = [stemmer.stem(word) for word in tokens]

    # Lemmatization
    if LEMMATIZATION:
        tokens = [lemmatizer(word)[0].lemma_ for word in tokens]

    return tokens
```

```
print(THE_CHOSEN)
THE_CHOSEN = _tokenizer(THE_CHOSEN)
print(THE_CHOSEN)
```

```
vectorizers = Namespace(
    bow = CountVectorizer(tokenizer=_tokenizer, ngram_range=(1,1)),
    tfidf = TfidfVectorizer(tokenizer=_tokenizer,
ngram_range=(1,1)),
)
```

```
if VECTORIZER == 'bow':
```

```
    vectorizer = vectorizers.bow
elif VECTORIZER == 'tfidf':
    vectorizer = vectorizers.tfidf
else:
    raise ValueError('Invalid vectorizer')
```

```
classifiers = Namespace(
    bayes = MultinomialNB(),
    svm = SVC(),
    logistic_regression = LogisticRegression(),
)
```

```
if CLASSIFIER == 'bayes':
    classifier = classifiers.bayes
elif CLASSIFIER == 'svm':
    classifier = classifiers.svm
elif CLASSIFIER == 'logistic_regression':
    classifier = classifiers.logistic_regression
else:
    raise ValueError('Invalid classifier')
```

```
pipe = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier),
])
```

```
X_train = trainDF[X_col]
X_test = testDF[X_col]
y_train = trainDF.encoded_sentiment
y_test = testDF.encoded_sentiment
```

```
pipe.fit(X_train, y_train)
```

```
predicted = pipe.predict(X_test)
```

```
datetime = time.strftime("%d-%m-%Y_%H-%M-%S")
```

```
report = classification_report(y_test, predicted, output_dict=True)
```

```
report = pd.DataFrame(report).transpose()
report.to_csv(f'../reports/{LANGUAGE}_{VECTORIZER}_{CLASSIFIER}_{STEMMING}_{LEMMATIZATION}_{datetime}.csv')

print(classification_report(y_test, predicted))
```

```
print(pipe.predict(allDF[X_col].iloc[0:1]))
```

```
# save the model to disk
filename = f'../models/{LANGUAGE}_{VECTORIZER}_{CLASSIFIER}_{STEMMING}_{LEMMATIZATION}_{datetime}.pkl'
pickle.dump(pipe, open(filename, 'wb'))

# load the model from disk
loaded_pipe = pickle.load(open(filename, 'rb'))
result = loaded_pipe.score(X_test, y_test)
print(result)
```

Apêndice B — *Notebook* Python utilizado para o processamento dos dados e geração de gráficos

```
import regex as re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
LANGUAGE = 'english'
MARKDOWN_PATH = '..//runnings/runnings_english.md'
```

```
with open(MARKDOWN_PATH, 'r') as file:
    markdown_content = file.read()

markdown_content[:500]
```

```
# Split markdown content into sections
sections = markdown_content.split("\n\n---\n\n")
```

```
def convert_time_to_seconds(time_str):
    '''Convert time string in format 'Xm Y.Zs' to seconds.'''
    minutes = 0
    seconds = 0.0
    if 'm' in time_str:
        minutes = int(time_str.split('m')[0])
        seconds = float(time_str.split('m')[1].replace('s',
'').strip())
    else:
        seconds = float(time_str.replace('s', '').strip())

    return minutes * 60 + seconds
```

```
def extract_execution_details(execution_content):
    '''Robustly extract details from a given execution content.'''

    lines = execution_content.split('\n')

    details = {}

    for line in lines:
        if 'Linguagem do Dataset' in line:
            details['Linguagem do Dataset'] =
line.split(':')[1].strip()
        elif 'Técnica para Redução de Palavras' in line:
            details['Técnica para Redução de Palavras'] =
line.split(':')[1].strip()
        elif 'Técnica de Representação Textual' in line:
            details['Técnica de Representação Textual'] =
line.split(':')[1].strip()
        elif 'Modelo Utilizado' in line:
            details['Modelo Utilizado'] =
line.split(':')[1].strip()
        elif 'Tempo de Treinamento' in line:
            details['Tempo de Treinamento'] =
convert_time_to_seconds(line.split(':')[1].strip())
        elif 'Tempo de Predição' in line:
            details['Tempo de Predição'] =
convert_time_to_seconds(line.split(':')[1].strip())

    metrics_pattern = r'\| (\d+|accuracy|macro avg|weighted avg) \|
([^\|]+) \| ([^\|]+) \| ([^\|]+) \| ([^\|]+) \|'
```

```
    metrics_matches = re.findall(metrics_pattern,
execution_content)

    details['Metrics'] = {match[0]: {'precision':
float(match[1])*100,
                                     'recall': float(match[2])*100,
                                     'f1-score':
float(match[3])*100,
                                     'support': int(match[4])} for
match in metrics_matches}

    return details
```

```
# Extracting details for all execution sections again using the
updated robust function
all_executions_details = []

# We will start from the second section since the first section is
the summary
for section in sections[1:]:
    if 'Execução' in section:  # Checking if the section
corresponds to an execution
        details = extract_execution_details(section)
        all_executions_details.append(details)

# Displaying the details for the first few executions to verify
all_executions_details[:3]  # Displaying details for the first 3
executions
```

```
# Creating a list to hold the consolidated data
consolidated_data = []

# Iterating through the extracted details to consolidate data
for details in all_executions_details:
    # Only considering entries with metrics
    if details.get('Metrics'):
        row = {
            'Modelo': details['Modelo Utilizado'],
            'Redução de Palavras': details['Técnica para Redução de
Palavras'],
            'Representação Textual': details['Técnica de
```

```
Representação Textual'],
            'Precision (Macro Avg)': details['Metrics']['macro
avg']['precision'],
            'Recall (Macro Avg)': details['Metrics']['macro
avg']['recall'],
            'F1-score (Macro Avg)': details['Metrics']['macro
avg']['f1-score'],
            'Tempo de Treinamento': details['Tempo de
Treinamento'],
            'Tempo de Predição': details['Tempo de Predição']
        }
        consolidated_data.append(row)

# Converting the consolidated data to a pandas DataFrame
df_metrics = pd.DataFrame(consolidated_data)

df_metrics
```

```
# Most efficient combination
most_efficient = df_metrics[df_metrics['F1-score (Macro Avg)'] ==
df_metrics['F1-score (Macro Avg)'].max()]

most_efficient
```

```
# Least efficient combination
least_efficient = df_metrics[df_metrics['F1-score (Macro Avg)'] ==
df_metrics['F1-score (Macro Avg)'].min()]

least_efficient
```

```
# Creating a new column for combination details
df_metrics['Combinação'] = df_metrics['Modelo'] + ' + ' +
df_metrics['Redução de Palavras'] + ' + ' +
df_metrics['Representação Textual']

# Sorting the data by F1-score for better visualization
df_metrics_sorted = df_metrics.sort_values(by='F1-score (Macro
Avg)', ascending=False)
```

```
# Average Precision (Macro Avg) by Model
```

```
avg_precision_by_model = df_metrics.groupby('Modelo')['Precision
(Macro Avg)'].mean()

avg_precision_by_model
```

```
# Average Recall (Macro Avg) by Model
avg_recall_by_model = df_metrics.groupby('Modelo')['Recall (Macro
Avg)'].mean()

avg_recall_by_model
```

```
# Average F1-score (Macro Avg) by Model
avg_f1score_by_model = df_metrics.groupby('Modelo')['F1-score
(Macro Avg)'].mean()

avg_f1score_by_model
```

```
# Average Precision (Macro Avg) by Reduction Technique
avg_precision_by_reduction = df_metrics.groupby('Redução de
Palavras')['Precision (Macro Avg)'].mean()

avg_precision_by_reduction
```

```
# Average Recall (Macro Avg) by Reduction Technique
avg_recall_by_reduction = df_metrics.groupby('Redução de
Palavras')['Recall (Macro Avg)'].mean()

avg_recall_by_reduction
```

```
# Average F1-score (Macro Avg) by Reduction Technique
avg_f1score_by_reduction = df_metrics.groupby('Redução de
Palavras')['F1-score (Macro Avg)'].mean()

avg_f1score_by_reduction
```

```
# Average Precision (Macro Avg) by Representation Technique
avg_precision_by_representation = df_metrics.groupby('Representação
Textual')['Precision (Macro Avg)'].mean()
```

```
avg_precision_by_representation
```

```
# Average Recall (Macro Avg) by Representation Technique
avg_recall_by_representation = df_metrics.groupby('Representação
Textual')['Recall (Macro Avg)'].mean()

avg_recall_by_representation
```

```
# Average F1-score (Macro Avg) by Representation Technique
avg_f1score_by_representation = df_metrics.groupby('Representação
Textual')['F1-score (Macro Avg)'].mean()

avg_f1score_by_representation
```

```
def save_plot_as_image(figure, filename):
    figure.savefig('../insights/' + LANGUAGE + '/' + filename +
'.png', dpi=300, bbox_inches='tight')
```

```
# Comparison of F1-score (Macro Avg) by Combination
fig_f1score_combination, ax_f1score_combination =
plt.subplots(figsize=(15, 10))
df_metrics_sorted.set_index('Combinação')['F1-score (Macro
Avg)'].plot(kind='barh', color='skyblue',
ax=ax_f1score_combination, fontsize=20)
# ax_f1score_combination.set_xlim([0, 100])  # Set x-axis limit to
start at 80
ax_f1score_combination.set_ylabel('Combinação de Modelo, Redução e
Representação', fontsize=20)
ax_f1score_combination.set_title('Comparação do F1-score (Macro
Avg) por Combinação', fontsize=20)
ax_f1score_combination.invert_yaxis()  # To display the highest
scores at the top

save_plot_as_image(fig_f1score_combination, 'f1score_combination')
```

```
# Average Precision by Model, Reduction Technique, and
Representation Technique
fig_avg_precision, axes_avg_precision = plt.subplots(nrows=1,
ncols=3, figsize=(20, 6))
avg_precision_by_model.sort_values().plot(kind='bar',
```

```
ax=axes_avg_precision[0], color='coral')
axes_avg_precision[0].set_title('Média de Precisão por Modelo',
fontsize=15)
axes_avg_precision[0].set_ylabel('Precisão (Macro Avg)',
fontsize=13)
axes_avg_precision[0].set_xlabel('Modelo', fontsize=13)
axes_avg_precision[0].grid(axis='y', linestyle='--', alpha=0.7)

avg_precision_by_reduction.sort_values().plot(kind='bar',
ax=axes_avg_precision[1], color='lightseagreen')
axes_avg_precision[1].set_title('Média de Precisão por Técnica de
Redução', fontsize=15)
axes_avg_precision[1].set_ylabel('Precisão (Macro Avg)',
fontsize=13)
axes_avg_precision[1].set_xlabel('Técnica de Redução', fontsize=13)
axes_avg_precision[1].grid(axis='y', linestyle='--', alpha=0.7)

avg_precision_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_precision[2], color='lightblue')
axes_avg_precision[2].set_title('Média de Precisão por Técnica de
Representação', fontsize=15)
axes_avg_precision[2].set_ylabel('Precisão (Macro Avg)',
fontsize=13)
axes_avg_precision[2].set_xlabel('Técnica de Representação',
fontsize=13)
axes_avg_precision[2].grid(axis='y', linestyle='--', alpha=0.7)

save_plot_as_image(fig_avg_precision, 'avg_precision')
```

```
# Average Recall by Model, Reduction Technique, and Representation
Technique
fig_avg_recall, axes_avg_recall = plt.subplots(nrows=1, ncols=3,
figsize=(20, 6))
avg_recall_by_model.sort_values().plot(kind='bar',
ax=axes_avg_recall[0], color='coral')
axes_avg_recall[0].set_title('Média de Recall por Modelo',
fontsize=15)
axes_avg_recall[0].set_ylabel('Recall (Macro Avg)', fontsize=13)
axes_avg_recall[0].set_xlabel('Modelo', fontsize=13)
axes_avg_recall[0].grid(axis='y', linestyle='--', alpha=0.7)

avg_recall_by_reduction.sort_values().plot(kind='bar',
```

```
ax=axes_avg_recall[1], color='lightseagreen')
axes_avg_recall[1].set_title('Média de Recall por Técnica de
Redução', fontsize=15)
axes_avg_recall[1].set_ylabel('Recall (Macro Avg)', fontsize=13)
axes_avg_recall[1].set_xlabel('Técnica de Redução', fontsize=13)
axes_avg_recall[1].grid(axis='y', linestyle='--', alpha=0.7)

avg_recall_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_recall[2], color='lightblue')
axes_avg_recall[2].set_title('Média de Recall por Técnica de
Representação', fontsize=15)
axes_avg_recall[2].set_ylabel('Recall (Macro Avg)', fontsize=13)
axes_avg_recall[2].set_xlabel('Técnica de Representação',
fontsize=13)
axes_avg_recall[2].grid(axis='y', linestyle='--', alpha=0.7)

save_plot_as_image(fig_avg_recall, 'avg_recall')
```

```
# Average F1-score by Model, Reduction Technique, and
Representation Technique
fig_avg_f1score, axes_avg_f1score = plt.subplots(nrows=1, ncols=3,
figsize=(20, 6))
avg_f1score_by_model.sort_values().plot(kind='bar',
ax=axes_avg_f1score[0], color='coral')
axes_avg_f1score[0].set_title('Média de F1-score por Modelo',
fontsize=15)
axes_avg_f1score[0].set_ylabel('F1-score (Macro Avg)', fontsize=13)
axes_avg_f1score[0].set_xlabel('Modelo', fontsize=13)
axes_avg_f1score[0].grid(axis='y', linestyle='--', alpha=0.7)

avg_f1score_by_reduction.sort_values().plot(kind='bar',
ax=axes_avg_f1score[1], color='lightseagreen')
axes_avg_f1score[1].set_title('Média de F1-score por Técnica de
Redução', fontsize=15)
axes_avg_f1score[1].set_ylabel('F1-score (Macro Avg)', fontsize=13)
axes_avg_f1score[1].set_xlabel('Técnica de Redução', fontsize=13)
axes_avg_f1score[1].grid(axis='y', linestyle='--', alpha=0.7)

avg_f1score_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_f1score[2], color='lightblue')
axes_avg_f1score[2].set_title('Média de F1-score por Técnica de
Representação', fontsize=15)
```

```
axes_avg_f1score[2].set_ylabel('F1-score (Macro Avg)', fontsize=13)
axes_avg_f1score[2].set_xlabel('Técnica de Representação',
fontsize=13)
axes_avg_f1score[2].grid(axis='y', linestyle='--', alpha=0.7)

save_plot_as_image(fig_avg_f1score, 'avg_f1score')
```

```
# Average precision, recall, and F1-score by Model
fig_avg_metrics_model, axes_avg_metrics_model =
plt.subplots(nrows=1, ncols=3, figsize=(30, 6))
avg_precision_by_model.sort_values().plot(kind='bar',
ax=axes_avg_metrics_model[0], color='coral', fontsize=20)
axes_avg_metrics_model[0].set_title('Média de Precisão por Modelo',
fontsize=20)
axes_avg_metrics_model[0].set_ylabel('Precisão (Macro Avg)',
fontsize=20)
axes_avg_metrics_model[0].set_xlabel('Modelo', fontsize=20)
axes_avg_metrics_model[0].grid(axis='y', linestyle='--', alpha=0.7)
axes_avg_metrics_model[0].set_ylim([30, 65]) # set the y-axis
limits to start from 0.8 and end at 1.0

avg_recall_by_model.sort_values().plot(kind='bar',
ax=axes_avg_metrics_model[1], color='lightseagreen', fontsize=20)
axes_avg_metrics_model[1].set_title('Média de Recall por Modelo',
fontsize=20)
axes_avg_metrics_model[1].set_ylabel('Recall (Macro Avg)',
fontsize=20)
axes_avg_metrics_model[1].set_xlabel('Modelo', fontsize=20)
axes_avg_metrics_model[1].grid(axis='y', linestyle='--', alpha=0.7)
axes_avg_metrics_model[1].set_ylim([30, 65]) # set the y-axis
limits to start from 0.8 and end at 1.0

avg_f1score_by_model.sort_values().plot(kind='bar',
ax=axes_avg_metrics_model[2], color='lightblue', fontsize=18)
axes_avg_metrics_model[2].set_title('Média de F1-score por Modelo',
fontsize=20)
axes_avg_metrics_model[2].set_ylabel('F1-score (Macro Avg)',
fontsize=20)
axes_avg_metrics_model[2].set_xlabel('Modelo', fontsize=20)
axes_avg_metrics_model[2].grid(axis='y', linestyle='--', alpha=0.7)
axes_avg_metrics_model[2].set_ylim([30, 65]) # set the y-axis
```

```
limits to start from 0.8 and end at 1.0

save_plot_as_image(fig_avg_metrics_model, 'avg_metrics_model')
```

```
# Global average by Model (precision + recall + F1-score / 3)
global_avg_by_model = (avg_precision_by_model + avg_recall_by_model
+ avg_f1score_by_model) / 3

global_avg_by_model
```

```
# Average precision, recall, and F1-score by Reduction Technique
fig_avg_metrics_reduction, axes_avg_metrics_reduction =
plt.subplots(nrows=1, ncols=3, figsize=(30, 6))
avg_precision_by_reduction.sort_values().plot(kind='bar',
ax=axes_avg_metrics_reduction[0], color='coral', fontsize=20)
axes_avg_metrics_reduction[0].set_title('Média de Precisão por
Técnica de Redução', fontsize=20)
axes_avg_metrics_reduction[0].set_ylabel('Precisão (Macro Avg)',
fontsize=20)
axes_avg_metrics_reduction[0].set_xlabel('Técnica de Redução',
fontsize=20)
axes_avg_metrics_reduction[0].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_reduction[0].set_ylim([30, 65]) # set the y-axis
limits to start from 0.8 and end at 1.0

avg_recall_by_reduction.sort_values().plot(kind='bar',
ax=axes_avg_metrics_reduction[1], color='lightseagreen',
fontsize=20)
axes_avg_metrics_reduction[1].set_title('Média de Recall por
Técnica de Redução', fontsize=20)
axes_avg_metrics_reduction[1].set_ylabel('Recall (Macro Avg)',
fontsize=20)
axes_avg_metrics_reduction[1].set_xlabel('Técnica de Redução',
fontsize=20)
axes_avg_metrics_reduction[1].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_reduction[1].set_ylim([30, 65]) # set the y-axis
limits to start from 0.8 and end at 1.0

avg_f1score_by_reduction.sort_values().plot(kind='bar',
```

```
ax=axes_avg_metrics_reduction[2], color='lightblue', fontsize=18)
axes_avg_metrics_reduction[2].set_title('Média de F1-score por
Técnica de Redução', fontsize=20)
axes_avg_metrics_reduction[2].set_ylabel('F1-score (Macro Avg)',
fontsize=20)
axes_avg_metrics_reduction[2].set_xlabel('Técnica de Redução',
fontsize=20)
axes_avg_metrics_reduction[2].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_reduction[2].set_ylim([30, 65]) # set the y-axis
limits to start from 0.8 and end at 1.0

save_plot_as_image(fig_avg_metrics_reduction,
'avg_metrics_reduction')
```

```
# Global average by Reduction Technique (precision + recall +
F1-score / 3)
global_avg_by_reduction = (avg_precision_by_reduction +
avg_recall_by_reduction + avg_f1score_by_reduction) / 3

global_avg_by_reduction
```

```
# Average precision, recall, and F1-score by Representation
Technique
fig_avg_metrics_representation, axes_avg_metrics_representation =
plt.subplots(nrows=1, ncols=3, figsize=(30, 6))
avg_precision_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_metrics_representation[0], color='coral', fontsize=20)
axes_avg_metrics_representation[0].set_title('Média de Precisão por
Técnica de Representação', fontsize=20)
axes_avg_metrics_representation[0].set_ylabel('Precisão (Macro
Avg)', fontsize=20)
axes_avg_metrics_representation[0].set_xlabel('Técnica de
Representação', fontsize=20)
axes_avg_metrics_representation[0].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_representation[0].set_ylim([30, 65]) # set the
y-axis limits to start from 0.8 and end at 1.0

avg_recall_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_metrics_representation[1], color='lightseagreen',
```

```
fontsize=20)
axes_avg_metrics_representation[1].set_title('Média de Recall por
Técnica de Representação', fontsize=20)
axes_avg_metrics_representation[1].set_ylabel('Recall (Macro Avg)',
fontsize=20)
axes_avg_metrics_representation[1].set_xlabel('Técnica de
Representação', fontsize=20)
axes_avg_metrics_representation[1].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_representation[1].set_ylim([30, 65]) # set the
y-axis limits to start from 0.8 and end at 1.0

avg_f1score_by_representation.sort_values().plot(kind='bar',
ax=axes_avg_metrics_representation[2], color='lightblue',
fontsize=18)
axes_avg_metrics_representation[2].set_title('Média de F1-score por
Técnica de Representação', fontsize=20)
axes_avg_metrics_representation[2].set_ylabel('F1-score (Macro
Avg)', fontsize=20)
axes_avg_metrics_representation[2].set_xlabel('Técnica de
Representação', fontsize=20)
axes_avg_metrics_representation[2].grid(axis='y', linestyle='--',
alpha=0.7)
axes_avg_metrics_representation[2].set_ylim([30, 65]) # set the
y-axis limits to start from 0.8 and end at 1.0

save_plot_as_image(fig_avg_metrics_representation,
'avg_metrics_representation')
```

```
# Global average by Representation Technique (precision + recall +
F1-score / 3)
global_avg_by_representation = (avg_precision_by_representation +
avg_recall_by_representation + avg_f1score_by_representation) / 3

global_avg_by_representation
```

Apêndice C — *Dataframes:* Métricas processadas

Figura 07: *Dataframe* das métricas processadas para o *dataset* em português

	Modelo	Redução de Palavras	Representação Textual	Precision (Macro Avg)	Recall (Macro Avg)	F1-score (Macro Avg)	Tempo de Treinamento	Tempo de Predição
0	Naive Bayes	Nenhuma	Bag of Words	96.0	96.0	96.0	5.5	1.1
1	Naive Bayes	Lematização	Bag of Words	96.0	96.0	96.0	702.7	151.4
2	Naive Bayes	Stemização	Bag of Words	96.0	96.0	96.0	15.6	2.7
3	Naive Bayes	Nenhuma	TF-IDF	92.0	92.0	92.0	5.5	1.0
4	Naive Bayes	Lematização	TF-IDF	93.0	93.0	93.0	694.3	149.3
5	Naive Bayes	Stemização	TF-IDF	92.0	92.0	92.0	15.1	3.0
6	SVM	Nenhuma	Bag of Words	99.0	100.0	99.0	95.5	5.2
7	SVM	Lematização	Bag of Words	99.0	99.0	99.0	824.2	158.6
8	SVM	Stemização	Bag of Words	99.0	99.0	99.0	100.5	7.2
9	SVM	Nenhuma	TF-IDF	100.0	100.0	100.0	323.7	11.3
10	SVM	Lematização	TF-IDF	99.0	99.0	99.0	979.9	152.1
11	SVM	Stemização	TF-IDF	100.0	100.0	100.0	274.1	12.0
12	Logistic Regression	Nenhuma	Bag of Words	100.0	100.0	100.0	10.3	1.0
13	Logistic Regression	Lematização	Bag of Words	100.0	100.0	100.0	695.2	140.4
14	Logistic Regression	Stemização	Bag of Words	100.0	100.0	100.0	18.8	3.0
15	Logistic Regression	Nenhuma	TF-IDF	99.0	99.0	99.0	12.1	1.0
16	Logistic Regression	Lematização	TF-IDF	99.0	99.0	99.0	727.9	149.0
17	Logistic Regression	Stemização	TF-IDF	99.0	99.0	99.0	19.2	2.9

Fonte: Elaborado pelos autores (2023)

Figura 08: *Dataframe* das métricas processadas para o *dataset* em inglês

	Modelo	Redução de Palavras	Representação Textual	Precision (Macro Avg)	Recall (Macro Avg)	F1-score (Macro Avg)	Tempo de Treinamento	Tempo de Predição
0	Naive Bayes	Nenhuma	Bag of Words	52.0	42.0	44.0	4.3	0.8
1	Naive Bayes	Lematização	Bag of Words	53.0	43.0	45.0	530.8	104.9
2	Naive Bayes	Stemização	Bag of Words	52.0	43.0	44.0	8.1	1.5
3	Naive Bayes	Nenhuma	TF-IDF	66.0	29.0	24.0	4.5	0.7
4	Naive Bayes	Lematização	TF-IDF	69.0	29.0	23.0	561.0	109.9
5	Naive Bayes	Stemização	TF-IDF	64.0	28.0	22.0	8.3	1.6
6	SVM	Nenhuma	Bag of Words	59.0	54.0	55.0	81.6	9.0
7	SVM	Lematização	Bag of Words	61.0	54.0	56.0	630.1	118.0
8	SVM	Stemização	Bag of Words	62.0	55.0	57.0	80.6	9.4
9	SVM	Nenhuma	TF-IDF	62.0	52.0	54.0	91.1	9.0
10	SVM	Lematização	TF-IDF	61.0	52.0	54.0	662.6	120.7
11	SVM	Stemização	TF-IDF	63.0	53.0	56.0	90.9	10.1
12	Logistic Regression	Nenhuma	Bag of Words	59.0	57.0	58.0	8.8	0.8
13	Logistic Regression	Lematização	Bag of Words	61.0	59.0	60.0	578.7	114.2
14	Logistic Regression	Stemização	Bag of Words	60.0	59.0	59.0	13.4	1.6
15	Logistic Regression	Nenhuma	TF-IDF	56.0	51.0	53.0	10.6	0.8
16	Logistic Regression	Lematização	TF-IDF	59.0	54.0	56.0	572.9	116.6
17	Logistic Regression	Stemização	TF-IDF	56.0	52.0	53.0	13.8	1.6

Fonte: Elaborado pelos autores (2023)