Análise de Sentimentos em Tweets Sobre Jogos de Futebol Utilizando PLN

1st Murilo Fontana Muniz
COENC. Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)
Apucarana, Brasil
murilo.2022@alunos.utfpr.edu.br

***Resumo*—This study conducted an in-depth investigation into the theoretical and practical foundations of Natural Language Processing (NLP) applied to Sentiment Analysis (SA), with a particular focus on traditional Machine Learning classifiers. We covered definitions and typologies of SA, its intrinsic challenges, and applications across various domains, as well as crucial steps in text preprocessing, vector representation, and the specific characteristics of classification models and their evaluation metrics. The tools and libraries employed, such as `twikit`, `pysentimiento`, `pandas`, `scikit-learn` and `nltk`, were detailed in their respective roles within the analytical pipeline.**

Experimental results demonstrated the critical importance of class balancing for the performance of Sentiment Analysis models. In the original ("Reduced") scenario, characterized by significant class imbalance, the models showed moderate performance, with Logistic Regression combined with CountVectorizer achieving the best F_1-Score of 0.62. However, the application of resampling strategies significantly altered the landscape.

The Oversampling scenario proved particularly effective, drastically enhancing the performance of all classifiers. In this context, classes were balanced to 196 samples each, and the SVC model combined with TfidfVectorizer achieved an impressive F_1-Score of 0.92 and accuracy of 0.92, demonstrating the models' ability to operate with high efficacy when classes are balanced through dataset expansion. Random Forest also exhibited excellent results in this scenario (F_1-Score of 0.90).

Conversely, the Undersampling scenario also yielded consistent improvements over the original setup, with classes balanced to 561 samples each. Here, the SGDClassifier with TfidfVectorizer reached an F_1-Score of 0.70, proving to be a valid strategy for mitigating model bias by reducing the majority class, though it might imply a loss of information.

The influence of vectorizers, while present, was secondary to the impact of balancing. TF-IDF generally showed a slight advantage over CountVectorizer in balanced scenarios, validating its effectiveness in weighting term relevance in sparse textual contexts. Classifiers like SVC and Random Forest demonstrated their strength and generalization capability in environments where Oversampling was applied , while SGDClassifier and PassiveAggressive Classifier excelled in the Undersampling scenario.

In summary, this work underscores that the sole application of traditional Machine Learning classifiers to raw textual data may not suffice to achieve optimal results in Sentiment Analysis, particularly when faced with class imbalance. The data balancing step emerges as a fundamental pillar to maximize the predictive capability of these models. These findings provide a robust foundation for future investigations, directing efforts towards hyperparameter optimization, exploration of hybrid balancing techniques, and integration with pre-trained language models, aiming for even more sophisticated and real-time Sentiment Analysis applications.

***Index Terms*—Sentiment Analysis, Natural Language Processing, Machine Learning, Class Balancing, Oversampling, Undersampling.**

I. Resumo

Este estudo realizou uma pesquisa aprofundada sobre os fundamentos teóricos e práticos do Processamento de Linguagem Natural (PLN) aplicado à Análise de Sentimentos (AS), com um foco particular na utilização de classificadores baseados em Machine Learning tradicional. Abordamos desde as definições e tipologias da AS, seus desafios intrínsecos e aplicações em diversos domínios, até as etapas cruciais de pré-processamento de texto, representação vetorial e as especificidades dos modelos de classificação e suas métricas de avaliação. As ferramentas e bibliotecas empregadas, como `twikit`, `pysentimiento`, `pandas`, `scikit-learn`e `nltk`, foram detalhadas em seus respectivos papéis no pipeline analítico.

Os resultados experimentais demonstraram a criticidade do balanceamento de classes para o desempenho dos modelos de Análise de Sentimentos. No cenário original ("Reduzido"), caracterizado por um desbalanceamento acentuado, os modelos apresentaram um desempenho moderado, com a Regressão Logística com CountVectorizer obtendo o melhor F_1-Score de 0.62. No entanto, a aplicação de estratégias de reamostragem alterou significativamente o panorama.

O cenário de **Oversampling** revelou-se particularmente eficaz, elevando o desempenho de todos os classificadores de forma drástica. Nele, as classes foram balanceadas para ter 196 amostras cada, e o modelo SVC combinado com TfidfVectorizer alcançou um impressionante F_1-Score de 0.92 e acurácia de 0.92, demonstrando a capacidade dos modelos em operar com alta eficácia quando as classes estão balanceadas pela expansão do dataset. O Random Forest também exibiu resultados excelentes neste cenário (F_1-Score de 0.90).

Por outro lado, o cenário de **Undersampling** também gerou melhorias consistentes em relação ao cenário original, com as classes balanceadas para 561 amostras cada. Neste contexto, o SGDClassifier com TfidfVectorizer atingiu um F_1-Score de 0.70, provando ser uma estratégia válida para mitigar o viés do modelo pela redução da classe majoritária, embora possa implicar em perda de informação.

A influência dos vetorizadores, embora presente, foi secundária ao impacto do balanceamento. O TF-IDF geralmente apresentou uma leve vantagem sobre o CountVectorizer em cenários balanceados, validando sua eficácia em ponderar a relevância dos termos em contextos textuais esparsos. Classificadores como SVC e Random Forest mostraram sua força e capacidade de generalização em ambientes onde o Oversampling foi aplicado, enquanto SGDClassifier e PassiveAggressive Classifier se destacaram no cenário de Undersampling.

Em síntese, este trabalho reforça que a simples aplicação de classificadores de Machine Learning tradicional a dados textuais brutos pode não ser suficiente para obter resultados ótimos em Análise de Sentimentos, especialmente diante de desbalanceamento de classes. A etapa de balanceamento de dados surge como um pilar fundamental para maximizar a capacidade preditiva desses modelos. Os achados servem como uma base robusta para futuras investigações, direcionando esforços para otimização de hiperparâmetros, exploração de técnicas híbridas de balanceamento e a integração com modelos de linguagem pré-treinados, visando aplicações ainda mais sofisticadas e em tempo real da Análise de Sentimentos.

***Index Terms*—Análise de Sentimentos, Processamento de Linguagem Natural, Machine Learning, Balanceamento de Classes, Oversampling, Undersampling.**

II. Introdução

Com o grande avanço da globalização, pôde-se perceber uma enorme importância da percepção de opiniões em diferentes contextos, como no trabalho de Hu e Liu [1] explora a importância da extração de opiniões para compreender tendências de comportamento em ambientes digitais. Os autores demonstram como a mineração de opiniões pode ser aplicada na formulação de estratégias de mercado, análise de produtos e decisões gerenciais. No contexto esportivo, essas técnicas podem ser adaptadas para avaliar o sentimento das torcidas, apoiar campanhas publicitárias e fortalecer a gestão da marca dos clubes.

Estudos como o de Zorron [2] e pesquisas da UTFPR [3] evidenciam os desafios específicos da análise de sentimentos na língua portuguesa. Entre esses desafios estão a variação linguística, a presença de gírias, regionalismos e menor disponibilidade de corpora (Em linguística e processamento de linguagem natural, um corpus, plural: corpora, é uma coleção de dados textuais ou orais, usada para análise e pesquisa da linguagem [**?**]) anotados para treinamento de modelos. Esses fatores exigem adaptações em algoritmos clássicos e refinamento das técnicas de pré-processamento.

A. *Fundamentos da Análise de Sentimentos (AS)*

A AS, também conhecida como mineração de opiniões, é um campo do Processamento de Linguagem Natural (PLN) que se dedica ao estudo computacional de opiniões, sentimentos e emoções expressas em textos [4]. Seu principal objetivo é determinar a polaridade sentimental de um texto, identificando se a atitude do autor é positiva, negativa ou neutra em relação a um determinado tópico ou entidade [5].

B. *Tipologias de AS (Polaridade, Emoção, Atomicidade*

A AS pode ser categorizada em diferentes níveis de granularidade e tipologias de sentimento:

- **Polaridade:** É o tipo mais comum de AS, classificando o texto em sentimentos **positivos**, **negativos** ou **neutros** [4].
- **Emoção:** Além da polaridade, a AS pode focar na detecção de emoções específicas, como alegria, raiva, tristeza, surpresa, medo e nojo, baseando-se em modelos categóricos (e.g., Ekman, Izard) ou dimensionais (e.g., valência, excitação, dominância) [6], [7].
- **Atomicidade:** A análise pode ser realizada em diferentes níveis:
  - **Nível de Documento:** Classifica o sentimento geral de um documento inteiro [4].
  - **Nível de Frase:** Determina o sentimento de cada frase individualmente dentro de um documento [4].
  - **Nível de Entidade/Aspecto:** Identifica o sentimento associado a entidades ou aspectos específicos mencionados no texto, permitindo uma compreensão mais detalhada das opiniões [4]. Por exemplo, em uma avaliação de um smartphone, pode-se analisar o sentimento sobre a ”câmera”versus a ”bateria”.

C. *Modelos de AS Baseados em Léxicos/Regras*

Modelos baseados em léxicos e regras utilizam dicionários de palavras com polaridades pré-definidas e conjuntos de regras gramaticais para determinar o sentimento de um texto.

1) VADER (Valence Aware Dictionary and sEntiment Reasoner):

- **Arquitetura/Princípios Operacionais:** VADER é um modelo baseado em regras e léxicos otimizado para textos de mídias sociais [8]. Ele utiliza uma lista de palavras com pontuações de valência (positiva, negativa, neutra) e um conjunto de regras heurísticas para levar em conta intensificadores, negações, pontuações e o uso de maiúsculas, gerando uma pontuação composta que varia de -1 (totalmente negativo) a +1 (totalmente positivo) [9].
- **Vantagens:** É rápido, eficaz em textos curtos e informais como tweets, não requer treinamento de dados e é de interpretabilidade (as regras são transparentes) [8].
- **Desvantagens:** Sua capacidade é limitada para capturar sarcasmo, ironia e nuances contextuais mais complexas. Não se adapta facilmente a domínios muito específicos ou a linguagens com estruturas morfológicas complexas, sendo predominantemente focado no inglês [9].
- **Cenários de Aplicação Ideais:** Ideal para análise rápida de grandes volumes de dados de mídias sociais, feedback de clientes em tempo real e cenários onde a velocidade e a interpretabilidade são cruciais e a complexidade linguística não é o principal desafio.

2) TextBlob:

- **Arquitetura/Princípios Operacionais:** TextBlob é uma biblioteca de PLN que oferece uma API simples para tarefas como part-of-speech tagging, extração de frases

nominais, classificação, tradução e, crucialmente, análise de sentimentos [10]. Sua análise de sentimento é baseada no léxico Pattern, que atribui pontuações de polaridade (de -1 a 1) e subjetividade (de 0 a 1) às palavras. A pontuação de uma frase é agregada a partir das pontuações das palavras, com algumas regras básicas para negações [11], [12].
- **Vantagens:** É extremamente fácil de usar e integrar em projetos, sendo uma ótima opção para prototipagem rápida e tarefas de AS básicas. Oferece tanto polaridade quanto subjetividade, o que pode ser útil para entender a natureza da expressão [11].
- **Desvantagens:** Por ser baseado em léxico e regras fixas, TextBlob tem dificuldade em lidar com sarcasmo, ironia, negações complexas, gírias e linguagem altamente contextual ou específica de domínio. Sua performance é limitada em comparação com modelos de Machine Learning treinados em grandes datasets [13], [14].
- **Cenários de Aplicação Ideais:** Melhor para tarefas de análise de sentimento que exigem simplicidade, rapidez e onde a profundidade da análise contextual não é o principal requisito. Adequado para análises exploratórias iniciais e para usuários com pouca experiência em PLN.

D. Técnicas de Pré-processamento de Texto para PNL/AS

O pré-processamento de texto é uma etapa crucial em qualquer pipeline de PLN, especialmente para a Análise de Sentimentos. Ele transforma dados de texto brutos em um formato que os modelos de Machine Learning podem processar de forma eficaz, reduzindo ruído e melhorando a qualidade das features.

E. Tokenização

Tokenização é o processo de dividir um texto em unidades menores, chamadas **tokens**. Esses tokens podem ser palavras, frases, caracteres ou subpalavras. O objetivo é segmentar o texto em elementos significativos para análise [15], [16].

- **Tipos:**
  - **Word Tokenization:** Divide o texto em palavras individuais. É o tipo mais comum e fundamental.
  - **Sentence Tokenization:** Divide o texto em frases, útil para análises de sentimento em nível de frase ou para entender a estrutura do discurso.
- **Considerações para Textos de Mídias Sociais e Tweets em Português:** Textos de mídias sociais apresentam características únicas, como o uso frequente de emojis, menções (`@username`), hashtags (`#topic`), URLs e abreviações. Tokenizadores tradicionais podem ter dificuldade em lidar com esses elementos. O **TweetTokenizer** da biblioteca NLTK é projetado especificamente para essa finalidade, sendo capaz de preservar ou separar esses elementos de forma inteligente, mantendo o contexto e o significado em tweets, inclusive em português [17], [18].

F. Desafios Intrínsecos à AS

Apesar de suas amplas aplicações, a Análise de Sentimentos enfrenta diversos desafios devido à complexidade e nuances da linguagem humana:

- **Ambiguidade Lexical:** Palavras com múltiplos significados que podem ter diferentes polaridades dependendo do contexto (e.g., ”bater”em ”bater um papo”vs. ”bater na porta”) [19], [20].
- **Sarcasmo e Ironia:** Expressões em que o significado pretendido é o oposto do literal, sendo extremamente difíceis de detectar por sistemas automatizados [19], [20].
- **Negação:** A presença de palavras de negação (e.g., ”não”, ”nunca”) pode inverter completamente o sentimento de uma frase (e.g., ”Este filme **não** é bom”) [19].
- **Dependência de Domínio:** Um mesmo termo pode ter polaridades diferentes em domínios distintos (e.g., ”grande”para um carro pode ser positivo, mas para uma conta de internet pode ser negativo) [19].
- **Variação Linguística (Dialetos, Gírias):** A presença de dialetos, gírias e regionalismos pode confundir os modelos, especialmente aqueles baseados em léxicos fixos [19].
- **Sentimentos Implícitos:** Sentimentos que não são expressos diretamente, mas inferidos a partir do contexto ou de subentendidos.

G. Modelos de AS Baseados em Léxicos/Regras

Modelos baseados em léxicos e regras utilizam dicionários de palavras com polaridades pré-definidas e conjuntos de regras gramaticais para determinar o sentimento de um texto.

1) VADER (Valence Aware Dictionary and sEntiment Reasoner):

- **Arquitetura/Princípios Operacionais:** VADER é um modelo baseado em regras e léxicos otimizado para textos de mídias sociais [8]. Ele utiliza uma lista de palavras com pontuações de valência (positiva, negativa, neutra) e um conjunto de regras heurísticas para levar em conta intensificadores, negações, pontuações e o uso de maiúsculas, gerando uma pontuação composta que varia de -1 (totalmente negativo) a +1 (totalmente positivo) [9].
- **Vantagens:** É rápido, eficaz em textos curtos e informais como tweets, não requer treinamento de dados e é de interpretabilidade (as regras são transparentes) [8].
- **Desvantagens:** Sua capacidade é limitada para capturar sarcasmo, ironia e nuances contextuais mais complexas. Não se adapta facilmente a domínios muito específicos ou a linguagens com estruturas morfológicas complexas, sendo predominantemente focado no inglês [9].
- **Cenários de Aplicação Ideais:** Ideal para análise rápida de grandes volumes de dados de mídias sociais, feedback de clientes em tempo real e cenários onde a velocidade e a interpretabilidade são cruciais e a complexidade linguística não é o principal desafio.

2) TextBlob:

- **Arquitetura/Princípios Operacionais:** TextBlob é uma biblioteca de PLN que oferece uma API simples para tarefas como part-of-speech tagging, extração de frases nominais, classificação, tradução e, crucialmente, análise de sentimentos [10]. Sua análise de sentimento é baseada no léxico Pattern, que atribui pontuações de polaridade (de -1 a 1) e subjetividade (de 0 a 1) às palavras. A pontuação de uma frase é agregada a partir das pontuações das palavras, com algumas regras básicas para negações [11], [12].
- **Vantagens:** É extremamente fácil de usar e integrar em projetos, sendo uma ótima opção para prototipagem rápida e tarefas de AS básicas. Oferece tanto polaridade quanto subjetividade, o que pode ser útil para entender a natureza da expressão [11].
- **Desvantagens:** Por ser baseado em léxico e regras fixas, TextBlob tem dificuldade em lidar com sarcasmo, ironia, negações complexas, gírias e linguagem altamente contextual ou específica de domínio. Sua performance é limitada em comparação com modelos de Machine Learning (ML) treinados em grandes datasets [13], [14].
- **Cenários de Aplicação Ideais:** Melhor para tarefas de análise de sentimento que exigem simplicidade, rapidez e onde a profundidade da análise contextual não é o principal requisito. Adequado para análises exploratórias iniciais e para usuários com pouca experiência em PLN.

H. Técnicas de Pré-processamento de Texto para PNL/AS

O pré-processamento de texto é uma etapa crucial em qualquer pipeline de PLN, especialmente para a Análise de Sentimentos. Ele transforma dados de texto brutos em um formato que os modelos de Machine Learning podem processar de forma eficaz, reduzindo ruído e melhorando a qualidade das features.

I. Tokenização

Tokenização é o processo de dividir um texto em unidades menores, chamadas **tokens**. Esses tokens podem ser palavras, frases, caracteres ou subpalavras. O objetivo é segmentar o texto em elementos significativos para análise [15], [16].

- **Tipos:**
  - **Word Tokenization:** Divide o texto em palavras individuais. É o tipo mais comum e fundamental.
  - **Sentence Tokenization:** Divide o texto em frases, útil para análises de sentimento em nível de frase ou para entender a estrutura do discurso.
- **Considerações para Textos de Mídias Sociais e Tweets em Português:** Textos de mídias sociais apresentam características únicas, como o uso frequente de emojis, menções (`@username`), hashtags (`#topic`), URLs e abreviações. Tokenizadores tradicionais podem ter dificuldade em lidar com esses elementos. O **TweetTokenizer** da biblioteca NLTK é projetado especificamente para essa finalidade, sendo capaz de preservar ou separar esses elementos de forma inteligente, mantendo o contexto e o significado em tweets, inclusive em português [17], [18].

J. Normalização de Texto

A normalização de texto visa padronizar o texto, reduzindo variações e ruídos que podem confundir os modelos.

- **Limpeza:** Envolve a remoção de elementos irrelevantes ou ruído que não contribuem para o significado sentimental do texto:
  - **Remoção de URLs:** Links da web geralmente não carregam valor sentimental.
  - **Remoção de Menções:** Nomes de usuário (`@username`) em mídias sociais raramente são relevantes para o sentimento da mensagem em si.
  - **Remoção de Pontuações Não Essenciais:** Sinais de pontuação excessivos ou fora de contexto podem ser removidos, embora em AS, a pontuação pode ser relevante para intensificar ou atenuar o sentimento (e.g., múltiplos pontos de exclamação).
  - **Remoção de Caracteres Especiais:** Caracteres não alfanuméricos que não agregam valor.
- **Stemming vs. Lematização:**
  - **Stemming:** É o processo de reduzir palavras flexionadas (ou, por vezes, derivadas) à sua raiz ou ”radical”(stem). O stem não é necessariamente uma palavra real ou linguisticamente correta [21].
    - **Vantagens:** Mais rápido e computacionalmente menos custoso.
    - **Desvantagens:** Pode gerar stems que não são palavras válidas e não lida bem com sinônimos ou palavras com raízes diferentes mas significados semelhantes.
    - **Exemplo (Português):** ”correr”, ”correndo”, ”corria”$\rightarrow$ ”corr”
  - **Lematização:** É o processo de reduzir palavras flexionadas à sua forma básica ou ”lema”(forma canônica de dicionário). O lema é sempre uma palavra linguisticamente válida [22].
    - **Vantagens:** Produz resultados mais precisos e semanticamente corretos, pois considera o contexto da palavra e seu Part-of-Speech (POS).
    - **Desvantagens:** É mais complexa e computacionalmente mais intensiva, exigindo léxicos e algoritmos mais sofisticados.
    - **Importância para o Português:** A lematização é particularmente crucial para o português devido à sua rica morfologia. Verbos, substantivos e adjetivos possuem um grande número de flexões de gênero, número, tempo verbal, pessoa e modo. A lematização garante que diferentes formas de uma mesma palavra sejam tratadas como a mesma feature, reduzindo a dimensionalidade e melhorando a precisão do modelo [23]. Por exemplo, ”correr”, ”correndo”, ”corria”, ”correm”seriam todos lematizados para ”correr”. Bibliotecas como o spaCy com seus modelos de português são eficazes para essa tarefa [24].

- **Minúsculas (Lowercase Conversion):** Converter todo o texto para minúsculas é uma prática comum para garantir que palavras como ”Amor”e ”amor”sejam tratadas como o mesmo token, reduzindo a esparsidade e a dimensionalidade dos dados [25], [26].
- **Remoção de Stop Words:**
  - **O que são:** Stop words são palavras comuns e de alta frequência em um idioma (e.g., ”o”, ”a”, ”de”, ”e”, ”para”) que geralmente não carregam significado semântico ou sentimental por si mesmas [27].
  - **Por que remover (ou não):** A remoção de stop words é feita para reduzir a dimensionalidade do vetor de features e focar nas palavras mais significativas para a análise. No entanto, em alguns contextos de AS, a remoção pode ser prejudicial, pois algumas stop words (e.g., ”não”) são cruciais para a negação e podem inverter o sentimento de uma frase. A decisão de remover ou não deve ser baseada no domínio e na complexidade da análise.
  - **Especificidade para o Português:** O português, assim como outros idiomas, possui suas próprias listas de stop words. Bibliotecas como NLTK e spaCy fornecem listas de stop words específicas para o português [27].
- **Tratamento de Dados Ausentes (NaNs):**
  - No contexto de dados textuais, dados ausentes (NaNs) podem surgir por problemas na coleta de dados ou na etapa de pré-processamento.
  - **Estratégias:**
    - **Remoção de Linhas/Colunas:** Se a quantidade de NaNs for pequena ou se a ausência de dados for sistemática, a remoção das linhas (documentos) ou colunas (features) com valores ausentes pode ser uma solução.
    - **Preenchimento com Valor Padrão:** NaNs em campos textuais podem ser preenchidos com uma string vazia ou um token especial (e.g., ”[UNK]”para ”unknown”) para que o modelo possa processá-los sem erros.
  - **Justificativa:** O tratamento adequado de NaNs é essencial para evitar erros durante o treinamento do modelo e garantir que todos os dados válidos sejam utilizados de forma eficaz [28], [29]. A escolha da estratégia depende da quantidade de dados ausentes e do impacto esperado na análise.

*K. **Representação Vetorial de Texto (Feature Engineering para PLN)***

Após o pré-processamento, o texto precisa ser convertido em um formato numérico (vetores) que os algoritmos de ML possam entender. Isso é conhecido como representação vetorial de texto ou *feature engineering*.

*L. **Bag of Words (BoW)***

- **Princípios:** O modelo BoW (Saco de Palavras) representa um documento como uma coleção (um ”saco”) de suas palavras, desconsiderando a ordem gramatical, mas mantendo a contagem da frequência de cada palavra [30]. Cada palavra única no corpus forma uma dimensão no vetor, e o valor em cada dimensão é a frequência (ou presença) dessa palavra no documento.
- **Vantagens:** É simples de entender e implementar. Funciona razoavelmente bem para muitas tarefas de classificação de texto devido à sua capacidade de capturar a importância das palavras pela frequência [31].
- **Desvantagens:**
  - **Esparsidade:** Gera vetores de alta dimensionalidade e com muitos zeros, especialmente em grandes vocabulários, o que pode levar a um alto consumo de memória e tempo de processamento [31].
  - **Perda de Ordem:** Ignora completamente a ordem das palavras, perdendo informações contextuais cruciais (e.g., ”bom não”vs. ”não bom”teriam a mesma representação BoW se a negação não for tratada em pré-processamento).
  - **Semanticamente Cego:** Não entende o significado das palavras ou as relações semânticas entre elas (e.g., ”carro”e ”automóvel”são tratadas como palavras distintas).

M. Term Frequency-Inverse Document Frequency (TF-IDF)

- **Princípios:** O TF-IDF é uma estatística numérica que reflete a importância de uma palavra em um documento em relação a um corpus de documentos [32]. Ele é calculado a partir de dois componentes:
  - **Term Frequency (TF):** A frequência de um termo (t) em um documento (d).

    $$TF(t,d) = \frac{\text{Nmro que o termo t aparece no doco d}}{\text{Nmro total de termos no doc d}}$$

  - **Inverse Document Frequency (IDF):** Uma medida de quão rara ou comum um termo é em todo o corpus. Termos raros têm um IDF alto, enquanto termos comuns (como stop words) têm um IDF baixo.

    $$IDF(t,D) = \log\left(\frac{\text{Nmro total de doc no corpus D}}{\text{Nmro de doc que tem termo t}}\right)$$

  - **TF-IDF:** O score final é o produto de TF e IDF.

    $$TF - IDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

  A intuição é que uma palavra é importante se aparece frequentemente em um documento específico, mas é rara em outros documentos do corpus [33].
- **Vantagens sobre BoW:** TF-IDF pondera a importância das palavras, reduzindo o peso de palavras muito comuns (stop words) e aumentando o peso de palavras que são mais discriminativas para um documento específico. Isso geralmente leva a uma melhor representação para tarefas

de classificação de texto em comparação com o BoW simples [32].

- **Desvantagens:**
  - **Perda de Contexto e Semântica:** Assim como BoW, TF-IDF não captura o significado semântico das palavras ou a ordem das palavras em uma frase, tratando cada palavra de forma independente [34].
  - **Ignora Relações Semânticas:** Palavras sinônimas ou relacionadas semanticamente são tratadas como distintas, mesmo que contribuam para o mesmo conceito.
  - **Problema de "Zero-Value":** Novos termos que não estavam presentes no corpus de treinamento terão um IDF de zero, resultando em um TF-IDF de zero, o que os torna inúteis para o modelo [35].

N. Outras Representações Básicas: N-grams

- **N-grams:** São sequências contíguas de 'n' itens (palavras ou caracteres) de um dado texto ou fala [36].
  - **Unigram:** Um único token (o mesmo que BoW).
  - **Bigram:** Uma sequência de duas palavras (e.g., "muito bom").
  - **Trigram:** Uma sequência de três palavras (e.g., "não muito bom").
- **Princípios:** Ao incluir sequências de palavras, os N-grams podem capturar um certo nível de contexto e ordem que BoW e TF-IDF perdem. Por exemplo, o bigram "não gosto"tem um significado muito diferente de "gosto não", e os N-grams conseguem diferenciar isso.
- **Vantagens:** Melhoram a capacidade de capturar o contexto local e padrões de frase, o que é crucial para tarefas como análise de sentimento e modelagem de linguagem. Reduzem a ambiguidade de palavras com múltiplos significados ao considerar seu contexto imediato [37].
- **Desvantagens:**
  - **Alta Dimensionalidade e Esparsidade:** O número de N-grams únicos pode ser exponencialmente maior que o número de palavras únicas, levando a vetores de altíssima dimensionalidade e ainda mais esparsos, o que aumenta a complexidade computacional e de armazenamento.
  - **Falta de Compreensão Semântica Profunda:** Embora capturem contexto local, os N-grams ainda são "cegos"à semântica mais profunda e às relações de longo alcance entre as palavras.

III. Metodologia

Nesse artigo, a pipeline de execução foi definida como mostra na figura a seguir:

Primeiramente é feita a coleta dos dados com a Application Programming Interface (API) Twikit [38] para a coleta dos dados que foram utilizados para o data-set (DS) utilizado na elaboração do PLN. Esses dados então seguiram para o pré-processamento, onde ficaram limpos e formatados para aprimorar o processo de ML. Em seguida, foi utilizada a API

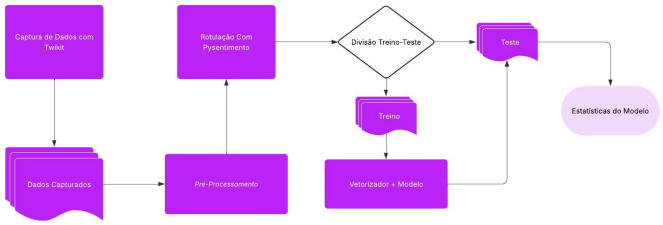

Figura 1. Fluxograma Utlizado Para Cronograma

da biblioteca Pysentimento [39] com a chamada 1 para rotular os tweets coletados, e salva esses dados em um data-frame (DF) para uma melhor manipulação desses dados no script.

Listing 1. Chamada da API Pysentimento

```
import pandas as pd
import re
import os
import torch
from pysentimiento import create_analyzer

print("\nCarregando o modelo de sentimento
     'bertweet-pt-sentiment'...")
    analyzer = create_analyzer(task="sentiment", lang="pt")

    print(f"\nAnalisando {len(df)} tweets... Por favor,
         aguarde.")
    textos_para_analise = df['texto_limpo'].tolist()

    # O analyzer processa a lista inteira de uma vez
    previsoes = analyzer.predict(textos_para_analise)

    sentimentos_finais = [MAPA_SENTIMENTO.get(p.output)
         for p in previsoes]
    df['sentimento'] = sentimentos_finais
    print("áAnlise de sentimento íconcluda!")

    print(f"\nSalvando resultados no arquivo
         '{ARQUIVO_SAIDA}'...")
    df_final = df[['texto_limpo', 'sentimento',
         'Query']].copy()

    # Salva o arquivo final
    df_final.to_csv(ARQUIVO_SAIDA, index=False,
         encoding='utf-8')

    print("\n--- PROCESSO ÍCONCLUDO COM SUCESSO! ---")
    print(f"O arquivo '{ARQUIVO_SAIDA}' áest pronto para
         ser usado no treinamento do seu modelo.")
    print("\çãnDistribuio dos sentimentos encontrados:")
    print(df_final['sentimento'].value_counts())
```

Após esse pré-processamento dos dados que serão utilizados no treinamento, os dados ficaram rotulados da maneira como mostra na Figura 2:

Com esses dados já rotulados e limpos, pode-se dar início ao treinamento dos modelos, como mostra a Tabela I.

Vetorizador / Classificador	CountVectorizer	TFIDF Vectorizer
LogisticRegression	X	X
PassiveAgressiveClassifier	X	X
Random Forest	X	X
SGDCClassifier	X	X
SVC	X	X

Tabela I
TABELA DE VETORIZADOR X CLASSIFICADOR

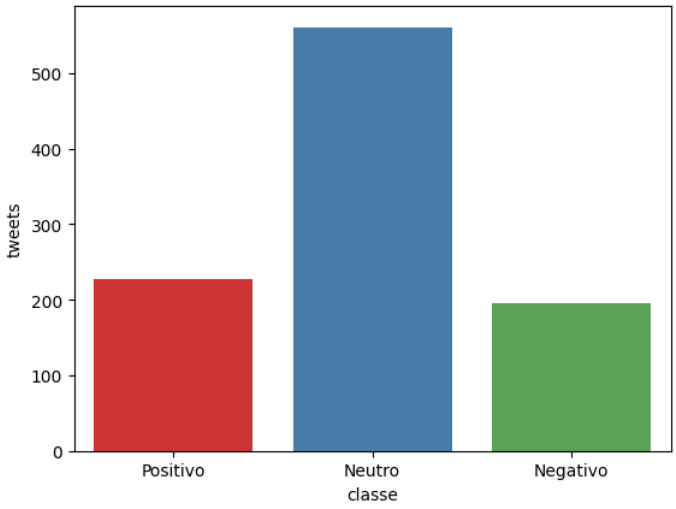

Figura 2. Gráfico de Rótulos no DS

A. *Dados Reduzidos*

Em primeira análise foi realizado o treinamento com os dados sem nenhuma modificação em sua proporção original, como mostra a Tabela II

	Quantidade
Positivo	**228**
Negativo	**196**
Neutro	**561**

Tabela II
TABELA DE CONTAGEM DE RÓTULOS

em seguida foram divididos entre os dados que foram utilizados em treinamento e os usados, em seguida mostra o relatório de cada modelo, e ao final faz um ranking decrescente dos valores de "$f_1 score$"

Listing 2. Chamada de treinamento do Modelo

```
import pandas as pd
import re
import os
import torch
from sklearn.model_selection import train_test_split,
    cross_val_predict
from sklearn.pipeline import Pipeline;
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score,
    f1_score, accuracy_score
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix


#vetorizadores
from sklearn.feature_extraction.text import
    CountVectorizer, TfidfVectorizer

#classificadores
from sklearn.linear_model import LogisticRegression,
    PassiveAggressiveClassifier, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from nltk.tokenize import TweetTokenizer

tweet_tokenizer = TweetTokenizer()

def escolhe_pipeline(pipe):
    match pipe:
        case 1:
            modelo1 = Pipeline([
                ('countVectorizer',
                    CountVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo',
                    LogisticRegression(solver='liblinear'))
            ])
            return modelo1
        case 2:
            modelo2 = Pipeline([
                ('TfidfVectorizer',
                    TfidfVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo',
                    LogisticRegression(solver='liblinear'))
            ])
            return modelo2
        case 3:
            modelo3 = Pipeline([
                ('countVectorizer',
                    CountVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', PassiveAggressiveClassifier())
            ])
            return modelo3
        case 4:
            modelo4 = Pipeline([
                ('TfidfVectorizer',
                    TfidfVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', PassiveAggressiveClassifier())
            ])
            return modelo4
        case 5:
            modelo5 = Pipeline([
                ('countVectorizer',
                    CountVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', RandomForestClassifier())
            ])
            return modelo5
        case 6:
            modelo6 = Pipeline([
                ('TfidfVectorizer',
                    TfidfVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', RandomForestClassifier())
            ])
            return modelo6
        case 7:
            modelo7 = Pipeline([
                ('countVectorizer',
                    CountVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', SGDClassifier(max_iter=1000))
            ])
            return modelo7
        case 8:
            modelo8 = Pipeline([
                ('TfidfVectorizer',
                    TfidfVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', SGDClassifier(max_iter=1000))
            ])
            return modelo8
        case 9:
            modelo9 = Pipeline([
                ('countVectorizer',
                    CountVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', SVC())
            ])
            return modelo9
        case 10:
            modelo10 = Pipeline([
                ('TfidfVectorizer',
                    TfidfVectorizer(analyzer='word',
                    tokenizer=tweet_tokenizer.tokenize)),
                ('modelo', SVC())
            ])
            return modelo10
```

```
def matriz_confusao(y_teste, modelo_predicao):
    print('óRelatrio de çãClassificao:\n',
          metrics.classification_report(y_teste,
          modelo_predicao))

    print("áAcurcia: {:.4f}\n",
          format(accuracy_score(y_teste, modelo_predicao)))

    print("Matriz de ãConfuso:\n", pd.crosstab(y_teste,
          modelo_predicao, rownames=['Real'],
          colnames=['Predito'], margins=True), '')

    mc = confusion_matrix(y_target=y_teste,
          y_predicted=modelo_predicao, binary=False)

    fig, ax = plot_confusion_matrix(conf_mat=mc)
    plt.show()
    print('\n')

x = tweet_df['texto_limpo']
y = tweet_df['sentimento']

x.shape, y.shape

#separar entre treino e teste
x_treino, x_teste, y_treino, y_teste = train_test_split(x,
     y, test_size=0.3, random_state=123)

x_treino.shape, x_teste.shape, y_treino.shape,
     y_teste.shape

relatorios=[]
modelos=[]

for i in range(1, 11):
    tipo_modelo = escolhe_pipeline(i)
    nome = str(tipo_modelo)
    modelo = tipo_modelo.fit(x_treino, y_treino)
    modelo_pred = modelo.predict(x_teste)

    relatorios.append(metrics.classification_report(y_teste,
          modelo_pred, output_dict=True))
    modelos.append(nome)

    print("*"*132)
    print(f'Modelo: {nome}\n')
    print("AVALIANDO MODELO\n")
    matriz_confusao(y_teste, modelo_pred)

    print("\ÇÃnVALIDAO CRUZADA\n")
    validacao_cruzada = cross_val_predict(modelo, x, y,
          cv=10)

colunas = ['modelo', 'f1_score', 'accuracy_score']
relatorio_df = pd.DataFrame(columns=colunas)
for i in range(len(relatorios)):
    relatorio_df.loc[i] = [modelos[i],
          round(relatorios[i]['weighted avg']['f1-score'],
          2), round(relatorios[i]['accuracy'], 2)]

def formatar_nome_modelo(modelo):
    vec = modelo[18:33]

    ini_nome = (modelo.rfind('modelo')+9)
    fim_nome = (modelo.rfind('('))

    classif = modelo[ini_nome: fim_nome]
    return(f'{classif} - {vec}')

relatorio_df['modelo'] =
     relatorio_df['modelo'].apply(formatar_nome_modelo)
relatorio_df.sort_values(by='f1_score', ascending=False)
```

B. *Undersampling*

Em seguida, os dados foram ”cortados”para igualar o menor valor rotulado, a diferença no código fica como mostra aqui:

Listing 3. Chamada de treinamento do Modelo Com Undersampling

```
from sklearn.svm import SVC
from nltk.tokenize import TweetTokenizer

[...] óCdigo se émantm igual
#equilibrando dados

sentimentos = []

sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
     "Neutro"]['sentimento'].count())
sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
     "Positivo"]['sentimento'].count())
sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
     "Negativo"]['sentimento'].count())

sentimentos.sort()
sentimentos

contagens_sentimento =
     tweet_df['sentimento'].value_counts()

df_positivo = tweet_df[tweet_df['sentimento'] ==
     'Positivo'][['texto_limpo', 'sentimento']].copy()
df_neutro = tweet_df[tweet_df['sentimento'] ==
     'Neutro'][['texto_limpo', 'sentimento']].copy()
df_negativo = tweet_df[tweet_df['sentimento'] ==
     'Negativo'][['texto_limpo', 'sentimento']].copy()

lista_classes = [
    (len(df_positivo), 'Positivo', df_positivo),
    (len(df_neutro), 'Neutro', df_neutro),
    (len(df_negativo), 'Negativo', df_negativo)
]

lista_classes_ordenada = sorted(lista_classes, key=lambda
     x: x[0])

minoria = lista_classes_ordenada[0][2] # O DataFrame da
     classe áminoritria
meio = lista_classes_ordenada[1][2]    # O DataFrame da
     classe áintermediria
maioria = lista_classes_ordenada[2][2] # O DataFrame da
     classe ámajoritria

print(f"Maioria\n{maioria['sentimento'].count()}\n")
print(f"Meio\n{meio['sentimento'].count()}\n")
print(f"Minoria\n{minoria['sentimento'].count()}\n")

maior_menor = resample(maioria, replace=True,
     n_samples=len(minoria), random_state=123)
meio_menor = resample(meio, replace=True,
     n_samples=len(minoria), random_state=123)

tweet_df_equilibrado_over = pd.concat([minoria,
     meio_menor, maior_menor])

tweet_df_equilibrado_over =
     tweet_df_equilibrado_over.reset_index()

tweet_df_equilibrado_over.drop(columns=['index'],
     inplace=True)

tweet_df_equilibrado_over.groupby(['sentimento']).count()

x = tweet_df_equilibrado_over['texto_limpo']
y = tweet_df_equilibrado_over['sentimento']ó

Cdigo segue igual [...]

relatorio_df['modelo'] =
     relatorio_df['modelo'].apply(formatar_nome_modelo)
relatorio_df.sort_values(by='f1_score', ascending=False)
```

C. *Oversampling*

Em última instância todos os rótulos são equalizados com base no maior rótulo reincidente, assim como mostra o código:

Listing 4. Chamada de treinamento do Modelo Com Oversampling

```
from sklearn.svm import SVC
from nltk.tokenize import TweetTokenizer

[...] óCdigo se émantm igual
```

```
#equilibrando dados

sentimentos = []

sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
    "Neutro"]['sentimento'].count())
sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
    "Positivo"]['sentimento'].count())
sentimentos.append(tweet_df.loc[tweet_df['sentimento'] ==
    "Negativo"]['sentimento'].count())

sentimentos.sort()
sentimentos

contagens_sentimento =
    tweet_df['sentimento'].value_counts()

df_positivo = tweet_df[tweet_df['sentimento'] ==
    'Positivo'][['texto_limpo', 'sentimento']].copy()
df_neutro = tweet_df[tweet_df['sentimento'] ==
    'Neutro'][['texto_limpo', 'sentimento']].copy()
df_negativo = tweet_df[tweet_df['sentimento'] ==
    'Negativo'][['texto_limpo', 'sentimento']].copy()

lista_classes = [
    (len(df_positivo), 'Positivo', df_positivo),
    (len(df_neutro), 'Neutro', df_neutro),
    (len(df_negativo), 'Negativo', df_negativo)
]

lista_classes_ordenada = sorted(lista_classes, key=lambda
    x: x[0])

minoria = lista_classes_ordenada[0][2] # O DataFrame da
    classe áminoritria
meio = lista_classes_ordenada[1][2]    # O DataFrame da
    classe áintermediria
maioria = lista_classes_ordenada[2][2] # O DataFrame da
    classe ámajoritria

print(f"Maioria\n{maioria['sentimento'].count()}\n")
print(f"Meio\n{meio['sentimento'].count()}\n")
print(f"Minoria\n{minoria['sentimento'].count()}\n")

from sklearn.utils import resample

menor_maior = resample(minoria, replace=True,
    n_samples=len(maioria), random_state=123)
meio_maior = resample(meio, replace=True,
    n_samples=len(maioria), random_state=123)

tweet_df_equilibrado_over = pd.concat([menor_maior,
    meio_maior, maioria])

tweet_df_equilibrado_over =
    tweet_df_equilibrado_over.reset_index()

tweet_df_equilibrado_over.drop(columns=['index'],
    inplace=True)

tweet_df_equilibrado_over.groupby(['sentimento']).count()
x = tweet_df_equilibrado_over['texto_limpo']
y = tweet_df_equilibrado_over['sentimento']

x.shape, y.shape

#separar entre treino e teste
x_treino, x_teste, y_treino, y_teste = train_test_split(x,
    y, test_size=0.3, random_state=123)

x_treino.shape, x_teste.shape, y_treino.shape,
    y_teste.shapeó

Cdigo segue igual [...]

relatorio_df['modelo'] =
    relatorio_df['modelo'].apply(formatar_nome_modelo)
relatorio_df.sort_values(by='f1_score', ascending=False)
```

IV. Resultados e Discussão

Após todas as etapas descritas na Figura 1 terem sido executadas, foi possível observar as matrizes de confusão e o relatório de cada abordagem de treinamento, descritas na Tabela I. Assim, possibilitando uma análise comparativa entre elas. Para a coleta dos dados, foi utilizado um $array$ de $Querys$ de 5 times brasileiros diferentes: $Corinthians$, $Flamengo$, $Gremio$, $Palmeiras$ e $SaoPauloFC$, com o intuito de ter uma variedade, mesmo que baixa, em uma visão nacional da língua portuguesa, de dialetos, a fim de uma maior área de reconhecimento dos modelos gerados.

A. Dados Reduzidos

A análise com os dados reduzidos gerou os modelos com as especificações seguindo a Tabela III, mostrando que o modelo que melhor performou foi o classificador $LogisticRegression$ com o vetorizador $countVectorizer$.

modelo	$f_1 score$	$accuracy\ score$
LogisticRegression - countVectorizer	0.62	0.64
PassiveAggressiveClassifier - TfidfVectorizer	0.61	0.62
SGDClassifier - TfidfVectorizer	0.61	0.61
SGDClassifier - countVectorizer	0.59	0.60
PassiveAggressiveClassifier - countVectorizer	0.59	0.59
RandomForestClassifier - countVectorizer	0.58	0.64
RandomForestClassifier - TfidfVectorizer	0.54	0.61
LogisticRegression - TfidfVectorizer	0.51	0.60
SVC - TfidfVectorizer	0.51	0.61
SVC - countVectorizer	0.47	0.59

Tabela III
Tabela de Resultados com Dados Reduzidos

A Regressão Logística com CountVectorizer obteve o melhor desempenho em F1-Score (0.62). Isso sugere que, para a distribuição original dos dados, a combinação da simplicidade da contagem de palavras com a linearidade da Regressão Logística foi eficaz.

PassiveAggressive Classifier e SGDClassifier (ambos online learning) apresentaram resultados competitivos, especialmente com TF-IDF, indicando sua robustez mesmo com datasets desbalanceados.

Os modelos baseados em árvore como Random Forest, apesar de geralmente poderosos, tiveram um F1-Score intermediário (0.58 com CountVectorizer e 0.54 com TfidfVectorizer). A acurácia mais alta para Random Forest com CountVectorizer (0.64) em comparação com seu F1-Score (0.58) pode ser um indício de que ele previu a classe majoritária com mais frequência. SVM (SVC) obteve os piores F1-Scores neste cenário (0.47 com CountVectorizer e 0.51 com TfidfVectorizer), o que pode indicar sensibilidade à distribuição original dos dados ou a necessidade de otimização de hiperparâmetros mais profunda.

B. Undersampling

A análise com $Undersampling$ contou com dados seguindo a Tabela IV gerou os modelos com as especificações seguindo a Tabela V, mostrando que o modelo que melhor performou foi o classificador $SGDClassifier$ com o vetorizador $TfidfVectorizer$.

	Quantidade
Positivo	**196**
Negativo	**196**
Neutro	**196**

Tabela IV
CONTAGEM DE RÓTULOS COM UNDERSAMPLING

modelo	$f_1 score$	$accuracy$
SGDClassifier - TfidfVectorizer	0.70	0.70
SGDClassifier - countVectorizer	0.68	0.68
PassiveAggressiveClassifier - TfidfVectorizer	0.67	0.67
LogisticRegression - TfidfVectorizer	0.67	0.67
SVC - TfidfVectorizer	0.67	0.67
LogisticRegression - countVectorizer	0.64	0.64
PassiveAggressiveClassifier - countVectorizer	0.63	0.63
RandomForestClassifier - countVectorizer	0.61	0.61
SVC - countVectorizer	0.60	0.60
RandomForestClassifier - TfidfVectorizer	0.59	0.59

Tabela V
TABELA DE RESULTADOS COM UNDERSAMPLING

O SGDClassifier com TfidfVectorizer emergiu como o modelo de melhor desempenho, atingindo um F1-Score e Accuracy de 0.70. Isso é significativo, pois o balanceamento permitiu que o SGD, eficiente para grandes datasets, aprendesse melhor as nuances das classes minoritárias.

Modelos como PassiveAggressive Classifier, Logistic Regression e SVC, todos com TfidfVectorizer, também mostraram melhorias notáveis, atingindo F1-Scores de 0.67. Isso reforça a ideia de que o TF-IDF, ao ponderar a importância dos termos, complementa bem o balanceamento de classes. A melhora mais evidente é observada no F1-Score, que é mais sensível ao balanceamento de classes do que a acurácia. O aumento geral nos F1-Scores para as classes minoritárias indica que os modelos estão agora mais aptos a identificá-las corretamente.

C. Oversampling

A análise com $Oversampling$ contou com dados seguindo a Tabela VI gerou os modelos com as especificações seguindo a Tabela VII, mostrando que o modelo que melhor performou foi o classificador SVC com o vetorizador $TfidfVectorizer$.

	Quantidade
Positivo	**562**
Negativo	**562**
Neutro	**562**

Tabela VI
CONTAGEM DE RÓTULOS COM OVERSAMPLING

O SVC com TfidfVectorizer alcançou um F1-Score e Accuracy impressionantes de 0.92. Isso sugere que a redução do dataset para um tamanho balanceado, combinada com a capacidade do SVM de encontrar hiperplanos ótimos em espaços de alta dimensionalidade, foi extremamente eficaz.

Random Forest também demonstrou um desempenho excepcional (0.90 com ambos os vetorizadores), indicando que a remoção de exemplos da classe majoritária que poderiam atuar como ruído ou dificultar o aprendizado das classes minoritárias beneficiou significativamente os modelos baseados em árvores.

modelo	$f_1 score$	$accuracy\ score$
SVC - TfidfVectorizer	0.92	0.92
RandomForestClassifier - TfidfVectorizer	0.90	0.90
RandomForestClassifier - countVectorizer	0.90	0.90
PassiveAggressiveClassifier - TfidfVectorizer	0.88	0.89
SGDClassifier - TfidfVectorizer	0.88	0.89
PassiveAggressiveClassifier - countVectorizer	0.88	0.88
SGDClassifier - countVectorizer	0.87	0.87
LogisticRegression - countVectorizer	0.87	0.87
LogisticRegression - TfidfVectorizer	0.84	0.84
SVC - countVectorizer	0.83	0.83

Tabela VII
TABELA DE RESULTADOS COM OVERSAMPLING

Todos os modelos testados no Undersampling apresentaram F1-Scores acima de 0.83, um salto significativo em relação ao cenário original.

D. Influência dos Tipos de Classificadores e Vetorizadores

Os resultados experimentais demonstram claramente como a estratégia de balanceamento do dataset e, em menor grau, a escolha do vetorizador e do classificador, influenciam o desempenho final dos modelos.

- **Impacto do Balanceamento de Classes:**
  - O **Oversampling** mostrou ser a estratégia mais eficaz para este dataset específico, levando a ganhos substanciais e consistentes em todas as métricas para a maioria dos modelos (aumento de F_1-Score de até 0.45 para SVC com TfidfVectorizer em comparação com o cenário original). Este resultado pode ser atribuído à remoção de redundâncias ou ruídos da classe majoritária que estavam dificultando o aprendizado e a generalização dos modelos. Contudo, é importante considerar a potencial perda de informações valiosas que uma subamostragem agressiva pode acarretar em datasets maiores.
  - O **Undersampling** também gerou melhorias consistentes em comparação com o cenário original (aumento de F_1-Score de até 0.08 para SGDClassifier com TfidfVectorizer), indicando que a criação de amostras sintéticas para as classes minoritárias ajudou os modelos a aprenderem padrões mais generalizáveis sem a perda de dados.
- **Influência dos Vetorizadores (CountVectorizer vs. TfidfVectorizer):**
  - Em geral, o **TfidfVectorizer** tendeu a apresentar resultados ligeiramente melhores ou comparáveis ao CountVectorizer na maioria dos modelos, especialmente nos cenários balanceados (Oversampling e Undersampling). Isso é esperado, pois o TF-IDF pondera a importância dos termos, atribuindo menor peso a palavras muito comuns e maior peso a termos mais distintivos, o que é benéfico para a discriminação de sentimentos.
  - A diferença de desempenho entre os vetorizadores não é tão acentuada quanto o impacto do balanceamento de classes, mas o TF-IDF frequentemente se

mostra uma escolha sólida para representação textual em tarefas de Análise de Sentimentos.

- **Desempenho dos Classificadores:**
  - **SGDClassifier e PassiveAggressive Classifier:** Destacaram-se como os melhores desempenhos nos cenários balanceados (Oversampling e Undersampling), mostrando sua eficiência e boa performance com volumes de dados alterados ou grandes.
  - **SVC:** Embora tenha tido um desempenho mediano no cenário original (F_1-Score de 0.47) , o SVC teve uma melhora notável e se tornou o melhor classificador no cenário de Undersampling (F_1-Score de 0.92). Isso reforça sua capacidade de encontrar limites de decisão ótimos em espaços de alta dimensionalidade quando os dados são bem distribuídos e não enviesados.
  - **Random Forest:** Também apresentou uma performance excelente no cenário de Undersampling (F_1-Score de 0.90), o que pode indicar que, com um dataset mais limpo e balanceado, as árvores de decisão conseguem capturar padrões complexos de forma mais eficaz.
  - **Regressão Logística:** Apesar de ser o melhor no cenário original, seu desempenho, embora melhorando com o balanceamento, foi superado por outros modelos mais robustos nos cenários de Oversampling e Undersampling.

V. Aplicação Futura

Esse trabalho serviu de base para a criação de modelos significativamente bem treinados para AS em tweets sobre jogos de futebol. Uma maneira de aplicar seria buscar o melhor classificado entre os modelos, e fazer uma análise por exemplos do retrospecto dos comentários na rede social sobre o desempenho do time em algumas partidas, utilizando-se de aparatos da API da Biblioteca [40], por exemplo.

VI. Conclusão

Este estudo realizou uma pesquisa aprofundada sobre os fundamentos teóricos e práticos do PLN aplicado à AS, com um foco particular na utilização de classificadores baseados em ML tradicional. Abordamos desde as definições e tipologias da AS, seus desafios intrínsecos e aplicações em diversos domínios, até as etapas cruciais de pré-processamento de texto, representação vetorial e as especificidades dos modelos de classificação e suas métricas de avaliação. As ferramentas e bibliotecas empregadas, como `twikit`, `pysentimiento`, `pandas`, `scikit-learn`, `nltk` e `spaCy`, foram detalhadas em seus respectivos papéis no pipeline analítico.

Os resultados experimentais demonstraram a criticidade do balanceamento de classes para o desempenho dos modelos de Análise de Sentimentos. No cenário original ("Reduzido"), caracterizado por um desbalanceamento acentuado (561 Neutros, 228 Positivos, 196 Negativos), os modelos apresentaram um desempenho moderado, com a Regressão Logística com CountVectorizer obtendo o melhor F_1-Score de 0.62. No entanto, a aplicação de estratégias de reamostragem alterou significativamente o panorama.

O cenário de **Oversampling** revelou-se particularmente eficaz, elevando o desempenho de todos os classificadores de forma drástica. Nele, as classes foram balanceadas para ter 196 amostras cada, e o modelo SVC combinado com TfidfVectorizer alcançou um impressionante F_1-Score de 0.92 e acurácia de 0.92, demonstrando a capacidade dos modelos em operar com alta eficácia quando as classes estão balanceadas pela expansão do dataset. O Random Forest também exibiu resultados excelentes neste cenário (F_1-Score de 0.90).

Por outro lado, o cenário de **Undersampling** também gerou melhorias consistentes em relação ao cenário original, com as classes balanceadas para 561 amostras cada. Neste contexto, o SGDClassifier com TfidfVectorizer atingiu um F_1-Score de 0.70, provando ser uma estratégia válida para mitigar o viés do modelo pela redução da classe majoritária, embora possa implicar em perda de informação.

A influência dos vetorizadores, embora presente, foi secundária ao impacto do balanceamento. O TF-IDF geralmente apresentou uma leve vantagem sobre o CountVectorizer em cenários balanceados, validando sua eficácia em ponderar a relevância dos termos em contextos textuais esparsos. Classificadores como SVC e Random Forest mostraram sua força e capacidade de generalização em ambientes onde o Oversampling foi aplicado, enquanto SGDClassifier e PassiveAggressive Classifier se destacaram no cenário de Undersampling.

Em síntese, este trabalho reforça que a simples aplicação de classificadores de ML tradicional a dados textuais brutos pode não ser suficiente para obter resultados ótimos em Análise de Sentimentos, especialmente diante de desbalanceamento de classes. A etapa de balanceamento de dados surge como um pilar fundamental para maximizar a capacidade preditiva desses modelos. Os achados servem como uma base robusta para futuras investigações, direcionando esforços para a a otimização de hiperparâmetros, exploração de técnicas híbridas de balanceamento e a integração com modelos de linguagem pré-treinados, visando aplicações ainda mais sofisticadas e em tempo real da Análise de Sentimentos.

Referências

[1] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proc. 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2004, [Online; acessado em 2025-06-29]. [Online]. Available: https://www.cs.uic.edu/~liub/publications/kdd04-revSummary.pdf

[2] A. F. S. Zorron, "Análise de sentimentos em textos em português," Trabalho de Conclusão de Curso, Universidade de Brasília, 2023, [Online; acessado em 2025-06-29]. [Online]. Available: https://bdm.unb.br/bitstream/10483/37005/1/2023_ArturFSZorron_tcc.pdf

[3] Universidade Tecnológica Federal do Paraná (UTFPR), "Classificação de sentimentos em textos em português usando aprendizado de máquina," [Online], [Acessado em 2025-06-29]. [Online]. Available: https://repositorio.utfpr.edu.br/jspui/handle/1/10764

[4] L. B. Souza, R. A. Piao, and E. M. Matos, "Análise de sentimentos em tweets: Um estudo comparativo de técnicas de classificação," *Anais do VIII Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*, 2016. [Online]. Available: https://lume.ufrgs.br/bitstream/handle/10183/140910/000991520.pdf

[5] N. M. G. Silva, "Análise de sentimentos em redes sociais: Abordagens e desafios," Master's thesis, Universidade Federal de Pernambuco (UFPE), 2016. [Online]. Available: https://repositorio.ufpe.br/bitstream/123456789/11441/1/dissertacao_nelson.pdf

[6] A. Alhashmi, A. Al-Ayash, M. Al-Wajeeh, and A. Abu-Shaikha, "A review on sentiment analysis and emotion detection from text," *PMC (PubMed Central)*, 2021. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC8402961/

[7] "Sentiment analysis," https://en.wikipedia.org/wiki/Sentiment_analysis, 2023.

[8] Hex, "Vader sentiment analysis," https://hex.tech/templates/sentiment-analysis/vader-sentiment-analysis/.

[9] P. Calderon, "Vader sentiment analysis explained," *Medium*, 2023. [Online]. Available: https://medium.com/@piocalderon/vader-sentiment-analysis-explained-f1c4f9101cd9

[10] TextBlob, "Textblob documentation," https://textblob.readthedocs.io/.

[11] P. Libraries, "Text analysis with python: Textblob," https://guides.library.upenn.edu/penntdm/python/textblob.

[12] N. Desk, "Textblob for sentiment analysis," https://www.nectardesk.com/blog/textblob-for-sentiment-analysis/.

[13] A. Vidhya, "Beginner nlp product sentiment analysis with textblob," https://www.analyticsvidhya.com/blog/2022/01/beginner-nlp-product-sentiment-analysis-with-textblob/.

[14] MoldStud, "Textblob for sentiment analysis," https://moldstud.com/textblob-for-sentiment-analysis/.

[15] DataCamp, "What is tokenization in nlp?" https://www.datacamp.com/blog/what-is-tokenization-in-nlp.

[16] Coursera, "Tokenization in nlp," https://www.coursera.org/articles/tokenization-in-nlp.

[17] Kaggle, "Beginner nlp: Product sentiment analysis (textblob)," https://www.kaggle.com/code/blessondensil294/beginner-nlp-product-sentiment-analysis-textblob.

[18] M. Sardana, "Text preprocessing for nlp," *Medium*, 2022. [Online]. Available: https://medium.com/@mohit.sardana/text-preprocessing-for-nlp-906fddb2e2d9

[19] AIMultiple, "Sentiment analysis challenges & how to overcome them," https://research.aimultiple.com/sentiment-analysis-challenges/.

[20] R. World, "10 challenges of sentiment analysis and how to overcome them - part 2," https://researchworld.com/articles/10-challenges-of-sentiment-analysis-and-how-to-overcome-them-part-2.

[21] S. North, "Stemming and lemmatization in nlp," https://seonorth.ca/pt-br/nlp/stemming-and-lemmatization/.

[22] IBM, "Stemming and lemmatization," https://www.ibm.com/br-pt/think/topics/stemming-lemmatization.

[23] G. Davedovicz, "Nlp para iniciantes: Lematização," *Medium*, 2023. [Online]. Available: https://medium.com/@guilherme.davedovicz/nlp-para-iniciantes-lematiza%C3%A7%C3%A3o-d3f723fa9ee3

[24] spaCy, "Models for portuguese," https://spacy.io/models/pt.

[25] Kaggle, "Text preprocessing in nlp," https://www.kaggle.com/code/abdmental01/text-preprocessing-nlp-steps-to-process-text.

[26] V. Agarwal, M. Shah, and M. Aggarwal, *Hands-On Natural Language Processing with Python*. O'Reilly Media, 2020. [Online]. Available: https://www.oreilly.com/library/view/hands-on-natural-language/9781838644337/ch01s03.html

[27] S. NLP, "Stop words cleaner," https://nlp.johnsnowlabs.com/docs/en/annotators#stopwordscleaner.

[28] A. Vidhya, "How to handle missing values in a dataset?" https://www.analyticsvidhya.com/blog/2021/04/how-to-handle-missing-values-in-a-dataset/.

[29] N. Kaur, "Handling missing values in text data," *Medium*, 2021. [Online]. Available: https://medium.com/@navdeepk25/handling-missing-values-in-text-data-274e1d520336

[30] IBM, "Bag-of-words," https://www.ibm.com/think/topics/bag-of-words.

[31] AIML.com, "What are the advantages and disadvantages of bag of words model?" https://aiml.com/what-are-the-advantages-and-disadvantages-of-bag-of-words-model/.

[32] N. Analytics, "Tf-idf: Comprehensive guide," https://www.numberanalytics.com/blog/tf-idf-comprehensive-guide.

[33] Semrush, "Tf-idf," https://www.semrush.com/blog/tf-idf/.

[34] BytePlus, "Tf-idf," https://www.byteplus.com/en/topic/400333.

[35] GeeksforGeeks, "Understanding tf-idf (term frequency-inverse document frequency)," https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/.

[36] Dremio, "N-grams in nlp," https://www.dremio.com/wiki/n-grams-in-nlp/.

[37] A. Vidhya, "What are n-grams and how to implement them in python?" https://www.analyticsvidhya.com/blog/2021/09/what-are-n-grams-and-how-to-implement-them-in-python/.

[38] Twikit, "Twikit: A modern, asynchronous python wrapper for the twitter api," [Online], [Acessado em 2025-06-29]. [Online]. Available: https://github.com/d60/twikit

[39] PyPI, "pysentimiento 0.5.2rc3," https://pypi.org/project/pysentimiento/0.5.2rc3/.

[40] twikit, "twikit documentation," https://twikit.readthedocs.io/en/latest/twikit.html.