CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
(CEFET-MG)

OMAR COSTA MARTINS

MACHINE LEARNING: CLASSIFICAÇÃO DE VINHOS BRANCOS POR MEIO DA APLICAÇÃO DE ÁRVORE DE DECISÃO

Belo Horizonte (MG)
2023

Omar Costa Martins

MACHINE LEARNING: CLASSIFICAÇÃO DE VINHOS BRANCOS POR MEIO DA APLICAÇÃO DE ÁRVORE DE DECISÃO

Trabalho de conclusão de curso apresentado como requisito parcial para a obtenção do título de Bacharel em Química Tecnológica.
Orientador: Prof. Dr. Cleverson Fernando Garcia.
Coorientadora: Profª Dra. Maria Aparecida Vieira Teixeira Garcia.

CEFET-MG
Belo Horizonte (MG)
2023

Omar Costa Martins

MACHINE LEARNING: CLASSIFICAÇÃO DE VINHOS BRANCOS POR MEIO DA APLICAÇÃO DE ÁRVORE DE DECISÃO

Trabalho de conclusão de curso do Bacharelado em
Química Tecnológica
CEFET-MG

Belo Horizonte, 24 de novembro de 2023

Prof. Dr. Cleverson Fernando Garcia
(Orientador – CEFET-MG)

Profª. Dra. Maria Aparecida Vieira Teixeira Garcia
(Coorientadora – FAFAR/UFMG)

Profª. Dra. Flávia Augusta Guilherme Gonçalves Resende
(avaliadora – CEFET-MG)

Prof. Me. Reginaldo Ferreira de Oliveira
(avaliador – CEFET-MG)

RESUMO

MARTINS, O. C.; GARCIA, C. F.; GARCIA, M. A. V. T. *Machine Learning*: Classificação de vinhos brancos por meio da aplicação de Árvore de Decisão.

A importância do vinho vai além do simples ato de beber. O vinho engloba áreas como a agricultura, o turismo, o comércio internacional e a cultura. A complexidade e a variedade associadas ao vinho o tornam um produto fascinante e economicamente substancial em escala global. Gerando emprego e renda diretamente ou indiretamente para várias famílias. Com o aumento da população as indústrias do setor alimentício tiveram grande expansão e o *Machine Learning* tem se tornado um grande aliado na análises de dados, ajudando tanto na análise de qualidade, logística, consumo entre outros. Este trabalho teve como objetivo a criação de um modelo de *Machine Learning* baseado em árvore de decisão que prevê a qualidade de vinhos brancos baseado em suas características físico-químicas. O modelo de árvore de decisão foi construído a partir da linguagem de programação Python, utilizando um conjunto de dados com 11 parâmetros físico-químicos (Variáveis preditoras) e sua classificação de qualidade sensorial (Variável resposta ou Classe) de vinhos brancos. Empregando técnicas de pré-processamento, como subamostragem via TomekLinks, reclassificação das classes de qualidade e remoção de *outliers*, o modelo foi treinado e testado, revelando uma acurácia média de treinamento de 0,913 e uma acurácia média de teste de 0,815 para o modelo com melhor acurácia com 100 processamentos. A análise detalhada destacou discrepâncias entre as categorias, evidenciando resultados piores, com base nas métricas, da Categoria 2 em comparação com a Categoria 1, um resultado previsto devido ao desbalanceamento nas amostras.

Palavras-chave: Classificação. Vinhos Brancos. Árvore de Decisão.

LISTA DE ILUSTRAÇÕES

FIGURAS

QUADROS

GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ML	*Machine Learning*
AI	Inteligência Artificial

SUMÁRIO

1 INTRODUÇÃO

É inegável que depois da Quarta Revolução Industrial, a quantidade de informações e dados teve um crescimento exponencial. Em 16 de dezembro de 2020, o volume de dados gerados diariamente em todo o mundo era de 59 zettabytes. Prevê-se que atinja 149 zettabytes em 2024 (NTI et al., 2022). Essa expansão notável ocorre em todas as áreas da ciência, comércio, saúde, entre vários outros. Embora o potencial desses conjuntos massivos de dados seja indiscutivelmente significativo, compreendê-los de maneira abrangente exige uma abordagem inovadora e a aplicação de novas técnicas de aprendizado para lidar com os diversos desafios que se apresentam (QIU et al, 2016).

A área de *Machine Learning* (ML) ou Aprendizado de Máquina, é uma área da Ciência da Computação dedicado a capacitar computadores a aprenderem sem uma programação direta. Originado a partir do movimento de inteligência artificial (IA) na década de 1950, o foco está em aplicações práticas, principalmente em previsão e otimização. Em vez de programação convencional, os computadores aprimoram seu desempenho em tarefas por meio da experiência (BI et al., 2019). O *Machine Learning* fornece técnicas que podem construir automaticamente um modelo computacional de relações complexas, processando os dados disponíveis e maximizando um critério de desempenho específicos para problema (BAŞTANLAR; OZUYSAL, 2014).

Um dos modelos de classificação do *Machine Learning* é a Árvore de Decisão, que divide o conjunto de dados em subconjuntos mais simples, com base nas características dos dados. Sua característica marcante são os resultados excepcionalmente interpretáveis que apresentam uma exibição intuitiva em forma de árvore, pois melhora a compreensão e a disseminação dos resultados. As origens computacionais das árvores de decisão são modelos de processos biológicos e cognitivos (DE VILLE, 2013).

A indústria de alimentos e bebidas pode se beneficiar consideravelmente com a aplicação de ferramentas de *Machine Learning* em diversas áreas. Por exemplo, no controle de qualidade, o uso de imagem e análises de dados pode ajudar a identificar e remover produtos defeituosos da linha de produção, economizando recursos e evitando desperdícios (ZHU et al., 2021; ZHOU et al., 2019). Para bebidas de alto valor agregado, como o vinho, a aplicação de *Machine Learning* pode fornecer métodos de avaliação da qualidade sensorial a partir de análises físico-químicas. Esse modelo desempenha um papel importante em complementar as avaliações de degustação feitas por enólogos, contribuindo para aprimorar o processo de produção de vinho. Além disso, abordagens semelhantes têm o potencial de impulsionar

estratégias de marketing direcionado, permitindo a criação de modelos que capturam as preferências dos consumidores em segmentos de mercado específicos (CORTEZ et al., 2009).

Assim, buscando contribuir no estudo de vinhos brancos, o presente trabalho tem como objetivo desenvolver um algoritmo de *Machine Learning*, baseado em Árvores de Decisão, para classificar sensorialmente os referidos vinhos, utilizando como base suas características físico-químicas.

2 REVISÃO BIBLIOGRÁFICA

2.1 Machine Learning

A expressão *Machine Learning* (ML), ou Aprendizado de Máquina, foi criada para designar uma área da Inteligência Artificial (IA) com o objetivo de desenvolver técnicas computacionais, como algoritmos e sistemas computacionais, capazes de adquirir conhecimento de forma automática (MONARD; BARANAUSKAS, 2003).

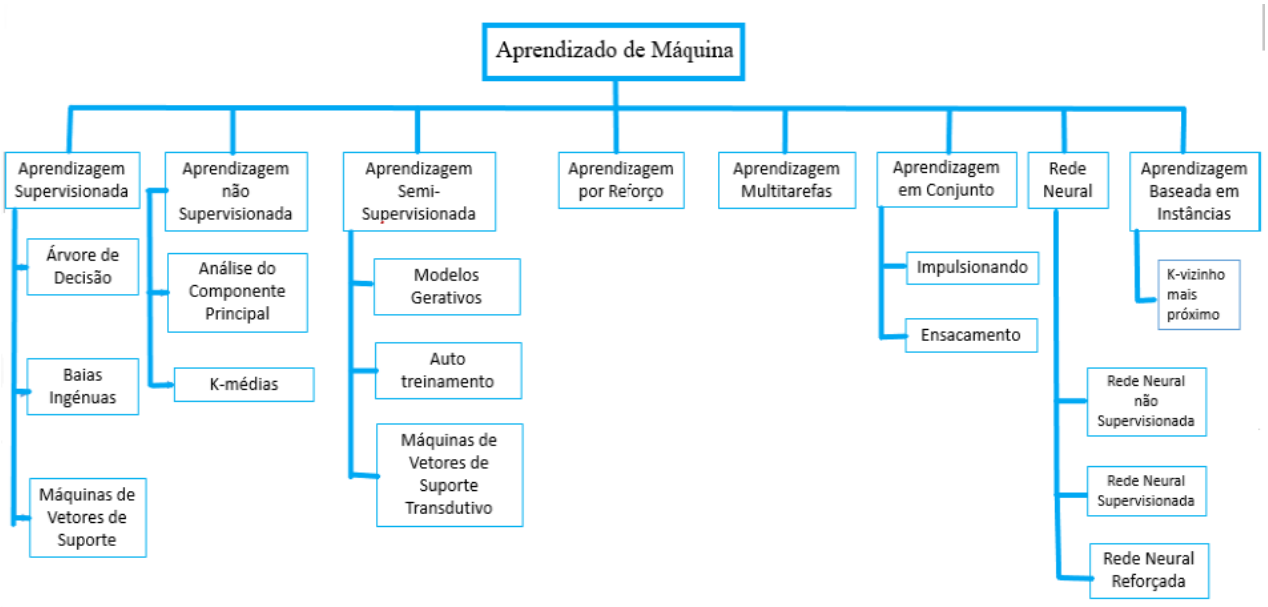
A aplicação de ML é vasta, abrangendo áreas como visão computacional, previsão, análise semântica, processamento de linguagem natural e recuperação de informação. A visão computacional lida com o reconhecimento, detecção e processamento de objetos. Na previsão, destacam-se subdomínios como classificação, análise e recomendação. O processamento de linguagem natural envolve a programação de computadores para entender e processar dados em linguagem natural. Já a recuperação de informação trata da busca de informações em documentos e bancos de dados de forma eficiente (SHINDE; SHAH, 2018).

Em geral, para criar um sistema de ML, é essencial definir claramente o problema de aprendizagem, identificando três fatores principais: a classe das tarefas, a medida de desempenho a ser aprimorada e a fonte da experiência. Para ilustrar esses fatores, Mitchell (1997) usou o exemplo de criar um programa para aprender a jogar damas e ganhar um torneio mundial. Um desafio foi escolher o tipo de experiência de treinamento para o sistema, como diferenciar quais jogadas levaram à vitória ou à derrota. Além disso, foi necessário estabelecer um grau de controle sobre as jogadas do programa e criar uma medida de desempenho confiável.

O *Machine Learning* é utilizado, ainda, para ensinar máquinas a lidarem com dados de forma eficiente, com o objetivo principal de aprender com os dados. Ele depende de diferentes algoritmos para resolver problemas de dados, não havendo um algoritmo melhor que outro, apenas algoritmos mais eficazes em situações específicas. Portanto, a escolha do algoritmo depende do problema a ser solucionado. Conforme ilustrado na Figura 1, existem vários subconjuntos de *Machine Learning* (MAHESH, 2020).

Os principais subgrupos do *Machine Learning* são o Aprendizado Supervisionado e o Aprendizado Não Supervisionado. O Aprendizado Supervisionado envolve a tarefa de aprender uma função que mapeia uma entrada para uma saída com base em exemplos de pares entrada-saída. Ele infere essa função a partir de dados de treinamento rotulados, que consistem em um conjunto de exemplos de treinamento (HAN; KAMBER; PEI, 2012).

Figura 1- Sistema organizacional relacionado ao *Machine Learning*



Fonte: Adaptado de Mahesh, 2020

Os algoritmos de MLSupervisionado necessitam de assistência externa. O conjunto de dados de entrada é dividido em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento possui uma variável de saída que precisa ser prevista ou classificada. Os algoritmos aprendem padrões a partir dos dados de treinamento e aplicam esses padrões ao conjunto de teste para fazer previsões ou classificações (MAHESH, 2020).

Por outro lado, na Aprendizagem Não Supervisionada não há um atributo alvo para orientar o processo. Os algoritmos são feitos para elaborar e apresentar estruturas eficientes. Algoritmos de Aprendizagem Não Supervisionada extraem características dos dados. Quando novos dados são introduzidos, esses algoritmos usam as características aprendidas anteriormente para reconhecer a classe dos dados (BERRY; MOHAMED; YAP, 2020). Os algoritmos de aprendizagem não supervisionada podem ser usados para criar rótulos nos dados que poderão ser usados para melhorar a eficiência de um algoritmo de Aprendizagem Supervisionada (HOFMANN, 2001).

O aprendizado por reforço tem seu algoritmo baseado em mapear situações, maximizando um sinal de reforço ou recompensa. Ou seja, essa técnica é baseada em tentativa e erro. Quando o algoritmo executa uma ação, essa ação pode ser classificada como boa ou ruim, e a partir dessa resposta, ele vai aprendendo. Não existem instruções de resultados corretos, apenas informações mais avaliativas do que instrucionais (BOADA; BOADA; LOPEZ, 2004).

Um dos algoritmos mais usados, mostrado na Figura 1, são as Máquinas de Vetores de Suporte. Esse método opera plotando dados de planejamento em um espaço de várias dimensões e tenta separar as classes usando um hiperplano. Se as classes não forem imediatamente distintas nesse espaço, o algoritmo aplicará medidas adicionais para melhor separá-las. Esse processo é repetido até que as informações de treinamento possam ser isoladas em suas duas classes distintas usando um hiperplano (KUMAR, 2016).

Já os algoritmos de Redes Neurais foram desenvolvidos para funcionar de maneira semelhante ao cérebro humano em termos de processamento de informações. Eles consistem em unidades de processamento simples que imitam dois aspectos do cérebro humano: a capacidade de aprender com o ambiente e as forças das conexões entre neurônios, chamadas de pesos sinápticos, que armazenam o conhecimento adquirido. Isso resulta em benefícios como não-linearidade, generalização, resposta a evidências em tarefas de classificação de padrões e tolerância a falhas (HAYKIN, 2001).

Como mencionado anteriormente, existe uma grande quantidade de algoritmos de *Machine Learning*. Portanto, são necessárias métricas para avaliar o desempenho do algoritmo. Nesse caso, o tipo de métrica utilizada também dependerá do problema analisado. O Quadro 1 evidencia uma matriz de confusão que representa os resultados da classificação de dados. Já no Quadro 2 são apresentadas as principais métricas utilizadas para avaliar o desempenho de algoritmos.

Quadro 1 – Representação geral de uma matriz de confusão

		Classe Prevista	
		Positivo	**Negativo**
Classe Real	**Positivo**	Verdadeiro positivo (VP)	Falso Negativo (FN)
	Negativo	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Fonte: Adaptado de Favan, 2015

O Quadro 1 apresenta quatro resultados possíveis: a quantidade de verdadeiros positivos (classificações corretas de classe positiva); a quantidade de verdadeiros negativos (classificações corretas de classe negativa); a quantidade de falsos positivos (classificações incorretas de classe positiva) e a quantidade de falsos negativos (classificações incorretas de classe negativa) (FAVAN, 2015). As métricas são calculadas a partir dos resultados na matriz de confusão. A acurácia é uma das métricas mais simples e importantes, calculando a

porcentagem de acertos do algoritmo. Há dois tipos de acurácia, a de treinamento e a de teste, relacionadas às bases de dados correspondentes (SHEDTHI; SIDDAPPA; SHETTY, 2019)

Quadro 2 – Principais métricas e suas equações

Método	**Fórmula**
Acurácia de Treinamento	(VP + VN)/Total
Acurácia de Previsão	(VP + VN)/Total
Precisão das Categorias (P)	VP/(VP + FP)
Revocação das Categorias (S)	VP/(VP + FN)
F1-Score	2 * (P x S)/(P + S)

Nota*: VP = Verdadeiro positivo; VN = Verdadeiro negativo; FP = Falso positivo; FN = Falso negativo; Total = quantidade total de resultados*

Fonte: Adaptado de Mariano (2021)

A precisão das categorias mostra a proporção das classificações corretas de determinada categoria em relação ao total de dados classificados nessa mesma categoria. A taxa de revocação refere-se à proporção das categorias corretamente classificadas e localizadas nos resultados retornados em relação ao total de categorias relacionadas (ZHANG; SONG; ZHANG, 2020). O F1-score é calculado a partir da precisão e da revocação, sendo uma média harmônica dessas duas métricas (MARIANO, 2021).

É importante entender quando cada métrica é mais apropriada para avaliar um algoritmo. Por exemplo, na criação de um sistema de detecção de spam, é crucial evitar detectar erroneamente uma mensagem legítima como spam, pois isso pode causar inconveniências ao usuário. Portanto, a métrica mais adequada para avaliar e comparar sistemas de detecção de spam é a precisão. Por outro lado, em um sistema de detecção de falhas em aeronaves, não identificar corretamente uma falha é extremamente sério. Assim, um falso negativo, ou seja, o sistema não identificar corretamente a falha, deve ser minimizado. Nesse contexto, a revocação é a métrica adequada para avaliar e comparar diferentes sistemas (MARIANO, 2021).

O primeiro passo para criar um bom algoritmo de ML e ter boas métricas é o tratamento dos dados. O refinamento dos dados antes do treinamento, em muitos casos, melhora o desempenho do algoritmo. Kamiri (2021) analisa artigos relacionados à *Machine Learning* e identifica uma metodologia comum na maioria deles: o pré-processamento dos dados, independentemente de o algoritmo ser de aprendizagem supervisionada, não supervisionada ou por reforço. Os principais tratamentos incluem limpeza de dados, normalização e redução de

ruídos. A limpeza de dados envolve preencher valores nulos ou eliminar valores quando o preenchimento não é possível. A normalização dos dados envolve a formatação para que o algoritmo compreenda melhor durante o treinamento, incluindo o escalonamento e a garantia de que os dados de treinamento e teste estejam no mesmo formato. A redução de ruído envolve a eliminação de valores discrepantes ou dados inconsistentes com as demais informações. Alguns desses artigos focam especificamente no pré-processamento de dados.

Um dos desafios encontrados em modelos de ML é o *overfitting*, devido à facilidade com que esse problema pode surgir se não forem tomadas precauções durante o processo de modelagem. Uma definição formal do problema é apresentada por Mitchell (1997): "Dado um modelo H, diz-se que H está sofrendo de *overfitting* no conjunto de dados de treinamento se existe outra hipótese H', na qual a taxa de erro de H no conjunto de dados de treinamento é menor que a de H', mas a taxa de erro de H' é menor que a de H quando aplicada ao conjunto de dados completo".

Isso significa que um modelo que apresenta alta precisão em um conjunto de dados de treinamento não garante necessariamente um desempenho igualmente bom em novas instâncias. Por exemplo, um classificador que alcança 100% de precisão no conjunto de treinamento, mas apenas 50% de precisão nas instâncias de teste, está exibindo sintomas de *overfitting* nos dados de treinamento. Comumente, o *overfitting* decorre da presença de ruídos nos dados de treinamento, mas também pode surgir devido a outras causas, como a inadequação do conjunto de treinamento em representar adequadamente a totalidade dos dados, ou ainda quando o conjunto de treinamento é insuficientemente amplo, entre outras razões. Assim, a seleção criteriosa dos dados de treinamento é fundamental para prevenir esse tipo de problema, embora, em algumas situações, essa precaução por si só possa não ser o suficiente.

Conforme o próprio nome sugere, o *underfitting* pode ser visto como um contraponto ao problema do *overfitting*. Enquanto um modelo com *overfitting* apresenta um desempenho excelente no conjunto de dados de treinamento, mas falha ao lidar com novas instâncias, um modelo que sofre de *underfitting* enfrenta dificuldades mesmo ao lidar com o conjunto de dados de treinamento em si. Nesse cenário, um modelo de *Machine Learning* com *underfitting* não consegue discernir os padrões subjacentes contidos no conjunto de dados de treinamento (DOMINGOS, 2012).

O *underfitting* geralmente ocorre quando o modelo produzido por um algoritmo não consegue se ajustar eficazmente às instâncias do conjunto de dados utilizado. Isso indica que o algoritmo escolhido não é apropriado para resolver o problema específico em questão. Por isso

deve-se escolher com cuidado os dados de treinamento, verificando se determinado algoritmo consegue solucionar o problema proposto (DOMINGOS, 2012).

Uma das soluções para lidar com o problema de *underfiting* e *overfitting* dos dados é podar a Árvore de Decisão após tê-la construído. Esse processo envolve a redução do número de nós internos, diminuindo a complexidade da árvore, ao mesmo tempo em que melhora o desempenho em relação à árvore original (MONARD; BARANAUSKAS, 2003).

Em um estudo comparativo, o trabalho de Ramalho et al. (2018) propõe um método automático de classificação de linguagens de programação em duas etapas. Primeiro, uma base de dados contendo as ocorrências de palavras nos trechos de código é gerada. Em seguida, os algoritmos de *Machine Learning* Naive Bayes, Redes Bayesianas, Árvore de Decisão (J48) e Redes Neurais Artificiais de Múltiplas Camadas são usados para classificar os trechos de código. Os resultados indicam que todos os algoritmos alcançaram uma acurácia superior a 95,4%, mas o desempenho está relacionado à construção da base de dados de palavras. Destaca-se que as Árvores de Decisão J48 alcançaram uma acurácia de 98,9% na classificação de linguagens de programação. Esse estudo ressalta a importância de uma base de dados sólida e que diferentes algoritmos de ML podem ser utilizados para a mesma tarefa, proporcionando resultados variados.

Em outro trabalho realizado por De Campos e Vasconcelos (2021), foi realizada uma revisão da literatura sobre artigos sobre aplicação de algoritmos de *Machine Learning* na área farmacêutica, no período de 2003 a 2021. Os pesquisadores concluirão que, o uso de algoritmos no desenvolvimento de medicamentos é fundamental. A árvore de decisão ajuda a antecipar desafios no processo. A regressão linear fornece estimativas seguras de estabilidade. *O Support Vector Machine* é valioso na classificação de dados. A regressão logística é essencial para modelar informações estatísticas. O Naive Bayes se baseia no teorema de Bayes, assumindo independência entre os recursos (variáveis preditoras) para uma dada classe, famoso por sua aplicação em probabilidade, é essencial criar métodos confiáveis para avaliar toxicidade de novos medicamentos.

Já em um trabalho desenvolvido por Kwekha-Rashid, Abduljabbar e Alhayani (2021), estudou-se pesquisas que aplicaram o aprendizado de máquina à COVID-19. A maioria dos 16 artigos usou aprendizado supervisionado, sendo apenas um vinculado ao aprendizado não supervisionado. Dentre esses, utilizou-se algoritmos de regressão logística, enquanto outros optaram por redes neurais artificiais. Ambos os métodos obtiveram resultados promissores na detecção de COVID-19 e na melhoria da assistência médica. Os pesquisadores concluíram que as aplicações de aprendizado de máquina na medicina têm demonstrado alta precisão,

sensibilidade e especificidade. Notavelmente, os modelos de aprendizado supervisionado superaram os de aprendizado não supervisionado na detecção da COVID-19, alcançando mais de 92% de precisão, em comparação com os 7,1% dos últimos.

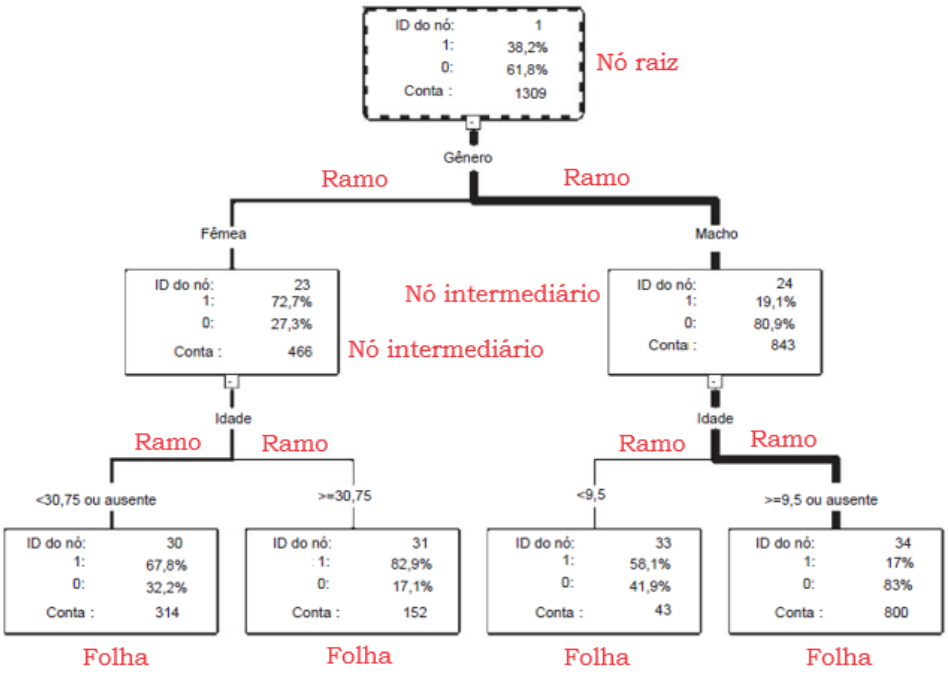
Falqueto e Cezar (2021) utilizaram o ML na área de e-commerce. Os pesquisadores desenvolveram a partir do algoritmo K-means uma proposta de segmentação da base de consumidores do e-commerce de uma empresa multinacional do varejo esportivo. Foram obtidos do banco de dados de pedidos, selecionados com base no histórico de vendas e nos padrões de compra dos consumidores, abrangendo o período de outubro de 2019 a setembro de 2020. Foram utilizados nesse estudo, dados de 526.686 clientes. A metodologia utilizada pelos pesquisadores foram divididas na avaliação da segmentação vigente, construção da base de dados, clusterização da base de dados e comparação dos resultados com o segmento vigente. O uso deste algoritmo de *Machine Learning* se provou ser eficiente, uma vez que permitiu a identificação de três segmentos válidos que eram distintos da segmentação atual. Além disso, houve uma melhoria nos atributos avaliados, uma vez que a nova abordagem de segmentação agora leva em consideração não apenas a demanda, como era feito anteriormente, mas também o número de pedidos e o tempo de inatividade dos consumidores.

Em outro estudo, foi criado um sistema baseado em redes neurais artificiais (RNA) cujo objetivo era detectar doenças foliares em árvores de eucalipto. Utilizando técnicas de visão computacional e treinamento de RNA *Perceptron* multicamadas com o algoritmo *Backpropagation* em Python, o sistema processou imagens de folhas doentes e saudáveis, resultando em uma média de 92% de precisão em várias topologias de redes neurais. A topologia mais eficaz, contendo uma camada oculta com 86 neurônios, alcançou métricas superiores a 93% em acurácia, precisão, revogação e F1 Score, com baixo esforço computacional, demonstrando um desempenho eficaz do sistema especialista (DOS SANTOS, 2021).

2.2 Árvore de decisão

A Árvore de Decisão é um algoritmo de *Machine Learning*, ou seja, uma ferramenta utilizada em bases de dados compostas por variáveis preditoras e por uma variável resposta. Neste tipo de modelo tem-se uma estrutura de nós, ramos e folhas, relacionados à constituição de uma árvore (Figura 2).

Figura 2 – Modelo didático de Árvore de Decisão relacionado ao naufrágio do Titanic



Nota: *Conta =número de exemplos; ID = identidade do nó; 0 = Mortos; 1 = Sobreviventes*

Fonte: Adaptado de De Ville, 2013

O nó raiz fica na parte de cima da árvore e sempre será o primeiro nó, ou seja, será o primeiro teste a ser feito. A partir do nó raiz serão criados ramos vinculados a nós intermediários e terminam nas folhas (nós folha ou nós externos), consideradas a classificação final (GUARNIZO, 2021).

Em resumo, a Árvore de Decisão é construída avaliando esses casos repetidamente para diferentes subconjuntos de treinamento, o que nos permite tomar decisões precisas com base nos dados disponíveis. Isso ajuda a simplificar a complexidade e a tornar as decisões mais claras (MONARD; BARANAUSKAS, 2003).

Na Árvore de decisão, o atributo mais importante é o nó raiz na árvore, enquanto os atributos de menor relevância são representados nos nós subsequentes. Isso oferece uma vantagem significativa, pois permite a tomada de decisões com foco nos atributos mais importantes, ao mesmo tempo em que tornando o processo mais compreensível. Ao apresentar os atributos em ordem de importância, as Árvores de Decisão possibilitam que a identificação de quais fatores exercem maior influência em suas atividades. Essa abordagem hierárquica e priorizada torna o processo de tomada de decisão mais claro e informado. Isso contribui para uma compreensão aprofundada dos fatores-chave envolvidos, auxiliando na escolha das melhores ações a serem tomadas (LEMOS; STEINER; NIEVOLA, 2005).

Os teste a serem realizados em cada nó vai depender do atributo dele, podendo ser quantitativos (valores numéricos) ou qualitativos (categorias ou classes) (GARCIA, 2022).

De modo geral, para se ter uma boa árvore de decisão existe uma grande dependência nos critérios utilizados para escolher o atributo em cada nó e a ordem com que esses atributos vão ser dispostos na árvore. Sarker (2021) relata que os critérios mais populares utilizados para se construir uma Árvore de Decisão a partir dos dados de treinamento é ordenar os nós a partir da Gini (Equação 1), para Impureza de Gini, e entropia (Eq. 2) para ganho de informação.

$$Gini(S) = 1 - \sum_{j=1}^{m} p_j^2 \qquad \text{Eq. (1)}$$

Onde:

S é o conjunto de exemplos

m é o número de classes

p_j é a frequência relativa da classe j em S

O índice Gini é um índice de dispersão estatística e é uma medida que avalia a variação ou heterogeneidade dos dados.

Quando seu valor é zero significa que o conjunto de dados é puro, ou seja, todos os registros pertencem à mesma classe. Por outro lado, quando se aproxima do valor 1, indica que os registros estão igualmente distribuídos entre todas as classes. Ao aplicar o índice Gini na construção de árvores de decisão binárias, tendemos a agrupar os registros que representam a classe mais frequente em um único ramo, utilizando o atributo com o menor valor de índice para a classificação (GARCIA, 2003).

Por sua vez, a Entropia (Equação 2) avalia o quão uniformemente as classes dos exemplos de treinamento estão distribuídas, com valores variando de 0 a 1.

$$Entropia(S) = \sum_{j=1}^{m} - p_j \log_2 p_j \qquad \text{Eq. (2)}$$

Onde:

S é o conjunto de exemplos

m é o número de classes

p_j é a proporção de S pertencer à classe j

Em um cenário de árvore de decisão com apenas duas classes, como "sim" e "não", a entropia zero indica que todos os exemplos de treinamento pertencem a uma única classe, enquanto entropia 1 sugere que metade dos exemplos pertence a uma classe e a outra metade à outra classe (REIS, 2006).

No contexto da Figura 2 (p. 10), De Ville (2013) conduziu uma análise de árvore de decisão utilizando dados de 1309 passageiros do Titanic. O nó raiz representa a distribuição global do campo 'alvo', que indica sobrevivência versus não sobrevivência. A taxa de sobrevivência global, representada por “1” nos dados, é de 38%. O gênero é escolhido como o primeiro critério de particionamento abaixo do nó raiz, formando dois nós descendentes para passageiros do sexo feminino e masculino, respectivamente. O efeito do gênero é notável, seguindo um padrão que reflete o protocolo "mulheres e crianças primeiro" nos botes salva-vidas. Enquanto a taxa de sobrevivência global é de 38%, ela aumenta para cerca de 73% entre as mulheres, mas diminui para aproximadamente 19% entre os homens. Os nós descendentes, resultado da partição recursiva dos grupos feminino e masculino, ilustram uma característica marcante das árvores de decisão. Aqui, observamos o efeito contextual da idade na taxa de sobrevivência. Entre as mulheres, as idades mais avançadas têm maior probabilidade de sobreviver, com uma taxa de sobrevivência de 83% entre as mulheres mais velhas em comparação com 68% entre as mulheres mais jovens. Na população masculina, o efeito é completamente inverso: os homens mais velhos têm uma taxa de sobrevivência substancialmente mais baixa (17% versus 58% para os homens mais jovens)

Em alguns casos para melhorar a eficiência da Árvore de Decisão e resolver problemas como o *overfintting* é utilizado o método de poda. A poda remove partes da estrutura da árvore que diminua ou não ajudem para uma melhor precisão do algoritmo, consequentemente a Árvore de Decisão terá uma estrutura menor e apresentará um melhor desempenho. (KINGSFORD, 2008). A poda poderá ser feita de duas maneira: 1) se ocorrer algum critério de parada enquanto a árvore está sendo construída como, por exemplo, o atributo não atingiu um valor de entropia pré-determinado, a construção da árvore é interrompida. Este tipo de poda é denominada pré-poda; 2) a poda é feita após a construção da árvore, visando diminuir seu tamanho. Sua denominação é pós-poda tendo a vantagem de evitar a construção que será removida posteriormente. Além disso, é mais confiável, pois todos os exemplos serão utilizados na construção da árvore (GARCIA, 2002).

Em um trabalho realizado por Lemos, Silva e Bernardino (2020), foram criados dois questionários para auxiliar o diagnóstico de distúrbios de humor e distúrbios de ansiedade. As 9 perguntas realizadas (atributos) nesses dois questionários foram utilizados para criar um

algoritmo de Árvore de Decisão. O atributo resposta teve duas possibilidades: sim ou não. O algoritmo utilizado para a construção da Árvore de Decisão foi o CART (do inglês, *Classification and Regression Trees*) que tem a capacidade de criar árvores de tamanhos reduzidos e com elevado desempenho. Durante o processo de criação da árvore, o algoritmo CART recebeu um conjunto de dados de treinamento. Como resultado foram criadas duas árvores, uma para diagnosticar distúrbios de humor e outra para diagnosticar distúrbios de ansiedade. Na Árvore de humor foi necessário entre 4 e 8 perguntas para se chegar ao diagnóstico. Já na árvore de ansiedade, dependendo da resposta, seria necessário apenas 3 perguntas para se chegar a uma classificação. Mostrado, assim, a eficiência em se utilizar o índice gini para se criar Árvores de Decisão. O trabalho não realizou testes para saber a eficiência do algoritmo.

Em outro trabalho, realizado por Trevisam (2023), apresentou-se a aplicação de algoritmos de Árvore de Decisão, Rede Bayesiana e Máquinas de Vetores de Suporte com o objetivo de classificar textos oriundos dos registros de histórico de manutenções em transformadores de potência. Boa parte do trabalho é destinado para o pré-processamento dos dados brutos, mas após isso é comparados os três algoritmos. No caso da Árvore de Decisão, ela foi realizada a partir do algoritmo CART. A partir dele foram criadas duas árvores, uma baseada no índice Gini e outra baseada na entropia. Os resultados apresentados pela Árvore de decisão com índice Gini foram F1-score de 0,89 e percentual de acerto de rótulos válidos de 89%. Já a Árvore de Decisão baseada na entropia teve F1-score de 0,92 e o percentual de acerto de rótulos validos de 92%. Vale ressaltar que os algoritmos de Rede Bayesiana e SVM tiveram um bom desempenho apenas classificando dados nulos. Já as duas Árvores de decisão tiveram bons resultados de modo geral.

Por sua vez, o trabalho realizado por Meira, Rodrigues e Moraes (2008) teve como objetivo criar uma árvore de decisão para ajudar na compreensão de manifestações epidêmicas da ferrugem do cafeeiro causada por *Hemileia vastatrix.* A criação da Árvore de Decisão foi realizada por meio da ferramenta “tree” do SAS® Enterprise Miner™. Para tanto, foram utilizadas duas regras de parada pré-poda. Uma limitou o tamanho da árvore, permitindo ter, no máximo, seis níveis. A outra limitou a fragmentação do conjunto de treinamento, precisando de, no mínimo, dez exemplos em cada nó para poder ir para uma nova divisão e pelo menos cinco exemplos em cada nó folha. As taxas de infecção foram agrupadas em três classes: redução ou estagnação (TX1); crescimento moderado, até 5p.p (TX2) e crescimento acelerado, acima de 5p.p (TX3). A árvore foi treinada com 364 exemplos, classificando corretamente 78% do conjunto de treinamento e teve a acurácia em 73% para a classificação de novos exemplos.

O acerto do algoritmo para a classificação TX1 foi de 88%, para a TX2 foi de 57% e para a TX3 foi de 79%. Os autores salientam o potencial da árvore de Decisão como modelo de representação simbólico e interpretável, auxiliando na verificação de quais variáveis e suas interações afetam as epidemias da ferrugem do cafeeiro.

Em outro trabalho, realizado por Lemos, Steiner e Nievola (2005), foram analisados registros históricos de uma agência bancária para a predição de crédito bancário. Para isso, foram utilizados algoritmos de Redes Neurais e Árvores de Decisão, com 24 atributos preditores para a classificação de inadimplente ou adimplente (atributo resposta). A base de dados consistia em informações de 339 empresas, divididas em adimplentes (266) e inadimplentes (73), com as 24 características. O algoritmo J48 (C4.5 *release* 8) foi usado para classificação. Foram conduzidos oito conjuntos de testes no total. O primeiro incluiu todos os dados das 339 empresas. Nos demais, os dados foram divididos em dois conjuntos: um de treinamento com informações de 306 empresas, com 241 empresas adimplentes e 65 inadimplentes. O segundo conjunto foi para testes, consistindo em 33 empresas, com 25 adimplentes e 8 inadimplentes. Exceto no primeiro caso, o conjunto de teste foi criado de maneira aleatória para evitar qualquer viés nos resultados. A Árvore de Decisão teve um erro médio de 11,49% para o conjunto de treinamento e de 28,13% para o conjunto de testes. Um erro grande comparado ao algoritmo de Redes Neurais que teve um erro de 4,09% para o conjunto de treinamento e 9,96 para o conjunto de testes. Porém, os autores salientam a fácil interpretação da Árvore de Decisão, detalhando quais informações foram mais importantes para a classificação.

Já no trabalho realizado por Pianucci e Pitombo (2019), objetivou-se introduzir o uso da Árvore de Decisão na determinação de classes domiciliares relacionadas às taxas de viagem, como uma alternativa ao método de Classificação Cruzada. A pesquisa foi conduzida na cidade de São Carlos (SP) com dados coletados durante a Pesquisa Origem-Destino (O/D) realizada entre 2007 e 2008. O método consiste em sete etapas principais:

1. Seleção das variáveis relevantes para o estudo;
2. Aplicação da Árvore de Decisão (70% amostras de treinamento e 30% amostras de teste;
3. Definição das classes domiciliares;
4. Aplicação do Modelo de Regressão Linear (RLM);
5. Estimativa da geração de viagens obtida na etapa 2;
6. Estimativa da geração de viagens obtida na etapa 4;
7. Comparação dos resultados das etapas 5 e 6;

O algoritmo utilizado para a criação da Árvore de Decisão foi o CART. Na construção das Árvores de Classificação, os critérios de divisão ou medidas de impureza mais amplamente reconhecidos se fundamentam na entropia e no índice Gini. Em contrapartida, nas Árvores de Regressão, a medida de impureza é denominada "redução da variância", que quantifica a diminuição na variabilidade da variável dependente em cada nó, ou seja, o algoritmo da árvore torna os subconjuntos resultantes cada vez mais homogêneos em relação à variável resposta. Os resultados mostraram que o número de moradores nas residências é a variável mais importante, pois melhor explica a variabilidade dos dados relacionados à geração de viagens por domicílio. Ambas as técnicas produziram resultados muito similares. O uso da Árvore de Decisão como ferramenta para determinar estratos e prever a geração de viagens se mostrou adequado, apresentando boa precisão na previsão de viagens domiciliares e utilizando a redução da variância como critério para a determinação das classes.

Por fim, no estudo realizado por Santhanam e Sundaram (2010), utilizou-se técnicas de mineração de dados para identificar e classificar o comportamento de pessoas doadoras de sangue utilizando uma Árvore de Decisão, a partir do algoritmo CART. O banco de dados incluía 748 doadores aleatórios e com 4 atributos: meses desde a última doação; número de doações; total de sangue doado e meses desde a primeira doação. O atributo resposta, por sua vez, foi composto pelas classes “se doou sangue” ou “não doou não sangue” em março de 2007. O conjunto de dados foi estendido com a criação de uma classe nominal estendida com o nome de doador voluntário regular. Os resultados experimentais numéricos nos dados de transfusão de sangue com as melhorias ajudaram a identificar a classificação dos doadores. Ao final do trabalho concluiu-se que o modelo derivado do CART, juntamente com a definição estendida para identificar doadores voluntários regulares, forneceram um bom modelo baseado na precisão da classificação. A precisão da classe de doador voluntário regular foi de 1 e o de doador voluntário não regular de 0,99.

2.3 Linguagem Python

A linguagem de programação Python foi criada em 1990 nos Países Baixos, por Guido van Rossum, no Instituto Nacional de Pesquisas para Matemática e Ciências da Computação (CWI). Essa linguagem de programação surgiu a partir de outra linguagem já existente conhecida como ABC (BORGES, 2014). Apresenta acesso gratuito e pode ser utilizada nos principais sistemas operacionais como Linux, Microsoft Windows e Mac OS (MENEZES, 2014).

A Linguagem de Programação Python tornou-se a principal escolha para a construção de algoritmos de *Machine Learning* por diversas razões. Primeiramente, sua facilidade de aprendizado fez com que se tornasse uma linguagem comum no campo educacional, desde o ensino escolar até a Universidade. Isso resultou em sua popularidade, tanto na indústria quanto entre profissionais que utilizam programação para fins de pesquisa. Soma-se a isso que a comunidade Python desenvolveu um amplo conjunto de ferramentas para a execução interativa de código e visualização de dados, contribuindo especialmente para a pesquisa científica. No entanto, Python enfrenta um grande desafio com relação ao seu desempenho, o que pode ser contornado escrevendo partes críticas do software em linguagens compiladas, geralmente C ou C++, ou fazendo uso do tradutor Cython. Muitas bibliotecas de *Machine Learning* são desenvolvidas em duas ou mais linguagens. Normalmente, a parte central dos cálculos, denominada de backend, é implementada em C ou C++, enquanto o Python é utilizado na parte de interface, conhecida como frontend, a fim de criar uma interface amigável e de fácil uso (GEVORKYA et al., 2019).

Atualmente, os avanços computacionais têm beneficiado várias atividades que os seres humanos realizam, tanto em áreas relacionadas ao trabalho quanto sociais. Um exemplo disso é o celular, que, após se tornar digital e conectado à internet, passou a ser indispensável na vida das pessoas, sendo usado em múltiplas tarefas. E, por ser programável, sua utilidade aumenta a cada dia (DE PINA; MORIMOTO, 2020).

Com esses avanços e os dispositivos sendo conectados e programáveis, o uso de softwares para otimizar tarefas, desde as mais simples, como montar uma planilha com dados, até a automação de tarefas que antes precisavam ser feitas manualmente, torna-se cada vez mais comum. Portanto, infinitas possibilidades se abrem para novos programas resolverem problemas passados e atuais (DE PINA; MORIMOTO, 2020).

A comunidade Python cresceu significativamente na última década e a principal força motriz por trás do crescimento do Python é uma comunidade em rápida expansão de profissionais e aprendizes de Ciência de Dados. Isso se deve em parte à facilidade de uso que linguagens como Python e seu ecossistema de suporte criaram. Isso também se deve à viabilidade do aprendizado profundo, bem como ao crescimento da infraestrutura em nuvem e de soluções escaláveis de processamento de dados capazes de lidar com grandes volumes de dados, o que torna possíveis fluxos de trabalho antes intratáveis em um período de tempo razoável. Estas capacidades de computação simples, escaláveis e aceleradas permitiram um surgimento de recursos digitais úteis que estão a ajudar a moldar ainda mais a Ciência de Dados

num campo próprio e distinto, atraindo indivíduos de muitas origens e disciplinas diferentes (RASCHKA; PATTERSON; NOLET, 2020).

Em um estudo feito por Da Silva (2023), foi estudado um mapeamento sistemático para saber as áreas que mais utilizavam algoritmos de ML escritos na linguagem de Python. Foi constatado que dos 25 artigos selecionados, 8 eram da área de Medicina, 2 de estudos de plantas, 2 de Geologia e 2 de Cibersegurança e o restante em áreas diversas. Já com relação as bibliotecas utilizadas, destacou-se a Scikit Learn, sendo menos frequentes Keras, Tensorflow, ArcPy, OpenCV, PyTorch e Tensorflow. Por sua vez, os algoritmos mais utilizados foram Redes Neurais, Floresta Aleatória e K-means.

Para o processamento de dados, Coelho (2015) utilizou a biblioteca NumPy que fornece suporte para arrays multidimensionais altamente otimizados, sendo estas as estruturas de dados fundamentais na maioria dos algoritmos de última geração. O SciPy que utiliza essas matrizes para oferecer processamentos de dados rápidos. E, por fim, o Matplotlib que possui diversos recursos para criar gráficos de alta qualidade.

Já em um trabalho realizado por Ferreira et al (2022), teve como objetivo investigar a elaborar estratégias para a detecção de *Fake News* sobre a COVID-19, a partir da coleta de um conjunto de dados em língua portuguesa, com o intuito de treinar um algoritmo de Redes Neurais Recorrentes, desenvolvido em linguagem Python. Como ponto de partida, um conjunto de dados de 7,2 mil textos foi selecionado, originado de diversas fontes, como *websites* e agências de notícias. E, cada texto nesse conjunto, já havia sido previamente categorizado como verdadeiro ou falso. Entre a etapa de coleta de textos e a subsequente classificação foi preciso o pré-processamento dos dados. Esse procedimento foi simplificado através do uso das ferramentas presentes nas bibliotecas NLTK (do inglês, *Natural Language Toolkit*) e Regex (do inglês, *Regular Expression*) do Python, que são especificamente desenvolvidas para o tratamento de textos. Esse conjunto de dados foi utilizado para o treinamento e validação do modelo, constituindo uma base sólida para o desenvolvimento do sistema de detecção de *Fake News* relacionadas à COVID-19. O algoritmo foi construído usado as bibliotecas Keras e Tensorflow. O Algortimo teve um índice de acerto de 95% durante o treinamento e de 70% com os dados de teste.

Por sua vez, Da Silva, De Castro e Oliveira Filho (2022) fizeram uma revisão de escopo com o intuito de identificar modelos de predição aplicados no diagnóstico do Acidente Vascular Cerebral (AVC). Conforme mencionado na pesquisa de escopo realizada, 615 trabalhos foram inicialmente identificados, dos quais apenas 9 atenderam aos critérios de inclusão e exclusão, sendo selecionados para análise detalhada e extração de informações. A análise da pesquisa

evidenciou que a maioria dos estudos abordou o desenvolvimento de modelos de aprendizagem, seguido pela comparação de algoritmos e criação de algoritmos específicos. Quanto às ferramentas utilizadas, a linguagem de programação Python, juntamente com a biblioteca Scikit-learn, foram amplamente empregadas. Além disso, os modelos e algoritmos predominantes incluíram a Árvore de Decisão, Naive Bayes, *Random Forest* e KNN (do inglês, *K-Nearest Neighbors*). Os estudos analisados recorrentemente empregaram métricas de avaliação, tais como Recall, Precisão, F1-Score e Acurácia para validar as soluções propostas. Contudo, vale ressaltar que foram identificadas limitações relacionadas à avaliação de desempenho das soluções propostas e à ausência de aspectos relevantes nos estudos examinados.

Pereira, Moraes e Mattos (2023) desenvolvem um sistema de *Machine Learning* para a detecção de intrusão. Criando o compartilhamento de um modelo de Árvore de Decisão, em que as árvores compartilhadas se tornam uma Floresta de Decisão Distribuída. No âmbito da pesquisa, foi desenvolvido um protótipo do sistema em linguagem Python, empregando as Árvores de Decisão da função DecisionTreeClassifier da biblioteca Scikit-learn. Para resolver possíveis problemas de *overfitting* nos dados de treino, foi estabelecida uma profundidade máxima (max-depth) de 7 para as árvores de decisão. Para determinar o valor ideal desse parâmetro, um intervalo de 5 a 15 foi avaliado por meio do método GridSearch da biblioteca Scikit-learn. A profundidade 7 foi selecionada empiricamente, pois demonstrou ser o compromisso mais adequado entre acurácia e generalidade do classificador. Além disso, a função que mede a qualidade da divisão dos ramos da árvore foi configurada com o valor Gini referente à métrica impureza Gini, que se mostrou mais eficaz em comparação com a métrica de entropia. Os autores compararam o desempenho da Floresta de Decisão com Redes Neurais. A floresta de Decisão teve mediana da acurácia em 79%, já a Rede Neural teve a mediana de acurácia em 86%, mas com dez rodadas de treinamento e com maior tempo execução.

Por fim, Rathee, Joshi e Kaur (2018) analisaram as avaliações postadas por pessoas em quatro sites diferentes: airlinequality.com, Amazon, Yelp e IMDB, utilizando técnicas de *Machine Learning* com linguagem Python. O conjunto de dados da airlinequality.com foram classificados como: ruim, médio e excelente. Os outros três modelos foram classificados como: negativo e positivo. O pré-processamento dos dados foi realizado utilizando as bibliotecas RE (do inglês, *Regular Expression*) e NLTK (do inglês, *Natural Language Tool Kit*). Depois foi feita a mudança das avaliações em vetores, utilizando a biblioteca Scikit-learn. Os dados de treinamento foram 75% do total e os dados de teste 25% do total. Em seguida, foram criados 8 modelos de *Machine Learning* utilizando também a biblioteca Scikit-learn: *Logistic*

Regression, KNN, Máquina de Vetores de Suporte, Árvore de Decisão, Floresta aleatória, AdaBoost, Gaussian Naives Bayes e classificador Bagging. Os melhores classificadores foram: Floresta aleatória com acurácia de 76% no conjunto de dados IMDB e Yelp; Classificador Bagging com acurácia de 76,4% no conjunto de dados da Amazon; Ada Boost com acurácia de 59,74% no conjunto de dados da airlinequality.com. Os pesquisadores concluíram que quanto maior o número de classes menor a acurácia e que avaliações maiores tendem a baixar a precisão dos algoritmos.

2.4 Aplicação de *Machine Learning* nas áreas de alimentos e bebidas

A *Machine Learning*, devido à sua aplicabilidade em diversas áreas, oferece inúmeras oportunidades para a indústria de alimentos e bebidas. Com o aumento da população, surgem desafios crescentes nesse setor. Um exemplo é o estudo de Zhou et al. (2019), que revisaram a aplicação em alimentos de algoritmos de *Machine Learning* Profundo ou *Deep Learning*, que é uma subcategoria do *Machine Learning* que se baseia em redes neurais artificiais para modelagem e resolução de problemas complexos. O termo "profundo" refere-se ao uso de arquiteturas de rede neural com várias camadas (conhecidas como redes neurais profundas) para representar e aprender padrões de dados. Redes neurais profundas têm a capacidade de aprender representações hierárquicas de dados, o que é particularmente eficaz para tarefas como reconhecimento de imagem, processamento de linguagem natural e outras aplicações complexas. O artigo abordou a utilização de algoritmos para estimar calorias, avaliar a qualidade de vegetais, frutas, carnes e produtos aquáticos, rastreamento da cadeia de abastecimento e detecção de contaminação alimentar. Esses estudos combinaram algoritmos de análise de dados, visão computacional, Sensometria e Quimiometria, entre outras técnicas. Os métodos de processamento de dados, utilizando algoritmos de *Machine Learning* e *Machine Learning* Profundo, foram utilizados para analisar grandes conjuntos de dados e identificar correlações entre parâmetros sensoriais, características físico-químicas, conteúdo calórico, preferências dos consumidores e dados coletados por meio de tecnologias e equipamentos de detecção. Os autores citam que na maioria dos métodos de reconhecimento de alimentos em imagens, a preparação de dados envolve coletar imagens e realizar o pré-processamento, normalizando e redimensionando as imagens. O treinamento é realizado com Redes Neurais Convolucionais (CNNs) populares, como AlexNet e GoogLeNet, adaptadas para essa tarefa. O conjunto de dados foi dividido em treinamento, validação e teste, com avaliação baseada em indicadores como Top-1% e Top-5%. Os autores ressalvam que embora os resultados sejam promissores, é importante avaliar a generalização do modelo em diferentes conjuntos de dados

no futuro e explorar informações adicionais, como cheiros e pesos dos alimentos, para melhorar o reconhecimento.

Outra revisão realizada por Zhu et al. (2021) destacou o uso de algoritmos de *Machine Learning* em áreas como segurança alimentar, avaliação de qualidade, monitoramento de embalagens e processos de produção alimentar, além da detecção de corpos estranhos. A aplicação dessas técnicas na indústria de alimentos e bebidas demonstra o potencial dessa abordagem para resolver desafios complexos e aprimorar a qualidade e segurança dos produtos alimentícios. É possível notar pelos trabalhos que são citados pelos autores que ocorre a utilização de diversos algoritmos de *Machine Learning* para a execução das tarefas citadas acima, como Máquina de Vetores de Suporte, *Logistic Regression*, KNN, K-means, Árvores de Decisão e Florestas Randômicas. Os autores demostram que a eficiência de cada algoritmo depende do tipo de atividade desejada, da quantidade e tipo de dados e de seu tratamento.

Já Gkikas (2023) fez um estudo com o objetivo de fornecer um método que auxiliaria os tomadores de decisão no gerenciamento de grandes conjuntos de dados, diminuindo o risco de decisões e destacando subconjuntos significativos de dados com determinado peso. Dessa maneira, os métodos de Árvore de Decisão Binária e algoritmo genético foram combinados usando uma técnica de empacotamento. O algoritmo de Árvore de Decisão Binária foi utilizado para classificar os dados em uma estrutura de árvore, enquanto o algoritmo genético foi empregado para identificar as melhores combinações de atributos de um conjunto de combinações possíveis, conhecidas como gerações. O estudo visou resolver o problema de *overfitting* que poderia ocorrer ao classificar grandes conjuntos de dados, reduzindo o número de atributos utilizados na classificação. O algoritmo gerou diversos conjuntos de atributos, os quais foram classificados utilizando o algoritmo Árvore de Decisão Binário e receberam um número de aptidão baseado em sua precisão. O processo de treinamento utilizou dados de uma análise química de vinhos cultivados na mesma região, mas oriundos de três cultivos diferentes. Os resultados demonstraram a eficácia dessa abordagem inovadora na determinação de certos ingredientes e pesos de origem do vinho.

Habib et al. (2020) utilizou um algoritmo de *Machine Learning* para criar um sistema de reconhecimento de doenças do mamão baseado em imagens. Segundo os pesquisadores, o reconhecimento da doença do mamão envolveu principalmente dois problemas desafiadores: a detecção da doença e a classificação da doença. Nesse cenário, foi desenvolvido um sistema especializado agromédico baseado em visão artificial que processava imagens capturadas por dispositivos móveis ou portáteis, com o objetivo de determinar as doenças e auxiliar agricultores distantes na resolução desse problema. A pesquisa incluiu uma série de

experimentos para demonstrar a utilidade do sistema especialista proposto. Inicialmente, foram definidas características distintivas para a análise das imagens. O algoritmo de agrupamento K-means foi utilizado para segmentar as áreas afetadas pela doença nas imagens capturadas, e em seguida, os recursos relevantes foram extraídos para a classificação das doenças com o auxílio de máquinas de vetores de suporte. Os resultados demonstraram que o sistema alcançou uma precisão de classificação de mais de 90%.

Vargas, Cunha e Fernandes (2023) propuseram utilizar os métodos Holt-Winters e K-Means para prever valores de alimentos da cesta básica tanto para o atacado quanto para o varejo. Os alimentos selecionados foram: arroz, feijão, batata inglesa e tomate. Os dados foram retidos do IBGE, EPAGRI/CIRAN e DIEESE e processados inicialmente no Excel e posteriormente em Python, no Jupyter Notebook. Esses processos envolveram normalização, padronização e unificação dos dados presentes em uma base PostgreSQL. O método Holt-Winters foi aplicado, configurado com parâmetros multiplicativos para tendência e sazonalidade, sendo o ciclo anual considerado para a sazonalidade. Após encontrar o número de clusters, os dados foram agrupados em dois conjuntos. O primeiro conjunto continha informações sobre o clima, cotação do dólar, cotação do barril do petróleo e preços dos alimentos. O segundo combinava os resultados do Holt-Winters com dados climáticos. O algoritmo K-Means foi aplicado, gerando pesos para cada registro com base na quantidade de clusters, que variavam de zero a quatro, sendo denominados "K-classes." A tendência de alta ou baixa dos preços dos alimentos foi determinada pela comparação entre cada registro e o intervalo de preços de sua K-classe correspondente. Em termos de resultados, o método Holt-Winters obteve taxas de acerto de 68,35%, com a maior precisão para o arroz, alcançando 100%. Já o K-means apresentou uma taxa geral de acerto de 57,28%, com destaque para o feijão preto no varejo, com 100% de acerto.

Por fim, Angus ([s.d.]) propôs criar uma análise para um conjunto de dados de vinho com o objetivo de prever a classificação, a partir de um algoritmo de Redes Neurais, com a biblioteca Scikit-Learn. Os dados de vinho utilizados neste estudo foram obtidos do repositório de aprendizado de máquina UCI. Esses dados se referem ao vinho que abrangem duas categorias: vinho tinto e vinho branco. O conjunto de dados é composto por onze variáveis, sendo dez delas propriedades físico-químicas e uma variável de pontuação de qualidade sensorial, classificada de 0 a 10. A biblioteca Scikit-Learn foi utilizada para analisar a seleção de recursos e, com base nessa análise, foi determinado que a utilização dos onze recursos resultaria no melhor desempenho de previsão. O código foi elaborado no ambiente PyCharm e a normalização dos conjuntos de dados foi realizada com o auxílio da função StandardScaler

do Scikit-Learn. Foram desenvolvidos dois modelos: um classificador multiclasse (classifica a qualidade do vinho) e um classificador binário (distingue se é um vinho branco ou tinto). Depois foi feita a classificação multiclasse com os dados sendo divididos em treinamento, validação e teste. A regularização dos dados foi necessária, pois vários experimentos apresentaram *overfitting*. O algoritmo conseguiu superar os outros classificadores testados durante os experimentos Scikit-Learn no conjunto de dados de vinho tinto (precisão 0,95) e chegou perto dos melhores classificadores no conjunto de dados de vinho branco (precisão 0,89). Já o classificador binário teve precisão de 98%.

3 MATERIAIS E MÉTODOS

3.1 Plataforma e linguagem de programação

O presente trabalho utilizou a plataforma Google Colab (https://colab.research.google.com) e a linguagem de programação Python, sendo utilizandas as seguintes bibliotecas de funções: Pandas, Numpy, Matplotlib, Seaborn e Scikit-learn.

3.2 Base de dados

O banco de dados utilizado foi obtido a partir do estudo de Cortez et al. (2009) (https://archive.ics.uci.edu/ml/datasets/wine+quality), em arquivo de extensão csv, sendo composto pelos dados de 11 tipos de análises físico-químicas (atributos preditores) (Quadro 3). Os dados da análise sensorial corresponderam à variável resposta, sendo geradas por pelo menos três avaliadores sensoriais (com recurso a provas cegas), que classificaram o vinho em uma escala numérica entre 1 (péssima qualidade) a 10 (qualidade excelente) (Quadro 4). Além disso, a base de dados contou com 4898 registros de vinhos brancos, sendo todos os dados experimentais adquiridos por Cortez et al. (2009) entre maio de 2004 a fevereiro de 2007.

Quadro 3 - Análises físico-químicas realizadas nas amostras de vinho branco

Análise físico-química
Acidez fixa (g(ácido tartárico) dm^{-3})
Acidez volátil (g(ácido acético) dm^{-3})
Teor de ácido cítrico (g dm^{-3})
Teor de açúcar residual (g dm^{-3})
Teor de cloretos (g(cloreto de sódio) dm^{-3})
Teor de dióxido de enxofre livre (mg dm^{-3})
Teor de dióxido de enxofre total (mg dm^{-3})
Densidade (g cm^{-3})
pH
Teor de sulfatos (g(sulfato de potássio) dm^{-3})
Álcool (% v v^{-1})

Fonte: Adaptado de Cortez et al., 2009

Após a aquisição dos dados foi feito o tratamento dos dados na seguinte ordem:

- Conversão de dados em formato de texto para formato de número;
- Conversão da base de dado em dataframes;
- Verificação da presença de dados faltantes;
- Análise descritiva dos atributos preditores;
- Avaliação e exclusão de *outliers*;

- Divisão da base de dados dando origem a duas novas bases de dados: atributos preditores e atributos classe;
- Padronização dos atributos preditores;
- Reorganização dos atributos classe em novas classes;
- Aplicação do processo de subamostragem Tomeklinks;
- Avaliação da Árvore de Decisão quanto à classificação dos vinhos brancos.

Quadro 4 – Classificação original das análises sensoriais

Nota	**Qualidade sensorial**
1	Péssimo
2	Muito Ruim
3	Ruim
4	Insatisfatório
5	Médio
6	Satisfatório
7	Bom
8	Muito Bom
9	Excelente
10	Excepcional

Após a aquisição dos dados foi realizado o seu tratamento. Em um primeiro momento foi feita a troca de todos os dados que estavam configurados em formato de texto para o formato de número no próprio Excel.

3.3 Desenvolvimento do *script* em Python

Esta seção explica o passo a passo do desenvolvimento do *script* em Python para a classificação dos vinhos brancos, sendo as etapas apresentado a seguir.

- Estruturação das bases de dados de treinamento e de teste;
- Escolha das métricas para a avaliação dos modelos de classificação;
- Aplicação do modelo de GridSearchCV para diferentes testes (Quadro 5) para a otimização dos Hiperparâmetros;
- Geração e avaliação do melhor modelo de classificação utilizando os dados obtidos através do GridSearchCV. A avaliação foi realizada por meio das

métricas acurácia de treinamento, acurácia de previsão, precisão, revocação e escore F1.

Quadro 5 – Relação dos testes de GridSearchCV

Teste	**Exclusão de outliers**	**Quantidade de categorias**
1	Dispersão de pontos	Duas
2	Dispersão de pontos	Três
3	Boxplots	Duas
4	Boxplots	Três

4 RESULTADOS E DISCUSSÃO

4.1 Análises físico-químicas

O presente trabalho teve como atributos preditores os valores das análises realizadas no controle de qualidade físico-químico dos vinhos brancos (CORTEZ et al., 2009). No total foram 11 análises físico-químicas, sendo sua descrição apresentada a seguir.

A acidez fixa é uma medida da acidez dos vinhos considerando a acidez total associada especificamente ao ácido tartárico (g(ácido tartárico) dm^{-3}),ou seja, nessa análise é levada em consideração a quantidade de outros ácidos mas sua concentração final é simplificada apenas para a concentração de ácido tartárico. Esta análise é muito importante, pois desconsidera a influência dos componentes voláteis ácidos presentes no vinho como, por exemplo, o ácido acético.

A acidez volátil, por sua vez, quantifica a acidez da bebida (g(ácido acético) dm^{-3}) em relação ao ácido acético. Na verdade, todos os componentes ácidos voláteis serão quantificados na forma de ácido acético.

Ainda no âmbito dos compostos ácidos do vinho, o ácido cítrico (g dm^{-3}) é quantificado separadamente. Fica claro que, por não ser um composto volátil, seu teor faz parte do obtido na acidez fixa.

O pH, por sua vez, corresponde ao logaritmo, na base 10, da concentração dos íons H^+ da bebida, sendo o resultado multiplicado por -1. No caso dos vinhos, devido à maior contribuição de ácidos fracos, o pH tem valores menores que 7, porém não extremos.

Os açúcares residuais (g dm^{-3}) correspondem aos carboidratos restantes após o processo de fermentação somado aos açúcares adicionados ao vinho, caso esta etapa seja realizada.

Por sua vez, o teor de cloretos (g(cloreto de sódio) dm^{-3}) corresponde à concentração de íons cloreto, considerando para efeito de cálculo, que todos os cloretos compõem o cloreto de sódio.

O teor de SO_2 livre (mg dm^{-3}) equivale à concentração dos sais derivados do dióxido de enxofre. Por sua vez, o teor de SO_2 total (mg dm^{-3}) considera o teor de SO_2 livre somado à concentração dos derivados formados pela reação entre o ânion bissulfito com açúcares, aldeídos, proteínas etc. Logo, da mesma forma que o teor de ácido cítrico está contido no teor de ácidos fixos, o teor de SO_2 livre também está para o teor de SO_2 total.

Já a densidade do vinho corresponde à relação entre sua massa (g) e seu volume, em cm^3. Apesar de simples, é uma medida muito importante, pois é afetada diretamente pelo

processo de fermentação. Processos mais longos levam à formação de mais etanol, reduzindo a densidade da bebida. E o oposto também pode ocorrer.

O teor de sulfatos (g(sulfato de potássio) dm^{-3}) equivale à concentração dos íons sulfato no vinho em termos de sulfato de potássio.

Por fim, o teor de álcool (% v v^{-1}) é o teor do álcool etílico, em volume, majoritário entre os álcoois na bebida.

4.2 Aquisição dos dados

Inicialmente foi realizada a importação das bibliotecas que foram usadas no tratamento de dados, desenvolvimento do algoritmo, treinamento dos dados e avaliação das métricas (Figura 3).

Figura 3 – Importação das bibliotecas para o Google Colab

```
import pandas as pd #bibliotecas pandas
import numpy as np #biblioteca numpy
import matplotlib.pyplot as plt #biblioteca matplotlib
%matplotlib inline
import seaborn as sns #biblioteca seaborn


from sklearn.metrics import confusion_matrix, accuracy_score #matriz de confusão
from sklearn.tree import DecisionTreeClassifier #arvore de decisão
from sklearn.model_selection import GridSearchCV #otimização de parâmetros com validação cruzada
from sklearn.metrics import accuracy_score, classification_report #dados matriz de confusão
from sklearn.preprocessing import StandardScaler #pradonização dos dados
from sklearn.model_selection import train_test_split # geração das bases de dados de treinamento e teste
from sklearn.metrics import ConfusionMatrixDisplay # geração de matriz  de confusão
from sklearn import datasets, tree #árvore de decisão

from scipy.stats import shapiro # teste de hipóteses Shapiro-Wilk
from statsmodels.stats.outliers_influence import variance_inflation_factor #fator de inflação de variância
from imblearn.under_sampling import TomekLinks #técnica de subamostragem
```

Após isso, foi feito o *upload* do arquivo Winequality-WhiteWine.csv que contém a base de dados dos vinhos brancos. O conteúdo do arquivo csv compôs o dataframe do pandas utilizando o método read_csv(), biblioteca pandas. O dataframe foi renomeado para vinho. Depois as colunas foram renomeadas traduzindo seus nomes para português, usando um dicionário (nomes_port) e o método rename() (Figura 4).

Para facilitar a visualização de uma parte dos dados contidos nos dataframes, foram realizadas configurações específicas. Isso foi feito através do método set_option(), no qual o número de linhas exibidas foi ajustado para 10 e a quantidade de casas decimais dos dados numéricos foi fixada em 3. Vale ressaltar que os dados com menos casas decimais não sofreram alterações (Figura 5).

Figura 4 – Conversão das bases de dados em dataframes

```
vinho = pd.read_csv('/content/winequality-white (4).csv', sep=';')
nomes_port = {'fixed acidity':'acidez fixa', 'volatile acidity':'acidez volátil', 'citric acid': 'teor de ácido cítrico',
              'residual sugar': 'teor de açúcares residuais', 'chlorides': 'teor de cloretos', 'free sulfur dioxide':
              'teor de SO2 livre','total sulfur dioxide': 'teor de SO2 total', 'density': 'densidade', 'sulphates': 'teor de sulfatos',
              'alcohol':'teor de álcool', 'quality':'qualidade'}
vinho = vinho.rename(columns=nomes_port)

print('Dimensões da base de dados dos vinhos brancos:', vinho.shape)
vinho.head()
```

```
Dimensões da base de dados dos vinhos brancos: (4898, 12)
```

	acidez fixa	acidez volátil	teor de ácido cítrico	teor de açúcares residuais	teor de cloretos	teor de SO2 livre	teor de SO2 total	densidade	pH	teor de sulfatos	teor de álcool	qualidade
0	7.0	0.27	0.36	20.7	0.045	45	170	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14	132	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30	97	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.40	9.9	6

Figura 5 – Visualização de parte da base de dados vinho

```
[ ] # arpesentação dos dataframes

    pd.set_option('display.min_rows',10)
    pd.set_option('display.precision',3)

    vinho
```

	acidez fixa	acidez volátil	teor de ácido cítrico	teor de açúcares residuais	teor de cloretos	teor de SO2 livre	teor de SO2 total	densidade	pH	teor de sulfatos	teor de álcool	qualidade
0	7.0	0.27	0.36	20.7	0.045	45	170	1.001	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14	132	0.994	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30	97	0.995	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47	186	0.996	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47	186	0.996	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24	92	0.991	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57	168	0.995	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30	111	0.993	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20	110	0.989	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22	98	0.989	3.26	0.32	11.8	6

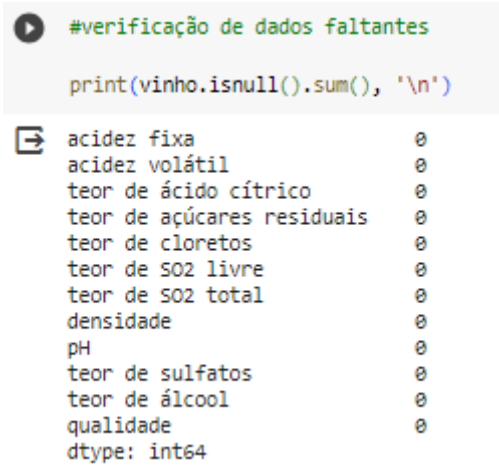
4898 rows × 12 columns

4.3 Análises preliminares

Na etapa seguinte da análise da base de dados ‘vinho’, foi avaliada a presença de informações ausentes. Para realizar essa verificação em um conjunto de dados extenso, foi aplicado o método isnull(). A fim de quantificar a extensão das informações faltantes para cada atributo, utilizou-se o método sum(), conforme ilustrado na Figura 6.

É importante ressaltar que, após a aplicação dessas técnicas, observou-se que todos os atributos da base de dados se encontravam completos, ou seja, não continham informações faltantes. Diante dessa constatação, não foi necessário proceder com a eliminação de registros ou a substituição de dados em falta por medidas de tendência central, como a média aritmética ou a mediana.

Figura 6 – Verificação de dados faltantes



```
#verificação de dados faltantes

print(vinho.isnull().sum(), '\n')
```

```
acidez fixa                 0
acidez volátil              0
teor de ácido cítrico       0
teor de açúcares residuais  0
teor de cloretos            0
teor de SO2 livre           0
teor de SO2 total           0
densidade                   0
pH                          0
teor de sulfatos            0
teor de álcool              0
qualidade                   0
dtype: int64
```

Para garantir que todos os atributos preditores e de classificação foram transformados para elementos numéricos, utilizou-se o método select_dtypes() com o atributo exclude = number. Dessa forma, dados que não são numéricos seriam mostrados. Como se pode constatar na Figura 7, não existem dados de outro tipo, que não sejam numéricos.

Figura 7 – Verificação de outros tipos de dados

```
[ ]  # verificação de outros tipos de dados
     vinho.select_dtypes(exclude = 'number')
```

0
1
2
3
4
...
4893
4894
4895
4896
4897

4898 rows × 0 columns

Essa parte na construção do modelo de ML é muito importante, pois a falta de valores acarreta um menor desempenho do algoritmo. Já os valores do tipo texto não podem ser usados em equações pelo algoritmo assim causando erros em sua execução.

4.4 Análise descritiva

A próxima fase envolveu a exploração dos dados. Iniciando pela avaliação da existência de valores discrepantes, conhecidos como *outliers*, em cada um dos atributos preditores. Para realizar essa análise, foi selecionado os atributos do dataframe de vinho usando o atributo iloc e, em seguida, aplicado o método describe() para apresentar uma visão descritiva desses dados, conforme Figura 8.

Figura 8 - Tabela com os dados descritivos dos atributos do dataframe

```
[ ] # tabela com os dados descritivos dos atributos preditores de vinho

    vinho.iloc[:, 0:12].describe()
```

	acidez fixa	acidez volátil	teor de ácido cítrico	teor de açúcares residuais	teor de cloretos	teor de SO2 livre	teor de SO2 total	densidade	pH	teor de sulfatos	teor de álcool	qualidade
count	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000	4898.000
mean	6.855	0.278	0.334	6.391	0.046	35.314	138.363	0.994	3.188	0.490	10.514	5.878
std	0.844	0.101	0.121	5.072	0.022	17.014	42.503	0.003	0.151	0.114	1.231	0.886
min	3.800	0.080	0.000	0.600	0.009	2.000	9.000	0.987	2.720	0.220	8.000	3.000
25%	6.300	0.210	0.270	1.700	0.036	23.000	108.000	0.992	3.090	0.410	9.500	5.000
50%	6.800	0.260	0.320	5.200	0.043	34.000	134.000	0.994	3.180	0.470	10.400	6.000
75%	7.300	0.320	0.390	9.900	0.050	46.000	167.000	0.996	3.280	0.550	11.400	6.000
max	14.200	1.100	1.660	65.800	0.346	289.000	440.000	1.039	3.820	1.080	14.200	9.000

Analisando os dados da Figura 8 é possível notar que os valores extremos estão bem maiores do que o 3° quartil na maioria dos atributos preditores. Para encontrar os *outliers* foi utilizada a técnica de boxplots, onde o limite superior é delimitado por $Q_3 + 1{,}5x(Q_3 - Q_1)$ e o limite inferior é $Q_1 - 1{,}5x(Q_3 - Q_1)$. Utilizando o método boxplot(), com cada atributo sendo selecionado usando o atributo iloc. Para identificar e visualizar os boxplots de cada atributo, foi aplicado o método set_title() para adicionar títulos informativos a cada um deles. E, buscando organizar esses gráficos em grupos de até seis unidades por figura, foi utilizado o método subplots() da biblioteca matplotlib, como ilustrado na Figura 9.

Figura 9 – Criação do conjunto de Boxplots dos atributos preditores

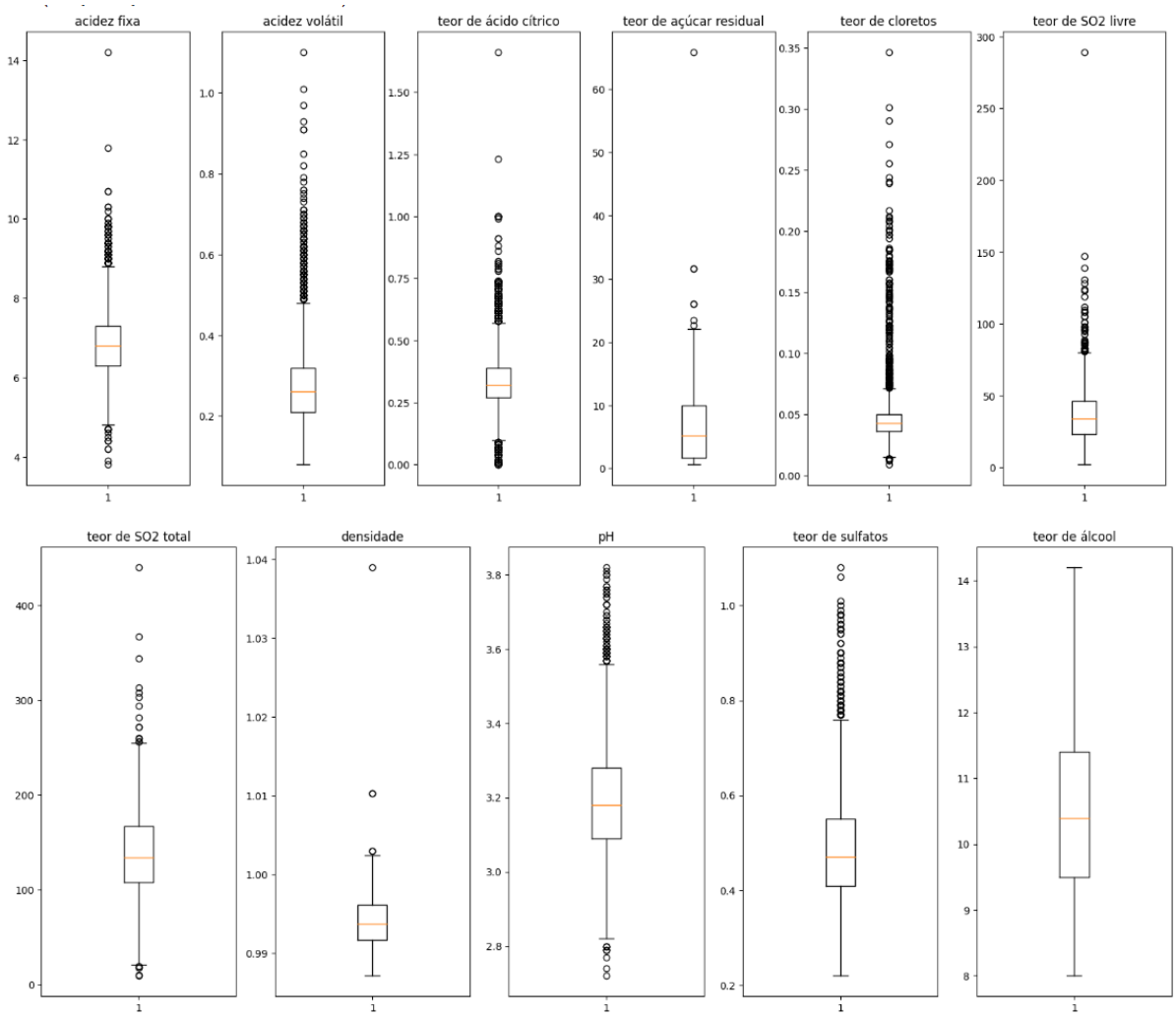
```
# Mostar os boxplots dos atributos preditores de vinho

fig, bp = plt.subplots(1,6, figsize=(20,8))
bp[0].boxplot(vinho.iloc[:,0]); bp[0].set_title('acidez fixa')
bp[1].boxplot(vinho.iloc[:,1]); bp[1].set_title('acidez volátil')
bp[2].boxplot(vinho.iloc[:,2]); bp[2].set_title('teor de ácido cítrico')
bp[3].boxplot(vinho.iloc[:,3]); bp[3].set_title('teor de açúcar residual')
bp[4].boxplot(vinho.iloc[:,4]); bp[4].set_title('teor de cloretos')
bp[5].boxplot(vinho.iloc[:,5]); bp[5].set_title('teor de SO2 livre')
```

Figura 9 - Continuação

```
fig, bp = plt.subplots(1, 5, figsize=(20, 8))
bp[0].boxplot(vinho.iloc[:,6]); bp[0].set_title('teor de SO2 total')
bp[1].boxplot(vinho.iloc[:,7]); bp[1].set_title('densidade')
bp[2].boxplot(vinho.iloc[:,8]); bp[2].set_title('pH')
bp[3].boxplot(vinho.iloc[:,9]); bp[3].set_title('teor de sulfatos')
bp[4].boxplot(vinho.iloc[:,10]); bp[4].set_title('teor de álcool')
```

Após processar os dados, os boxplots foram criados (Gráfico 1), sendo os *outliers* representados como círculos.

Gráfico 1 – Boxplots dos atributos preditores da base de dados vinho



Com base nos resultados acima, as faixas de exclusão para cada atributo preditor, utilizando o método de boxplot, estão exibidos no Quadro 6.

Quadro 6 – Faixa de exclusão para cada atributo preditor pelo método boxplot

Atributo Preditore	**Faixa de Exclusão**
Acidez Fixa	Faixa ≤ 4,8 (g(ácido tartárico) dm^{-3}) ou Faixa ≥ 8,8 (g(ácido tartárico) dm^{-3})
Acidez Volátil	Faixa ≤ 0,045 (g(ácido acético) dm^{-3}) ou Faixa ≥ 0,485 (g(ácido acético) dm^{-3})
Teor de Ácido Cítrico	Faixa ≤ 0,09 (g dm^{-3}) ou Faixa ≥ 0,57 (g dm^{-3})
Teor de Açucares Residuais	Faixa ≥ 22,2 (g dm^{-3})
Teor de Cloretos	Faixa ≤ 0,015 (g(cloreto de sódio) dm^{-3}) ou Faixa ≥ 0,071 (g(cloreto de sódio) dm^{-3})
Teor de SO_2 Livre	Faixa ≥ 80,5 (mg dm-$^{-3}$)
Teor de SO_2 Total	Faixa ≤ 19,5 (mg dm^{-3}) ou Faixa ≥ 225,5 (mg dm^{-3})
Densidade	Faixa ≤ 0,986 (g cm^{-3}) ou Faixa ≥ 1,002 (g cm^{-3})
pH	Faixa ≤ 2,805 ou Faixa ≥ 3,565
Teor de Sulfetos	Faixa ≤ 0,2 (g(sulfato de potássio) dm^{-3}) ou Faixa ≥ 0,76 (g(sulfato de potássio) dm^{-3})
Teor de Álcool	Faixa ≤ 6,65 (% v v^{-1}) ou Faixa ≥ 14,25 (% v v^{-1})

Utilizando o método drop(), associado ao atributo index, deletou-se as faixas de valores considerados como *outliers*. Depois, foi feita a verificação das dimensões atualizadas do dataframe vinhos, realizado por meio do atributo shape (Figura 10). Verificou-se que a eliminação dos outliers reduziu o número de registros para 3879 (redução de 20,8% dos registros).

Figura 10 – Exclusão de registros com outliers pelo método de Boxplot

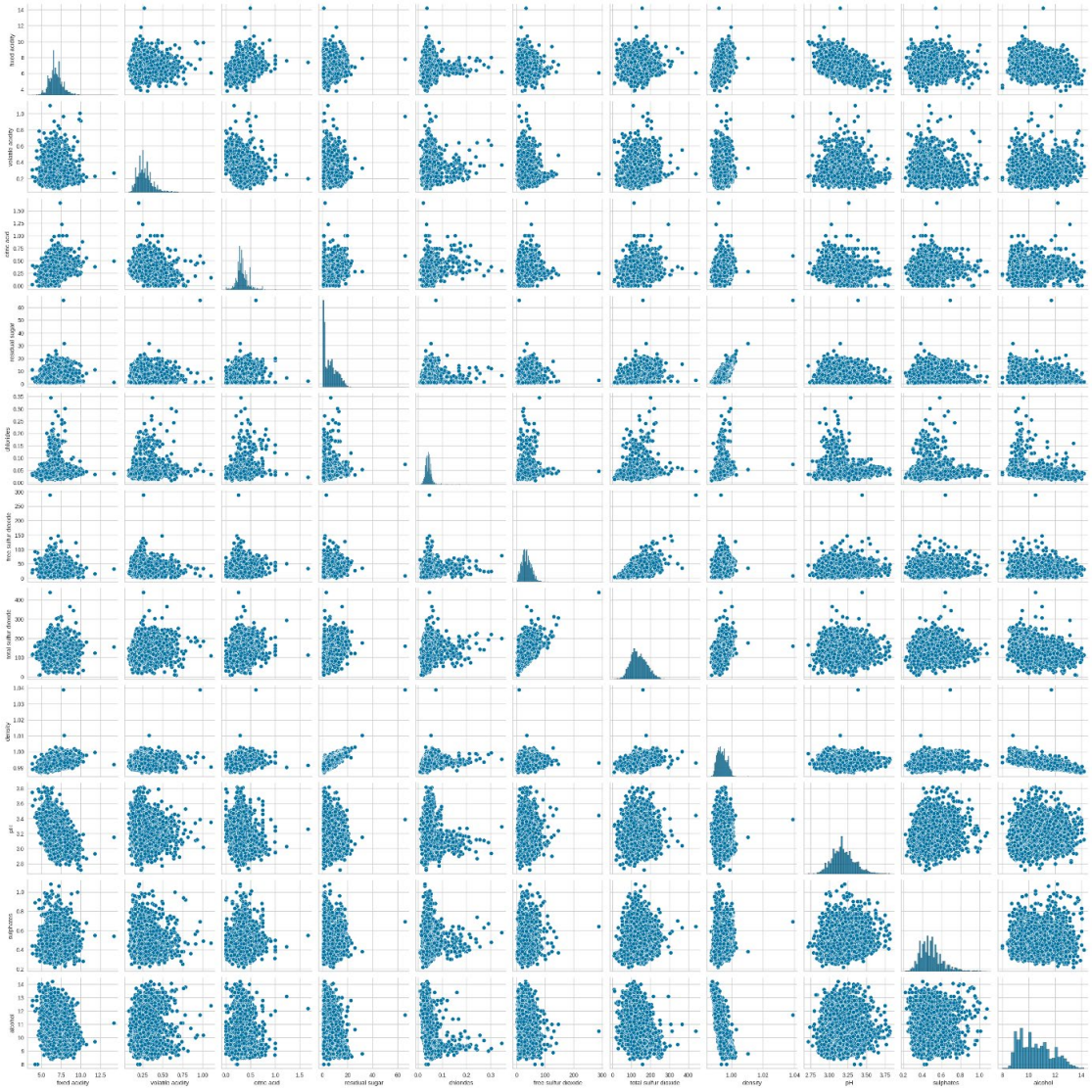
```
#exclusão de registros com outliers

vinho.drop(vinho[vinho['acidez fixa']>=8.8].index, inplace = True)
vinho.drop(vinho[vinho['acidez fixa']<=4.8].index, inplace = True)
vinho.drop(vinho[vinho['acidez volátil']>=0.485].index, inplace = True)
vinho.drop(vinho[vinho['teor de ácido cítrico']>=0.57].index, inplace = True)
vinho.drop(vinho[vinho['teor de ácido cítrico']<=0.09].index, inplace = True)
vinho.drop(vinho[vinho['teor de açúcares residuais']>=22.2].index, inplace = True)
vinho.drop(vinho[vinho['teor de cloretos']<=0.015].index, inplace = True)
vinho.drop(vinho[vinho['teor de cloretos']>=0.071].index, inplace = True)
vinho.drop(vinho[vinho['teor de SO2 livre']>=80.5].index, inplace = True)
vinho.drop(vinho[vinho['teor de SO2 total']>=225.5].index, inplace = True)
vinho.drop(vinho[vinho['teor de SO2 total']<=19.5].index, inplace = True)
vinho.drop(vinho[vinho['densidade']>=1.002].index, inplace = True)
vinho.drop(vinho[vinho['pH']>=3.565].index, inplace = True)
vinho.drop(vinho[vinho['pH']<=2.805].index, inplace = True)
vinho.drop(vinho[vinho['teor de sulfatos']>=0.76].index, inplace = True)

print('dimensões igual a:',vinho.shape)
```

```
dimensões igual a: (3879, 12)
```

A eliminação completa dessas informações pode resultar na perda de exemplos que desempenham um papel fundamental na construção de modelos de ML capazes de atingir altos níveis de desempenho. Então, buscou-se também avaliar visualmente os dados discrepantes por meio de uma matriz de dispersão (Gráfico 2). Para tanto, utilizou-se o método pairplot() da biblioteca Seaborn.

Ao considerar visualmente os dados discrepantes, catalogou-se as faixas de exclusão para cada atributo preditor no Quadro 7. Utilizando o método drop(), associado ao atributo index para retirar os exemplos que tivessem valores outliers e utilizou-se o atributo inplace = True para assegurar que as exclusões realizadas fossem mantidas, tendo como referência o dataframe vinhos (Figura 11). Depois, foi feita a verificação das dimensões atualizadas do dataframe vinhos, realizado por meio do atributo shape.

Gráfico 2 – Matriz de dispersão dos atributos preditores



Quadro 7 – Faixa de exclusão para cada atributo preditor utilizando a matriz de dispersão

Atributo Preditor	**Faixa de Exclusão**
Acidez Fixa	Faixa ≤ 4 (g(ácido tartárico) dm^{-3}) ou Faixa ≥ 11 (g(ácido tartárico) dm^{-3})
Acidez Volátil	Faixa ≥ 0,8 (g(ácido acético) dm^{-3})
Teor de Ácido Cítrico	Faixa ≤ 0,01 (g dm^{-3}) ou Faixa ≥ 1 (g dm^{-3})
Teor de Açucares Residuais	Faixa ≥ 30 (g dm^{-3})
Teor de Cloretos	Faixa ≥ 0,25 (g(cloreto de sódio) dm^{-3})
Teor de SO_2 Livre	Faixa ≥ 150 (mg dm^{-3})
Teor de SO_2 Total	Faixa ≥ 300 (mg dm^{-3})

Quadro 7 – Continuação

Densidade	Faixa ≥ 1,02(g cm^{-3})
pH	Faixa ≥ 3,8
Teor de Sulfetos	Faixa ≥ 1(g(sulfato de potássio) dm^{-3})
Teor de Álcool	Não houve outliers pelo método de dispersão

Figura 11 – Processo de eliminação dos exemplos com outliers pelo método de dispersão

```
#exclusão de registros com outliers

vinho.drop(vinho[vinho['acidez fixa']>=11].index, inplace = True)
vinho.drop(vinho[vinho['acidez fixa']<=4].index, inplace = True)
vinho.drop(vinho[vinho['acidez volátil']>=0.8].index, inplace = True)
vinho.drop(vinho[vinho['teor de ácido cítrico']>=1].index, inplace = True)
vinho.drop(vinho[vinho['teor de ácido cítrico']<=0.01].index, inplace = True)
vinho.drop(vinho[vinho['teor de açúcares residuais']>=30].index, inplace = True)
vinho.drop(vinho[vinho['teor de cloretos']>=0.25].index, inplace = True)
vinho.drop(vinho[vinho['teor de SO2 livre']>=150].index, inplace = True)
vinho.drop(vinho[vinho['teor de SO2 total']>=300].index, inplace = True)
vinho.drop(vinho[vinho['densidade']>=1.02].index, inplace = True)
vinho.drop(vinho[vinho['pH']>=3.8].index, inplace = True)
vinho.drop(vinho[vinho['teor de sulfatos']>=1].index, inplace = True)

print('dimensões igual a:',vinho.shape)
```

```
dimensões igual a: (4832, 12)
```

Observando a Figura 11 é possível notar que após a retirada dos dados com valores considerados *outliers*, o dataframe ficou com 4832 dados, ou seja, houve uma diminuição de 66 exemplos (1,3% dos dados). Comparativamente, o uso da matriz de dispersão foi menos rigorosa com valores discrepantes do que o método do boxplot. Salienta-se que o método de seleção via matriz de dispersão foi selecionado para a continuidade do trabalho, mas a comprovação de sua maior eficácia só ocorrerá no Quadro 7 (p. 34).

Seguindo com a análise dos dados, foram criadas duas arrays do NumPy a partir do dataFrame vinho. A primeira, denominada a_vinho, contendo os dados dos 11 atributos preditores, enquanto a segunda, chamada c_vinho, contém o atributo classe. Essa ação foi realizada utilizando o método iloc em conjunto com o atributo values (Figura 12).

Figura 12 – Geração das bases de dados dos atributos preditores e de classificação

```
# Dividir a base de dados em atributos e classificação
a_vinho = vinho.iloc[:, 0:11].values
c_vinho = vinho.iloc[:,11].values
print('dimensões da base preditoria:',a_vinho.shape)
print('dimensões da base de classe:',c_vinho.shape)
```

```
dimensões da base preditoria: (4832, 11)
dimensões da base de classe: (4832,)
```

Para entender as dimensões das arrays resultantes, foi utilizado o atributo shape, como ilustrado na Figura 12. Contudo, notou-se que os dados em a_vinhos apresentavam um possível problema para a aplicação dos futuros algoritmos de classificação: a escala de unidades era diferente entre os atributos. Buscando resolver esse problema, foi aplicado os métodos StandardScaler() da biblioteca scikit-learn e fit_transform(), gerando a array a_vinhos_p, conforme visto na Figura 13.

Figura 13 – Geração da base de dados padronizada dos atributos preditores

```
#padronização dos dados de avinho

pad = StandardScaler()
a_vinho_p = pad.fit_transform(a_vinho)
```

O processo de padronização dos dados envolveu o uso da Equação 3, na qual Z_i representa o valor padronizado com base no dado absoluto x_i de um atributo específico, $\overline{x}$ é a média amostral e s é o desvio-padrão amostral dos dados desse atributo.

$$z_i = \frac{x_i - \bar{x}}{s} \qquad \text{Eq. 3}$$

Após a aplicação da padronização, cada atributo preditor em a_vinhos_p passou a ter dados com média zero e desvio-padrão igual a 1, como evidenciado na Figura 14. Esse processamento foi realizado utilizando os métodos mean() (para calcular a média aritmética) e std() (para calcular o desvio-padrão amostral) da biblioteca NumPy. É importante observar que o parâmetro axis, quando igual a zero, permite o cálculo para cada coluna da array.

Figura 14 – Comprovação da padronização dos dados

```
#comprovação da padronização dos dados da base a_vinho

print((np.mean(a_vinho_p,axis=0)), '\n')
print((np.std(a_vinho_p, axis=0)))
```

```
[ 3.76446483e-16 -4.52912174e-16 -1.29403478e-16  2.13956888e-16
 -5.88197629e-18  1.31609219e-16  2.11751146e-16  7.50540175e-15
  3.25861486e-15  2.04398676e-16 -7.05837155e-16]

[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Além de avaliar os atributos preditores, analisou-se a situação dos dados relativos ao atributo classe. Inicialmente, procurou-se examinar a frequência das categorias geradas na avaliação sensorial dos vinhos. Para isso, utilizou-se o método unique() da biblioteca NumPy, combinado com o atributo return_counts = True para garantir a exibição das categorias e suas frequências. Para utilizar o método catplot(), foi necessária a criação um DataFrame (c_vinho_df) a partir da array c_vinhos, utilizando o método DataFrame() da biblioteca Pandas mostrado na Figura 15. Confirmou-se que a distribuição dos valores do atributo classe não é homogênea, com a categoria 6 sendo a mais frequente (2176 registros), sendo seguida pela categoria 5 com 1438 registros. A categoria 7, por sua vez, teve 875 registros. As categorias 8 e 4 tiveram 174 e 152, respectivamente. E as categorias menos frequentes foram 3 e 9, com 25 e 5 registros, respectivamente.

Essa distribuição não homogênea dos registros por categoria pode acarretar problemas na fase de treinamento e teste, pois o algoritmo pode favorecer as classes mais numerosas em registros (GARCIA, 2022).

O gráfico de barras, ilustrado na Figura 15, foi elaborado utilizando-se o método catplot() da biblioteca Seaborn. Os valores do atributo classe foram atribuídos ao eixo das abscissas, e as frequências foram alocadas ao eixo das ordenadas. Os rótulos dos eixos foram adicionados por meio dos métodos xlabel() e ylabel(), associados à biblioteca Matplotlib.

É importante destacar que, originalmente, Cortez et al. (2009) planejaram a faixa de categorias entre zero e 10 ao estabelecer as bases de dados para os vinhos tintos e brancos. No entanto, após a coleta dos dados sensoriais, as categorias zero, 1, 2 e 10 não apresentaram registros.

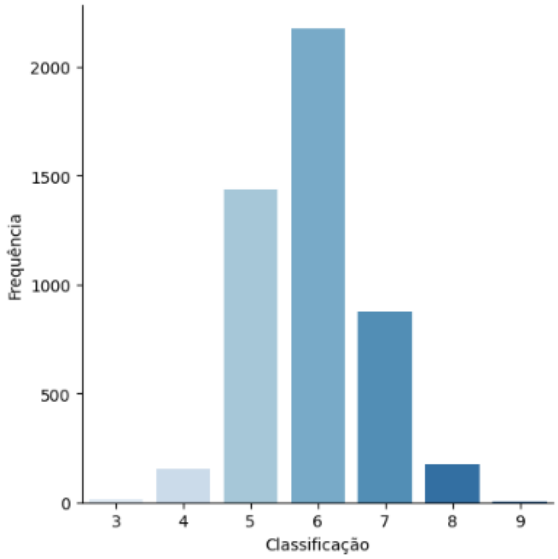
Para resolver o problema da distribuição não homogênea dos dados, foi necessário a utilização de técnicas de subamostragem. A primeira foi a reorganização das classes originais em outras três, sendo elas: Classe 1: vinhos de baixa qualidade (Classes originais 3 e 4); Classe 2: vinhos de qualidade moderada (Classes 5 e 6); Classe 3: vinhos de boa e alta qualidade

(Classes 7, 8 e 9). A segunda foi a reorganização das classes originais em outras duas, sendo elas: Classe 1: vinhos de baixa e média qualidade (Classes 3, 4, 5, 6); Classe 2: vinhos de boa e ótima qualidade (Classes 7, 8 e 9).

Figura 15 – Estruturação e gráfico de barras das classificações

```
#verificação da frequência das categorias de classificação

print(np.unique(c_vinho, return_counts=True))
c_vinho_df = pd.DataFrame(c_vinho)
sns.catplot(x = 0, kind = 'count', data = c_vinho_df, palette = 'Blues')
plt.xlabel('Classificação')
plt.ylabel('Frequência')
```

```
(array([3, 4, 5, 6, 7, 8, 9]), array([  16,  152, 1438, 2172,  875,  174,    5]))
Text(-3.6805555555555554, 0.5, 'Frequência')
```



A geração das novas categorias foi realizada por meio da associação de comandos condicionais "if" com o comando de repetição "for". Em termos gerais, o comando "for" foi utilizado para percorrer todos os dados da base de dados "c_vinhos". Os comandos "if" avaliaram os valores e os categorizaram em 1, 2 ou 3, organizando-os em três categorias com base nas faixas determinadas. Posteriormente, a frequência dos novos valores foi analisada por meio do método "unique" da biblioteca NumPy. É possível notar pela Figura 16 que a nova distribuição das classes ficaram da seguinte maneira: Classe 1 com 168 exemplos; Classe 2 com 3610 exemplos; Classe 3 com 1054 exemplos.

Para estruturar o gráfico de barras, conforme apresentado na Figura 17, o método catplot() da biblioteca Seaborn foi utilizado novamente.

Figura 16 – Geração da classificação com 3 categorias da base de dados c_vinho

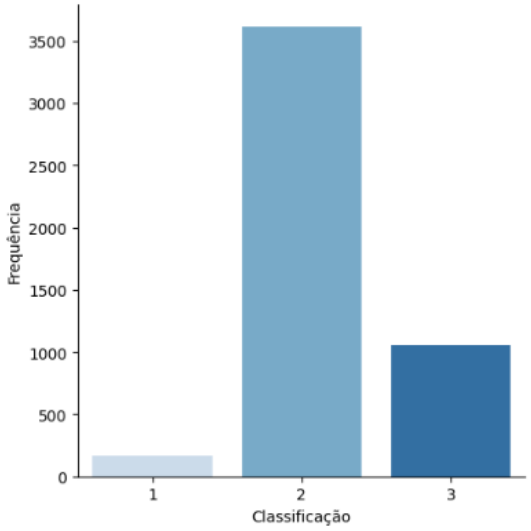
```
#distribuição dos dados de atributo classe em  novas categorias

for i in range(len(c_vinho)):
  if c_vinho[i] <=4:
    c_vinho[i] = 1
  if c_vinho[i] > 4 and c_vinho[i]<= 6:
    c_vinho[i] = 2
  if c_vinho[i] >= 7:
    c_vinho[i] = 3

np.unique(c_vinho, return_counts= True)
```

```
(array([1, 2, 3]), array([ 168, 3610, 1054]))
```

Figura 17 - Distribuição dos dados para a classificação em 3 Classes

```
c_vinho_df = pd.DataFrame(c_vinho)
sns.catplot(x = 0, kind = 'count', data = c_vinho_df, palette = 'Blues')
plt.xlabel('Classificação')
plt.ylabel('Frequência')
```

```
Text(-3.6805555555555554, 0.5, 'Frequência')
```



Após a reorganização das categorias, ainda se observou uma discrepância nas frequências, mesmo após a soma das categorias originais 3 e 4, além da soma das categorias originais 7, 8 e 9. Com o objetivo de reduzir a frequência da categoria 2, que era majoritária, optou-se pela aplicação do método de subamostragem chamado TomekLinks. Esse algoritmo visa eliminar o desequilíbrio entre as classes, removendo os Tomek Links de classes majoritárias até que todos os pares de vizinhos mais próximos pertençam à mesma classe.

Para isso, utilizou-se os métodos TomekLinks() e fit_resample() da biblioteca imbalanced-learn, configurando o parâmetro sampling_strategy como majority, com o intuito de preservar o número de registros das categorias 1 e 3. As bases de dados utilizadas no

processamento foram a_vinho_p e c_vinho, resultando nas criações de a_vinhos_p_sub (atributos preditores) e c_vinhos_sub (atributo classe), conforme ilustrado na Figura 18.

Figura 18 – Aplicação do processo de subamostragem Tomeklinks

```
#subamostragem Tomeklink
tl = TomekLinks(sampling_strategy='majority')
a_vinho_p_sub, c_vinho_sub = tl.fit_resample(a_vinho_p, c_vinho)

np.unique(c_vinho_sub, return_counts=True)
```

```
(array([1, 2, 3]), array([ 168, 3467, 1054]))
```

É possível observar na Figura 18, a frequência da categoria 2 foi reduzida de para 3467, uma diminuição de 143 exemplos ou 3,9%, enquanto as frequências das categorias 1 e 3 foram mantidas constantes. Mesmo com a redução da frequência da categoria 2, não foi possível atingir a condição ideal de similaridade desejada. No entanto, é importante notar que as categorias originais apresentam um perfil desafiador em relação à aplicação de modelos de aprendizado de máquina.

A criação da base de dados com duas categorias foi feita também utilizando os comandos if e for, tendo 3778 registros na Classe 1 e 1054 registros na Classe 2, como mostrado na Figura 19.

Figura 19 – Criação do banco de dados com duas classes

```
[11] #distribuição dos dados de atributo classe em  novas categorias

     for i in range(len(c_vinho)):
       if c_vinho[i] <=6:
         c_vinho[i] = 1
       if c_vinho[i] >= 7:
         c_vinho[i] = 2
     np.unique(c_vinho, return_counts = True)
```

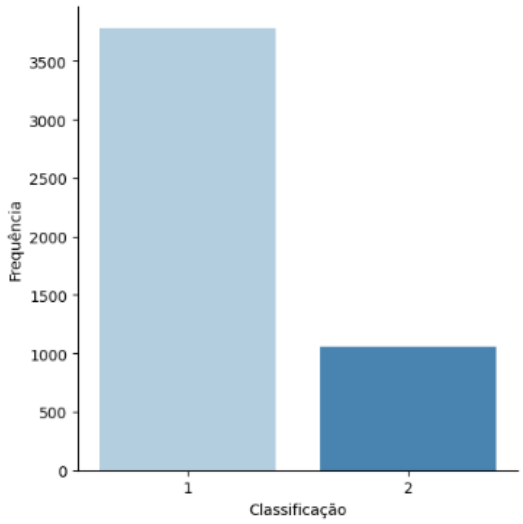
```
(array([1, 2]), array([3778, 1054]))
```

Para estruturar o gráfico de barras, conforme apresentado na Figura 20, o método catplot() da biblioteca Seaborn foi utilizado novamente.

Figura 20 – Criação do gráfico com barras com duas classes

```
c_vinho_df = pd.DataFrame(c_vinho)
sns.catplot(x = 0, kind = 'count', data = c_vinho_df, palette = 'Blues')
plt.xlabel('Classificação')
plt.ylabel('Frequência')
```

```
Text(-3.6805555555555554, 0.5, 'Frequência')
```



É possível notar pelo gráfico que mesmo utilizando apenas duas classificações ainda há uma discrepância entre as duas classes, por isso foi utilizado novamente o método de subamostragem TomekLinks, ilustrado na Figura 21. Após a aplicação do método a classe, que é a majoritária teve uma redução em seus dados de 112 registros (2,9%).

Figura 21 – Aplicação do método de subamostragem Tomeklinks para o banco de dados com duas classes

```
[14] #subamostragem Tomeklink
     tl = TomekLinks(sampling_strategy='majority')
     a_vinho_p_sub, c_vinho_sub = tl.fit_resample(a_vinho_p, c_vinho)

     np.unique(c_vinho_sub, return_counts=True)
```

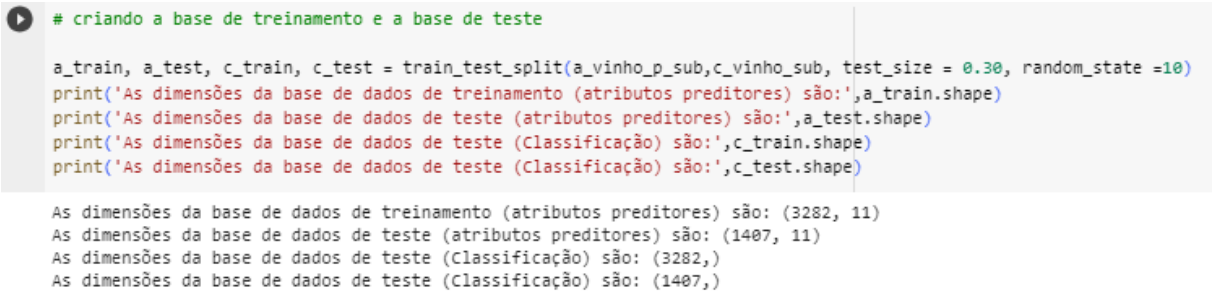
```
(array([1, 2]), array([3666, 1054]))
```

Vale ressaltar que mesmo depois da criação de novas distribuições de Classes (2 e 3 Classes) e a utilização do método Tomeklinks, a base de dados não apresenta a disposição ideal para o funcionamento do algoritmo. Ainda assim, comparada às Classes originais e à proposta de 3 Classes, apresentou maior eficácia. Entretanto, este resultado será provado apenas no Quadro 7 (p. 34).

4.5 Treinamento dos dados

Com o propósito de estruturar os modelos de *Machine Learning*, as bases de dados a_vinho_p_sub e c_vinho_sub foram utilizadas na formação das bases de dados de treinamento denominadas a_train e c_train e de teste denominadas a_test e c_test, conforme representado na Figura 22. Esse procedimento foi feito por meio da utilização do método train_test_split() disponível na biblioteca Scikit-learn.

Figura 22 – Geração das bases de treinamento e teste



```
# criando a base de treinamento e a base de teste

a_train, a_test, c_train, c_test = train_test_split(a_vinho_p_sub,c_vinho_sub, test_size = 0.30, random_state =10)
print('As dimensões da base de dados de treinamento (atributos preditores) são:',a_train.shape)
print('As dimensões da base de dados de teste (atributos preditores) são:',a_test.shape)
print('As dimensões da base de dados de teste (Classificação) são:',c_train.shape)
print('As dimensões da base de dados de teste (Classificação) são:',c_test.shape)

As dimensões da base de dados de treinamento (atributos preditores) são: (3282, 11)
As dimensões da base de dados de teste (atributos preditores) são: (1407, 11)
As dimensões da base de dados de teste (Classificação) são: (3282,)
As dimensões da base de dados de teste (Classificação) são: (1407,)
```

Foi decidido que a base de dados de teste conteria 30% dos rótulos originais, em vez do padrão de 25%. Essa decisão se justifica pelo fato de haver categorias de baixa frequência dentro do atributo classe. Assim, assegura-se um volume maior de dados para a fase de teste, sem prejudicar o processo de treinamento. Considerando que a quantidade de rótulos antes do processamento era de 4689, 70% (3282) foram alocados nas bases de dados de treinamento, enquanto 30% (1407) permaneceram nas bases de dados de teste. Além disso, optou-se por estabelecer uma semente para o gerador aleatório, utilizando random_state = 10, para garantir a replicabilidade dos resultados, assegurando que, em processamentos subsequentes, os mesmos rótulos selecionados para as bases de dados de treinamento e teste fossem mantidos.

Para otimizar o desempenho do modelo foi empregado o método GridSearchCV da biblioteca Scikit-learn. Este método permitiu a seleção dos melhores hiperparâmetros.

O modelo de Árvore de Decisão foi submetido ao método GridSearchCV(), que considerou os hiperparâmetros criterion (função para medir a qualidade de uma divisão sendo eles a entropia e gini); max_depth (extensão máxima da árvore); min_samples_split (número mínimo de amostras necessárias para dividir um nó interno) e min_samples_leaf (número mínimo de amostras necessárias para estar em um nó folha) (Figura 23). O Quadro 8 disponibiliza o resultados para cada teste.

Figura 23 – Implementação do GridSearchCV com o modelo de árvore de decisão

```
[16] #gridSearchCV - Árvore de Decisão

     param_dict = {'criterion':['gini','entropy'],
                   'max_depth': range(1, 10),
                   'min_samples_split': range(1, 10),
                   'min_samples_leaf': range(1, 5)}
     grid = GridSearchCV(DecisionTreeClassifier(), param_grid = param_dict, cv = 10, verbose = 1, n_jobs = -1)
     grid.fit(a_train, c_train)

     print('\n', '\n', 'Os melhores parâmetros são', grid.best_params_)
     print('A acurácia é igual a', grid.best_score_, '\n', '\n')

     Fitting 10 folds for each of 648 candidates, totalling 6480 fits


      Os melhores parâmetros são {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 1, 'min_samples_split': 5}
     A acurácia é igual a 0.8181213952210932
```

Quadro 8 – Resultado dos testes GridSearchCV para a árvore de decisão

Teste	Resultado
1	Os melhores parâmetros são ('criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 1, 'min_samples_split': 5) **A acurácia igual a 0.8181213952210932**
2	Os melhores parâmetros são ('criterion': 'gini', 'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 2) A acurácia é igual a 0.787012565794351
3	Os melhores parâmetros são ('criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 5) A acurácia é igual a 0.7953695702269846
4	Os melhores parâmetros são ('criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 1, 'min_samples_split': 2) A acurácia é igual a 0.7586538182451454

Concluído o experimento, o modelo que alcançou a seguinte acurácia de treinamento foi o Teste 1 (exclusão de *outliers* por meio da matriz de dispersão; reorganização das Classes originais em duas) com os seguintes hiperparâmetros: criterion = gini, max_depth = 9, min_samples_leaf = 1 e min_samples_split = 5.

Considerando o número de categorias o valor da acurácia foi menor nos modelos usando três categorias, pois duas das três classes tinham poucos exemplos considerando a classe majoritário. Estima-se que o algoritmo teve dificuldade em classificar exemplos dessas duas categorias, o que levou a uma diminuição do desempenho. Um outro ponto importante é que mesmo os modelos com duas categorias tiveram uma discrepância entre suas classes o que também acarreta um menor desempenho na classe minoritária, por isso os valores das acurácias não tiveram uma diferença muito expressiva.

Após isso, um *loop* for foi implementado com 100 interações, cada uma representando um processo de processamento de dados independente. O loop foi desenvolvido apenas para o modelo de melhor acurácia e levando em conta os hiperparâmetros selecionados pelo método de GridSearchCV. Cada iteração englobou as seguintes ações: reformulação das bases de dados de treinamento e teste usando o método train_test_split() da biblioteca scikit-learn, com um atributo test_size igual a 0,30; associação do método do modelo classificador a um objeto, o qual foi utilizado para treinar as bases de dados por meio do método fit(); comparação dos valores da base de dados c_test com os valores de c_pred, a base de dados de valores previstos pelo modelo de classificação, utilizando o método predict() com base nos dados a_test.

Em cada interação, os valores das métricas de avaliação foram adicionados nas listas correspondentes por meio do método append(). A acurácia de treinamento foi calculada usando o método score() e a acurácia de previsão por meio do método accuracy_score(), da biblioteca Scikit-learn, com base nas bases de dados c_test e c_pred.

Para o cálculo das demais métricas de avaliação, os dados da matriz de confusão, para classificação binárias, foram utilizados. Essa matriz foi obtida por meio do método confusion_matrix() da biblioteca scikit-learn, empregando as bases de dados c_test e c_pred. A cada valor obtido, o resultado foi incluído na lista correspondente por meio do método append().

A análise dos resultados foi conduzida com a utilização do método print(), no qual foram incorporados textos informativos, destacados entre apóstrofos. O cálculo das médias aritméticas foi realizado por meio do método mean(), enquanto o cálculo dos desvios-padrão foi executado com o método std(). Ambos esses métodos fazem parte da biblioteca numpy. Com o propósito de facilitar a interpretação dos dados, foi empregado o método round() para arredondar os valores para apenas três casas decimais, todos esses passos estão ilustrado na Figura 24.

Figura 24 – Aplicação do modelo de Árvore de Decisão e obtenção das médias das métricas

```
# calculo das metricas
ac_trein = []
ac_prev = []
prec_cat1 = []
prec_cat2 = []
prec_cat3 = []
rec_cat1 = []
rec_cat2 = []
rec_cat3 = []
f1_cat1 = []
f1_cat2 = []
f1_cat3 = []
```

Figura 24 - Continuação

```
for i in range(0,100):
  a_train, a_test, c_train, c_test = train_test_split(a_vinho_p_sub, c_vinho_sub, test_size = 0.30)
  arv_dec = DecisionTreeClassifier(criterion = 'gini',max_depth= 9, min_samples_leaf= 1, min_samples_split= 5)
  arv_dec = arv_dec.fit(a_train, c_train)
  c_pred = arv_dec.predict(a_test)
  cm = confusion_matrix(c_test, c_pred)
  ac_trein.append(arv_dec.score(a_train, c_train))
  ac_prev.append(accuracy_score(c_test, c_pred))
  prec_cat1.append(cm[0,0]/(cm[0,0]+cm[1,0]))
  prec_cat2.append(cm[1,1]/(cm[1,1]+cm[0,1]))
  rec_cat1.append(cm[0,0]/(cm[0,0]+cm[0,1]))
  rec_cat2.append(cm[1,1]/(cm[1,1]+cm[1,0]))
  f1_cat1.append(2*((cm[0,0]/(cm[0,0]+cm[1,0]))*(cm[0,0]/(cm[0,0]+cm[0,1])))/
                 ((cm[0,0]/(cm[0,0]+cm[1,0]))+(cm[0,0]/(cm[0,0]+cm[0,1]))))
  f1_cat2.append(2*((cm[1,1]/(cm[1,1]+cm[0,1]))*(cm[1,1]/(cm[1,1]+cm[1,0])))/
                 ((cm[1,1]/(cm[1,1]+cm[0,1]))+(cm[1,1]/(cm[1,1]+cm[1,0]))))
  f1_cat3.append(2*((cm[0,0]/(cm[0,0]+cm[1,0]))*(cm[0,0]/(cm[0,0]+cm[0,1])))/
                 ((cm[0,0]/(cm[0,0]+cm[1,0]))+(cm[0,0]/(cm[0,0]+cm[0,1]))))
```

```
print('Acurácia de treinamento:', round(np.mean(ac_trein), 3), '+/-', round(np.std(ac_trein), 3))
print('Acurácia de previsão:', round(np.mean(ac_prev), 3), '+/-', round(np.std(ac_prev), 3))

print('Precisão da categoria 1:', round(np.mean(prec_cat1), 3), '+/-', round(np.std(prec_cat1), 3))
print('Precisão da categoria 2:', round(np.mean(prec_cat2), 3), '+/-', round(np.std(prec_cat2), 3))


print('f1-score da categoria 1:', round(np.mean(f1_cat1), 3), '+/-', round(np.std(f1_cat1), 3))
print('f1-score da categoria 2:', round(np.mean(f1_cat2), 3), '+/-', round(np.std(f1_cat2), 3))


ac_prev_arv = np.copy(ac_prev)
prec_cat1_arv = np.copy(prec_cat1)
prec_cat2_arv = np.copy(prec_cat2)

rec_cat1_arv = np.copy(rec_cat1)
rec_cat2_arv = np.copy(rec_cat2)

f1_cat1_arv = np.copy(f1_cat1)
f1_cat2_arv = np.copy(f1_cat2)
```

```
Acurácia de treinamento: 0.913 +/- 0.01
Acurácia de previsão: 0.815 +/- 0.01
Precisão da categoria 1: 0.873 +/- 0.012
Precisão da categoria 2: 0.593 +/- 0.034
f1-score da categoria 1: 0.882 +/- 0.007
f1-score da categoria 2: 0.566 +/- 0.028
```

Os resultados da obtidos pela árvore de decisão serão discutidos no próximo tópico.

4.6 Avaliação dos dados

O modelo criado baseado na Árvore de Decisão obteve as seguintes medias métricas (Quadro 9).

Observando o Quadro 9 é possível notar que as métricas relacionadas à Classe 2 foram menores que as da Classe 1. Isso já era esperado já que houve uma grande diferença no número de exemplos de cada classe. Em outras palavras, o modelo foi mais eficiente na previsão da Classe 1 (vinhos de qualidade sensorial pior).

Quadro 9 – Dados descritivos das métricas do modelo de Árvore de Decisão considerando 100 processamentos

Métrica	**Média ± Desvio-padrão**
Acurácia de treinamento	0,913 ± 0,010
Acurácia de teste	0,815 ± 0,010
Precisão Classe 1	0,873 ± 0,012
Precisão Classe 2	0,593 ± 0,034
F1-score Classe 1	0,882 ± 0,007
F1-score Classe 2	0,566 ± 0,028

De modo geral, pode-se afirmar que a base de dados de treinamento (70% dos registros selecionados) apresentou elevada acurácia, ou seja, elevada porcentagem de acertos de classificação considerando todos os registros. A acurácia de teste também foi alta, destacando-se que 30% dos registros foram utilizados, um volume representativo de dados. Assim, se o atual modelo fosse utilizado na avaliação de novas amostras de vinhos brancos, espera-se sucesso nas classificações corretas próximo a 81,5%.

Mas, quando se busca apenas a classificação dos vinhos da Classe 1 (menor qualidade sensorial), a precisão tem valores mais próximos do ideal, pois corresponde à proporção de resultados verdadeiros positivos no conjunto de todos os resultados classificados como positivos (Vp e Fp). Assim, tem-se alta incidência de classificações realmente corretas da Classe 1 em todas as que o modelo supõe serem corretas.

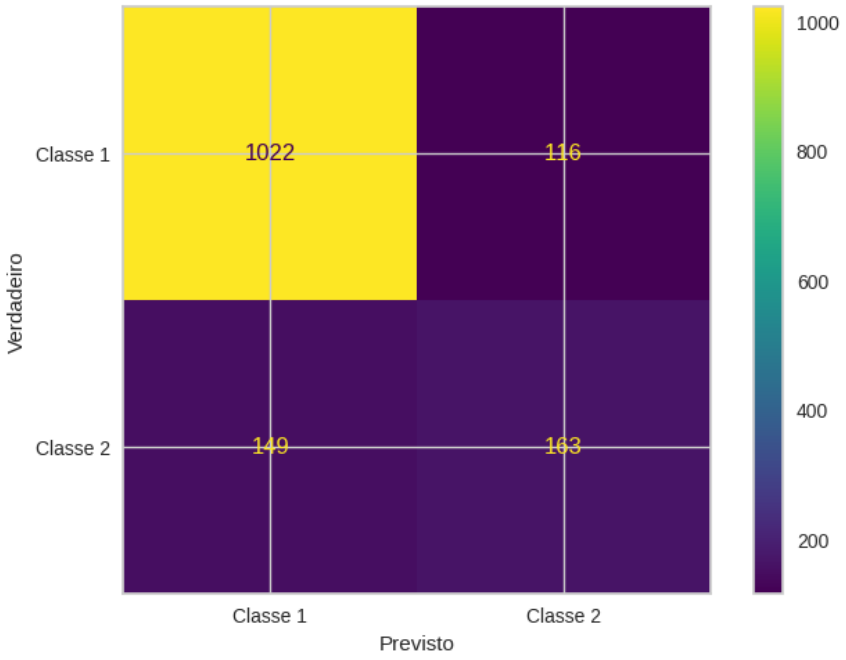
O mesmo comportamento ocorreu com os dados da métrica F1-score. Como ela relaciona a precisão com a revocação, maiores mais elevados indicam que o modelo é capaz de classificar de forma mais correta os resultados verdadeiros positivos dentro do conjunto de dados que realmente é positivo e no conjunto de dados classificado como positivo. Isso indica que a hipótese de que a maior proporção dos registros da classe 1 favorece sua classificação foi confirmada.

Como forma didática de ilustração dos dados previamente explicados, apresenta-se na Figura 25 uma matriz de confusão para um único processamento dos dados de teste por meio do modelo de Árvore de Decisão adquirido.

Figura 25 – Estrutura e gráfico de matriz de confusão do processamento único da base de dados

```
#Árvore de Decisão

AD = tree.DecisionTreeClassifier(criterion = 'gini', max_depth = 9,
                                 min_samples_leaf = 1, min_samples_split = 5, random_state= 5)
AD.fit(X_train, y_train)

y_pred = AD.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Classe 1', 'Classe 2'])
cmd.plot()
cmd.ax_.set(xlabel='Previsto', ylabel='Verdadeiro')
plt.savefig("Matriz de confusão - RN.png", dpi = 300)
```



Mesmo considerando um único processamento, verifica-se que o número de classificações verdadeiras positivas para a Classe 1 é muito representativo considerando o número de registros da base de dados de teste, motivo pelo qual as métricas de precisão e F1-score desta classe são elevados.

5 CONCLUSÃO

O trabalho oferece uma visão abrangente do processo de construção de um modelo de *Machine Learning* destacando a importância da preparação adequada dos dados e da escolha de técnicas para lidar com desequilíbrios nas classes. Os resultados obtidos são promissores e sugerem que, quando submetido a um processo cuidadoso de pré-processamento, o modelo revelou-se capaz de discernir padrões complexos, oferecendo uma boa acurácia tanto no treinamento quanto no teste.

Constatou-se, com base na acurácia do modelo, que a eliminação dos outliers usando o critério do boxplot afetaria negativamente a métrica. Por outro lado, foi necessário abdicar das Classes originais de análise sensorial para reduzir o impacto da discrepância de distribuição dos registros. Ao final, a reorganização em duas Classes foi a que apresentou melhor acurácia.

A realização dos 100 processamentos foi muito adequada, pois reduz o erro de seleções enviesadas de registros na formação das bases de dados de treinamento e de teste.

Ao final obteve-se elevada acurácia de treinamento e bons resultados de acurácia de teste, precisão da Classe 1 e F1-score da Classe 1. Na prática, o modelo seria mais adequado para classificar vinhos brancos de baixa qualidade sensorial. Mas, há a possibilidade de otimizar o modelo considerando a necessidade de manutenção de todas as variáveis preditoras e, se possível, fazendo uso de mais dados de vinhos brancos classificados sensorialmente com de alta qualidade.

REFERÊNCIAS

ANGUS, DALE; **Modeling Wine Quality from Physicochemical Properties.** [s.d] Disponivel em: https://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/25895690.pdf. Acesso em: 25 Out. 2023.

BERRY, M, W; MOHAMED, A; YAP, B, W; **Supervised and Unsupervised Learning for Data Science.** 1. ed. Cham: Editora Springer, 2020. Doi: https://doi.org/10.1007/978-3-030-22475-2

BOADA, J, L; BOADA, B, L; LÓPES, V, D; Algoritmo de Aprendizaje por Refuerzo Continuo para el Control de um Sistema de Suspensión Semi-Activa. **Revista Iberoamericana de Ingenieria Mecánica**, Madrid, v. 9, n. 2, p. 77-91, nov. 2005. Disponível em: http://62.204.194.45/fez/eserv/bibliuned:iberoingmecanica-2005-vol09-n2-08/Documento.pdf. Acesso em: 13 Out. 2023.

BORGES, L, E; **Python Para Desenvolvedores.** 1 ed. São Paulo: Editora Novatec .2014. Disponível em: https://books.google.com.br/books?hl=ptBR&lr=&id=eZmtBAAAQBAJ&oi=fnd&pg=PA14&dq=python&ots=VETnpkBlim&sig=4pvMMpuiara8AoKoaNWlfaQ02YI#v=onepage&q=python&f=false. Acesso em: 28 Mar 2023

COELHO, L,P; RICHERT, W. **Building Machine Learning Systems with Python.** 2. ed. Birmingham: Editora Packt Publishing, 2015.

CORTEZ, P et al; Modeling Wine Preferences by Data Mining From Physicochemical Properties. **Decision Support Systems**, v. 47, n. 4, p. 547–533, Editora Elsevier, 2009. Doi: 10.1016/j.dss.2009.05.016

DA SILVA, M, A, F; DE CASTRO, A, F; OLIVEIRA FILHO, I, L; Modelos de Predição Aplicados no Diagnóstico do AVC: Uma Revisão de Escopo. **Journal of Health Informatics,** v. 15, n. 2, p. 39 – 45, 2023. DOI: https://doi.org/10.59681/2175-4411.v15.i2.2023.980. Disponível em: https://jhi.sbis.org.br/index.php/jhi-sbis/article/view/980. Acesso em: 24 Out. 2023.

DA SILVA, R, R; **Áreas em que Python vem Sendo Utilizado No Machine Learning:** Um Mapeamento Sistemático. Dissertação (Pós-graduação em Tecnologia Python para Negócios). Universidade Tecnológica Federal do Paraná. Dois vizinhos.- PR, 2023. Disponível em: https://riut.utfpr.edu.br/jspui/bitstream/1/31614/3/areaspythonaprendizadomaquina.pdf. Acesso em: 11 Out. 2023.

DE CAMPOS, T, C; DE VASCONCELOS, T, C, L; Aplicação de Algoritmo de Machine Learning na Área Farmacêutica: Uma Revisão. **Research, Society and Development**, v. 10, n. 15, p. 1-9, 2021. DOI: 10.33448/rsd-v10i15.22862. Disponível em: https://rsdjournal.org/index.php/rsd/article/view/22862. Acesso em: 25 out. 2023.

DE PINA JR, J, C; MORIMOTO, C, H; **Introdução à Computação com Python:** Um Curso Interativo. 2020. Disponível em: https://panda.ime.usp.br/cc110/static/cc110/00-prefacio.html. Acesso em: 28 Mar. 2023.

DE VILLE, B; Decision trees. **Wiley Periodicals,** vol. 5, n. 6, p. 448 - 455, Dez. 2013. doi: 10.1002/wics.1278

DOMINGOS, P. A few useful things to know about machine learning. **Communications of the ACM**, New York, v. 55, n. 10, p. 78–87, out. 2012.

DOS SANTOS, A, L; **Uso de Rede Neural para Desenvolvimento de Sistema Especialista para Diagnose de Doenças Foliares em Eucaliptos.** Dissertação **(**Pós-Graduação em Planejamento e Uso de Recursos Renováveis). Universidade Federal de São Carlos. Sorocaba - SP, 2021.

FALQUETO, A, A; CEZAR, L, C; Segmentação via Machine Learning: Proposta de Clusterização de Consumidores do E-Commerce de uma Empresa Multinacional do varejo Esportivo. **Holos,** v. 4, n. 10, 2021. DOI: 10.15628/holos.2021.12032

FAVAN J. R.; **Utilização de redes neurais artificias aplicadas na discriminação de padrões de doenças florestais**. Dissertação (Mestrado em ciência Florestal) - Faculdade de Ciências Agronômicas da UNESP – Campus de Botucatu, Botucatu, 2015.

FERREIRA, F, V et al; Uso de Python para Detecção de Fake News sobre a Covid-19: Desafios e Possibilidades. **Revista Eletrônica de Comunicação, Informação & Inovação em Saúde**, Rio de Janeiro, v. 16, n. 2, p. 266-280, abr.-jun. 2022. DOI: https://doi.org/10.29397/reciis.v16i2.3253. Disponível em: https://www.reciis.icict.fiocruz.br/index.php/reciis/article/view/3253. Acesso em: 24 Out. 2023.

GARCIA, C. F. **Avaliação de diferentes modelos de Machine Learning para a classificação da qualidade de vinhos**. Trabalho de Conclusão de Curso (Pós-graduação em Ciência de Dados e Big Data). Pontifícia Universidade Católica de Minas Gerais. Belo Horizonte - MG, 2022.

GARCIA, S, C; **Uso de Árvore de Decisão na Descoberta de Conhecimento na Área da Saúde.** Dissertação (Mestrado em Ciência da Computação). Universidade Federal do Rio Grande do Sul. Porto Alegre, **2003.**

GEVORKYA, M, N et al; Review and Comparative Analysis of Machine Learning Libraries for Machine Learning. **Discrete e Continuous Models e applied Computational Science**, Moscow, v 27, n. 4, p. 305-315, dez. 2019.

GKIKAS, D, C; et al; Finding Good Attribute Subsets for Improved Decision Trees Using a Genetic Algorithm Wrapper; a Supervised Learning Application in the Food Business Sector for Wine Type Classification**. Informatics**. v. 10, n. 3, p. 1-30, 2023. Disponível em: https://www.mdpi.com/2227-9709/10/3/63. Acesso em: 03 Out. 2023.

GUARNIZO, J, A, Y; **Métodos Supervisionados de Machine Leaarning Aplicados à Produtividade Agrícola de Cana-de-Açúcar.** Dissertação (Mestrado em Engenharia Agrícula, na área de Máquinas Agrículas). Faculdade de Engenharia Agrícola. Universidade Estadual de Campinas. Campinas, 2021. Disponível em: https://repositorio.unicamp.br/acervo/detalhe/1166814. Acesso em: 01 Out. 2023.

HABIB, M, T; **Machine Vision Based Papaya Disease Recognition. Journal of King Saud University-Computer and Information Sciences.** v. 32, n 10, p.300-309, 2020. Disponível em: https://www.sciencedirect.com/science/article/pii/S1319157818302404?via%3Dihub. Acesso em: 03 Out 2023.

HAN, J; KAMBER, M; PEI, J. **Data mining:** Concepts and Techniques. 3. ed. Waltham: Editora Elsevier, 2012.

HAYKIN, S. **Redes neurais: princípios e prática**. 2.Ed. Porto Alegre: Bookman, 2001. p. 27–32.

HOFMANN, T. Unsupervised learning by probabilistic latent semantic analysis. **Machine Learning**, Providence, v. 42, p. 177–196, 2001.

KAMIRI, J; MARIGA, G; Research Methods in Machine Learning: A Content Analysis. **International Journal of Computer and Information Tecchnology,** v. 10, n. 2, p. 78 - 91, mar. 2021.

KINGSFORD, C; SALZBERG, S, L; What are Decision trees? **Nature Biotechnology,** v. 26, n. 9, p. 1011-1013 set. 2008. Doi: https://doi.org/10.1038/nbt0908-1011

KUMAR, P et al; Analysis of Various Machine Learning Algorithmss for Enhanced Opinion Mining using Twitter Data Streams. **2016 International Conference on Micro-Electronics and Telecommunication Engineering** (ICMETE), Ghaziabad, p.265-270, 2016 Disponível em: https://ieeexplore.ieee.org/abstract/document/7938923. Acesso em: 02 Out. 2023.

KWEKHA-RASHID, A, S; ABDULJABBAR, H, N; ALHAYANI. B; Coronavirus Disease (COVID-19) Cases Analysis Using Machine-Learning Applications. **Applied Nanoscience.** v. 13, p. 2013-2025, mai. 2021. DOI: https://doi.org/10.1007/s13204-021-01868-7.

LEMOS, E, P; STEINER, M, T, A; NIEVOLA, J, C; Análise de Crédito Bancário por Meio de Redes Neurais e Árvores de Decisão: um Aplicação Simples de Data Mining. **Revista de Administração-RAUSP,** vol. 40, n. 3, p 225 – 234, Jul-set. 2005. Disponível em: https://www.redalyc.org/pdf/2234/223417392002.pdf. Acesso em: 22 Out. 2023.

LEMOS, G; SILVA, R; BERNARDINO, J; **Decision Tree Algorithm Application for diagnosis of Mental Disorders Symptoms.** CAPSI, Porto2020. Disponível em: https://aisel.aisnet.org/capsi2020/2. Acesso em: 22 Out. 2023.

MAHESH, B; Machine Learning Algorithms**:** A Review. **International Journal of Science and Research.** v. 9, n. 1, p 381-386, jan. 2019.

MEIRA, C, A, A; RODRIGUES, L, H, A; MORAES, S, A; Análise de Epidemia da Ferrugem do Cafeeiro com Árvore de Decisão. **Tropical Plant Pathology**, vol. 33, n. 2, p. 114-124, mar-abr. 2008. Disponível em: https://www.scielo.br/j/tpp/a/gwhNLwQB58hkwJSSWdJ7gbB/?format=pdf&lang=pt. Acesso em: 22 out 2023.

MENEZES, N, N ,C; **Introdução à Programação com Python:** Algoritmos e lógicas de programação para iniciantes**.** 2. ed. São Paulo: editora Novatec. 2014. 25-26 p. Disponível em: https://s3.novatec.com.br/capitulos/capitulo-9788575222508.pdf. Acesso em: 6 out. 2022.

MITCHELL, T, M; **Machine Learning.** 1. ed. Ithaca: McGraw-Hill Science/Engineering/Math, 1997. p. 432.

MONARD, M, C; BARANAUSKAS, J, A; **Conceitos sobre Machine Learning.** ed. 1. Barueri: editora Manole Disponivel em: https://dcm.ffclrp.usp.br/~augusto/publications/2003-sistemas-inteligentes-cap4.pdf. Acesso em: 01 Out. 2023.

NTI, I, K et al; A Mini-Review of Machine Learning in Big Data Analytics: Applications, Challenges, and Prospects. **Big Data Mining ad Analytics**, v. 5, n. 2, p. 81 – 97, 2022. DOI: 10.26599/BDMA.2021.9020028

PIANUCCI, M, N; PITOMBO, C, S; Uso de Árvore de Decisão para Previsão de Geração de Viagens como Alternativa ao Método de Classificaçao Cruzada. **Revista de Engenharia Civil**, n. 56, p. 5–13, 2019.

PEREIRA, L, F, L; MORAES, I, M; MATTOS, D, M, F; Floresta de Decisão Distribuída: Um Sistema de Aprendizado de Máquina Colaborativo Par-a-Par para Detecção de Intrusão em Redes. *In*: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS (SBRC), 41. , 2023, Brasília/DF. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2023. p. 253-266. DOI: https://doi.org/10.5753/sbrc.2023.469.

QIU, J et al; A Survey of Machine Learning for Big Data Processing. **EURASIP Journal on Advances in Signal Processing**, n. 67, p. 1 – 16, 2016. DOI: 10.1186/s13634-016-0355-x

RAMALHO, G et al; Classificação de Linguagens de Programação Utillizando Técnicas de Machine Learning. ENCONTRO NACIONAL DE MODELAGEM COMPUTACIONAL, 21, 2018, Búzios/RJ. **Anais**[...]. Intituto Federal Fluminense, 2018. p. 1-10. Disponivel em: https://editoraessentia.iff.edu.br/index.php/enmc-ectm/article/view/13070/10490. Acesso: 10 out 2023.

RASCHKA, S; PARTTERSON, J; NOLET, C; **Machine Learning in Python:** Main Developments and Technology Trends in Data Science, Machine Learning and Artificial Intelligence. **Information,** Baltimore, v. 11, n. 4, p. 1-44, 2020. Disponível em:< https://www.mdpi.com/2078-2489/11/4/193>. Acesso em: 02 Out 2023.

RATHEE, N; JOSHI, N; KAUR, J; Sentiment Analysis using Machine Learning Techniques on Python. **Proceedings of the Second International Conference on Intelligent Computing and Control Systems**. v. 18, p. 779-786, 2018**.** DOI: 10.1109/ICCONS.2018.8663224

REIS FILHO, J; **Sistema Inteligente Baseado em Árvore de Decisão para Apoio ao Combate às perdas Comerciais na Distribuição de Energia Elétrica.** Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Uberlândia. Uberlândia, 2006. Disponível em: https://repositorio.ufu.br/handle/123456789/14493. Acesso em: 02 Out 2023.

SANTHANAM, T; SUNDARAM, S. Application of CART in Blood Donors Classification. **Journal of Computer Science**, v. 6, n. 5, p. 548–552, 2010.

SHEDTHI, S, B.; SIDDAPPA, M.; SHETTY, S. Identification of plant leaf disease using machine learning techniques**. International Journal of Recent Technology and Engineering** v. 8, n. 3, p. 6077-6081. 2019 DOI: 10.35940/ijrte.C5621.098319.

SHINDE, P, P; SHAH, S. A Review of Machine Learning and Deep Learning Applications. In: INTERNATIONAL CONFERENCE ON COMPUTING COMMUNICATION CONTROL AND AUTOMATION, 14, 2018, Pune/India . **Anais**[…].Institute of Electrical and Electronics Engineers, 2018. p.1-6.

TREVISAM, B, A; Aplicação de Ferramentas Inteligentes de Classificação de Dados não Estruturados como Suporte a Gestão de Ativos em Sistemas Elétricos de Potência. (2023). DOI: https://doi.org/10.11606/D.18.2023.tde-16032023-075818

VARGAS, R, G; CUNHA, F, S; FERNANDES, A, M, R; Aplicação de Aprendizado de Máquina para Predição do Preço da Cesta Básica. Computer on the Beach, v. 14, Florianópolis/SC. 2023. p. 504-505.

VIEIRA, W, C; **Analisando Cerveja Artesanal por Meio de 3 Modelos de Classificação e Machine Learning.** Tese (Pós-Graduação em modelagem e otimozação) - Instituto de Biotecnologia. Universidade Federal de Catalão. Catalão. 2023. Disponível em: https://www.researchgate.net/publication/373803725_ANALISANDO_CERVEJA_ARTESANAL_POR_MEIO_DE_3_MODELOS_DE_CLASSIFICACAO_E_APRENDIZADO_DE_MAQUINA. Acesso em: 03 Out. 2023.

ZHANG Y.; SONG C.; ZHANG D., Deep Learning-Based Object Detection Improvement for Tomato Disease. **IEEE Access**, v. 8, p. 56607-56614, 2020, doi: 10.1109/ACCESS.2020.2982456

ZHOU, L et al; Application of Deep Learning in Food: A Review. **Comprehensive Reviews in Food Science and Food Safety,** v 0, p. 1 – 19, 2019. Doi: : 10.1111/1541-4337.12492

ZHU, L et al; Deep Learning and Machine Vision for Food Processing: A Survey. **Current Research in Food Science**, v 4, p. 233–249, 2021. Doi: https://doi.org/10.1016/j.crfs.2021.03.009