QUIMIOINFORMÁTICA

RAFAEL VIEIRA

APRENDIZADO DE MÁQUINA NO RECONHECIMENTO DE PADRÕES MOLECULARES EM PRODUTOS NATURAIS

AYA EDITORA
2026

QUIMIOINFORMÁTICA

APRENDIZADO DE MÁQUINA NO RECONHECIMENTO DE PADRÕES MOLECULARES EM PRODUTOS NATURAIS

RAFAEL VIEIRA

QUIMIOINFORMÁTICA

APRENDIZADO DE MÁQUINA NO RECONHECIMENTO DE PADRÕES MOLECULARES EM PRODUTOS NATURAIS

AYA EDITORA
2026

Direção Editorial
Prof.° Dr. Adriano Mesquita Soares
Executiva de Negócios
Ana Lucia Ribeiro Soares
Autor
Prof.° Dr. Rafael Vieira
Revisão
O Autor
Produção Editorial
AYA Editora©
Capa
AYA Editora©
Imagens de Capa
Acervo do Autor
Área do Conhecimento
Ciências Exatas e da Terra

V658 Vieira, Rafael

Quimioinformática: aprendizado de máquina no reconhecimento de padrões moleculares em produtos naturais [recurso eletrônico]. / Rafael Vieira. -- Ponta Grossa: Aya, 2026. 157 p.

Inclui biografia
Inclui índice
Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
ISBN: 978-65-5379-984-4
DOI: 10.47573/aya.5379.1.472

1. Engenharia química. 2. Simulação (computadores). 3. Métodos de simulação. 4. Ambientes virtuais compartilhados. 5. Produtos naturais. I. Título.

CDD: 660

Ficha catalográfica elaborada pela bibliotecária Bruna Cristina Bonini - CRB 9/1347

International Scientific Journals Publicações de Periódicos e Editora LTDA
AYA Editora©
CNPJ: 36.140.631/0001-53
Fone: +55 42 3086-3131
WhatsApp: +55 42 99906-0630
E-mail: contato@ayaeditora.com.br
Site: https://ayaeditora.com.br
Endereço: Rua João Rabello Coutinho, 557
Ponta Grossa - Paraná - Brasil
84.071-150

O Fluxo da Quimioinformática: Da Molécula à Inteligência Artificial

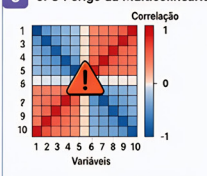
Como dados moleculares são transformados em modelos preditivos para acelerar descobertas e decisões.

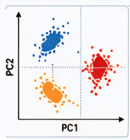
MÉTRICAS DE SUCESSO

- Acurácia
- Sensibilidade (Recall)
- Especificidade
- F1-Score

A eficácia é medida por Acurácia, Sensibilidade (Recall), Especificidade e F1-Score (média harmônica entre precisão e recall).

0. O Perigo da Multicolinearidade





Descritores altamente correlacionados (correlação próxima de 1 ou -1) devem ser removidos para evitar redundância e facilitar a interpretação dos resultados.

Redução de Dimensionalidade (PCA)

Técnicas como PCA resumem e projetam dados complexos em espaços menores, facilitando a visualização de padrões de agrupamento.

1. O Motor: Descritores Moleculares

- logP
- TPSA
- Fingerprint (RDKit)
- Etc.

Extração de Features via RDKit

Utiliza bibliotecas como RDKit para calcular descritores moleculares 1D, 2D e 3D (ex.: massa molecular) a partir da estrutura da molécula.

Os descritores incluem propriedades físico-químicas (ex.: logP, TPSA), estruturais (ex.: peso molecular) e de complexidade (ex.: ligações rotacionáveis).

2. O Refinamento: Análise Exploratória e Seleção

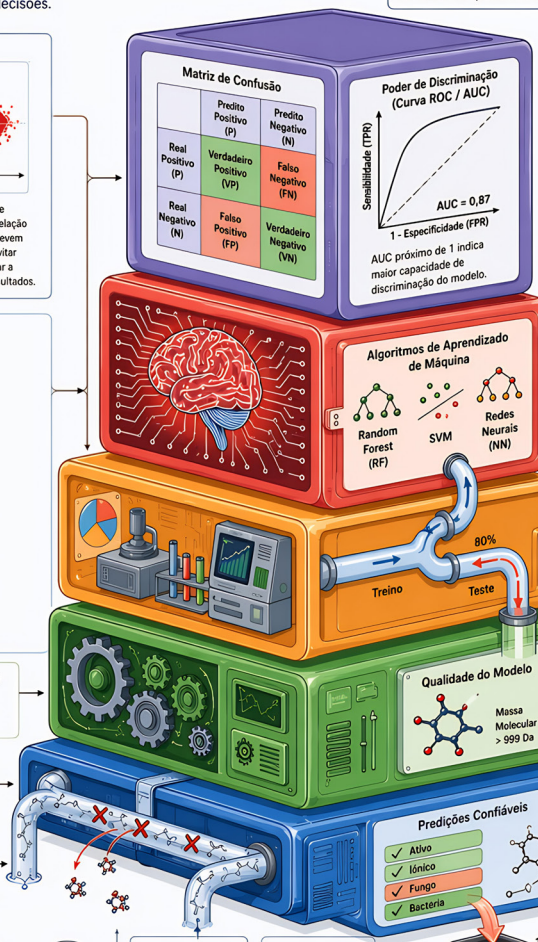
Análise de dados, seleção de variáveis relevantes e balanceamento para melhorar o desempenho dos modelos.

3. O Cérebro: Aprendizado de Máquina

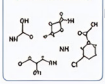
Algoritmos aprendem padrões nos dados e constroem modelos preditivos robustos.

4. O Resultado: Validação e Performance

Avaliação rigorosa com métricas e validação cruzada para garantir que o modelo generaliza bem para novos dados.



Fontes de Dados (Bancos de Dados)



CC(C)NCC1=CC=CC=C1
COC(=O)C1=CC=CC=C1
CNC(=O)CC=CC=O)O
CC(C)C1=CC=CC=C1=CN
C1=CC=CC(=C1)C(=O)N
C1=CC=CC=C11Cl

NPAtlas (Descritores baseados em propriedades)

PubChem (Descritores públicos)

Limpeza de Dados (KSAR)

Remoção de sais, correção de tautomeria, padronização de cargas e normalização de nitrogênios asseguram a qualidade dos dados.

Remoção de Ruídos e Duplicatas

1. A Fundação: Representação e Dados

Representação molecular padronizada e dados de alta qualidade são a base de qualquer modelo confiável.

SMILES: O DNA Digital

O sistema SMILES (Simplified Molecular-input Line-Entry System) converte estruturas químicas em texto, permitindo armazenamento, compartilhamento e uso computacional eficiente.

Limpeza e Padronização (KSAR)

Etapas críticas como remoção de sais, correção de tautomeria, padronização de cargas e normalização de nitrogênios asseguram a qualidade dos dados.

Filtragem de Outliers

Moléculas com comportamento atípico (ex.: massa molecular > 999 Da) podem ser tratadas separadamente para evitar viés no modelo.

Aplicação Supervisionada

O modelo treinado prevê a atividade de novas moléculas e auxilia na tomada de decisão, priorizando os compostos mais promissores.

Da molécula ao modelo, a quimioinformática transforma dados em conhecimento.

Um fluxo integrado de dados, métodos estatísticos e aprendizado de máquina impulsiona a descoberta de novos fármacos e materiais.

SUMÁRIO

APRESENTAÇÃO

A crescente integração entre química computacional, ciência de dados e aprendizado de máquina tem ampliado as possibilidades de investigação de moléculas bioativas e análise de grandes conjuntos de dados químicos. Inserida nesse contexto, esta obra apresenta uma proposta introdutória e aplicada voltada ao reconhecimento de padrões moleculares em produtos naturais, aproximando conceitos computacionais das demandas atuais da pesquisa química e farmacêutica.

Ao longo do livro, o leitor acompanha desde a preparação do ambiente computacional em Python até a construção de modelos preditivos aplicados à classificação molecular. A discussão envolve análise exploratória de dados, cálculo de descritores químicos, interpretação de correlações e utilização de bibliotecas amplamente empregadas na quimioinformática. Paralelamente, são apresentados fundamentos de aprendizado supervisionado e não supervisionado, incluindo métodos estatísticos e algoritmos aplicados à separação e identificação de classes moleculares.

A obra também dedica atenção à avaliação de desempenho dos modelos, explorando métricas, matrizes de confusão e estratégias de validação aplicadas a problemas binários e multiclasse. Estudos envolvendo moléculas relacionadas à dengue, zika e chikungunya demonstram como diferentes abordagens computacionais podem ser empregadas na análise de compostos de interesse biológico, aproximando teoria, programação e aplicação científica.

Ao integrar fundamentos conceituais e práticas computacionais em uma abordagem acessível e organizada, este livro contribui para a formação de estudantes, pesquisadores e profissionais interessados em utilizar ferramentas de aprendizado de máquina na análise de dados químicos e moleculares. Trata-se de uma leitura que amplia as possibilidades de investigação em quimioinformática e pesquisa em produtos naturais.

Boa leitura!

INTRODUÇÃO AO APRENDIZADO DE MÁQUINA E QUÍMICA COMPUTACIONAL

O aprendizado de máquina (AM) é uma disciplina da ciência da computação que se concentra no desenvolvimento de algoritmos e técnicas que permitem aos computadores aprenderem padrões a partir de dados e tomar decisões com base nesses padrões, sem a necessidade de programação explícita. Essa capacidade de aprendizado automático tem uma ampla gama de aplicações em diversas áreas, incluindo medicina, finanças, ciência dos materiais e química computacional.

Na interseção entre o aprendizado de máquina e a química, surge a química computacional, um campo de estudo que utiliza métodos computacionais para entender e prever propriedades químicas e comportamentos de moléculas em sistemas químicos. O uso do aprendizado de máquina na química computacional tem se mostrado extremamente poderoso, permitindo a análise de grandes conjuntos de dados moleculares, a modelagem de propriedades químicas complexas e a descoberta de novos compostos com potencial terapêutico, entre outras aplicações.

Um dos aspectos mais fascinantes da química computacional é o estudo de moléculas orgânicas de produtos naturais. Estas são moléculas derivadas de fontes naturais, como plantas, microrganismos e organismos marinhos, e são conhecidas por sua diversidade estrutural e atividade biológica. As moléculas orgânicas de produtos naturais têm sido fontes valiosas de fármacos e compostos bioativos, e a compreensão de suas estruturas e interações com proteínas-alvo é fundamental para a descoberta de novos medicamentos e o desenvolvimento de terapias mais eficazes.

A aplicação do aprendizado de máquina na análise de moléculas orgânicas de produtos naturais oferece uma abordagem inovadora e eficiente para identificar padrões, prever propriedades e realizar estudos de docagem molecular. Isso possibilita a rápida triagem de compostos, a identificação de candidatos a fármacos promissores e a otimização de moléculas para aumentar sua eficácia e seletividade.

Ao combinar as ferramentas do aprendizado de máquina com os princípios da química computacional, os pesquisadores podem explorar de forma mais abrangente o vasto espaço químico, acelerando o processo de desco-

berta e desenvolvimento de novos compostos, bem como a compreensão dos mecanismos subjacentes a sua atividade biológica.

Neste contexto, este livro explora as técnicas, algoritmos e aplicações do aprendizado de máquina na química computacional, com foco especial no uso de moléculas orgânicas de produtos naturais para reconhecimento de padrões e docagem molecular. Ao longo dos capítulos seguintes, vamos mergulhar mais fundo nesse fascinante campo de estudo, fornecendo uma visão abrangente das ferramentas e métodos disponíveis, bem como exemplos práticos de sua aplicação em pesquisa química e farmacêutica.

Produtos naturais e sua importância na pesquisa química.

Produtos naturais desempenham um papel fundamental na pesquisa química devido à sua diversidade estrutural e potencial terapêutico. Esses compostos, derivados de plantas, microrganismos e organismos marinhos, oferecem uma riqueza de moléculas bioativas que podem ser exploradas em várias áreas, desde a descoberta de medicamentos até a síntese de materiais avançados.

A importância dos produtos naturais na pesquisa química reside em várias esferas:

- **Diversidade Estrutural:** Produtos naturais abrangem uma ampla gama de estruturas químicas, desde simples compostos até moléculas altamente complexas. Essa diversidade estrutural oferece um vasto campo para a exploração de propriedades e atividades biológicas.
- **Potencial Terapêutico:** Muitos medicamentos modernos são derivados de produtos naturais ou inspirados por eles. A pesquisa em produtos naturais oferece oportunidades para identificar compostos com atividades farmacológicas interessantes, que podem ser desenvolvidos como tratamentos para uma variedade de doenças.
- **Fonte de Inspiração:** Além de serem fontes diretas de medicamentos, os produtos naturais também inspiram a síntese de compostos sintéticos. Muitas estratégias de síntese orgânica são influenciadas pela complexidade e eficácia das moléculas naturais. No contexto brasileiro, a rica biodiversidade torna-se um aliado valioso no estudo e na exploração de produtos naturais. A vasta va-

> riedade de espécies vegetais, microbianas e marítimas oferece um tesouro de moléculas com potencial terapêutico e industrial, que podem servir como fonte de inspiração para a síntese de novos compostos bioativos. A biodiversidade brasileira, com sua singularidade e complexidade, continua a ser uma fonte quase inesgotável de descoberta e inovação na pesquisa química e farmacêutica.

Assim, para todos nós que realizamos pesquisa em produtos naturais, temos que ter em mente que desempenhamos um papel crucial na investigação química, oferecendo oportunidades para descobrir novos medicamentos, entender processos biológicos e discutir proposituras de modificações estruturais mais eficazes e sustentáveis. Essa área continua a ser uma fonte valiosa de inovação e descoberta na ciência química.

Então, meu caro leitor, adentre neste vasto e fascinante universo dos produtos naturais, onde a ciência e a natureza convergem em uma relação intrigante de moléculas e descobertas. Imagine-se explorando suas matrizes complexas, desvendando dados em cada extrato de planta, em cada microrganismo, em cada material que estiver sendo estudado. A partir de agora, você começa a se equipar não apenas com o conhecimento tradicional, mas também com as mais avançadas ferramentas computacionais, capazes de revelar padrões extremamente interessantes.

É nesse cenário de possibilidades que convido você a embarcar em uma jornada única e empolgante. Em "*Aprendizado de máquina no reconhecimento de padrões moleculares de produtos naturais: Uma abordagem prática para iniciantes em quimionformática*", você será guiado por um caminho de descoberta e entendimento, mergulhando fundo na complexidade e na beleza das moléculas naturais.

Este livro não é apenas um convite, é um desafio, uma nova proposta. Um desafio para os curiosos, os inquietos, os que buscam compreender o mundo químico. Aqui, exploraremos a química dos produtos naturais através das abordagens computacionais mais avançadas e buscaremos insights significativos por meio das ciências de dados.

Aqui você será o explorador, o pesquisador, o descobridor de novos horizontes. Seja bem-vindo a este universo de possibilidades infinitas. Se você deseja desbravar espaços químicos, compreender a constelação de descritores moleculares e desvendar padrões moleculares em seus extratos, este livro é para você. O que está esperando? Abra seu computador e prepare seu ambiente de desenvolvimento! Vou te mostrar como.

PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO EM PYTHON USANDO ANACONDA

Python é uma linguagem de programação versátil e popular, adequada para uma variedade de aplicações, desde desenvolvimento web até análise de dados e inteligência artificial. Antes de começar a escrever código em Python, é importante configurar um ambiente de desenvolvimento adequado. Uma das maneiras mais convenientes de fazer isso é utilizando o Anaconda, uma distribuição de Python que inclui várias bibliotecas e ferramentas úteis para desenvolvedores.

O que é o Anaconda?

O Anaconda é uma distribuição de código aberto e gratuita de Python e R projetada para simplificar o gerenciamento de pacotes e ambientes em ciência de dados e computação científica. Ele inclui uma ampla gama de pacotes, bibliotecas e ferramentas populares pré-instalados, facilitando o trabalho com análise de dados, aprendizado de máquina, visualização e muito mais.

Tabela 1 - Recursos do Anaconda para Desenvolvimento e Ciência de Dados.

Recurso	Descrição
Gerenciamento de Pacotes Simplificado	O Anaconda facilita a instalação, atualização e gerenciamento de pacotes e dependências, eliminando muitas das dores de cabeça associadas à configuração de ambientes de desenvolvimento.
Ambientes Virtuais Isolados	Com o Anaconda, você pode criar ambientes virtuais isolados, permitindo que você trabalhe em projetos diferentes com diferentes versões de pacotes e dependências sem conflitos.
Ampla Gama de Pacotes Pré-instalados	O Anaconda vem com uma grande variedade de pacotes populares pré-instalados, incluindo numpy, pandas, scikit-learn, TensorFlow, Keras, entre outros, o que o torna uma escolha conveniente para cientistas de dados e desenvolvedores de aprendizado de máquina.

Recurso	Descrição
Plataforma Multiplataforma	O Anaconda é compatível com Windows, macOS e Linux, garantindo uma experiência consistente em diferentes sistemas operacionais.

Irei fornecer este guia passo a passo para configurar seu ambiente de desenvolvimento Python com Anaconda:

Baixando e instalando o Anaconda

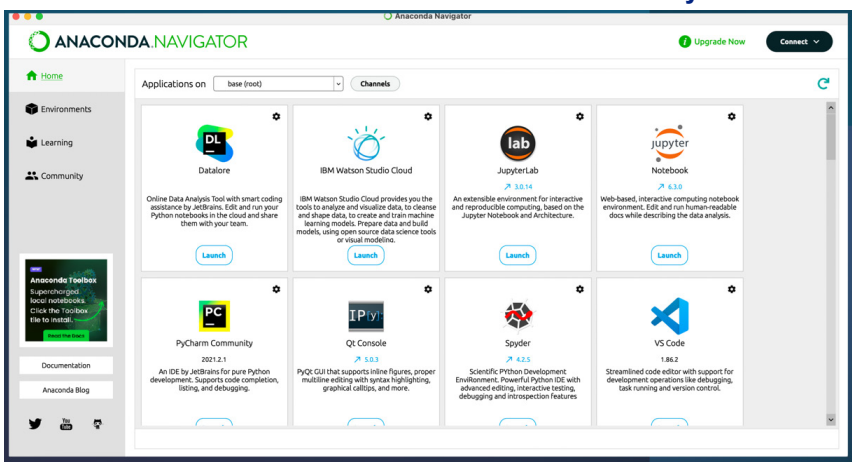
Acesse o site oficial do Anaconda em https://www.anaconda.com/products/distribution e faça o download da versão adequada para o seu sistema operacional (Windows, macOS ou Linux). Siga as instruções de instalação específicas para o seu sistema para concluir o processo de instalação.

Configurando o Anaconda

Após a instalação, você pode configurar o Anaconda através da linha de comando do terminal (a qual iremos usá-la bastante) ou usando a interface gráfica. Existe a possibilidade de iniciar o Anaconda Navigator, uma interface gráfica que permite gerenciar seus ambientes e pacotes Python de forma mais intuitiva.

Assim que a instalação estiver concluída, você poderá abrir a Anaconda Navigator. A Figura 1 mostra a interface bastante amigável e atrativa do navegador.

Figura 1 – Interface do Anaconda Navigator, uma das maneiras de se criar um ambiente de desenvolvimento Python.



Agora para criar um ambiente de desenvolvimento Python, basta seguir o passo a passo da Tabela 2.

Tabela 2 - Passo a passo para instalação do ambiente de desenvolvimento Python.

Passo	Descrição
1	Abra o Anaconda Navigator no seu sistema operacional.
2	Navegue até o Gerenciador de Ambientes (Environment tab) – **Figura 2-A.**
3	Clique no botão “Create” para iniciar o processo de criação de um novo ambiente – **Figura 2-B.**
4	Configure o novo ambiente: forneça um nome, selecione a versão do Python e escolha os pacotes adicionais a serem instalados – **Figura 2-C**
5	Clique em “Create” para iniciar o processo de criação do ambiente – **Figura 2-D.**
6	Aguarde até que o Anaconda Navigator conclua a criação do ambiente.
7	Verifique se o ambiente foi criado com sucesso.

Fantástico! Agora que você está pronto para explorar o mundo da química digital! Com seu ambiente configurado e pronto para uso, você tem uma plataforma sólida para mergulhar em uma variedade de tópicos químicos, desde análises de dados até treinamento de modelos de aprendizado de máquina!

Criando um Ambiente Virtual pelo terminal

Um ambiente virtual é uma instância isolada do Python que permite instalar pacotes e suas dependências sem interferir no Python global do sistema. No Anaconda Navigator ou através da linha de comando, você pode criar um novo ambiente virtual com o seguinte comando (Código 1):

Código 1 - Criação do ambiente virtual

```
#CÓDIGO 1
conda create –name meuambiente python=3.8
```

Substitua “meuambiente” pelo nome que deseja dar ao seu ambiente virtual. O Python 3.8 é especificado aqui como a versão do Python, mas você pode escolher outra versão, se preferir.

Ativando o Ambiente Virtual

Depois de criar o ambiente virtual, você precisa ativá-lo antes de começar a trabalhar nele. Você pode ativar o ambiente virtual usando o Código 2:

Código 2 - Ativação do ambiente de trabalho Python.

```
#CÓDIGO 2
conda activate livro
```

Você notará que o nome do ambiente virtual aparece agora no prompt de comando, indicando que está ativo.

Desativando o Ambiente Virtual

Quando terminar de trabalhar no seu projeto, você pode desativar o ambiente virtual usando o Código 3:

Código 3 - Desativação e fechamento do ambiente conda.

```
#CÓDIGO 3
conda deactivate
```

Isso retornará ao ambiente Python global do sistema. Com o Anaconda configurado e um ambiente virtual criado, você está pronto para começar a desenvolver seus trabalhos iniciais em Python de forma eficiente e muito organizada. Experimente explorar outras funcionalidades do Anaconda Navigator e do Conda para aprimorar sua experiência de desenvolvimento!

INICIANDO PROJETOS

Você já deu os primeiros passos instalando o Anaconda e criando seu ambiente de desenvolvimento. Acompanhe passo a passo enquanto exploramos informações químicas vindas de bases de dados. Vamos lá!

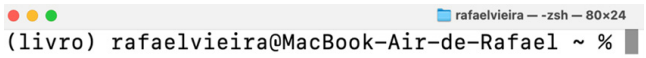
Pronto para elevar suas habilidades em Python? Primeiro passo: ative seu ambiente conda. No meu caso, criei um ambiente chamado 'livro' e, para ativá-lo, abri o terminal Anaconda e digitei o código 4:

Código 4 - Ativação do ambiente Conda chamado Livro. Para testagem e estudo dos códigos dessa obra.

```
#CÓDIGO 4
conda activate livro
```

Agora, estamos prontos para iniciar a instalação de pacotes e bibliotecas essenciais. Recomendo fortemente a instalação do Jupyter! Ele permite criar um 'caderno' onde você pode codificar e registrar anotações vitais para o seu estudo. Certifique-se de estar no ambiente Conda. No meu caso, o terminal exibirá o nome do ambiente:

Figura 2 - Terminal indicando o nome do ambiente de desenvolvimento Python.



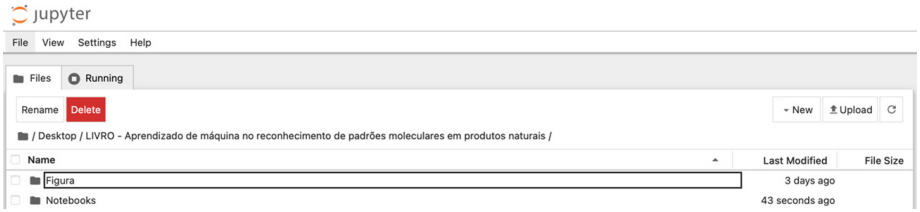
A instalação do Jupyter é rápida e fácil. Basta digitar o Código 5 no terminal:

Código 5 - Instalação do jupyter pelo terminal.

```
#CÓDIGO5
pip install jupyter
```

Após a conclusão da instalação, inicie o Jupyter Notebook digitando simplesmente no terminal: jupyter notebook .

Figura 3 - Representação de abertura do jupyter notebook.



No canto superior direito, clique em 'NEW' e selecione Python3. Um arquivo no formato .ipynb será criado. Aqui é onde você terá total liberdade para criar e explorar seus dados.

Notebook 1 – Análise Exploratória de Dados

Explorando a base de dados molecular Atlas

Com o avanço e a disponibilidade crescente de dados químicos, surgiram bases de dados públicas que oferecem insights valiosos sobre uma ampla gama de moléculas. Essas plataformas visam identificar padrões moleculares significativos que possam ser úteis na classificação e compreensão dessas substâncias. Neste contexto, uma plataforma que merece destaque é o NPAtlas.

A NPAtlas torna-se fundamental nos trabalhos de aprendizado de máquina usando produtos naturais que buscam explorar o vasto universo molecular. Sua interface intuitiva e acessível oferece uma variedade de recursos para análise e pesquisa de moléculas, permitindo o acesso a informações detalhadas sobre estruturas, propriedades e interações.

Para acessar o Atlas e explorar suas funcionalidades, basta visitar a página oficial da plataforma e clicar na aba de downloads, disponível em: https://www.npatlas.org . Lá, os usuários encontrarão uma riqueza de dados e ferramentas que os auxiliarão na busca por insights e descobertas no vasto mundo das moléculas.

Agora abra seu "jupyter notebook" para iniciar a exploração dos dados moleculares.

Biblioteca Pandas

No vasto ecossistema da linguagem Python, há uma biblioteca que se destaca como uma ferramenta indispensável para a manipulação e análise de dados: o Pandas. Com suas poderosas funcionalidades e sua facilidade de uso, o Pandas se estabeleceu como uma pedra angular para cientistas de dados, analistas e programadores em todo o mundo.

Imagine que você está diante de uma vasta biblioteca, repleta de ferramentas que simplificam a manipulação de conjuntos de dados complexos. Essa é a promessa do Pandas. Com apenas uma linha de código, você pode abrir as portas para um mundo de possibilidades analíticas e exploratórias.

Para trazer o Pandas para o seu ambiente de desenvolvimento, tudo o que você precisa fazer é digitar algumas linhas de comando. No seu notebook Python, basta executar o Código 6:

Código 6 - Instalação da biblioteca Pandas.

```
CÓDIGO 6
!pip install pandas
```

Em instantes, você terá acesso a todas as funcionalidades do Pandas, prontas para serem exploradas e utilizadas em seus projetos de análise de dados. Mas o que torna o Pandas tão especial? Sua simplicidade e poder. Com o Pandas, você pode carregar, limpar, transformar e analisar dados de maneira eficiente e intuitiva. Ele oferece estruturas de dados flexíveis, como o DataFrame, que permitem manipular dados tabulares com facilidade e elegância.

Além disso, o Pandas é altamente extensível. Com uma vasta gama de métodos e funções, você pode realizar operações complexas em seus dados com apenas algumas linhas de código. Seja filtrando dados, calculando estatísticas descritivas ou criando visualizações impressionantes, o Pandas oferece as ferramentas de que você precisa para transformar seus dados em insights valiosos.

Portanto, ao embarcar na jornada da análise de dados em Python, lembre-se do poder do Pandas. Ele será sua biblioteca confiável em todas as etapas do processo, ajudando você a analisar seus conjuntos de dados e a contar histórias impactantes através dos números.

Abrindo os dados do NPAtlas

Ao longo do desenvolvimento do código, você encontrará a necessidade de utilizar outras bibliotecas além do Pandas. Uma dessas bibliotecas essenciais é a openpyxl, amplamente reconhecida por sua capacidade de manipular arquivos Excel de forma eficiente e flexível.

Para instalar a openpyxl e garantir que você tenha acesso às suas funcionalidades poderosas, basta seguir o mesmo procedimento simples utilizado para instalar o Pandas. No seu ambiente Python, como em um notebook Jupyter, você pode executar o Código 7, que irá fazer a instalação rapidamente:

Código 7 - Instalação da biblioteca openpyxl, para manipulação de dataframes oriundos do Excel.

```
#CÓDIGO 7
!pip install openpyxl
```

Com essa única linha de código, você estará prontamente equipado com todas as capacidades da biblioteca openpyxl. A partir daí, você poderá explorar e manipular arquivos Excel com facilidade, realizando operações como leitura, escrita e edição de planilhas com apenas algumas linhas de código.

Você pode utilizar o site de apoio desse livro: https://vieira86.github.io/livroquimioinformatica/ e realizar o download do arquivo Atlas que está no formato .xlsx chamado “NPAtlas_download_2023_06.xlsx”, ele será aberto usando a biblioteca pandas através do Código 8 disponibilizado a seguir:

Código 8 - Código para leitura da base de dados utilizando o pandas.

```
#CÓDIGO 8
df = pd.read_excel('NPAtlas_download_2023_06.xlsx')
df.head()
```

Observe que estamos abrindo um conjunto de dados que contém 33.372 moléculas distribuídas em 32 colunas distintas. Cada uma dessas colunas representa uma informação crucial previamente depositada na base de dados NPAtlas. No entanto, para começarmos nossa análise, optaremos por selecionar apenas aquelas que são de nosso interesse primordial. (Caso

haja outras informações que você considere relevantes explorar, sinta-se à vontade para adicioná-las).

Dessa maneira, serão escolhidas as seguintes colunas para compor nosso conjunto de características, também conhecidas como “features”. Para isso, basta executar o Código 9:

Código 9 - Seleção apenas de colunas desejadas para exploração dos dados.

```
#CÓDIGO 9
df=df[['npaid','compound_names','compound_molecular_
formula','compound_smiles','origin_type']]
```

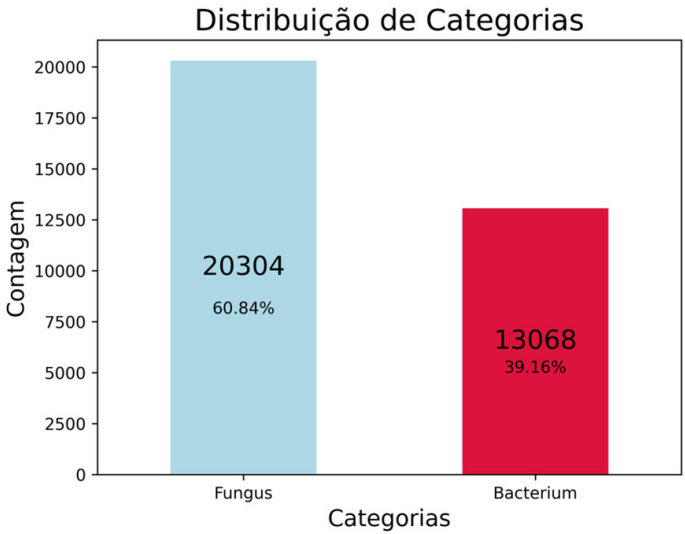
Atualmente, existem 5 colunas (ou features) que formarão a base para nossa análise exploratória inicial. Com a informação a respeito da origem da molécula, é possível visualizar a quantidade em cada uma dessas classes, utilizando o Código 10:

Código 10 - Visualização da análise exploratória de dados.

```
#CÓDIGO 10
import pandas as pd
import matplotlib.pyplot as plt
# Suponha que df seja o seu DataFrame
# Conte o número de ocorrências de cada categoria
contagem_categorias = df['origin_type'].value_counts()
# Defina as cores para cada categoria
cores = ['lightblue' if x == 'Fungus' else 'crimson' for x in contagem_
categorias.index]
# Crie um gráfico de barras
ax = contagem_categorias.plot(kind='bar', color=cores)
plt.title('Distribuição de Categorias')
plt.xlabel('Categorias')
plt.ylabel('Contagem')
# Adicione o valor em cada barra
for i, count in enumerate(contagem_categorias):
      ax.text(i, count / 2, str(count), ha='center', va='center',
fontsize=16)
# Rotacione os rótulos do eixo x em 90 graus
ax.set_xticklabels(contagem_categorias.index, rotation=0)
plt.show()
```

O código gerará o gráfico a seguir, permitindo visualizar a distribuição das moléculas entre as duas classes analisadas. No conjunto de dados, há 20.304 moléculas originadas de fungos, enquanto as restantes, totalizando 13.068, são de origem bacteriana. A distribuição das duas classes pode ser visualizada na Figura 4.

Figura 4 - Distribuição das moléculas entre fungos e bactérias.



No entanto, é crucial destacar uma feature de extrema importância: aquela que armazena a informação molecular conhecida como SMILES (simplified molecular-input line-entry system). A partir dessa informação, será possível calcular outros descritores químicos essenciais, os quais serão empregados para treinar os modelos de aprendizado de máquina.

O SMILES é uma ferramenta essencial para os químicos computacionais e modeladores moleculares, proporcionando uma maneira eficiente e padronizada de representar estruturas químicas complexas. Sua simplicidade e versatilidade o tornam uma peça fundamental no arsenal de ferramentas utilizadas na pesquisa e na indústria química moderna, e ele servirá de entrada para os algoritmos de cálculo de descritores químicos que irá ser apresentado a vocês agora.

Calculando descritores usando a biblioteca RDKIT

A linguagem Python, por ser bastante versátil, permite a criação de funções que podem ser acessadas como bibliotecas. Será necessário a instalação do pacote RDKIT, uma biblioteca Python amplamente utilizada para a manipulação de estruturas químicas e análise de dados químicos. Essa poderosa ferramenta oferece uma ampla gama de funcionalidades para cien-

tistas e pesquisadores que trabalham com química computacional e modelagem molecular. O RDKit permite a leitura, escrita e manipulação de estruturas químicas, além de fornecer ferramentas para cálculos de descritores moleculares que permitem o treinamento dos algoritmos de aprendizado de máquina aplicados a problemas de química medicinal. Neste exemplo, será importada uma função chamada "*calcular_descritores*". Faça download no site de suporte do livro[1].

No conjunto de dados da NPAtlas, uma coluna essencial denominada "compound_smiles" destaca-se entre as features disponíveis. Para simplificar a análise, faremos uma pequena modificação, renomeando-a para "Smiles". Execute o seguinte Código 11 para implementar essa alteração:

Código 11 - Renomear colunas. Importante para explicitar e corrigir a coluna SMILES.

```
#CÓDIGO 11
df.rename(columns = {"compound_smiles": 'Smiles'}, inplace=True)
```

Agora, com as informações devidamente organizadas, estamos prontos para calcular os descritores químicos de cada molécula presente no conjunto de dados NPAtlas. Esses descritores fornecerão insights valiosos sobre as características moleculares de cada composto. Para realizar esse processo, basta utilizar o Código 12:

Código 12 - Cálculo dos descritores moleculares. Chamada de código por meio de importação.

```
#CÓDIGO 12
from calcular_descritores import calc_descriptors
df_completo = calc_descriptors(df)
df_completo.head()
```

Após a execução dessas etapas, você poderá explorar as primeiras instâncias do novo DataFrame completo utilizando o método head(). Esse conjunto de dados ampliado oferecerá uma visão mais detalhada e abrangente das propriedades químicas das moléculas analisadas, enriquecendo assim sua compreensão do universo molecular presente na NPAtlas.

1 https://vieira86.github.io/livroquimioinformatica/

Explorando os Dados Químicos

Para compreender a natureza e o comportamento dos dados químicos, realizamos uma análise exploratória detalhada. Um simples comando (Código 13) nos proporciona uma visão geral do conjunto de dados:

Código 13 - Análise estatística rápida do dataset.

```
#CÓDIGO 13
df_completo.describe()
```

A tabela resultante, semelhante à Tabela 3, fornece uma análise estatística abrangente dos descritores químicos das moléculas da NPAtlas. Essas estatísticas essenciais são cruciais para compreender a distribuição e a variabilidade dos dados, proporcionando insights valiosos sobre as propriedades moleculares presentes no conjunto de dados.

Essa abordagem oferece uma visão mais ampla e convidativa para os leitores, ressaltando a importância da análise exploratória e os insights que podem ser obtidos através dela.

Tabela 3 - Exemplo de tabela obtida pelo Python, para visualização estatística dos descritores químicos usados.

	Fraction CSP^3	LogP	MolWt	Num Aromatic Rings	Num HAccep	Num HDonors	Num Rotatable Bonds	TPSA
count	33372	33372	33372	33372	33372	33372	33372	33372
mean	0,547	2,521	491,044	1,096	7,106	3,739	7,264	135,711
std	0,228	2,856	304,617	1,271	5,148	4,025	8,560	116,270
min	0,000	28,945	68,119	0,000	0,000	0,000	0,000	0,000
25%	0,382	1,191	294,259	0,000	4,000	2,000	2,000	69,670
50%	0,563	2,513	410,507	1,000	6,000	3,000	5,000	99,860
75%	0,731	4,025	567,548	2,000	9,000	4,000	9,000	153,750
max	1,000	37,082	5033,832	11,000	71,000	69,000	143,000	2035,240

Agora é possível analisar cada uma das variáveis na Tabela:

1. FractionCSP³ (Fração CSP³):

- A média de FractionCSP³ é de aproximadamente 0,547, com um desvio padrão de 0,228.

- A fração CSP^3 varia de 0 a 1, com valores mínimos e máximos de 0 e 1, respectivamente.

- A distribuição parece ser razoavelmente simétrica, já que a média e a mediana estão próximas.

2. MolLogP:

- A média de MolLogP é de aproximadamente 2,521, com um desvio padrão de 2,856.

- A distribuição de MolLogP é mais ampla, com valores mínimos e máximos de -28,945 e 37,082, respectivamente.

- A presença de valores extremos pode indicar a presença de moléculas altamente lipofílicas ou hidrofílicas.

3. MolWt (Massa Molecular):

- A massa molecular média das moléculas é de aproximadamente 491,044 g/mol, com um desvio padrão de 304,617 g/mol.

- A massa molecular varia de 68,119 g/mol a 5033,832 g/mol.

- A distribuição parece ser assimétrica à direita, com a média maior que a mediana.

4. NumAromaticRings (Número de Anéis Aromáticos):

- A média de NumAromaticRings é de aproximadamente 1,096, com um desvio padrão de 1,271.

- A quantidade de anéis aromáticos varia de 0 a 11.

- A maioria das moléculas parece ter um ou nenhum anel aromático.

5. NumHAcceptors (Número de Acceptores de Hidrogênio):

- A média de NumHAcceptors é de aproximadamente 7,106, com um desvio padrão de 5,148.

- O número de átomos de hidrogênio acceptores varia de 0 a 71.

- Há uma grande variabilidade no número de acceptores de hidrogênio nas moléculas.

6. NumHDonors (Número de Doadores de Hidrogênio):

- A média de NumHDonors é de aproximadamente 3,739, com um desvio padrão de 4,025.

- O número de doadores de hidrogênio varia de 0 a 69.

- Assim como os acceptores de hidrogênio, há uma grande variabilidade nos doadores de hidrogênio.

7. NumRotatableBonds (Número de Ligações Rotacionais):

- A média de NumRotatableBonds é de aproximadamente 7,264, com um desvio padrão de 8,560.

- O número de ligações rotacionais varia de 0 a 143.

- Existem moléculas com um grande número de ligações rotacionais, indicando sua flexibilidade estrutural.

8. TPSA (Área de Superfície Polar Total):

- A média de TPSA é de aproximadamente 135,711 Å, com um desvio padrão de 116,270 Å.

- A área de superfície polar total varia de 0 a 2035,240 Å.

- Há uma ampla variação na polaridade das moléculas, refletida na distribuição da área de superfície polar total.

Essa análise fornece uma compreensão inicial das características e distribuições das variáveis no conjunto de dados, ajudando a identificar padrões e peculiaridades que podem influenciar análises posteriores. Além disso, é possível visualizar as distribuições dos dados de cada descritor. Para isso, utilize o Código 14, que irá efetuar automaticamente a análise exploratória dos dados:

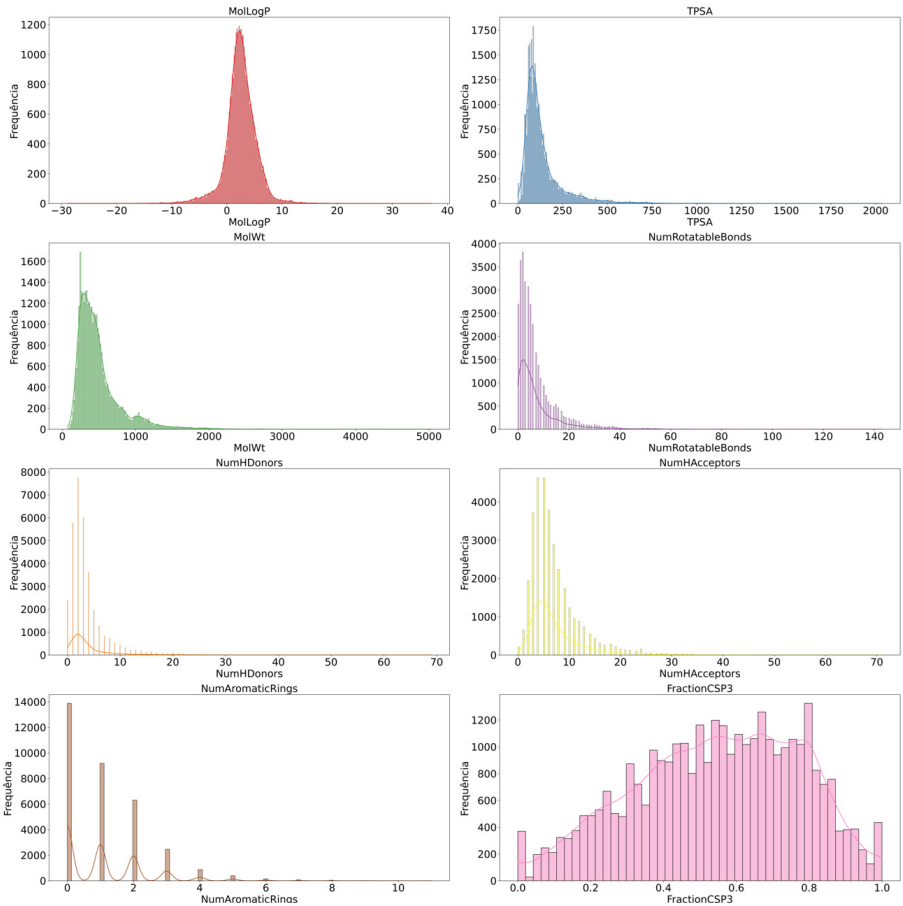
Código 14 - Análise exploratória dos principais descritores moleculares.

```
#CÓDIGO 14
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.cm as cm

# Defina os dados e os atributos
atributos = ['MolLogP', 'TPSA', 'MolWt', 'NumRotatableBonds', 'NumHDonors', 'NumHAcceptors', 'NumAromaticRings', 'FractionCSP3']
cmap = cm.get_cmap('Set1').colors  # mapa de cores

# Crie a figura e os subplots
fig, axes = plt.subplots(4, 2, figsize=(30, 30))

# Preencha os subplots com os gráficos
for i, atributo in enumerate(atributos):
    linha = i // 2  # Determina a linha onde o subplot deve estar
    coluna = i % 2   # Determina a coluna onde o subplot deve estar

        sns.histplot(data=df_completo, x=atributo, color=cmap[i], ax=axes[linha, coluna], kde=True)
```

```
    axes[linha, coluna].set_xlabel(atributo, fontsize=24)  # Ajuste
do tamanho da fonte no eixo x
     axes[linha, coluna].set_ylabel('Frequência', fontsize=24)   #
Ajuste do tamanho da fonte no eixo y
   axes[linha, coluna].set_title(atributo, fontsize=24)  # Ajuste do
tamanho da fonte no título
     axes[linha, coluna].tick_params(axis='both', labelsize=24)   #
Ajuste do tamanho da fonte nos números dos eixos

# Ajuste o layout para evitar sobreposição
plt.tight_layout()

# Salve a figura em 600 dpi
fig.savefig('histograma.png', dpi=600)
```

O código acima permitirá agrupar os histogramas de cada um dos descritores químicos, e o histograma apresenta a distribuição de frequência dos valores do respectivo atributo, permitindo a visualização das tendências e variações nos dados químicos (figura 5). O uso de diferentes cores e tamanhos de fonte ajuda a diferenciar e destacar cada atributo, facilitando a comparação entre eles.

Figura 5 - Histogramas dos descritores químicos selecionados.



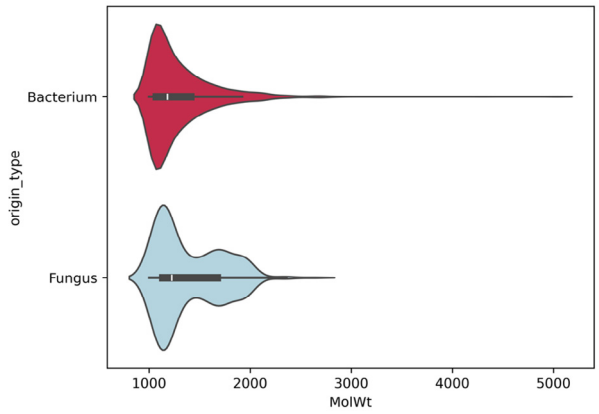
No estudo em questão, é evidente que a distribuição dos dados diverge consideravelmente entre as classes analisadas, principalmente em relação ao descritor da massa molecular (MolWt). Para a classe de bactérias, observa-se uma dispersão significativa dos dados, com a maioria das moléculas apresentando uma massa molecular superior a 1000 Dalton. Por outro lado, para a classe de fungos, identificam-se duas regiões de destaque: uma similar à das bactérias, mas com uma notável concentração de moléculas na faixa de 2000 Dalton, além de uma menor dispersão dos dados.

A visualização desses padrões é facilitada pelo uso do gráfico de violinplot (Figura 6), o qual oferece uma representação intuitiva e detalhada das distribuições de massa molecular em diferentes classes biológicas. O Código 15 é usado para gerar esse tipo de gráfico.

Código 15 - Criação de gráficos violinplot.

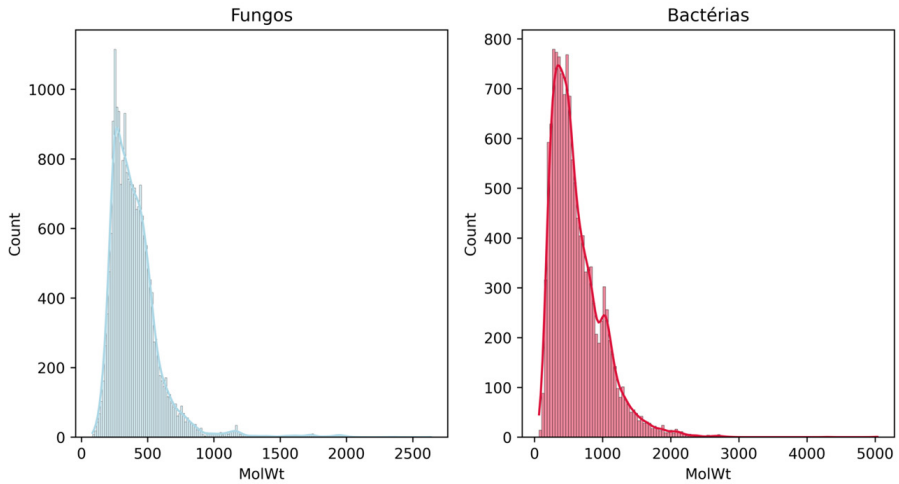
```
#CÓDIGO 15
import seaborn as sns
import matplotlib.pyplot as plt

# Definir a paleta de cores desejada
cores = ["crimson", "lightblue"]  # Por exemplo, uma lista de cores RGB

# Criar o gráfico de violinplot com a paleta de cores definida
sns.violinplot(data=moleculas_massa_mq1000,  x='MolWt',  y='origin_type', palette=cores)

# Adicionar título ao gráfico
# plt.title('Distribuição de Massa Molecular por Origem da molécula')
# Salvar a figura em 600 dpi
plt.savefig('violinplot_classes.png', dpi=600, bbox_inches='tight')
# Exibir o gráfico
plt.show()
```

Figura 6 - Distribuição de Massa Molecular por origem da molécula



É possível complementar a análise da distribuição de massa molecular junto à Figura 7, mostrando que a classe de “fungos” apresenta moléculas menores do que as produzidas por bactérias.

Figura 7 - Distribuição da massa molecular entre as classes de fungos e bactérias.

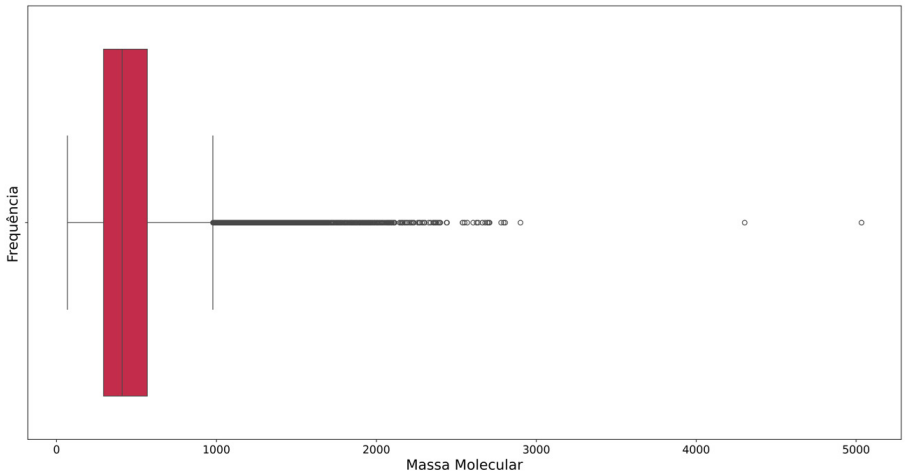


Esta representação gráfica, além de ser bastante interessante, proporciona uma compreensão mais profunda das características distintas entre as classes de bactérias e fungos em relação à distribuição da massa molecular das moléculas analisadas. (Você pode ampliar os estudos e fazer este gráfico para outros descritores químicos do seu dataset).

Visualização de moléculas com valores atípicos

Dentre o conjunto de dados de produtos naturais, existem moléculas com valores elevados de massa molecular (‘MolWt’), que, em estatística, podem ser consideradas outliers. O boxplot (Figura 8), também conhecido como diagrama de caixa, é uma ferramenta estatística poderosa para visualizar a distribuição e a dispersão dos dados, neste caso, a massa molecular das moléculas do conjunto de dados.

Figura 8 - Boxplot da massa molecular dos produtos naturais do NPAtlas.



Caso o boxplot seja algo novo para você, aqui está uma explicação sobre as partes de um boxplot e o que elas representam:

- **Caixa (Box):** O retângulo central representa o intervalo interquartil (IQR), que é a faixa que contém 50% dos dados. A linha no meio da caixa indica a mediana, que é o valor que divide os dados em duas metades iguais.
- **Whiskers (Hastes):** São linhas que se estendem a partir da caixa até os valores extremos dos dados, excluindo os outliers. Os outliers, caso existam, são pontos além dos limites determinados pelos whiskers. Eles são indicativos de valores incomuns ou anômalos na distribuição.
- **Pontos fora dos whiskers (Outliers):** São pontos individuais que são considerados extremos em relação à distribuição principal dos dados. São pontos que podem ser considerados incomuns ou atípicos.
- **Mínimo e Máximo:** Os pontos finais dos whiskers representam os valores mínimos e máximos dos dados, excluindo os outliers.

O boxplot permite identificar a dispersão dos valores da massa molecular das moléculas, bem como detectar a presença de outliers. Ele é útil para comparar a distribuição de diferentes grupos ou categorias de moléculas, identificar tendências centrais e avaliar a variabilidade dos dados. Para obter o intervalo sem outlier, execute o Código 16:

Código 16 - Verificação dos limites para outliers.

```
#CÓDIGO 16
stats = df_completo.describe()
Q1 = stats.loc['25%', 'MolWt']
Q3 = stats.loc['75%', 'MolWt']
IQR = Q3 - Q1
# intervalo sem outliers
lower_bound = Q1 - 1.5*IQR
upper_bound = Q3 + 1.5*IQR
print(f'Q1 = {Q1}')
print(f'Q3 = {Q3}')
print(f'IQR = {IQR}')
print(f'Intervalo sem outliers: [{lower_bound}, {upper_bound}]')
```

A Importância de considerar outliers em análises moleculares.

Ao lidar com conjuntos de dados moleculares em contextos como química medicinal, farmacologia e descoberta de medicamentos, frequentemente nos deparamos com moléculas que são identificadas como outliers, ou seja, pontos de dados que se desviam significativamente do padrão observado na maioria dos casos.

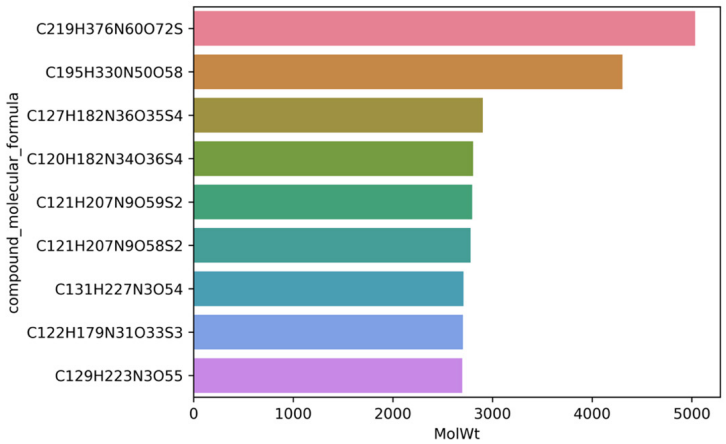
A abordagem tradicional em análises estatísticas e de machine learning tende a remover ou tratar esses outliers devido ao seu comportamento não convencional. Mas, é importante reconhecer que, em muitos casos, essas moléculas atípicas podem conter informações valiosas e insights cruciais para entender o comportamento molecular e, consequentemente, o desenvolvimento de fármacos.

Considerando que as moléculas de massa molecular acima de 999 Dalton (Da) seja considerada um outlier, será criado um conjunto de dados contendo apenas essas moléculas. Para isso, propôs-se o Código 17, que agrupa essas moléculas e destaca as 10 moléculas com maiores massas moleculares (Figura 9).

Código 17 - Análise exploratória dos dados. Construção de gráfico para ranqueamento de moléculas por massa molecular.

```
#CÓDIGO 17
moleculas_massa_mq1000 = df_completo.query('MolWt >= 977.48')
top10_moleculas_massa_mq1000 = moleculas_massa_mq1000.sort_values(by='MolWt', ascending=False).head(10)

import seaborn as sns
import matplotlib.pyplot as plt

# Define uma paleta de cores
sns.set_palette("husl")

# Plotando o mesmo gráfico, mas na horizontal para um melhor entendimento
sns.barplot(data=top10_moleculas_massa_mq1000, x='MolWt', y='compound_molecular_formula')

# Adiciona título ao gráfico
plt.title('Moléculas com maiores massas moleculares')

# Exibe o gráfico
plt.show()
```

Figura 9 - Ranking das 10 moléculas de produtos naturais da NPAtlas com maiores massas molecular.



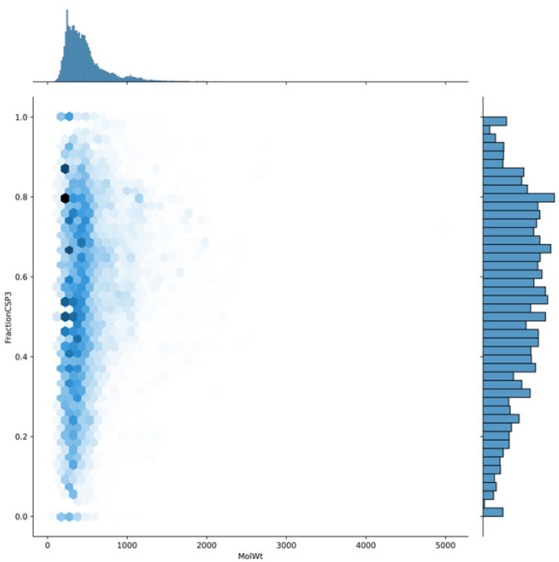
Existem diversas razões pelas quais moléculas são identificadas como outliers. Podem ser devidos a propriedades físico-químicas únicas, interações específicas com alvos moleculares ou mesmo erros de medição em experimentos laboratoriais. Independentemente da causa, ignorar essas moléculas pode levar a uma perda significativa de informações relevantes. Portanto, meu caro leitor, tome cuidado. Caso contrário, você perderá dados valiosos!

É crucial considerar o contexto do problema em questão ao decidir lidar com outliers. Por exemplo, em pesquisa farmacêutica, um composto que se destaca dos demais pode ser o precursor de um novo medicamento promissor. Da mesma forma, em estudos de toxicologia, uma molécula com comportamento atípico pode indicar efeitos adversos significativos.

Portanto, ao analisar conjuntos de dados moleculares, é fundamental adotar uma abordagem flexível e exploratória que permita a inclusão e investigação de outliers. Essas moléculas podem oferecer insights valiosos que não seriam descobertos de outra forma, impulsionando avanços importantes no campo da química de produtos naturais, na seleção por alvos biomacromoleculares diversos.

Assim, a decisão de considerar ou não outliers em análises moleculares deve ser cuidadosamente ponderada, levando em conta o potencial informativo desses pontos de dados excepcionais e o impacto que eles podem ter nas conclusões científicas e descobertas de medicamentos.

Figura 10 - Gráfico de dispersão hexagonal.



É possível criar gráficos, como os da Figura 10, que fazem a representação da relação entre a massa molecular e a fração de carbonos hibridizados em sp3 no conjunto de dados do NPAtlas, usando uma representação hexagonal para mostrar a densidade de pontos ao longo dessa relação. Isso

pode ajudar a identificar padrões ou correlações entre essas duas variáveis. Pela análise do gráfico, a maior parte das moléculas apresentam massa molecular em torno de 500 daltons, e esse grupo de estruturas orgânicas apresenta aproximadamente 50% dos seus carbonos na hibridização de sp3. Isso é relevante, pois a geometria dos átomos em uma molécula pode influenciar sua interação com alvos biológicos, como proteínas, enzimas ou receptores. Moléculas com átomos de carbono híbridos sp3 podem ter uma conformação tridimensional específica que é necessária para se ligar e interagir efetivamente com o alvo biológico. O arranjo espacial desses átomos pode determinar a atividade biológica da molécula, tornando a hibridização sp3 crucial para o design de medicamentos.

É possível criar gráficos de comparações hexagonais seguindo o Código 18 descrito abaixo:

Código 18 - Análise exploratória dos dados. Construção de gráficos comparativos.

```
#CÓDIGO 18
# utilize seu dataframe completo
sns.jointplot(data=df_completo,
x='MolWt', y='FractionCSP3', kind='hex', height=10)
# plt.yticks(np.arange(0, 1.01e8, 0.1e8))
plt.savefig('massa_csp3.png', dpi=600, bbox_inches='tight')
```

Correlação entre os descritores químicos.

Analisar correlações antes de aplicar técnicas de machine learning é crucial por várias razões:

- **Identificar relações relevantes:** A análise de correlação pode ajudar a identificar quais variáveis estão relacionadas entre si. Isso é importante porque em muitos conjuntos de dados, algumas variáveis podem estar altamente correlacionadas, o que pode afetar negativamente o desempenho do modelo de machine learning, causar multicolinearidade e até mesmo introduzir viés no modelo.
- **Feature Selection:** A análise de correlação pode ser usada para ajudar na seleção de variáveis (feature selection). Se duas variáveis estiverem altamente correlacionadas, talvez apenas uma delas seja necessária para explicar a variação nos dados. Isso pode reduzir a dimensionalidade do conjunto de dados e simplificar o modelo, evitando a inclusão de variáveis redundantes.

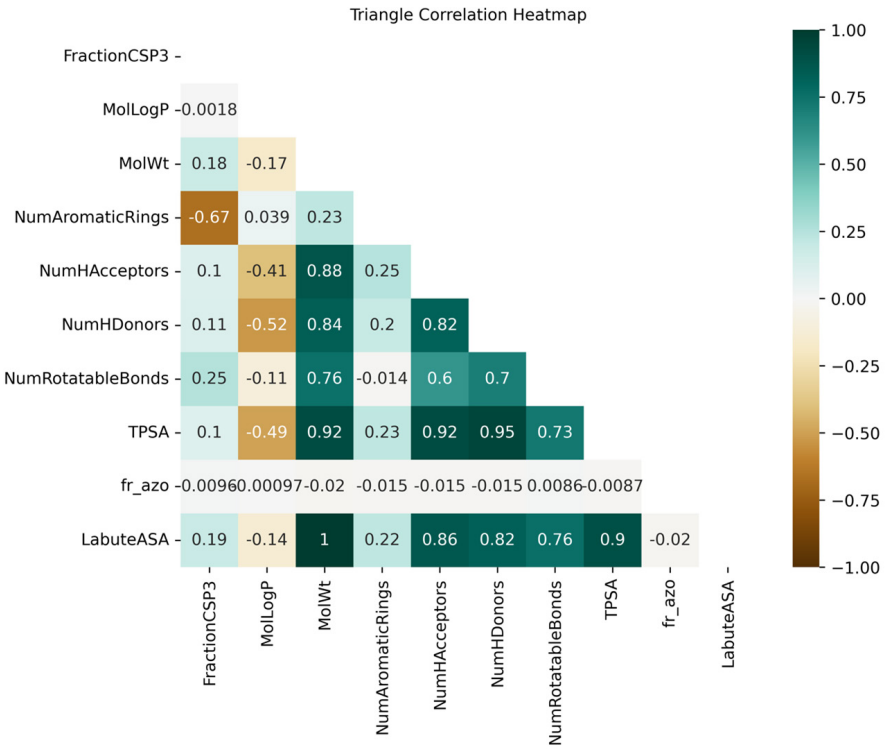
- **Evitar Overfitting:** Variáveis altamente correlacionadas podem introduzir overfitting nos modelos de machine learning, onde o modelo se ajusta muito bem aos dados de treinamento, mas não generaliza bem para novos dados. Reduzir a redundância nas variáveis pode ajudar a mitigar o overfitting.
- **Interpretabilidade:** Modelos mais simples e interpretação de resultados são favorecidos em muitos casos. Reduzir a redundância nas variáveis torna o modelo mais fácil de interpretar, pois podemos entender melhor quais características são mais importantes para fazer previsões.
- **Eficiência Computacional:** Modelos com menos variáveis são geralmente mais rápidos de treinar e executar. Portanto, remover variáveis altamente correlacionadas pode levar a um processo de modelagem mais eficiente.

Dessa forma, a análise de correlações antes da aplicação de técnicas de machine learning é importante para garantir a qualidade do modelo, evitar problemas como overfitting e multicolinearidade, e facilitar a interpretação dos resultados. Para verificar as correlações entre os descritores químicos utilizados, explore o Código 19 demonstrado abaixo:

Código 19 - Análise exploratória de dados. Construção de gráfico de correlação.

```
#CÓDIGO 19
#selecione apenas os descritores do seu conjunto de dados:
df_correlacao =
df_completo[['FractionCSP3',   'MolLogP','MolWt','NumAromaticRings',
'NumHAcceptors','NumHDonors'  'NumRotatableBonds',  'TPSA',  'fr_azo',
'LabuteASA']]


# faça a plotagem do gráfico
import numpy as np
plt.figure(figsize=(8, 6))
mask  =  np.triu(np.ones_like(df_correlacao.corr(),  dtype=bool))   #
creates a triangular matrix based on the pandas correlation matrix


heatmap  =  sns.heatmap(df_correlacao.corr(),  mask=mask,  vmin=-1,
vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Triangle          Correlation          Heatmap',
fontdict={'fontsize':10}, pad=6);
```

Figura 11 - Gráfico de correlação entre os descritores químicos propostos.



Pela Figura 11 é possível constatar que existem descritores químicos com alta colinearidade. A decisão de retirar variáveis altamente correlacionadas depende do contexto específico do problema e dos dados em questão. Aqui estão algumas considerações para ajudar a orientar essa decisão:

- **Relevância biológica ou teórica:** Se as variáveis correlacionadas são ambas relevantes para o problema em questão e têm significado biológico ou teórico, pode ser preferível mantê-las no modelo, mesmo que isso introduza alguma multicolinearidade. Mas, se uma das variáveis não tem um significado prático ou interpretação clara, pode ser razoável removê-la.
- **Impacto no Desempenho do Modelo:** Em alguns casos, variáveis altamente correlacionadas podem levar a problemas como overfitting ou instabilidade nos coeficientes estimados pelo modelo. Se isso for observado durante a validação do modelo, pode

ser aconselhável remover uma das variáveis correlacionadas para melhorar a generalização do modelo para novos dados.

- **Interpretabilidade do modelo:** Modelos mais simples e interpretação de resultados são preferidos em muitos casos. Remover variáveis altamente correlacionadas pode simplificar o modelo e tornar as interpretações dos coeficientes mais claras.
- **Custo computacional:** Se o conjunto de dados for muito grande e a presença de variáveis altamente correlacionadas estiver causando um aumento significativo no tempo de treinamento ou na carga computacional, pode ser benéfico remover algumas dessas variáveis para melhorar a eficiência do modelo.

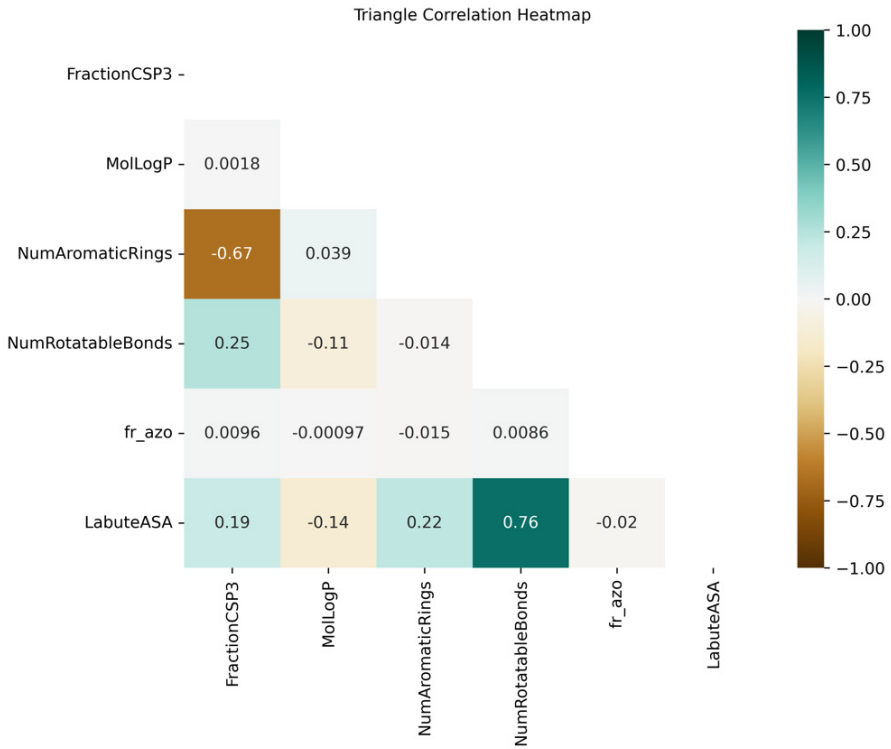
Para uma análise mais precisa das correlações entre os descritores, é viável rodar o Código 20, que apresentará de forma organizada os resultados na tela. Essa abordagem facilita uma análise minuciosa, auxiliando na identificação dos descritores que podem ser dispensáveis no conjunto de dados.

Código 20 – Verificação da existência de variáveis altamente correlacionadas

```
#CÓDIGO 20
from itertools import combinations
corr_matrix = df_correlacao.corr()
comb = combinations(corr_matrix.columns, 2)
for c1, c2 in comb:
    if abs(corr_matrix.loc[c1, c2]) > 0.8:
          print(f'r2 entre as variáveis {c1} e {c2} = {corr_matrix.
loc[c1, c2]:.3f}')
```

Ao examinar essas correlações, podemos identificar descritores que estão altamente correlacionados entre si, o que sugere a possibilidade de removê-los do conjunto de dados. Por exemplo, a massa molecular (MolWt) mostra forte correlação com o número de doadores e aceitadores de ligação de hidrogênio, a área topológica superficial e a área superficial acessível (Labute ASA). Portanto, ao construir o conjunto de dados, optamos por incluir apenas seis descritores químicos: FractionCSP3, MolLopP, NumAromaticRings, NumRotatableBonds, fr_azo e Labute ASA. A disposição dessas correlações pode ser observada de forma clara na Figura 12, permitindo apenas correlações menores do que 0,8.

Figura 12 - Gráfico de correlação entre os descritores químicos propostos (retirando os descritores altamente correlacionados).



Após a definição dos descritores que serão utilizados como fonte de treinamento para os modelos de aprendizado de máquina, assim como a obtenção de insights cruciais e o conhecimento do conjunto de dados, você está pronto para avançar para a próxima etapa. Agora, é possível proceder à inserção dos dados de treinamento e teste nos modelos de machine learning. Essa etapa é fundamental para o desenvolvimento e avaliação da eficácia dos modelos em prever ou classificar os dados com base nos padrões identificados nos descritores selecionados. Tais descritores são suficientes para permitir que os modelos possam aprender padrões e separar adequadamente uma molécula oriunda de fungo ou de bactéria

O QUE SERIA APRENDIZADO DE MÁQUINA?

Segundo uma definição clássica de Arthur Samuel, o termo *machine learning* refere-se a "um campo de estudo que capacita o computador a aprender sem ser explicitamente programado". Posteriormente, uma definição mais técnica foi proposta por Tom Mitchell, um dos principais nomes na popularização da inteligência artificial. Segundo ele, um computador aprende a partir de uma experiência E, em relação a um conjunto de tarefas T e uma medida de desempenho P, se o seu desempenho nas tarefas T, avaliado por P, melhora com a experiência E. Embora mais formal, essa definição contribui para uma compreensão mais precisa do conceito de aprendizado de máquina.

Consideremos o exemplo de codificar um algoritmo para detectar spam em e-mails. A tarefa T nesse caso seria classificar se um e-mail é spam ou não. E qual seria a experiência? Seria um conjunto de dados que inclui não apenas os e-mails, mas também os rótulos que indicam se são spam ou não. Essa experiência é o conjunto total de dados disponíveis para treinar o modelo, incluindo diferentes tipos de dados além dos rótulos, como metadados dos e-mails, por exemplo. Portanto, temos um conjunto de dados prévios para realizar uma determinada tarefa. Para medir se o algoritmo é eficaz nessa tarefa com base nessa experiência, precisamos de uma medida de desempenho, que poderia ser a acurácia.

Em outras palavras, um computador está aprendendo com uma coleção de dados, incluindo e-mails rotulados como spam ou não spam, para realizar a tarefa de classificação de e-mails, levando em conta a acurácia dessa classificação e se a performance melhora com essa experiência. Por exemplo, se o título do e-mail contiver palavras como "oferta" ou "promoção", podemos considerá-lo como spam. No entanto, no contexto do machine learning, permitimos que o programa aprenda com os dados, em vez de ser explicitamente programado com regras. Isso representa uma abordagem mais flexível e poderosa para lidar com problemas complexos como a detecção de spam.

Vamos considerar um exemplo de classificação de moléculas orgânicas como ativas ou inativas contra uma determinada doença, como o câncer. Nesse caso:

1. Experiência (E): A experiência consistiria em um conjunto de dados que inclui informações sobre diferentes moléculas orgânicas, como suas estruturas químicas, propriedades físico-químicas, atividades biológicas e resultados de ensaios experimentais. Esses dados seriam coletados de estudos anteriores, bancos de dados públicos ou experimentos laboratoriais.

2. Tarefa (T): A tarefa seria classificar as moléculas como ativas ou inativas contra a doença em questão. Isso seria determinado com base nos resultados dos ensaios biológicos ou testes de atividade farmacológica das moléculas.

3. Medida de Desempenho (P): A medida de desempenho poderia ser a precisão do modelo em prever corretamente se uma molécula é ativa ou inativa. Outras métricas como sensibilidade, especificidade, área sob a curva ROC, acurácia, também poderiam ser usadas, dependendo das características específicas do problema e das necessidades do projeto.

Assim, utilizando esses conceitos, podemos aplicar técnicas de machine learning para desenvolver um modelo capaz de classificar as moléculas orgânicas como ativas ou inativas contra a doença alvo. O modelo seria treinado com base nos dados disponíveis (experiência) e avaliado com base em sua capacidade de realizar corretamente essa classificação (tarefa) com uma medida apropriada de desempenho (precisão, sensibilidade, etc.).

A partir dessas duas definições, podemos perceber que nos dias de hoje, os dados são considerados o novo petróleo. Grandes empresas utilizam extensivamente dados para tomadas de decisão, sugestão de produtos e muito mais. Portanto, é crucial que nós utilizemos e processemos esses dados para desenvolver soluções automáticas para os nossos problemas.

Hoje em dia, uma das realidades mais significativas na área que está em constante crescimento é o machine learning, que engloba não apenas o treinamento de algoritmos a partir de dados para automatizar processos, mas também a ciência de dados como um todo. Machine learning vai além, possibilitando análises exploratórias e outras aplicações, não se limitando apenas ao treinamento de algoritmos. É uma ferramenta poderosa para a automação de processos e tomada de decisões eficazes em uma variedade de problemas

No contexto do machine learning, não precisamos programar explicitamente essas regras. Em vez disso, utilizamos dados existentes, nossa experiência e conhecimento prévio sobre o problema para treinar um algoritmo.

Esse algoritmo extrai implicitamente padrões e conhecimentos dos dados durante o treinamento, os quais incluem as regras que poderíamos ter definido na abordagem tradicional. Após o treinamento, avaliamos o desempenho do modelo e, se necessário, refinamos o processo, examinando os erros e iterando até alcançarmos uma solução satisfatória para colocar em produção.

De maneira mais sistemática para você, leitor, enquanto na abordagem tradicional escrevemos explicitamente as regras, no machine learning, deixamos que o algoritmo aprenda essas regras implicitamente a partir dos dados. Isso oferece uma abordagem mais flexível e adaptável, permitindo lidar com problemas complexos e não-lineares de forma mais eficaz.

Na abordagem do machine learning, o objetivo é aliviar a necessidade de criar manualmente regras complexas para alcançar resultados satisfatórios. Por exemplo, consideremos um conjunto de dados representado por quadrados azuis e triângulos vermelhos, onde os quadrados azuis denotam moléculas orgânicas inativas contra uma determinada doença, enquanto os triângulos vermelhos representam moléculas ativas contra essa doença. Esses dados são então utilizados para treinar modelos de machine learning.

Os algoritmos de machine learning são projetados para extrair automaticamente características relevantes dos dados, em vez de analisar as informações químicas das moléculas como um todo. Essas características são padrões ou atributos específicos que distinguem entre moléculas ativas e inativas contra a doença alvo.

Essa abordagem permite que os sistemas de machine learning tomem decisões automatizadas ou realizem tarefas sem intervenção humana. Por exemplo, em pesquisas envolvendo produtos naturais, ou drug discovery, um sistema pode analisar as propriedades químicas de diferentes moléculas para determinar sua eficácia contra uma doença específica. Essa avaliação é baseada nos padrões extraídos dos dados durante o treinamento do modelo.

Com um conjunto de dados representativo, os algoritmos de machine learning podem classificar as moléculas em categorias como “ativa” ou “inativa” contra a doença. Com base nessas categorias, pesquisadores podem identificar potenciais candidatos a fármacos e direcionar seus esforços de desenvolvimento de medicamentos de forma mais eficaz. Essa flexibilidade e capacidade de adaptação são características-chave do machine learning, tornando-o uma ferramenta poderosa na descoberta e desenvolvimento de novas terapias médicas, sobretudo, oriundas de fontes naturais, como é o propósito desse livro.

A análise de imagens moleculares a partir da fórmula estrutural do composto é fundamental para identificar sua atividade biológica. Nesse contexto, podemos classificar a molécula com base em sua atividade, mesmo sem conhecer necessariamente todos os descritores químicos associados à sua estrutura. O algoritmo para tomada de decisão precisa lidar com esses dados de entrada de forma eficaz, independentemente da natureza deles, seja estruturados ou não.

O aspecto central aqui é como lidar com esses dados de entrada e extrair características relevantes deles. Esse processo pode variar significativamente de acordo com a aplicação específica. Por exemplo, alguns algoritmos são mais adequados para lidar com imagens coloridas, enquanto outros são mais eficientes com imagens em tons de cinza. No entanto, o objetivo final é o mesmo: tomar decisões informadas com base nos dados disponíveis e buscar separar as classes, reconhecendo padrões específicos de cada molécula.

É importante destacar que cada aplicação apresenta suas próprias peculiaridades. Por exemplo, imagens médicas frequentemente possuem ruídos e características específicas que exigem cuidados adicionais durante a análise. Portanto, é essencial adaptar a abordagem de acordo com as necessidades e peculiaridades de cada problema.

Apesar das variações e desafios específicos de cada aplicação, todas elas compartilham uma base comum de processamento e análise de dados. Em última análise, a eficácia dos algoritmos depende da capacidade de extrair informações úteis, independentemente da complexidade ou da natureza dos dados. A Tabela 4 contempla exemplos de aplicações e a área do aprendizado de máquina associada a elas.

Tabela 4 - Exemplo de aprendizado de máquina e suas aplicações

Área de Machine Learning	Exemplos de Aplicações
Aprendizado Supervisionado	Predição de Atividade Biológica de Moléculas, Predição de Propriedades Químicas de Compostos
Aprendizado Não Supervisionado	Agrupamento de Moléculas com Estruturas Similares
Aprendizado por Reforço	Planejamento de Síntese Química, Otimização de Parâmetros de Reação
Processamento de Linguagem Natural (PLN)	Extração de Informações de Literatura Científica, Análise de Relatórios de Ensaios Clínicos

Área de Machine Learning	Exemplos de Aplicações
Visão Computacional	Análise de Estruturas Moleculares em Imagens, Segmentação de Imagens de Microscopia
Análise de Séries Temporais	Monitoramento de Reações Químicas ao Longo do Tempo
Aprendizado Profundo	Geração de Moléculas Novas, Reconhecimento de Padrões em Espectros Químicos

Aprendizado Supervisionado na Química:

Predição de Atividade Biológica de Moléculas: Nesse caso, modelos de aprendizado supervisionado são treinados com dados de moléculas conhecidas e suas atividades biológicas, como afinidade com determinadas proteínas ou capacidade de inibir enzimas. O objetivo é prever a atividade biológica de novas moléculas com base em suas características estruturais e químicas.

Predição de Propriedades Químicas de Compostos: Aqui, o objetivo é prever propriedades físico-químicas de compostos, como solubilidade, polaridade ou constante de acidez, a partir de características estruturais das moléculas. Isso é útil para otimizar o design de novos compostos com propriedades desejadas.

Aprendizado Não Supervisionado na Química:

Agrupamento de Moléculas com Estruturas Similares: Algoritmos de aprendizado não supervisionado são utilizados para agrupar moléculas com características estruturais semelhantes. Esses agrupamentos podem revelar padrões importantes nas propriedades químicas das moléculas e auxiliar na análise exploratória de dados.

Aprendizado por Reforço na Química

Planejamento de Síntese Química: Nessa aplicação, o aprendizado por reforço é utilizado para otimizar o processo de planejamento de síntese química, sugerindo sequências de reações e condições de reação que levem à síntese eficiente de compostos desejados.

Otimização de Parâmetros de Reação: Algoritmos de aprendizado por reforço também podem ser empregados para otimizar os parâmetros de reações químicas, como temperatura, pH e tempo de reação, visando maximizar o rendimento e a seletividade dos produtos.

Processamento de Linguagem Natural (PLN) na Química

Extração de informações de literatura científica: Técnicas de PLN são aplicadas para extrair informações relevantes de artigos científicos e patentes na área da química, como propriedades de compostos, métodos de síntese e resultados de ensaios biológicos.

Análise de relatórios de ensaios clínicos: O PLN pode ser usado para analisar relatórios de ensaios clínicos e identificar informações relevantes sobre a eficácia e segurança de novos medicamentos em desenvolvimento.

Visão Computacional na Química

Análise de estruturas moleculares em imagens: Algoritmos de visão computacional são aplicados para analisar imagens de estruturas moleculares obtidas por técnicas de microscopia, permitindo a identificação e quantificação de características relevantes das moléculas.

Segmentação de imagens de microscopia: A visão computacional também é utilizada para segmentar imagens de microscopia de células e tecidos, facilitando a análise e quantificação de estruturas biológicas em estudos de toxicologia e farmacologia.

Análise de Séries Temporais na Química

Monitoramento de reações químicas ao longo do tempo: Técnicas de análise de séries temporais são empregadas para monitorar o progresso de reações químicas ao longo do tempo, permitindo a identificação de intermediários, produtos e impurezas, bem como a otimização das condições de reação.

Aprendizado Profundo na Química

Geração de moléculas novas: Modelos de aprendizado profundo são utilizados para gerar novas estruturas moleculares com características específicas, como atividade biológica desejada ou propriedades físico-químicas favoráveis.

Reconhecimento de padrões em espectros químicos: O aprendizado profundo é aplicado para reconhecer padrões em espectros químicos, como espectros de massas ou espectros de RMN, auxiliando na identificação de compostos desconhecidos em misturas complexas.

Classificadores de aprendizado supervisionado

Logistic Regression

Ao explorar os domínios do aprendizado de máquina, um ponto de partida essencial é a compreensão do aprendizado supervisionado, uma técnica que se baseia na orientação fornecida por um conjunto de dados previamente rotulado. Neste contexto, emerge a classificação como um dos pilares fundamentais desse paradigma.

A classificação, intrinsecamente ligada aos algoritmos de aprendizado supervisionado, pressupõe a presença de um conjunto de treinamento composto por características distintas, ou “features”, acompanhadas de anotações denominadas rótulos, labels ou classes. Para ilustrar, consideremos uma situação em que nos deparamos com um conjunto de imagens representando moléculas de fungos e bactérias. Sem uma prévia organização ou identificação das imagens, surge a necessidade humana de examinar cada uma e atribuir a elas uma categoria apropriada.

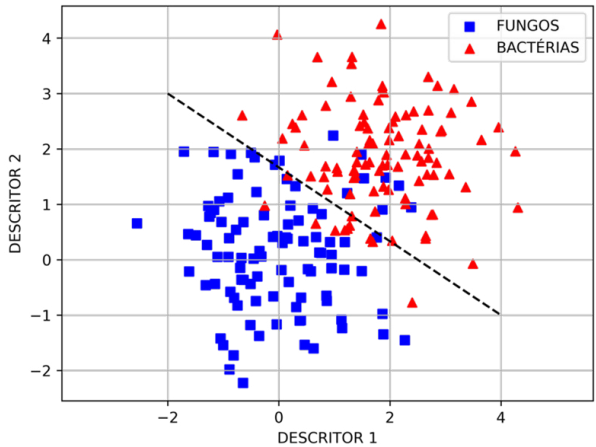
Essa organização é essencial para o treinamento eficaz de algoritmos de classificação. Requer-se, assim, a intervenção humana para designar esses rótulos, inserindo uma camada de supervisão no processo de aprendizado. No entanto, é pertinente destacar que esse processo é dispendioso em termos de recursos temporais e humanos, especialmente em cenários de conjuntos de dados volumosos ou com complexidades intrínsecas que exigem expertise especializada.

Diante dessa complexidade, surge a necessidade de estratégias mais eficientes para a rotulagem de dados, dando origem a abordagens como o aprendizado ativo. Essa técnica, alinhada ao conceito de aprendizado supervisionado, busca otimizar o processo de rotulagem, aproveitando-se de mecanismos automatizados para identificar padrões em dados não rotulados e, assim, guiar a atribuição de rótulos de forma mais inteligente.

O aprendizado ativo não apenas simplifica o processo de rotulagem, mas também oferece uma abordagem iterativa e adaptativa, onde o modelo em construção é constantemente refinado com base no feedback fornecido pelo especialista humano ou pelos próprios algoritmos. É algo muito parecido com o que acontece com o Google Fotos, que faz uma organização automática, mas quando ele está confuso, ele pergunta quem é aquela pessoa da foto, e assim, aprende outra característica e melhora sua performance classificatória. Tudo isso representa uma evolução significativa na busca por eficiência e precisão no aprendizado supervisionado, mitigando os desafios inerentes à rotulagem manual de conjuntos de dados extensos e complexos.

Para a separação das classes, uma abordagem comumente empregada é o uso de um regressor logístico, capaz de criar um limite de decisão (decision boundary). Dessa forma, o modelo é capaz de distinguir entre as classes, e quando apresentado a uma nova amostra não observada, ele a classifica com base em qual lado do hiperplano ela se encontra, de acordo com o limiar estabelecido. A Figura 13 ilustra um exemplo dessa separação de classes, demonstrando como é feita a distinção entre moléculas provenientes de fungos e de bactérias. E de maneira resumida, a regressão logística é um modelo de regressão usado para problemas de classificação binária. Ele estima a probabilidade de uma observação pertencer a uma determinada classe, utilizando uma função logística para calcular as probabilidades.

Figura 13 - Exemplo de classificação binária (separação) usando algoritmos de supervisionados de machine learning.

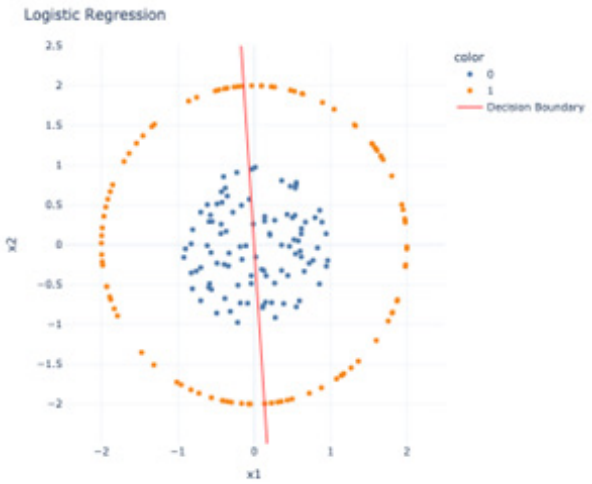


Imagine que seja criado um conjunto fictício de dados que representam descritores químicos, e que ao visualizá-los em um gráfico, observamos uma disposição semelhante à mostrada na Figura 14. Nesse cenário, é notável que uma das classes forma um círculo central, enquanto a outra classe se estende ao redor dele.

Ao treinar um algoritmo de regressão linear com esses dados, é provável que ele produza uma linha de decisão que não consiga separar eficientemente as duas classes. Se considerarmos essa linha de decisão, o modelo estaria sugerindo que todos os pontos à direita pertencem à classe 1 e os à esquerda à classe 0. No entanto, essa abordagem resultaria em muitos erros de classificação, indicando um viés significativo no modelo e sofrendo de underfitting, ou seja, uma incapacidade do modelo de se ajustar adequadamente aos dados.

Para evitar esse problema de underfitting e melhorar a capacidade do modelo de aprender com os dados, seria necessário empregar métodos mais avançados que possam capturar a complexidade do relacionamento entre os descritores químicos e as classes correspondentes.

Figura 14 - Disposição de diferentes classes moleculares sendo aplicada uma regressão logística.

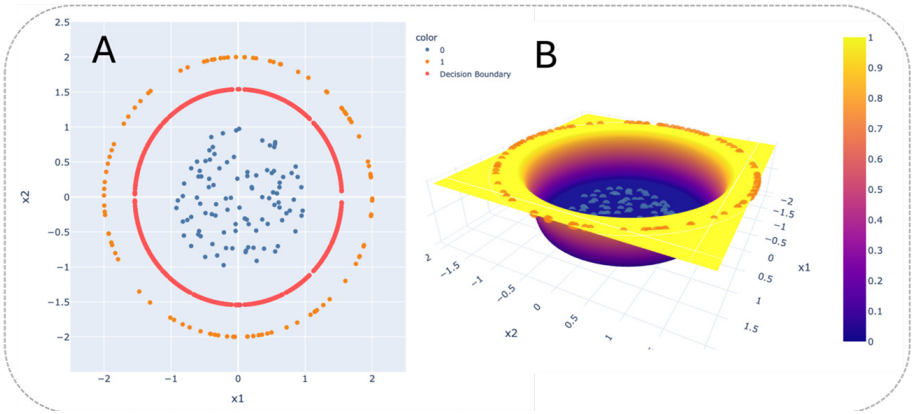


O modelo de regressão linear enfrenta dificuldades em aprender quando as suposições subjacentes não são atendidas, como é o caso da separabilidade linear dos dados. Porém, para contornar essa limitação, uma abordagem comum é expandir o modelo para incluir características polinomiais.

Ao aumentar o grau do polinômio, o modelo se torna capaz de capturar relações mais complexas entre os dados. Por exemplo, ao aplicar um polinômio de segundo grau, torna-se possível separar os dados de forma mais eficaz, como ilustrado na Figura 15. Isso amplia consideravelmente as oportunidades de resolver problemas que anteriormente pareciam insolúveis usando um modelo linear simples.

Essa flexibilidade adicional na modelagem permite lidar com casos mais desafiadores e melhorar a capacidade do modelo de se adaptar aos padrões presentes nos dados.

Figura 15 - Exemplo de classificação não linear utilizando regressão logística polinomial. (A) A transformação polinomial permite separar dados em formato circular. (B) A superfície mostra como o modelo estima probabilidades para cada ponto no espaço de entrada.



Neste contexto, podemos imaginar que dentro do círculo representado no gráfico estariam as moléculas da classe fungo, enquanto as moléculas da classe bactéria estariam localizadas fora do círculo. É importante ressaltar que essas são apenas suposições com base em valores hipotéticos utilizados para criar nosso conjunto de dados.

Dessa forma, para resolver esse tipo de problema, precisamos de um modelo mais flexível que possa capturar a complexidade das relações entre os dados. Um modelo não linear, como um círculo ou uma elipse, poderia ser mais adequado para separar efetivamente essas classes.

Do ponto de vista matemático, isso significa que estamos saindo de uma regressão linear simples, representada pela fórmula:

$$y=h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Para um modelo polinomial de grau 2, onde a equação seria:

$$y=h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 ,$$

Nesta nova equação, além de termos cada uma das variáveis (representadas por x_1 e x_2) elevadas ao quadrado, também temos o termo de interação $x_1.x_2$, o que não estava presente no modelo linear.

Essa abordagem mais complexa nos permite capturar relações não lineares entre as variáveis e, assim, melhorar a capacidade do modelo de se ajustar aos dados de forma mais precisa.

A regressão logística, embora seja um modelo estatístico relativamente simples, desempenha um papel fundamental na solução de problemas de classificação, inclusive na distinção entre classes químicas, como moléculas de fungos e bactérias. A sua utilidade está pautada na capacidade de prever a probabilidade de ocorrência de um evento, ou classe, com base em variáveis independentes. Embora seu nome contenha “regressão”, a regressão logística é comumente utilizada para problemas de classificação binária, onde o objetivo é separar as classes de interesse. Graças à sua simplicidade e interpretabilidade, além da capacidade de lidar com relacionamentos não lineares, a regressão logística é uma ferramenta valiosa para resolver uma ampla gama de problemas de separação de classes químicas e é frequentemente empregada em diversas áreas, desde a medicina até a química de produtos naturais, que é nosso objetivo central.

Naive Bayes (NaiveBayes)

O teorema de Bayes é uma ferramenta importante na análise estatística, especialmente quando se trata de inferência estatística e aprendizado de máquina. É rápido e eficiente para conjuntos de dados com muitas características, mas pode não funcionar tão bem em dados com dependências complexas entre as características. Antes de entrarmos nos detalhes do Naive Bayes, é essencial entender a distinção entre probabilidade e verossimilhança.

Probabilidade vs. Verossimilhança:

Probabilidade refere-se à chance de um evento ocorrer, enquanto verossimilhança representa a adequação de um modelo estatístico aos dados observados. Em outras palavras, a probabilidade quantifica nossas crenças sobre um evento antes de observá-lo, enquanto a verossimilhança mede a plausibilidade de os dados observados serem produzidos pelo modelo proposto.

Podemos calcular a probabilidade de pertencer a cada classe usando a distribuição normal previamente estabelecida. Se a probabilidade de pertencer à classe “A” for maior do que à classe “B”, classificamos a molécula como pertencente à classe “A”, e vice-versa.

Vamos realizar um exemplo prático. Para isso iremos separá-lo em 4 partes:

Parte 1 – O problema. O que queremos resolver?

Suponha que queremos classificar uma nova molécula com uma determinada massa molecular como molécula obtida de fungo ou de bactéria.

Podemos calcular a probabilidade de ser “Fungo” ou “Bactéria” usando a distribuição normal dos pesos moleculares em cada classe e aplicando o teorema de Bayes. A classe com a maior probabilidade é atribuída à molécula.

Parte 2 – Dados conhecidos.

Suponha que temos as seguintes probabilidades a priori[2]:

- Probabilidade de uma molécula ser um fungo: P(Fungo) = 0,6
- Probabilidade de uma molécula ser uma bactéria: P(Bactéria) = 0,4

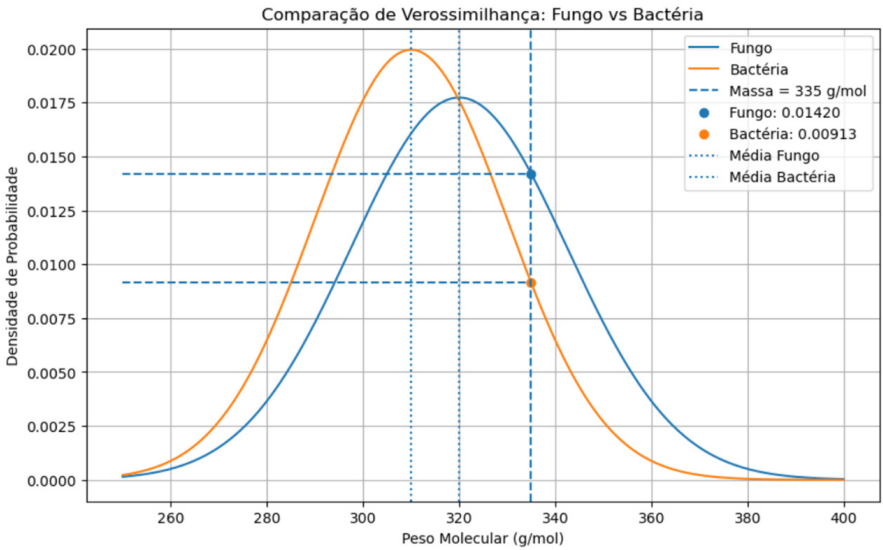
Além disso, temos as seguintes distribuições normais para os pesos moleculares em cada classe:

- Para “Fungo”: Média (μ) = 320 g/mol, Desvio Padrão (σ) = 22,5 g/mol
- Para “Bactéria”: Média (μ) = 310 g/mol, Desvio Padrão (σ) = 20 g/mol

Agora, suponha que temos uma nova molécula com peso molecular de 335 g/mol e queremos determinar a probabilidade de ser um fungo ou uma bactéria. Vamos calcular a verossimilhança de a nova molécula ter peso molecular de 335 g/mol, dada a distribuição normal em cada classe:

2 Esse valor de probabilidade você retira do seu conjunto de dados de treinamento. Exemplo: se você tem um dataset com 100 amostrar e 60 delas são fungos e 40 bactérias, então sua probabilidade é que seja 0,6 para fungo e 0,4 para bactéria.

Figura 16 - Curva de distribuição normal e comparação de verossimilhança entre as duas classes analisadas.



Notem que a verossimilhança está associada ao ponto em que há a projeção do valor de 335 do eixo x até encontrar a distribuição normal (Figura 16).

Para ambas as classes:

- Verossimilhança de peso molecular (PM) de 335 g/mol: P(PM = 335 | F)
- Verossimilhança de peso molecular (PM) de 335 g/mol: P(PM = 335 g/mol | B)

Onde F = Fungo; e B = Bactéria

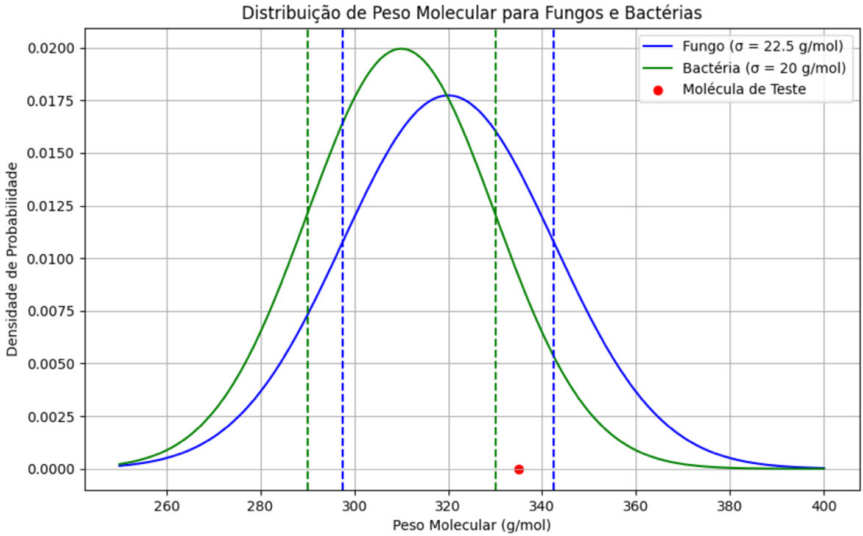
Para calcular essa probabilidade de pertencer à classe de “Fungos”, usamos a função de densidade de probabilidade (PDF) da distribuição normal com média 320 g/mol e desvio padrão de 22,5 g/mol para calcular a probabilidade de obter 335 g/mol.

Da mesma forma, calculamos a verossimilhança para a classe “Bactéria” usando a distribuição normal com média 310 g/mol e desvio padrão de 20 g/mol. Agora, podemos calcular as probabilidades a posteriori de a molécula ser um fungo ou uma bactéria:

- Para “Fungo”: P(F | PM= 335)
- Para “Bactéria”: P(B | PM= 335)

Essas probabilidades a posteriori nos dirão a probabilidade de a molécula ser um fungo ou uma bactéria, dada a observação de seu peso molecular. Na prática, esse processo é realizado utilizando as fórmulas estatísticas apropriadas e os dados disponíveis para as distribuições das classes. Acompanhe a resolução passo a passo, considerando a feature de massa molecular de cada molécula.

Figura 17 - Comparação entre as distribuições de peso molecular de fungos (curva azul) e bactérias (curva verde), ambas modeladas por distribuições normais. As linhas tracejadas indicam as médias e os desvios padrão de cada grupo. O ponto vermelho representa uma molécula de teste, cuja posição relativa sugere maior proximidade com a distribuição de fungos.



Analisando o gráfico de distribuição entre as duas classes (Figura 17), cria-se um palpite a respeito da probabilidade a priori entre cada classe.

$$P(\text{Classe} \mid \text{Peso Molecular}) = \frac{P(\text{Peso Molecular} \mid \text{Classe}) \cdot P(\text{Classe})}{P(\text{Peso Molecular})}$$

Onde:

A probabilidade *P* (Classe | Peso Molecular) representa a **probabilidade a posteriori**, isto é, a probabilidade de a molécula pertencer a uma determinada classe dado o valor observado de seu peso molecular.

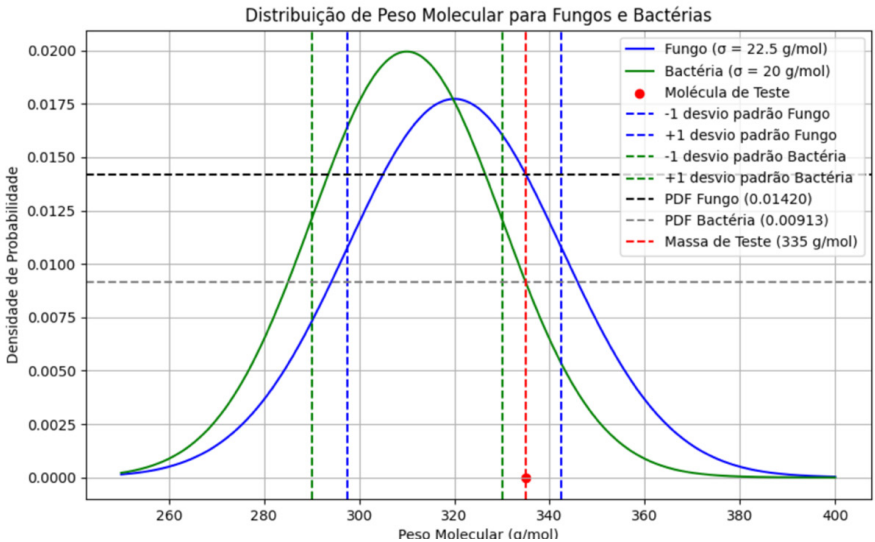
A quantidade *P* (Peso Molecular | Classe) corresponde à **verossimilhança**, que indica a probabilidade de observar um determinado valor de peso mole-

cular assumindo que a molécula pertence à classe considerada.

Já *P* (Classe) é a **probabilidade a priori**, que expressa a probabilidade inicial de uma molécula pertencer à classe antes de observar os dados.

Por fim, *P* (Peso Molecular) é a **probabilidade marginal**, também chamada de evidência, que representa a probabilidade total de observar o valor de peso molecular, independentemente da classe.

Figura 18 - Análise probabilística comparativa entre fungos e bactérias baseada em distribuições normais de peso molecular.



Complementando o gráfico com os valores de PDF para as duas classes, nota-se que a linha vermelha cruza com a linha cinza no gráfico verde (bactérias) em 0,009313 e em na distribuição normal azul (associada ao fungo) em 0,01419.

A fórmula para cálculo da densidade de probabilidade (PDF)[3] para o fungo é:

$$P\ (\text{PM= 335} \mid \text{F}) = \frac{1}{\sqrt{2\pi}*(\sigma)^2} * e\ \frac{-(x-\mu)^2}{2\sigma^2}$$

$$P\ (\text{PM = 335} \mid \text{F}) = \frac{1}{\sqrt{2\pi}*(22{,}5)^2} *\ e\ \frac{-(335-320)^2}{2(22{,}5)^2}$$

$$P\ (\text{PM= 335} \mid \text{F}) = \frac{1}{56{,}40} *\ e\ \frac{-(15)^2}{1012{,}5}$$

3 Para variáveis contínuas, a verossimilhança é calculada por meio da função densidade de probabilidade (PDF), sendo mais apropriado representá-la como f(x|C) ao invés de P(x|C), uma vez que não se trata de uma probabilidade pontual. Mas aqui estamos usando P para tornar-se mais didático.

P (PM= 335 | F) = $\frac{1}{56,40} * e \frac{-225}{1012,5}$

P (PM= 335 | F) = $0,0177 * 0,80$

P (Peso Molecular = 335g/mol | Fungo) = $0,01417$

Seguindo o mesmo raciocínio e calculando a densidade de probabilidade para a bactéria, temos:

P (PM= 335 | B) = $\frac{1}{\sqrt{2\pi}*(\sigma)^2} * e \frac{-(x-\mu)^2}{2\sigma^2}$

P (PM= 335 | B) = $\frac{1}{\sqrt{2\pi}*(20)^2} * e \frac{-(335-310)^2}{2(20)^2}$

P (PM= 335 | B) = $\frac{1}{50,13} * e \frac{-(15)^2}{800}$

P (PM= 335 | B) = $\frac{1}{50,13} * e \frac{-625}{800}$

P (PM= 335 | B) = $0,0199 * 0,458$

P (Peso Molecular = 335g/mol | Bactéria) = $0,00913$

Uma vez encontrada a verossimilhança, feita pelo cálculo de densidade de probabilidade de cada classe, é possível combinar tal métrica com as probabilidades a priori e encontrar a probabilidade marginal. Assim, teríamos:

P (PM) = P (PM= 335 | F) × P(F) + P (PM = 335 | B) × P(B)

P (335) = 0,01417 × 0,6 + 0,00913 × 0,4 = $8,502 \times 10^3 + 3,652 \times 10^3$ = **0,01215**

Portanto, todas as métricas necessárias para calcular a probabilidade a posteriori já foram explicitadas, basta combiná-las e obter a probabilidade que uma molécula de massa molecular 335 g/mol pertença a uma das classes. Vamos analisar como ficaria:

$$P(\text{F} \mid 335) = \frac{P(335 \mid \text{F}) \cdot P(\text{F})}{P(335)}$$

$$P(\text{F} \mid 335) = \frac{0,01417 \cdot 0,6}{0,01215}$$

$$P(\text{F} \mid 335) = \frac{0,008502}{0,01215}$$

$$\boldsymbol{P(\text{Fungo} \mid 335\ \text{g/mol}) = 0,70}$$

E, obviamente, a porcentagem restante, 30% seria associada à essa molécula pertencer às bactérias. Caso queira confirmar matematicamente, veja o cálculo a seguir:

$$P(\text{B} \mid 335) = \frac{P(335 \mid \text{B}) \cdot P(\text{Bactéria})}{P(335)}$$

$$P(\text{B} \mid 335) = \frac{0{,}00913 \cdot 0{,}4}{0{,}01215}$$

$$P(\text{B} \mid 335) = \frac{0{,}003652}{0{,}01215}$$

$$\boldsymbol{P(\text{Bactéria} \mid 335\ \text{g/mol}) = 0{,}30}$$

Agora as probabilidades foram atualizadas: já temos indicativos estatísticos de que uma molécula com massa molecular de 335 g/mol possui maior probabilidade de pertencer à classe dos fungos.

No entanto, sabemos que um dataset real é composto por diversas features, e é justamente nesse ponto que reside a principal ideia do algoritmo Naive Bayes: assumir que essas variáveis são independentes entre si dado a classe.

Suponha então que, além da massa molecular, passemos a considerar mais duas propriedades moleculares:

- **MLogP (coeficiente de partição)**
- **TPSA (área polar superficial**

A Tabela 5 foi feita para sistematizar a análise estatística agora para os três descritores moleculares.

Tabela 5 - Resumo das análises estatísticas utilizadas para o cálculo de Naive Bayes.

Parâmetro	Fungos	Bactérias	🧬 Nova Molécula
MolWt	μ = 320 ; σ = 22,5	μ = 310 ; σ = 20	335
MLogP	μ = 2,5 ; σ = 1,0	μ = 3,5 ; σ = 1,2	2,0
TPSA	μ = 120 ; σ = 30	μ = 90 ; σ = 25	110
Prior	**P = 0,6**	**P = 0,4**	—

Ou seja, com essas informações, aplica-se o cálculo da função densidade de probabilidade (pdf), cujos valores representam a verossimilhança de cada feature dada a classe. A verossimilhança conjunta é então obtida pelo produto dessas densidades, conforme a hipótese de independência do Naive Bayes, os resultados são mostrados na Tabela 6.

Tabela 6 - Probabilidades Condicionais (Modelo Naive Bayes Gaussiano) e cálculos de verossimilhança e probabilidade a posteriori das classes analisadas.

Feature	Fungos P(x | Fungo)	Bactérias P(x | Bactéria)
MolWt = 335	0,01417	0,0091
MLogP = 2,0	0,3520	0,121
TPSA = 110	0,0129	0,0079
Verossimilhança[4]	$6{,}43410^{-5}$	$8{,}69810^{-6}$
Priori	0,6000	0,4000
Posteriori (não padronizado)	$3{,}86010^{-5}$	$3{,}47910^{-6}$
Posteriori (padronizado)	0,9170	0,0830

VEROSSIMILHANÇA (FUNGO): $P(\text{Fungo}|X) \propto P(\text{Fungo}) \cdot P(\text{MolWt}|F) \cdot P(\text{MLogP}|F) \cdot P(\text{TPSA}|F)$

VEROSSIMILHANÇA (BACTÉRIA): $P(\text{Bactéria}|X) \propto P(\text{Bactéria}) \cdot P(\text{MolWt}|B) \cdot P(\text{MLogP}|B) \cdot P(\text{TPSA}|B)$

Observe que agora levando em consideração outras features, a probabilidade da amostra de teste pertencer à classe de fungo é de 91,70%! Bastante diferente se fosse utilizado apenas a massa molecular como feature, que apontava 70%. O algoritmo avalia o quanto cada propriedade molecular "combina" com cada classe e combina todas essas evidências multiplicativamente.

De maneira geral, esse algoritmo irá estimar a probabilidade condicional de uma classe dada os descritores químicos, usando a suposição ingênua (daí o "naive") de que os descritores são independentes entre si, dado o valor da classe.

Portanto, quando aplicado a um conjunto de dados com 10 features, O Naive Bayes calcula a probabilidade de cada feature dado a classe, assumindo independência entre elas, e combina essas probabilidades usando o Teorema de Bayes para estimar a probabilidade da classe dado o conjunto completo de features.

4 No Naive Bayes, assumimos independência entre as features, então:

- $P(X \mid \text{classe}) = P(\text{MolWt} \mid \text{classe}) \times P(\text{MLogP} \mid \text{classe}) \times P(\text{TPSA} \mid \text{classe})$
- $P(X) = 4{,}209 \times 10^{-5}$ (será utilizado para padronização)

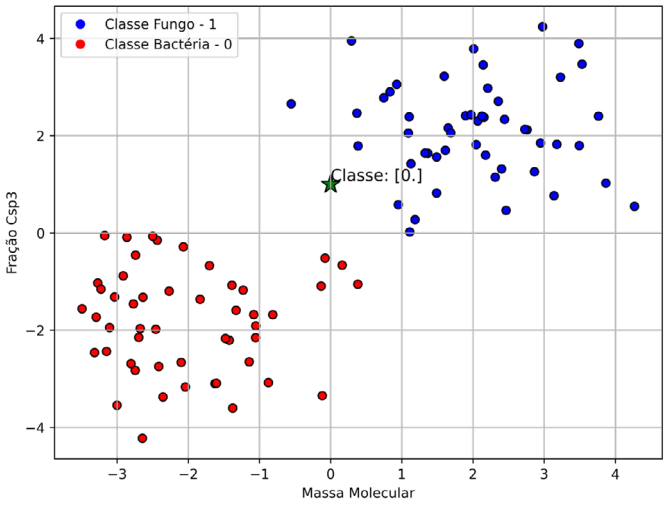
NAIVE BAYES

O Naive Bayes é um algoritmo de classificação probabilístico baseado no Teorema de Bayes, que estima a probabilidade de uma amostra pertencer a uma determinada classe com base nas probabilidades condicionais de suas características. O modelo assume, de forma simplificada, que as variáveis são independentes entre si, o que permite calcular a probabilidade conjunta como o produto das probabilidades individuais. Apesar dessa suposição, muitas vezes irreal, o Naive Bayes apresenta bom desempenho em diversos problemas, sendo eficiente, rápido e especialmente útil para conjuntos de dados de alta

K-Nearest Neighbors – KNN

O KNN é um algoritmo de classificação baseado em vizinhança. Durante a fase de treinamento, ele não otimiza parâmetros como em outros modelos; em vez disso, armazena as instâncias do conjunto de treinamento. Por exemplo, consideremos um problema de classificação de moléculas de produtos naturais com base em suas massas e frações de carbono sp3, com rótulos indicando se são de fungos ou bactérias. Para classificar uma nova molécula, representada por uma estrela verde, selecionamos um número k de vizinhos mais próximos no conjunto de treinamento. Usando uma medida de distância, como a euclidiana, calculamos a distância entre a nova amostra e todas as amostras de treinamento. As k amostras mais próximas são escolhidas como vizinhas mais similares. É importante observar que o KNN é adequado principalmente para dados numéricos, podendo lidar com dados categóricos mediante codificação apropriada, mas pode enfrentar desafios com alta dimensionalidade ou dados esparsos.

Figura 19 - Exemplo de funcionamento do algoritmo KNN.



O gráfico mostra um conjunto de dados de exemplo com duas características: Massa Molecular e Fração Csp3. Cada ponto no gráfico representa uma molécula, sendo que metade das moléculas pertence à classe de fungos (representadas em azul) e a outra metade pertence à classe de bactérias (representadas em vermelho). O ponto verde com forma de estrela indica uma nova molécula de teste para a qual queremos prever a classe usando o algoritmo KNN.

O algoritmo KNN (K-Vizinhos Mais Próximos) classifica a molécula de teste com base na classe das amostras vizinhas mais próximas no conjunto de treinamento. Neste exemplo, consideramos 3 vizinhos mais próximos. O texto anotado ao lado do ponto verde indica a classe prevista para a molécula de teste de acordo com o algoritmo KNN.

Na legenda do gráfico há indicação das cores associadas às classes de fungos e bactérias, além de identificar a molécula de teste. A Figura 19 ilustra como o algoritmo KNN pode ser usado para classificar novas amostras com base nas amostras de treinamento mais próximas no espaço de características.

K-NEAREST NEIGHBORS (KNN)

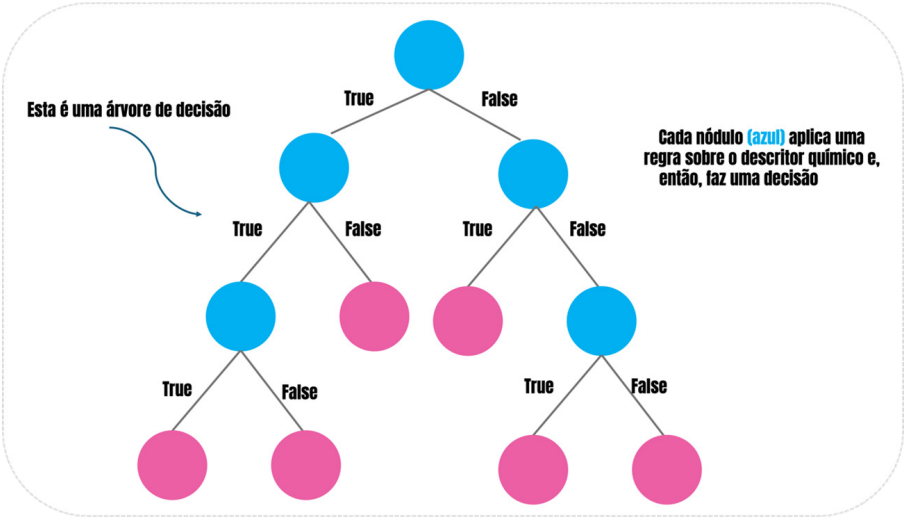
O K-Nearest Neighbors (KNN) é um algoritmo de classificação baseado em similaridade, que atribui a classe de uma nova amostra com base nas classes de seus vizinhos mais próximos no conjunto de treinamento.

Utilizando uma métrica de distância, como a euclidiana, o modelo identifica os k pontos mais próximos e realiza a classificação por meio de votação majoritária. Por não construir um modelo explícito durante o

Árvore de Decisão (CART – Classification and Regression Trees)

Uma árvore de decisão é uma representação em forma de grafos não cíclico. Em que são apresentados alguns níveis hierárquicos nessa árvore. O primeiro deles é a raiz, e derivando dela, aparecem as folhas. Nota-se que um nó-pai origina dois nós-filhos, baseados sempre em uma “regra” ou apontamento sobre determinado descrito químico que esteja analisando.

Figura 20 - Esquematização de uma árvore de decisão.



As árvores de decisão dividem o espaço de características em regiões. Cada região é associada a uma classe específica (no caso da classificação) ou a um valor (no caso da regressão). As árvores de decisão são fáceis de entender e interpretar, mas podem ser propensas a overfitting. Por isso existe a possibilidade de estabelecer critérios de parada para o algoritmo. Caso contrário, ao invés de aprender com os dados químicos disponíveis, ele irá decorar os padrões, e isso não é algo interessante para um modelo que precisa ser o mais generalista possível.

O gráfico contemplado na Figura 20 representa o funcionamento de uma árvore de decisão, que pode ser utilizada para tarefas de classificação e regressão também. No gráfico, os nós da árvore de decisão são representados por círculos de diferentes cores. O primeiro círculo, localizado no topo, representa o nó raiz da árvore, enquanto os círculos menores abaixo representam os nós de decisão subsequentes. Cada nó de decisão é rotulado com uma condição que determina a bifurcação do caminho na árvore.

As linhas conectando os nós representam as decisões tomadas com base nas características dos dados. Por exemplo, se a condição “X <= 0.5” for verdadeira, o caminho da esquerda é seguido, enquanto se for falsa, o caminho da direita é seguido. Essas linhas conectam os nós de acordo com as regras de divisão definidas durante o treinamento da árvore.

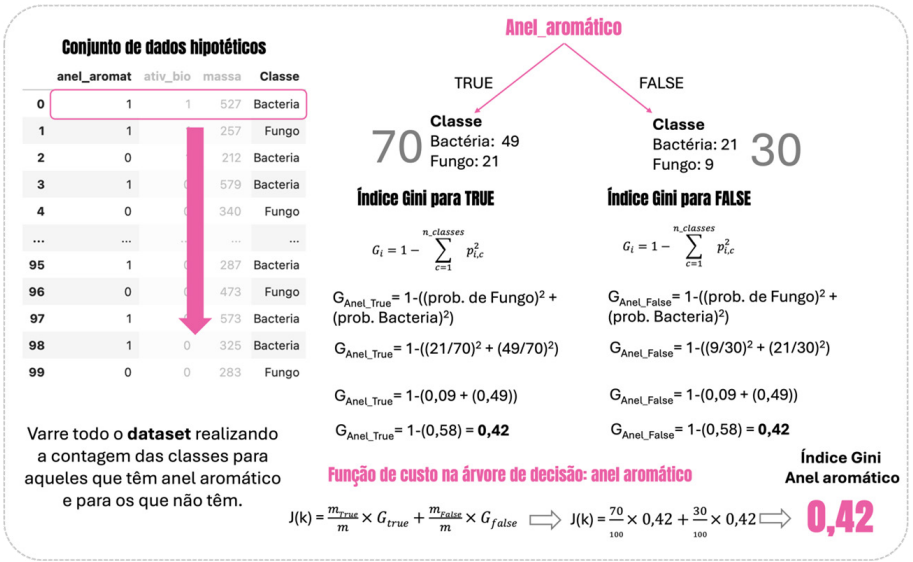
As folhas da árvore, representadas por círculos coloridos em azul ou rosa, representam as classes de saída ou os valores de previsão. Cada folha é rotulada com a classe ou valor correspondente.

Por exemplo, considere nosso problema de classificação em que estamos classificando amostras em duas classes: “Moléculas de Fungos” e “ Moléculas de Bactérias “. A árvore de decisão aprendeu a dividir as amostras com base em uma condição inicial, em seguida, com base em outras condições subsequentes para chegar à classificação final.

Tomaremos um exemplo hipotético em que existam 100 instâncias e três descritores químicos associados a elas, e que cada uma dessas instâncias esteja rotulada como pertencente à classe “Fungo” ou à classe “Bactéria”. Como features desse conjunto de instâncias, há dois descritores que são categóricos, o primeiro sinaliza a existência ou não de anel aromático na estrutura, o outro, faz um apontamento se é um bioativo ou não. O terceiro descritor é numérico, e faz a representação da massa molecular do composto. Dessa forma, para criar uma árvore de decisão adequada, precisamos estipular o tipo de função de custo que estará associada a essa classificação. Para isso, será utilizado o índice Gini, que é uma medida de impureza ou heterogeneidade dos dados. Assim, isso será apresentado como é realizado esse cálculo, com o intuito de fazer com que determinado nó da árvore seja criado com base em um índice Gini mínimo, ou seja, maximizando a pureza dos grupos resultantes.

A Figura 20 é uma representação genérica do que seria uma árvore de decisão. Mas qual é o grau hierárquico entre os descritores utilizados? Quem seria o escolhido para iniciar o crescimento da árvore, ou seja, ser o nó-raiz? Para isso, será calculado o índice Gini para todos os descritores, dessa forma, aquele que apresentar o menor índice Gini será selecionado como o nó-raiz. Veja o conceito abaixo, na Figura 21.

Figura 21 - Cálculo do índice Gini para o descritor anel aromático.

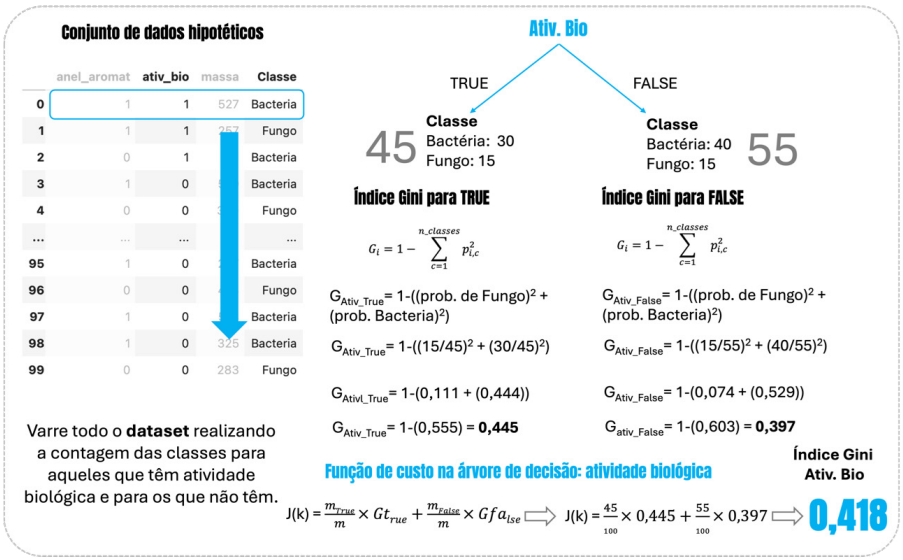


Verifiquem que para efetuar o cálculo de um descritor categórico, como o anel aromático, é feita toda a varredura do dataset realizando a contagem em dois blocos: True ou False. Ou seja, para aquelas instâncias que apresentam anel aromático, será computado quantas são pertencentes à classe dos fungos e quantas são de origem bacteriana; o mesmo é feito para aquelas amostras que não apresentam anel aromático, sendo direcionadas ao agrupamento False.

Notem que existem 70 instâncias, das 100 inicialmente analisadas, que apresentam anel aromático em sua constituição química, e nesse grupo 49 são de origem bacteriana e 21, originárias de fungo. O cálculo de índice Gini utiliza a probabilidade ao quadrado de cada classe aparecer, realizando uma somatória dos resultados e subtraindo de 1. Assim, foram obtidos os valores de 0,42 tanto para o direcionamento TRUE, quanto para o False. De posse dessas informações é possível, então, calcular a função de custo associada ao descritor anel aromático realizando uma média ponderada dos índices Gini de cada bloco, obtendo, portanto, o índice Gini de 0,42 quando se utiliza o descritor de anel aromático para separar os dados. Vale salientar que o índice de Gini é um grau de impureza dos dados. Ou seja, quanto menor, mais puro, melhor para classificar (separar) os dados. Para decidir qual é o nó-raiz, é necessário escolher o descrito que gere o menor índice de Gini.

A Figura 22 sintetiza o cálculo Gini para o outro descritor categórico, agora associado à atividade biológica.

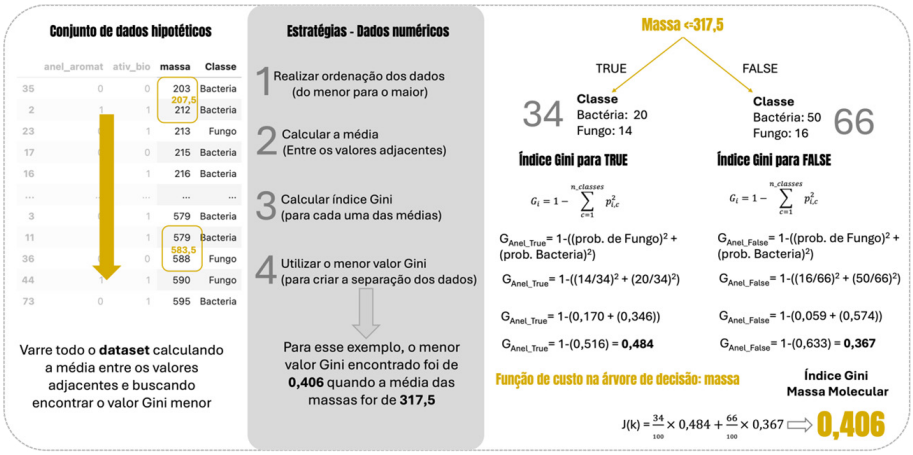
Figura 22 - Cálculo do índice de Gini para o descritor atividade biológica.



Na comparação entre o grau de impureza do entre os dois descritores já calculados, nota-se que existe mais impureza quando é utilizado o descritor associado a ter ou não anéis aromáticos na molécula, com um índice Gini 0,42 comparado com 0,418 do Gini relacionado à atividade biológica. Apesar de valores próximos, nesse caso, a atividade biológica gera menor impureza, ou seja, é mais puro, quando comparada com o primeiro descritor.

Porém, ainda não é possível decidir quem seria o nó-raiz para esse problema hipotético, uma vez que ainda falta outro descritor, o de massa molecular. Contudo, por se tratar de um descritor numérico, será necessário criar uma estratégia, de modo que seja encontrado um ponto de corte que consiga gerar um valor Gini menor. Analise na Figura 23 a estratégia para se tratar um dado numérico e relacioná-lo com o índice Gini.

Figura 23 - Cálculo do índice de Gini para o descritor massa molecular.



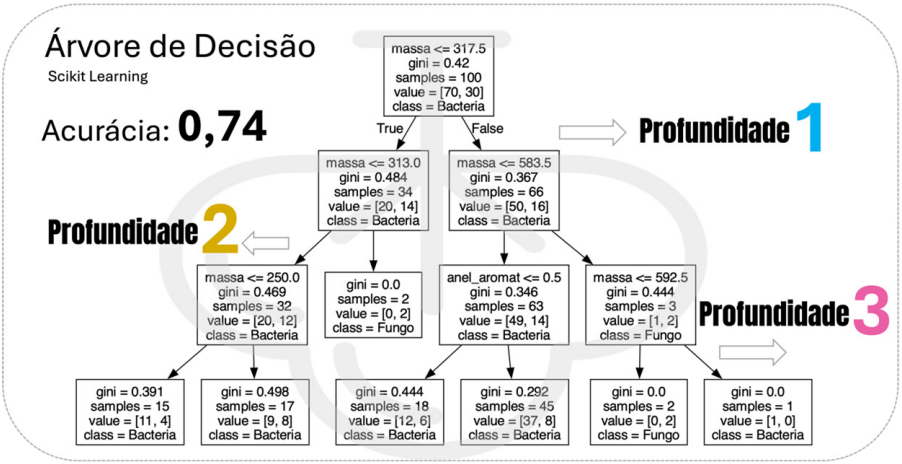
Existe uma diferença significativa para calcular o índice de Gini para um descritor que seja numérico. Para isso, é aplicada uma estratégia para que seja encontrado um ponto de corte ideal desses valores que faça com que o grau de pureza dos dados seja maior, ou seja, que seja encontrado um Gini menor. No exemplo explorado, o descritor de massa molecular, por ser numérico, foi inicialmente ordenado, do menor para o maior número. Assim, calculou-se a média entre cada valor adjacente e assim, estabeleceu-se um critério de separação dos dados, considerando valores menores ou iguais àquele limiar estabelecido. É feito o cálculo de Gini completo para todas as médias, e a menor delas é encontrada e utilizada, então, como o ponto de corte dos valores.

Dessa forma, quando se utiliza o valor de 317,5 para as massas moleculares, obtém-se uma separação de 34 instâncias que atendem a esse critério (sendo 20 de origem bacteriana e 14, fúngica) e o restante, 66, não atendem ao critério, subdividindo-se em 50 de origem bacteriana e o restante, originárias de fungos. Com isso, obteve-se o valor do índice Gini de 0,406, o menor entre os três descritores, por esse motivo, a massa molecular, utilizando o corte de 317,5 será o nó-raiz da árvore de decisão.

E assim, vão sendo aplicados todos esses critérios, de modo que a árvore cresça, gerando nós-filhos. É possível determinar a profundidade que essa árvore terá. O algoritmo de árvore de decisão, apesar de eficiente na maioria dos casos, pode se sobreajustar aos dados, ou seja, decorar os

padrões ao invés de aprendê-los, perdendo a capacidade de generalização. Muitas vezes, é adequado realizar uma restrição do tamanho da árvore, permitindo que ela tenha mais erros, mas que seja mais generalista. A Figura 24 mostra a árvore de decisão criada a partir dos dados hipotéticos apresentados, estabelecendo um critério de profundidade igual a três, fazendo com que a acurácia seja de 0,74.

Figura 24 - Árvore de decisão com profundidade 3.



Para acrescentar um breve resumo sobre as curiosas árvores de decisão, notem que o gráfico ilustra como uma árvore de decisão divide iterativamente o espaço de características com base em condições de divisão para realizar a classificação de amostras em diferentes classes (ou valores de saída, caso o seu estudo almeje isso). Essa abordagem é útil para entender como as decisões serão tomadas em diferentes etapas do processo de classificação de moléculas orgânicas.

ÁRVORES DE DECISÃO

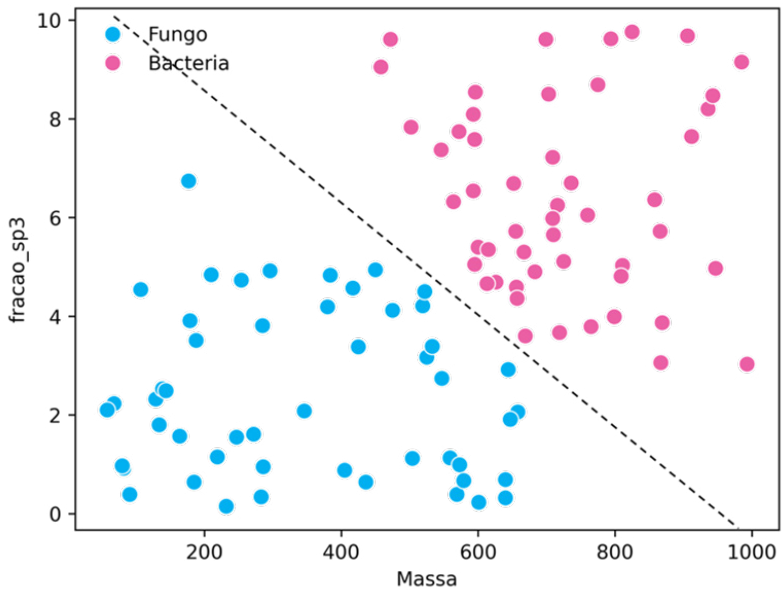
A Árvore de Decisão é um algoritmo de classificação que organiza os dados em uma estrutura hierárquica de decisões, semelhante a um fluxograma. A cada etapa, o modelo realiza divisões com base nas características das amostras, buscando separar os dados em grupos cada vez mais homogêneos. Essas divisões são feitas de forma a maximizar a pureza das classes em cada nó. Ao final do processo, cada caminho da raiz até uma folha representa uma regra de decisão, permitindo classificar novas amostras de forma interpretável e intuitiva.

SVM (Support Vector Machine) – Linear

Vamos considerar um exemplo hipotético para analisar dados químicos. Neste caso, queremos continuar realizando um reconhecimento de padrões capazes de separar as classes de moléculas oriundas de fungos e de bactérias. Para o Support Vector Machine – SVM, tais classes são expressas como -1 e +1.

A título de exemplificação didática, será proposto um conjunto de dados com descritores químicos, a massa molecular e a fração de carbonos que estão hibridizados em sp^3. A Figura 25 faz a dispersão dos dados utilizando esses dois descritores.

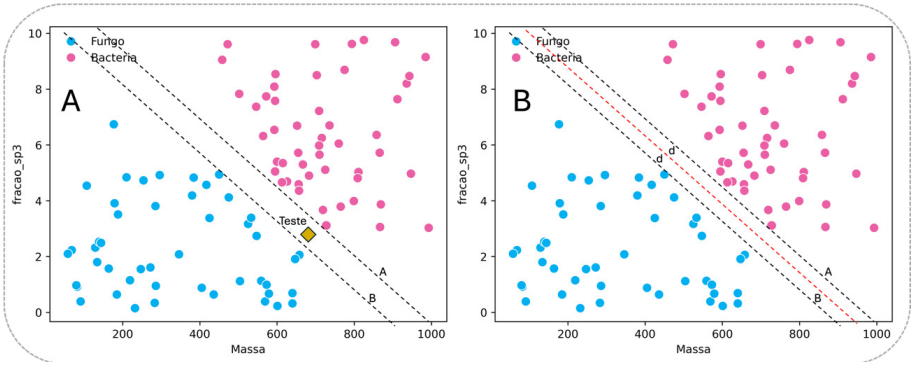
Figura 25 - Separação linear dos dados.



Como mostrado na Figura 25, trata-se de um problema linearmente separável. Diante de múltiplas retas capazes de separar os dados, surge uma questão fundamental: qual hiperplano deve ser escolhido? Suponha que seja utilizado o hiperplano apresentado. Ele consegue separar as duas classes corretamente. Em que todas as instâncias que estão acima da reta pertencem à classe positiva (bactérias), e as que estão abaixo, no grupamento azul, são da classe de fungos. Mas qual o problema de utilizar esse hiperplano específico?

Imagine que haja uma amostra que será testada (sinalizada pelo losango dourado na Figura 26-A). Se definido o hiperplano A, ela será classificada como pertencente à classe azul, ou seja, de origem fúngica, mas se for utilizado o hiperplano B, ela será classificada como de origem bacteriana. Há um problema claro de overfitting, quaisquer que sejam os hiperplanos, ajustam-se em demasia em relação aos dados. Ou seja, tais hiperplanos não parecem ser muito interessantes, pois pode acontecer de uma amostra muito próxima da classe negativa ser classificada como positiva. Então seria necessário buscar uma estratégia de encontrar o melhor hiperplano!

Figura 26 - Determinação de hiperplanos adequados à separação de dados.



Uma proposta bastante óbvia é inserir um hiperplano que esteja bem no centro de dois hiperplanos propostos anteriormente, assim, existiria uma margem de segurança para a classificação, como mostrado na Figura 26-B. Configurando em situações como mostradas acima. Na existência de outras amostras de testes, ela poderia classificá-las corretamente. Então, garante-se que haja uma margem de flexibilidade (chamada de “d”) no momento da classificação, mantendo as instâncias do lado classificatório correto. Justamente é essa a ideia do algoritmo de SVM, atribuir uma margem, uma distância entre as classes, e é chamado de classificador de classe máxima.

O hiperplano de decisão é chamado de decision boundary. Essa zona de separabilidade entre as classes é chamada de margem e a ideia do SVM é que o plano esteja exatamente no meio. A distância da margem até a borda da instância mais próxima será exatamente igual para ambas as classes, sendo representada pela letra d. Fazendo uma analogia: qual seria a rua reta mais larga possível para separar esses dois grupos de pessoas? A maior largura possível é justamente para estabelecer essa margem, para não ficar pendente de nenhum dos lados.

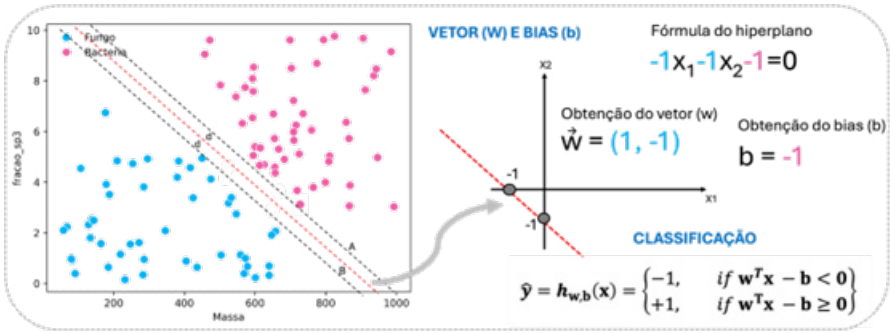
A margem é definida com as amostras que estão na borda dos grupos, ou seja, as amostras que estão nas bordas são candidatas a construir essa margem de separação, e essas instâncias que estão nas bordas são chamadas de vetores de suporte, daí o nome: Support Vector Machine. Afinal de contas, em uma abordagem mais aprofundada, todos os pontos são vetores, ou seja, em uma dispersão de dados 2D, cada ponto pode ser representado com um vetor que sai da origem do plano cartesiano.

Uma vez que foi definida qual é a margem para construção da decision boundary, utilizando somente as amostras candidatas que estariam nas bordas do grupo, as demais amostras contidas no interior da região não contribuem para a criação da margem. Se adicionasse ou removessem amostras que não fossem candidatas à margem, não haveria uma interferência na construção do hiperplano. O contrário também é válido, se fossem adicionadas mais instâncias, e estas não fossem candidatas à margem, não haveria influência no processo classificatório. As grandes estrelas do SVM, são as amostras que estão na borda, e que definem, portanto, a margem.

É importante destacar que o SVM é sensível às escalas dos descritores, então, é importante utilizar uma técnica de normalização dos dados para deixá-las na mesma dimensão.

Para a construção dos modelos, a proposta é que haja um w para a representação dos vetores (w_1 e w_2) e assim, relacioná-los com o valor de x (x_1 e x_2), além do bias (b).

Figura 27 - Encontrando os valores vetoriais e bias pelo algoritmo SVM.



Uma vez que forem encontrados os pesos que definirão um hiperplano, é possível encontrar o valor de w como sendo um vetor normal a este hiperplano. Vale reforçar que um vetor normal ao plano é um vetor que forma um ângulo de 90°. O bias (o intercepto do hiperplano, ou seja, -1) é o deslocamento no sentido do hiperplano. Em resumo, utiliza-se $w^T x+b=0$ para definir o hiperplano. Assim, a margem superior será dada por $w^T x+b=+1$ e a inferior: $w^T x+b=$-1.

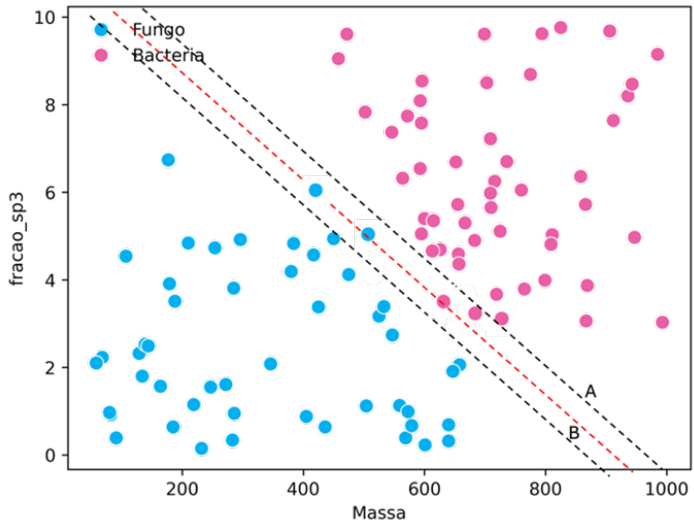
Logo, todas as instâncias que estiverem na direção do vetor w serão classificadas como pertencentes à classe positiva. E aquelas que estiverem opostas, classificadas como negativas.

A classificação empregada nos modelos de SVM se baseia em um conceito fundamental: o produto interno. Imagine um ponto a ser classificado com valores x_1 e x_2. Este ponto é multiplicado pelo vetor de pesos (w), resultando em um valor escalar. Ao subtrair o viés (bias), se o resultado for negativo, indica que o ponto pertence à classe negativa (ou seja, está do lado oposto do hiperplano), enquanto um valor não negativo sugere que o ponto pertence à classe positiva. Assim, o resultado reflete a distância da amostra em relação ao hiperplano. Essa abordagem, conhecida como Classificador de Margem Rígida (Hard Margin Classifier), não tolera erros de classificação durante o treinamento.

No entanto, quais são suas limitações? Caso um dos grupos apresente um outlier, ele faria com que as margens fossem muito estreitas e, portanto, o hiperplano gerado poderia causar muitos erros nas classificações futuras, pois o limiar é muito pequeno, causando um overfitting nos dados. Para contornar essa situação, pode-se permitir alguns erros de classificação, fazendo com que o modelo seja menos sensível a outliers.

A Figura 28 mostra que o problema não é separável linearmente. Mas se fosse sugerido um hiperplano como mostrado em vermelho, erra-se a classificação de algumas amostras, mas as separa. Caso fosse utilizado um algoritmo de bordas rígidas, não seria possível encontrar esse hiperplano.

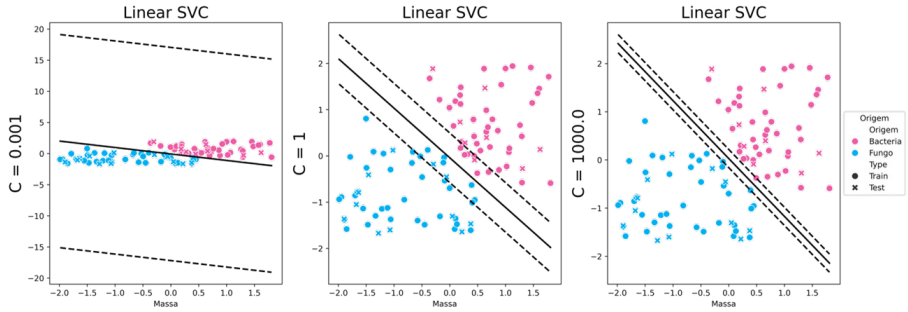
Figura 28 - Representação de instâncias entre as margens. Motivação para flexibilização do algoritmo.



Nesse cenário, há uma consideração de um certo limite de erro. As margens são relaxadas para permitir erros, conforme ilustrado na Figura 29, resultando no aumento do bias. Um aumento no erro do bias resulta na redução do erro de variância, tornando o modelo mais genérico. Essa abordagem tende a mitigar o overfitting. Permitir que o algoritmo cometa erros tem como objetivo manter uma margem ampla, embora limitada, permitindo algumas violações dessa margem, ou seja, erros de classificação. Essa estratégia é conhecida como regularização, introduzindo uma penalização para permitir que o algoritmo cometa alguns erros de classificação. Essa versão do SVM é denominada Classificador de Margem Flexível (Soft Margin Classifier), e é a versão que utilizaremos em nossa prática de classificação de moléculas orgânicas.

Ao programar um algoritmo SVM linear, nos deparamos com o parâmetro “C”, que desempenha um papel crucial na determinação da rigidez das margens de decisão. Quanto menor o valor de “C”, mais amplas são as margens, o que significa que o classificador é mais flexível e tolerante a erros. Nesse contexto, o algoritmo penaliza os erros de classificação de forma leve. A Figura 29 faz uma visualização de como o parâmetro “C” controla o equilíbrio entre a complexidade do modelo e a capacidade de generalização, sendo uma ferramenta essencial para ajustar o desempenho do SVM aos requisitos específicos do problema.

Figura 29 - Exemplos de utilização de regulação no algoritmo de SVM.



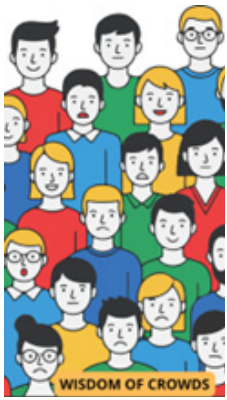
Note que um valor muito pequeno deixa as amostras todas dentro da margem, ao aumentar o valor de C por cem vezes, é possível separar as classes, e algumas instâncias ainda causam confusão na classificação. Aumentando ainda mais o valor de C para 1000, as margens ficam mais rígidas, podendo chegar ao valor inicial e apresentar overfitting.

SUPPORT VECTOR MACHINE
O Support Vector Machine (SVM) é um algoritmo de classificação que busca encontrar o hiperplano que maximiza a margem entre duas classes. As amostras mais próximas do hiperplano são chamadas de vetores de suporte e são fundamentais na definição da fronteira de decisão.

Ensemble Learning

Uma maneira interessante de começar a explorar o conceito de ensemble learning é por meio da ideia de “sabedoria da multidão”, também conhecida pelo termo em inglês “Wisdom of Crowds”. Esse conceito sugere que um grupo grande e diversificado de pessoas, quando reunido, pode tomar decisões melhores ou mais sábias do que especialistas individuais. A origem dessa ideia remonta ao famoso livro “The Wisdom of Crowds”, de James Surowiecki (2004). A essência é que, ao tomar uma decisão ou buscar uma resposta, a combinação dos conhecimentos e opiniões de diversas pessoas tende a ser mais precisa e confiável do que a opinião de um único especialista.

Para que essa “sabedoria coletiva” se manifeste, algumas características são necessárias:



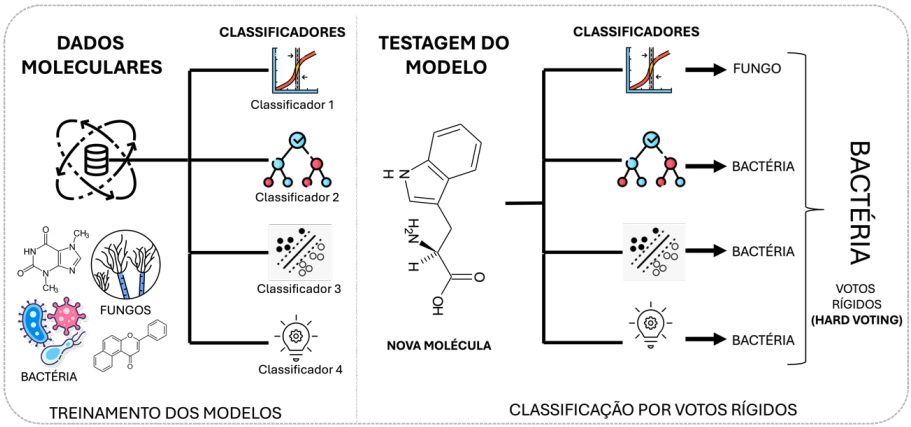
- diversidade de opiniões;
- a opinião de uma pessoa deve se manter independente das pessoas ao redor (não ser influenciada por outras pessoas);
- qualquer um da multidão deveria estar apto a fazer suas conclusões, baseado na sua experiencia individual;
- a multidão deve estar apta a agregar essas opiniões individuais desses membros para tentar chegar a uma decisão única coletiva.

A ideia por trás do ensemble learning é bastante semelhante ao conceito discutido anteriormente. Imagine que temos um grande conjunto de dados e treinamos um classificador, como uma regressão logística. Em seguida, usamos os mesmos dados para treinar outro algoritmo com características diferentes, como uma árvore de decisão. Depois, aplicamos outro algoritmo, como o SVM, e assim por diante, criando um conjunto de classificadores com naturezas diversas. Cada treinamento é realizado de forma independente, ou seja, um classificador não influencia o outro, mas todos usam os mesmos dados como fonte.

Esse conjunto de classificadores é chamado de ensemble. A ideia por trás dessa abordagem é que cada classificador fornece uma resposta individualmente, mas ao final, suas previsões são agregadas para formar um aprendiz mais robusto, conhecido como Strong Learner.

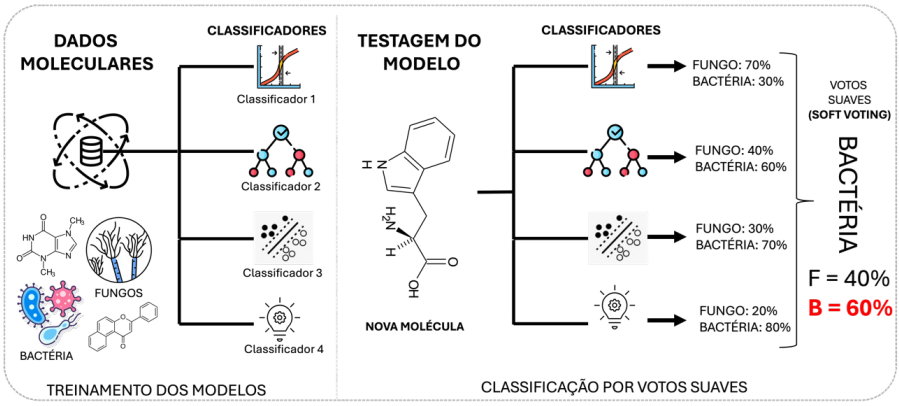
Uma proposta de iniciar a potencialidade do método de ensemble learning é através de uma votação rígida (Hard Voting Classifier). Vamos supor que seja considerado o nosso problema classificatório de reconhecer padrões em moléculas oriundas de fungos e de bactérias, e em cima disso, usa-se vários classificadores, que podem ser treinados em paralelo. Após o treinamento de todos os classificadores, surge uma nova molécula que não estava no conjunto de dados e queremos classificá-la, assim, cada um dos classificadores irá rotulá-la e definir a classe pertencente a essa nova instância de acordo com o voto majoritário dos classificadores. A classe que mais aparecer será aquela considerada no final. Essa abordagem está esquematizada na Figura 30.

Figura 30 - Votação por votos rígidos (Hard Voting Classifier).



Se todos os classificadores inseridos no ensemble fossem aptos a estimar a probabilidade de classes pode-se usar uma estratégia diferente, chamada de votação suave (Soft Voting). Porém, nesse caso, será utilizada a média das probabilidades de cada uma das classes possíveis, e assim, utilizar o valor máximo das probabilidades preditas. A Figura 31 mostra uma representação esquemática desse caso.

Figura 31 - Votação por votos suaves (Soft Voting Classifier).



Vale ressaltar que usaremos a biblioteca Scikit-Learn para realizar as classificações, e essa abordagem Soft Margin só será permitida se o algoritmo utilizado estiver implementado o método predict_proba(), caso contrário, essa ideia não irá funcionar.

Cada classificador no ensemble é considerado um aprendiz fraco, pois individualmente pode não ser muito preciso, mas anda é melhor do que um palpite aleatório. No entanto, quando combinamos o conhecimento de todos os classificadores, tendemos a obter um modelo mais poderoso.

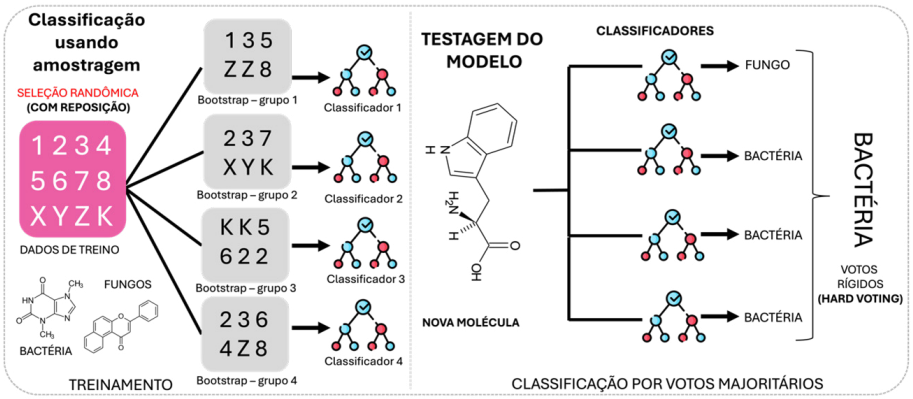
Agregação por bootstrap

Esta é uma estratégia de ensemble que emprega o mesmo algoritmo de treinamento para cada classificador. Os classificadores são treinados em diferentes subconjuntos de dados de treinamento, obtidos aleatoriamente, utilizando a estratégia de bootstrapping (reamostragem com reposição).

Suponha que tenhamos um conjunto hipotético de instâncias relacionadas a moléculas de fungos e bactérias. Podemos ter k classificadores, e então é criado um subconjunto com substituição. Note que, no primeiro bootstrap (grupo 1 – Figura 32), uma mesma amostra (a amostra Z) aparece duas vezes; já no próximo, nenhuma amostra é repetida; nos subconjuntos subsequentes, outras amostras podem aparecer duas vezes, e assim por diante. Dessa forma, são gerados novos subconjuntos que servirão como dados de treinamento para o classificador, como uma árvore de decisão, por exemplo.

A ideia é usar um valor *k* muito alto para que haja muitos subconjuntos diferentes, e cada subconjunto contenha amostras diferentes, gerando assim estimadores com características distintas. Ao fazer isso, aumentamos a diversidade dos estimadores. Quando uma nova amostra é testada, todos os classificadores atribuem um rótulo, e por meio de uma votação majoritária, a nova molécula pode ser classificada como pertencente a uma das classes propostas.

Figura 32 - Exemplificação de funcionamento de montagem de subconjuntos por amostragem (bagging).



Cada classificador treinado individualmente tende a ter um viés um pouco maior do que se ele fosse treinado no conjunto inicial inteiro. Então aumenta-se o erro durante o treinamento, para que o modelo fique mais ge-

neralista, diminuindo a variância. A técnica de bagging pode utilizar a técnica de soft voting também, utilizando a probabilidade média, considerando todas as classificações.

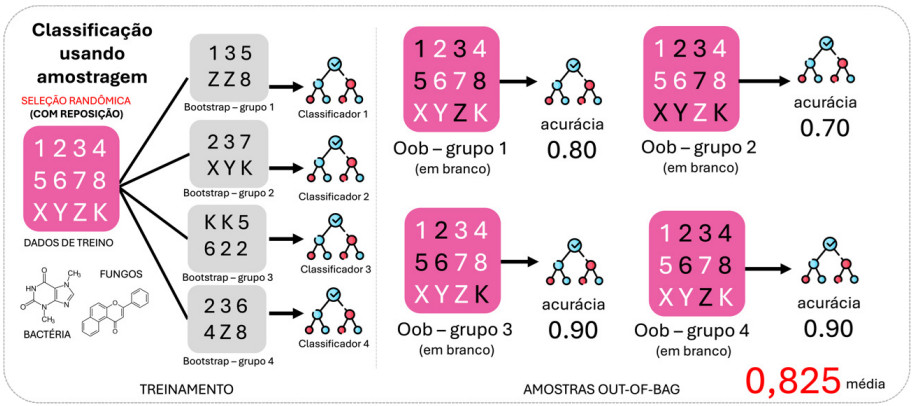
Observe que todas essas técnicas estão fundamentadas no mesmo framework básico. Embora cada uma delas possa se especializar em uma área específica, todas compartilham uma base matemática comum.

Vale enfatizar no nosso estudo de que não há garantias de que a abordagem de reconhecimento de padrões utilizando bagging terá uma performance melhor do que um classificador rodando com todas as amostras. Tudo depende dos hiperparâmetros definidos. É necessário muito testagem. Você pode ter muitas intuições, mas, na prática, quanto mais testar as ideias, melhor.

Bagging: avaliação "out-of-bag" (fora do sacola)

No exemplo anterior, representado na Figura 32, é possível que algumas instâncias não sejam amostradas para nenhum classificador, ou seja, não façam parte de nenhum conjunto de treinamento do bootstrap. Essas amostras não selecionadas são chamadas de out-of-bag (oob). Importante ressaltar que as amostras não selecionadas não são as mesmas para cada conjunto de bootstrap. Cada conjunto de bootstrap seleciona um conjunto de amostras e deixa outro conjunto de fora. Esse conjunto de amostras que ficou de fora, chamado de oob, será diferente para cada bootstrap realizado. Mas, então, qual é a utilidade dessas instâncias que ficaram de fora? Que tal utilizá-las como uma métrica de validação? Seria uma boa ideia?

Figura 33 – Exemplificação de funcionamento da amostragem "out-of-bag".



Então uma vez que o classificador nunca viu as instâncias desse grupo oob, ele pode ser testado em cima dessas instâncias, sem a necessidade de validação separada. E com essa validação pode-se verificar os indícios do modelo para amostras nunca vistas, se está tendo overfitting ou não.

Técnica de empilhamento (Pasting).

Existe uma outra técnica de ensemble chamada Pasting, que é exatamente a mesma estratégia do bagging, exceto que ele trabalha sem reposição das amostras no "sorteio" de instâncias para bootstrap. Usa-se o mesmo código para bootstrap com reposição, porém, basta fazer a alteração do hiperparâmetro "bootstrap" para **False**, fazendo isso, você está garantindo que você irá fazer amostras sem recolocação, ignorando a técnica de bootstrap.

Na prática mudar os subconjuntos gerados gerará uma redundância ou não? No geral, o bootstrapping introduz um pouco mais de diversidade nos subconjuntos em cada classificador. Pode-se repetir um pouco de amostras, mas as outras que sobrarem vão ser mais diversas. Dessa maneira, o bagging acaba tendo um viés um pouco maior do que o pasting, o que vai garantir melhor generalização. Mas essa diversidade extra significa que os classificadores acabam sendo menos correlatos e consequentemente pode acontecer da variância ser reduzida.

De maneira geral, o bagging resulta em modelos melhores e geralmente é preferido pela comunidade. Entretanto, se você tem tempo de sobra e poder computacional: teste e veja o que acontece com seus dados. Estudando todos esses métodos você terá um indicativo do que fazer, por quê esses erros podem acontecer ou não, o que pode ser feito para melhorar um dado modelo?

Subespaços e patches aleatórios – Random subspaces e random patches.

Outra técnica interessante de ensemble learning é a amostragem de descritores. Em vez de o classificador usar, por exemplo, 100 características, poderíamos criar um conjunto selecionando apenas algumas características (features) específicas. Geralmente, algumas features são menos importantes que outras e, eventualmente, um subconjunto de features pode ser mais eficaz do que usar todas juntas.

Assim, em vez de apenas amostrar as instâncias, por que não amostrar também as features? O objetivo dessas técnicas é obter um classificador ainda mais diversificado, trocando um pouco de viés por uma redução na variância.

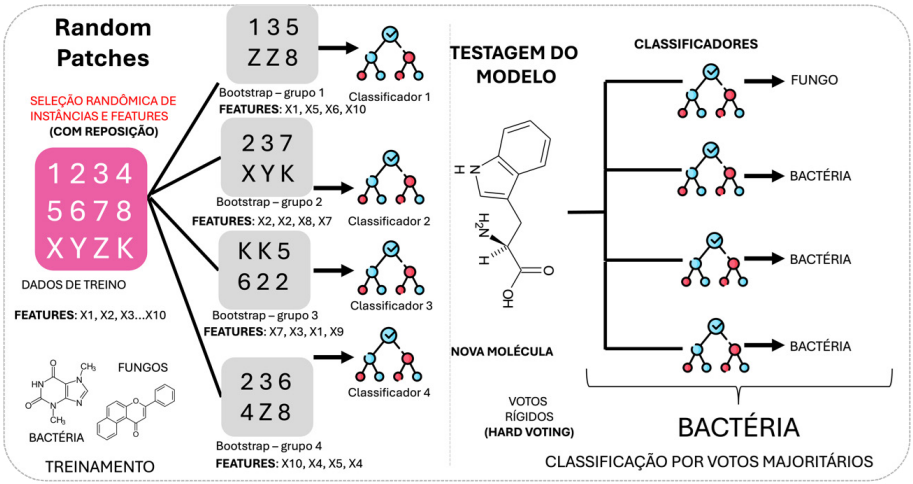
Na prática, usaremos o mesmo algoritmo várias vezes, com diferentes conjuntos de features, na esperança de que um subconjunto se destaque e produza um bom resultado de classificação. Portanto, estamos tentando, de maneira aleatória e exploratória, aproveitar a aleatoriedade e o grande número de possibilidades para alcançar um bom resultado. Ao utilizar técnicas de ensemble learning, eliminamos a necessidade de selecionar manualmente as melhores features ou de nos preocuparmos em como construir nosso modelo. A ideia é gerar muitas possibilidades aleatórias e esperar que, por acaso, tenhamos um bom resultado. Como o acaso muitas vezes funciona melhor do que um simples palpite, ao combinar várias tentativas, esperamos obter um resultado sólido.

Portanto, a ideia é introduzir variabilidade na escolha dos classificadores, o que pode ser especialmente útil ao lidar com conjuntos de dados de alta dimensão. Em vez de usar todas as features disponíveis, podemos criar subconjuntos menores e testar várias combinações, na esperança de encontrar uma que funcione bem.

Se for mantido todo o conjunto de treinamento intacto, mas amostrarmos as features, isso é chamado de método de subespaços aleatórios (random subspaces). Se fizermos amostragens tanto no conjunto de treinamento quanto nas features, isso é chamado de método de patches aleatórios (random patches).

No caso dos patches aleatórios, vamos supor que tenhamos um conjunto de dados moleculares (de fungos e bactérias) com 10 descritores físico-químicos. Ao treinar o classificador, primeiro usamos uma amostragem aleatória para obter o bootstrap, configurando os hiperparâmetros para obter o número desejado de features. No método de patches aleatórios, geralmente permitimos a reposição, mesmo que isso resulte em redundância na escolha das features. Por exemplo, em vez de representar as amostras com os 10 descritores originais, podemos selecionar, randomicamente, apenas quatro, demonstrado na Figura 34. Assim, representaremos o mesmo problema de várias maneiras diferentes, aumentando significativamente a variabilidade. A classificação final para uma nova amostra segue o mesmo critério das outras abordagens: o voto majoritário.

Figura 34 - Exemplificação de funcionamento da amostragem patches aleatórios.



Florestas Aleatórias – Random Forest

As florestas aleatórias (Random forest) são uma extensão das árvores de decisão, um método que, embora poderoso em muitos aspectos, pode enfrentar desafios de precisão devido ao potencial de overfitting, especialmente com árvores profundas. Uma solução promissora para esse problema é adaptar o conceito de bagging, que consiste em combinar múltiplas árvores de decisão diversificadas e flexíveis em um conjunto, resultando em melhorias significativas na precisão do modelo. Essa abordagem tem se destacado particularmente na análise de dados de moléculas orgânicas.

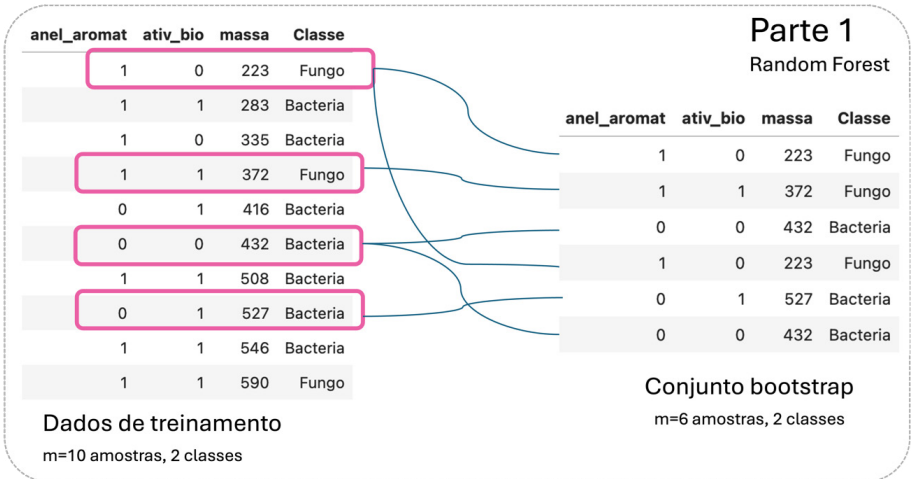
Ao aplicar essas técnicas, um conjunto de árvores é construído, com cada uma delas sendo treinada em um subconjunto aleatório dos dados. A combinação dessas árvores em um único modelo, através do bagging, cria o que é conhecido como uma floresta, daí o termo “floresta aleatória”. Essa técnica aproveita a aleatoriedade para explorar diferentes aspectos dos dados e mitigar os efeitos prejudiciais do overfitting, resultando em modelos mais robustos e generalizáveis.

Vamos supor que a floresta tenha *k* árvores, e o *k* tende a ser um número alto, dezenas de milhares, talvez. Será feita uma iteração de 0 até k-1, e para cada classificador será criado um conjunto bootstrap para treinar o classificador, e o segundo passo, treinar uma decision tree usando o bootstrap

set (do jeito feito pelo bagging), mas aqui tem o conceito especial da random forest, que é adicionar um pouco mais de aleatoriedade. O treinamento será feito, mas será utilizado apenas um subconjunto de features, sem reposição (algo muito parecido com random patches – que tem amostragem tanto nas instâncias quanto nas features, só que patches é feito com reposição), porém essa amostragem será feita em cada um dos passos do treinamento da árvore. Iremos considerar que cada nó terá um subconjunto próprio de features, e decidir qual será a melhor feature para aquele nó, repetindo o mesmo processo para os demais, isso é chamado de feature bagging. Note que em random forest, está sendo inserido aleatoriedade no momento do treinamento do algoritmo.

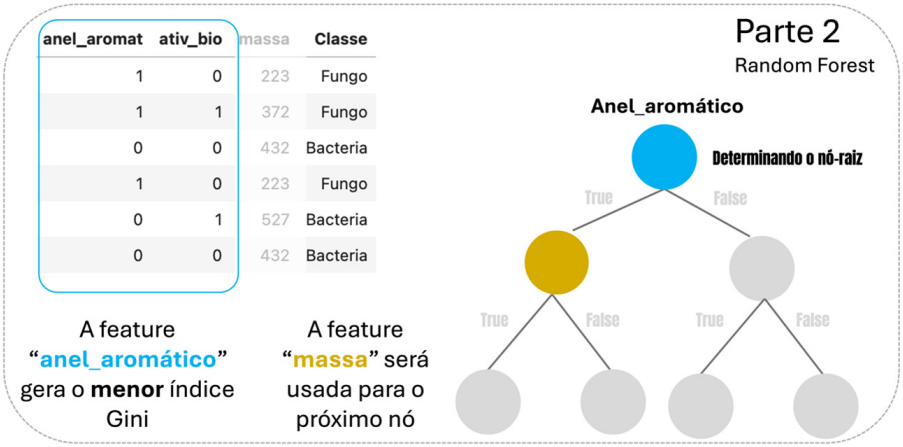
De maneira sistemática, a título de exemplificação, considere um conjunto de dados que gostaria de analisar perante o algoritmo de random forest, inicialmente deverá criar um bootstrap set, em cima desse conjunto, e assim, treinar a árvore com algumas peculiaridades como serão mostradas na Figura 35 a seguir.

Figura 35 - Criação do conjunto bootstrap para construção das árvores das florestas aleatórias.



O segundo passo é realizar a árvore de decisão. Recordam-se que a raiz de uma árvore é aquela que apresenta o menor grau de impureza? O mesmo conceito se mantém nesse caso. Porém, vamos supor que randomicamente, duas features serão sorteadas, sem reposição (verifique na Figura 36). Considere que foram selecionadas as features: anel_aromático e atividade_biológica. Entre as duas, qual geraria a menor impureza nos dados?

Figura 36 - Definição e construção das árvores de decisão das florestas aleatórias.

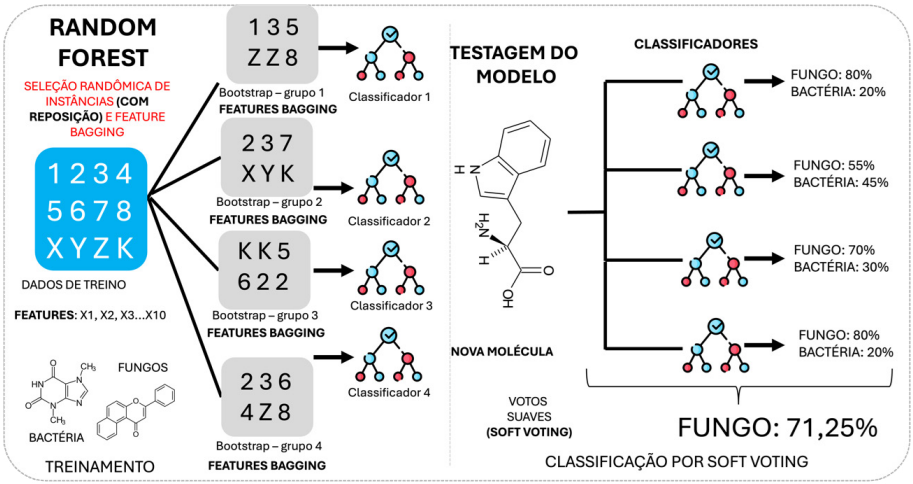


A descritor associado à presença ou não de anel aromático na molécula apresenta a menor impureza (menor índice Gini) quando comparado com o descrito de atividade biológica. Por este motivo, ele é escolhido como sendo o nó-raiz. A seguir, os demais nós serão montados seguindo esse mesmo critério, que no exemplo proposto, será a feature de massa molecular.

Uma feature pode ser randomicamente selecionada múltiplas vezes durante a etapa do feature bagging para fazer a separação em diferentes nós, verifique o esquema na Figura 37. Uma regra de ouro para definir a quantidade de features: se está sendo trabalhado com um problema de classificação, usa-se raiz quadrada do número de features, arredondado para baixo.

Note que além de pegar um conjunto diferente para cada árvore, para a construção de cada nó da árvore, usa-se também características diferentes. E isso gera uma aleatoriedade muito grande. É você ter tantas combinações possíveis, algumas até meio esdrúxulas, que na prática tende a ter um resultado melhor. A classificação é feita usando o mesmo critério do bagging (seja para *hard* voting ou soft voting). Por padrão o Scikit-Learn usa o soft voting.

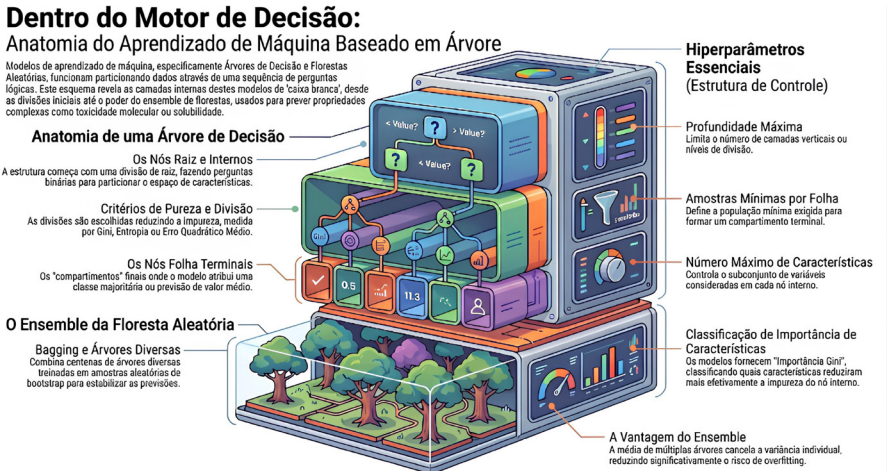
Figura 37 - Esquematização do funcionamento do algoritmo de random forest.



Uma característica interessante do Random Forest é sua capacidade de medir a importância de cada característica para resolver um determinado problema, um processo conhecido como seleção de características. Ele faz isso observando o quanto cada característica é usada nos nós da árvore e o quanto ela contribui para reduzir a incerteza na média.

Existem variações desse método, como o Extreme Trees, que, em resumo, não busca encontrar o limiar que resultará na menor incerteza; em vez disso, ele testa vários valores aleatórios. Há também o AdaBoost, GradientBoost e XGBoost, uma série de classificadores que se aprimoram mutuamente. Cada classificador atribui pesos às amostras, dando mais importância às amostras que foram classificadas incorretamente anteriormente, o que permite que o próximo classificador se concentre em corrigir esses erros.

Figura 38 - Esquematização do funcionamento do algoritmo de random forest.

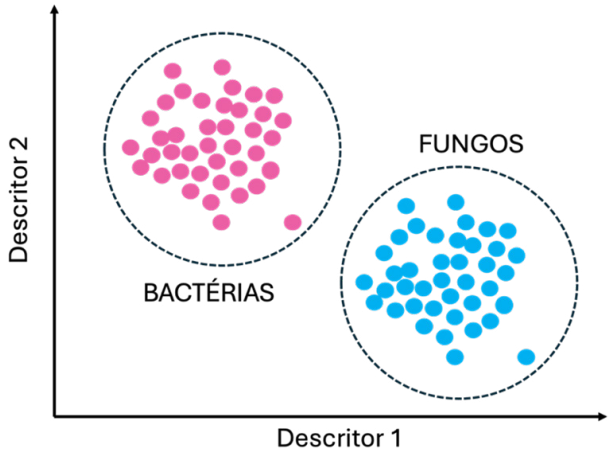


Fonte: imagem gerada pela inteligência artificial do Google Notebook LM.

Algoritmos não supervisionados.

Apesar de não ser o escopo desse livro, a título de curiosidade, o aprendizado não supervisionado ocorre sem o uso de rótulos ou anotações para orientar o modelo. Imagine que temos um conjunto de imagens, metade contendo moléculas de fungos e a outra metade de bactérias, porém não possuímos informações sobre quais imagens pertencem a cada categoria. Como podemos então diferenciar esses grupos? A solução está em utilizar um algoritmo capaz de extrair características significativas das imagens, criando um espaço onde essas características podem ser comparadas. Este espaço pode ser visualizado como um gráfico onde as características das imagens são representadas. Se as características das imagens de fungos e bactérias forem muito diferentes, é possível que o algoritmo consiga separar essas classes, como mostrado na Figura 39.

Figura 39 - Exemplo de clusterização usando algoritmos não supervisionados de machine learning.



Um extrator de características examinará cada imagem individualmente, com base em alguma medida de similaridade ou conceito de semelhança, procurando agrupar instâncias semelhantes. Se tivermos um algoritmo de agrupamento perfeito, como o de clustering, poderemos identificar precisamente os dois grupos desejados. No entanto, na realidade, as coisas não são tão simples como ilustrado na Figura 39.

No mundo real, é comum encontrar moléculas de fungos com características muito próximas às das bactérias. Quando isso ocorre, o sistema pode cometer erros de agrupamento, pois as características que estão sendo usadas para representar os dados não são capazes de separar completamente as diferentes classes. Nesses casos, as características não conseguem distinguir adequadamente as classes reais, tornando-se menos representativas.

Como existem muitos descritores químicos, talvez seja possível utilizar uma abordagem para lidar com esse problema, e reduzir a dimensionalidade dos dados, utilizando técnicas como Análise de Componentes Principais (PCA), que permite projetar os dados em um espaço de características de menor dimensão sem perder muita informação, tornando-os mais fáceis de visualizar e analisar.

Percebam, meus caros leitores, que para alguns de vocês que se aventuram ao isolamento ou à exploração de produtos naturais, podem ter visto potencial de utilizar o aprendizado de máquina para direcionar ou testar dados obtidos em laboratório. Agora chegou o momento de explorar adequadamente os dados moleculares e realizar a abordagem do aprendizado de máquina.

MODELAGEM E APRENDIZADO DE MÁQUINA

A partir de agora será explorado um conjunto de dados que contém informações de moléculas extraídas de cultivo fúngico ou de origem bacteriana. O objetivo é de verificar se existe uma distinção entre moléculas dessas duas classes, ou se há confusão no momento de reconhecer padrões entre tais fontes moleculares. No entanto, em algumas situações, pode não haver diferença significativa entre as moléculas de fungos e bactérias. Por exemplo, em estudos de metabolômica ou análise de compostos químicos presentes em organismos, as moléculas podem ser agrupadas com base em suas características químicas ou estruturais, independentemente da sua origem microbiana específica. Nesses casos, a distinção entre fungos e bactérias pode ser menos relevante do que a identificação das propriedades e dos efeitos das próprias moléculas. Por este motivo, é necessário ter clareza de qual objetivo você quer para explorar seus dados. Para esse exemplo, iremos aplicar o aprendizado de máquina visando reconhecimento de padrões de moléculas de fungos e de bactérias e, se possível, separá-las!

Importação das principais bibliotecas.

Agora no Jupyter Notebook você iniciará a preparação dos dados para aplicá-los em modelos de aprendizado de máquina. Para isso, a etapa inicial é realizar a importação das bibliotecas básicas para iniciar o projeto. Veja a proposta do Código 21 demonstrada a seguir:

Código 21- Bibliotecas necessárias para iniciar o projeto.

```
#CÓDIGO 21
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Note que além de bibliotecas, forneci um código relacionado com o design e configuração de parâmetros das figuras que serão criadas. Assim, estabelecemos uma padronização das informações que serão retiradas desta análise.

Carregamento dos dados.

Novamente, dando continuidade à exploração racional dos dados moleculares discutidos na fase de análise exploratória, utilizam o Código 22 para ler um arquivo CSV e carregar seus dados em um DataFrame usando a biblioteca Pandas.

Código 22 - Carregamento do conjunto de dados moleculares.

```
#CÓDIGO 22
import pandas as pd
df = pd.read_csv('df_completo.csv')
df
```

- **import pandas as pd:** Importa a biblioteca Pandas e a renomeia como 'pd', permitindo acessar as funções da biblioteca usando 'pd'.
- **df = pd.read_csv('df_completo.csv')[5]:** Esta linha de código lê o arquivo CSV chamado 'df_completo.csv' e carrega seus dados em um DataFrame, que é armazenado na variável 'df'. O método `read_csv()` do Pandas é usado para ler arquivos CSV e convertê-los em DataFrames.
- **df:** Por fim, o DataFrame 'df' é exibido, mostrando os dados que foram carregados a partir do arquivo CSV. O Jupyter Notebook ou o ambiente em que o código está sendo executado automaticamente exibirá os dados em forma de tabela.

Separando os dados entre treino e teste.

Dividir ou amostrar o conjunto de dados em um conjunto de treinamento e um conjunto de teste (também conhecido como conjunto de validação) é uma etapa crucial no desenvolvimento de soluções de aprendizado de máquina. Você treina seus modelos usando o conjunto de treinamento e os avalia usando o conjunto de teste. A taxa de erro em novos casos é chamada de erro de generalização, e ao avaliar seu modelo no conjunto de teste, você obtém uma estimativa desse erro. Esse valor indica o quão bem seu modelo pode se comportar em instâncias que nunca viu antes.

5 O conjunto de dados "completo.csv" pode ser baixado do site de apoio a esse livro: https://vieira86.github.io/livroquimioinformatica/

Muitos autores e profissionais de aprendizado de máquina não realizam essa tarefa nesta fase do pipeline de ML. Normalmente, eles utilizam o conjunto de dados completo para limpeza, pré-processamento e análise exploratória de dados.

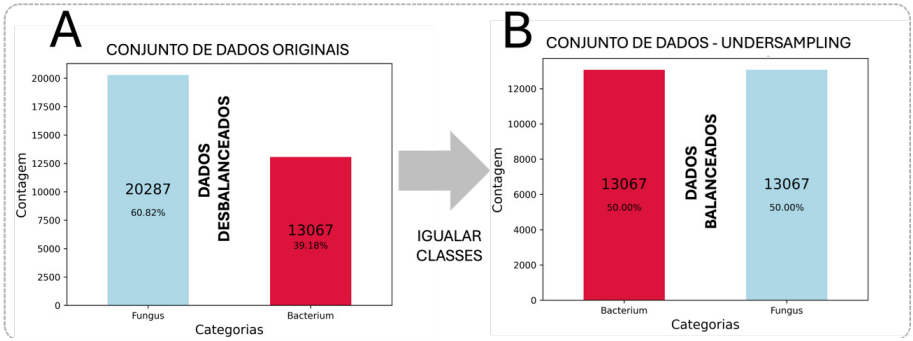
Por exemplo, ao usar o conjunto de dados completo para preencher valores ausentes em um atributo com a mediana, está-se olhando para amostras futuras de teste. Consequentemente, a estimativa do erro de generalização pode ser enviesada, e você pode lançar um sistema em produção que não apresenta o desempenho esperado. Isso é chamado de viés de espionagem de dados. Portanto, é recomendável realizar a amostragem do conjunto de dados nesta fase do pipeline de ML.

OBSERVAÇÕES: No entanto, existem algumas preocupações com isso. Por exemplo, se você tem um conjunto de dados com 100 amostras e o divide aleatoriamente em um conjunto de treinamento (80 amostras) e um conjunto de teste (20 amostras), durante a limpeza de dados, você pode descobrir que tanto o conjunto de treinamento quanto o de teste possuem amostras com valores ausentes ou duplicados. Ao remover essas amostras, a proporção inicial entre o treino e os conjuntos de teste muda, o que pode impactar o treinamento e a avaliação do modelo nos conjuntos de teste. Uma estratégia razoável é realizar a limpeza de dados em todo o conjunto de dados antes de dividi-lo **(como foi feito nesse conjunto que estamos explorando)**.

É comum usar 80% dos dados para treinamento e 20% para teste, mas isso depende do tamanho do conjunto de dados. Se o conjunto de dados for muito grande, manter uma porcentagem menor para teste ainda pode fornecer uma boa estimativa do erro de generalização.

Ao analisar a distribuição de amostras em cada classe, observa-se um desequilíbrio significativo no conjunto de dados, com uma classe contendo significativamente mais amostras do que a outra (Figura 39). Esse desequilíbrio pode prejudicar o desempenho dos modelos de aprendizado de máquina, uma vez que eles podem ter uma tendência a favorecer a classe majoritária, ignorando padrões importantes presentes na classe minoritária.

Figura 40 - Distribuição das classes no conjunto de dados originais (A) e alinhamento na quantidade de instâncias em cada classe usando técnica de undersampling (B).



Para lidar com esse problema, uma abordagem comum é o undersampling que envolve reduzir o número de amostras na classe majoritária para equilibrar as proporções entre as classes (Figura 40-B). Isso pode ser feito aleatoriamente, removendo uma parte das amostras da classe majoritária, ou de forma mais estratégica, utilizando técnicas que preservam informações importantes enquanto reduzem o desequilíbrio.

Os benefícios do undersampling incluem:

1. Melhor desempenho do modelo: Ao equilibrar as proporções entre as classes, os modelos de aprendizado de máquina podem aprender de maneira mais eficaz os padrões presentes em ambas as classes, resultando em uma melhor capacidade de generalização e desempenho em dados não vistos.

2. Redução do viés: O desequilíbrio de classes pode levar a um viés nos modelos, onde eles tendem a classificar erroneamente a maioria das amostras como pertencentes à classe majoritária. Com o undersampling, esse viés é reduzido, resultando em previsões mais equilibradas e precisas.

3. Economia de recursos computacionais: Reduzir o número de amostras na classe majoritária pode diminuir o tempo de treinamento e os recursos computacionais necessários para construir e avaliar os modelos, especialmente em conjuntos de dados grandes.

4. Foco nas amostras relevantes: Ao remover amostras redundantes da classe majoritária, o undersampling permite que os modelos se concentrem em amostras mais informativas e representativas de ambas as classes, melhorando a capacidade de discriminação entre elas.

No nosso estudo, a técnica de undersampling será muito eficaz para lidar com conjuntos de dados desbalanceados, melhorando o desempenho e a capacidade de generalização dos modelos de aprendizado de máquina visando o reconhecimento de padrões em moléculas de fungos e bactérias.

Para a separação das classes, optaremos por uma amostragem estratificada. Essa técnica é especialmente útil quando se tem por objetivo manter as proporções entre as classes nos dados de treinamento e nos dados de teste. Assim, a amostragem estratificada garante que as proporções entre as classes sejam mantidas em ambas as partes. Isso é fundamental para evitar distorções nos modelos de aprendizado de máquina, garantindo que eles sejam treinados e avaliados de forma justa em todas as classes.

Quando garantimos uma representação equilibrada de todas as classes nos conjuntos de dados de treinamento e teste, a amostragem estratificada ajuda a reduzir o viés nos resultados dos modelos de aprendizado de máquina. Utilize o Código 23 para criar uma estratificação baseado na massa molecular das amostras.

Código 23 - Código para extratificação das massas moleculares.

```
#CÓDIGO 23:
df_completo['massa_molecular']      =      pd.cut(df_completo['massa_
molecular'], bins=[2, 200, 400, 600, 1000, np.inf], labels=[1, 2, 3,
4, 5], include_lowest=True)
```

- pd.cut(): Esta função do Pandas é usada para dividir os valores da coluna 'MolWt' em intervalos (ou bins) específicos. Os intervalos são definidos pelos argumentos `bins`, que especificam os limites das faixas. Neste caso, os intervalos são `[2, 200, 400, 600, 1000, np.inf]`, o que significa que os valores serão agrupados em faixas de 2 a 200, 201 a 400, 401 a 600, 601 a 1000 e acima de 1000 (infinito).
- labels: Este argumento especifica os rótulos para cada intervalo. Neste caso, as faixas são rotuladas como 1, 2, 3, 4 e 5, respectivamente.
- include_lowest: Este argumento é definido como True para incluir o limite inferior do primeiro bin. Isso significa que o valor 2 será incluído na primeira faixa.

O código proposto acima cria uma nova coluna chamada 'massa_molecular' no nosso dataframe *df_completo*, que contém os rótulos das faixas (ou bins) de valores de outra coluna existente chamada 'massa_molecular'.

Portanto, após a execução deste código, a coluna ‘massa_molecular’ conterá os rótulos correspondentes às faixas de valores da coluna ‘massa_molecular’, de acordo com os intervalos e rótulos especificados. Realizando esse tipo de abordagem, você tem garantias de que a seleção das instâncias nos conjuntos de treinamento e teste tenha proporções muito próximas. Para realizar a separação, basta utilizar o Código 24 apresentado abaixo:

Código 24 - Separação dos dados entre treino e teste.

```
#CÓDIGO 24:

from sklearn.model_selection import train_test_split

X_train, X_test = train_test_split(df_completo, test_size=0.2, random_state=42, stratify=df_completo[‘massa_molecular’])
X_train = X_train.drop(columns=[‘massa_molecular’])
X_test = X_test.drop(columns=[‘massa_molecular’])
```

Este código utiliza a função ‘**train_test_split**’ do módulo ‘model_selection’ da biblioteca ‘Scikit-learn’ para dividir o DataFrame ‘df_completo’ em conjuntos de treinamento e teste. Aqui está o que cada argumento faz:

df_completo: O DataFrame que contém os dados a serem divididos em conjuntos de treinamento e teste.

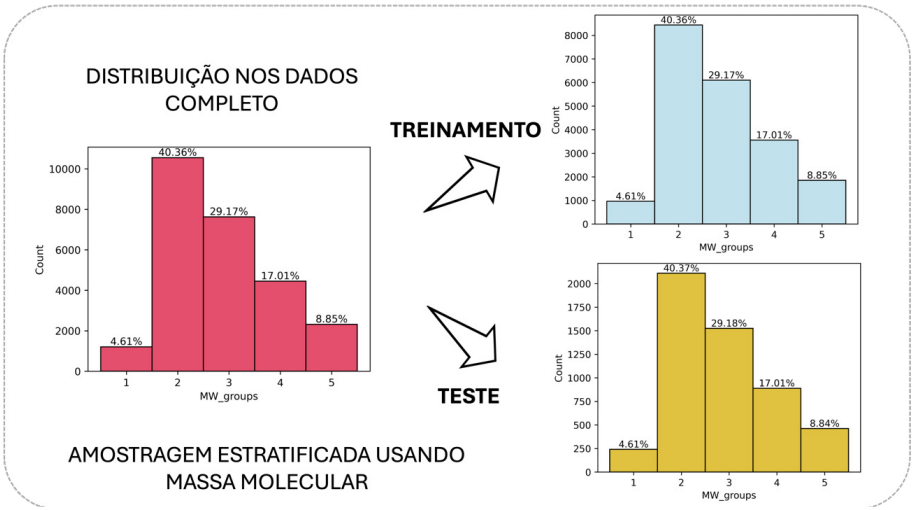
test_size=0.2: Especifica a proporção dos dados que serão atribuídos ao conjunto de teste. Neste caso, 20% dos dados serão usados para teste, enquanto os 80% restantes serão usados para treinamento.

random_state=42: Controla a aleatoriedade da divisão dos dados. Definir um valor específico para `random_state` garante que a divisão seja reproduzível, ou seja, ao executar o código várias vezes com o mesmo valor para `random_state`, você obterá a mesma divisão dos dados.

Após a execução deste código, ‘**X_train**’ conterá os dados de treinamento e `X_test’ conterá os dados de teste. Cada um desses conjuntos é uma amostra aleatória dos dados originais, com 80% dos dados no conjunto de treinamento e 20% no conjunto de teste, conforme especificado pelo argumento ‘test_size’. O ‘random_state’ garante que a divisão dos dados seja consistente entre diferentes execuções do código. Note que a coluna ‘massa_molecular’é deletada, pois ela foi construída apenas criar a estratificação.

Agora você tem disponível dois conjuntos de dado: um para utilizar como treinamento dos modelos de aprendizado de máquina (X_train – contendo 20907 instâncias, ou seja, 80% da quantidade das amostras do dataframe original) e o outro, será utilizado para testagem e obtenção das métricas dos modelos (X_test – contendo 5227, os 20% do dataframe original). Observe a distribuição no conjunto original e nos dados de treinamento/teste na Figura 41, mostrando que a distribuição seguiu a mesma proporção tanto nos dados de treino, quanto nos de teste.

Figura 41 - Separação estratificada dos dados de treino e teste.



Correlação dos descritores químicos utilizados.

Durante a análise exploratória dos dados realizada no início desse livro, foram explorados quais descritores apresentaram correlação. E assim, optou-se por removê-los, de modo que fossem evitados problemas durante a geração dos modelos de aprendizado de máquina.

Código 25 - Verificação de descritores correlacionados.

```
#CÓDIGO 25
# Verificar a existência de variáveis altamente correlacionadas


corr_matrix = X_train_correl.corr()
comb = combinations(corr_matrix.columns, 2)
variavel_deletada = []
```

```
for c1, c2 in comb:
    if abs(corr_matrix.loc[c1, c2]) > 0.8:
        if c1 not in variavel_deletada:  # Verifica se a variável já
está na lista
            variavel_deletada.append(c1)
        if c2 not in variavel_deletada:  # Verifica se a variável já
está na lista
            variavel_deletada.append(c2)

# Remover variáveis duplicadas das que serão deletadas
variavel_deletada = list(set(variavel_deletada))

# Criar lista das variáveis que não serão deletadas
variavel_nao_deletada = [col for col in corr_matrix.columns if col not
in variavel_deletada]

print("Variáveis deletadas:")
print(variavel_deletada)

print("\nVariáveis não deletadas:")
print(variavel_nao_deletada)

df_sem_correlacao = X_train_correl.drop(columns = variavel_deletada)
```

Utilizamos o Código 25 para verificar a existência de variáveis altamente correlacionadas em um DataFrame, e sugerir aquelas que poderão ser desconsideradas para alimentar os modelos de aprendizado de máquina.

1. Importação de bibliotecas adequadas:

from itertools import combinations

- 'itertools' é um módulo em Python que fornece ferramentas para trabalhar com iteradores. Neste caso, estamos usando a função 'combinations' para gerar todas as combinações possíveis de pares de colunas em nosso DataFrame.

2. Cálculo da matriz de correlação:

corr_matrix = X_train_correl.corr()

- 'corr()' é uma função do pandas que calcula a matriz de correlação entre as colunas do DataFrame. Aqui, em X_train_correl deixamos apenas as variáveis numéricas para buscar correlações, então 'X_train_correl' é o DataFrame contendo as variáveis de treinamento que queremos analisar.

3. Iteração sobre combinações de colunas:

```
comb = combinations(corr_matrix.columns, 2)
```

- ‘combinations’ gera todas as combinações possíveis de pares de colunas na matriz de correlação.

4. Verificação da correlação:

```
for c1, c2 in comb:
    if abs(corr_matrix.loc[c1, c2]) > 0.8:
        if c1 not in variavel_deletada:
            variavel_deletada.append(c1)
        print(f'r2 entre as variáveis {c1} e {c2} = {corr_matrix.loc[c1, c2]:.3f}')
```

- Para cada par de colunas (`c1` e `c2`) nas combinações:
- Verifica se a correlação absoluta entre as duas colunas é maior que 0.8.
- Se for maior, a primeira coluna (`c1`) é adicionada à lista `variavel_deletada` se ainda não estiver presente.
- Imprime a correlação entre as duas variáveis.
- No final, você disponibilizou um dataframe chamado **df_sem_correlacao**.

Este código será útil para identificar variáveis altamente correlacionadas, o que pode levar a problemas como multicolinearidade em modelos de machine learning. Identificar e remover variáveis altamente correlacionadas pode melhorar o desempenho e a interpretabilidade do modelo.

Após utilização do código acima, nota-se que o conjunto de dados agora apresenta 5 variáveis (['FractionCSP3', 'MolLogP', 'NumAromaticRings', 'NumRotatableBonds', 'fr_azo'] e serão estas que irão participar do treinamento do modelo. Agora precisamos separar as variáveis preditoras e a variável target. Para isso, basta utilizar o Código 26, que será gerado 4 novos conjuntos de dados, separando do X_train e X_test os rótulos de cada instância, gerando, assim, as labels, chamadas de y_train e y_test.

Código 26 – Geração das matrizes de treinamento e array de labels.

```
#CÓDIGO 26:
X_train_preditoras = X_train.drop(columns=['Class'])
X_train_target = X_train['Class'].copy()
X_test_preditoras = X_test.drop(columns=['Class'])
X_test_target = X_test['Class'].copy()
```

Mas antes de inseri-las nos algoritmos de aprendizado de máquina, precisamos padronizar os dados. Pois assim, teremos garantias de que o modelo não ficará enviesado para descritores que apresentem valores mais expressivos do que o outro.

Código 27 - Aplicação de pipeline para pre-processamento dos dados.

```
#CÓDIGO 27:
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('robust_scaler', RobustScaler())
])

# (name, transformer, columns)
preprocessed_pipeline = ColumnTransformer([
    ('numerical', num_pipeline, variavel_nao_deletada)
])

X_train = preprocessed_pipeline.fit_transform(X_train)
X_test = preprocessed_pipeline.transform(X_test)
y_train = y_train.values
y_test = y_test.values
```

O Código 27 cria um pipeline de pré-processamento de dados usando a biblioteca scikit-learn.

1. Importação de Módulos:

- 'from sklearn.impute import SimpleImputer': Importa a classe `SimpleImputer` do módulo `impute` para preencher valores ausentes.

- ‘from sklearn.preprocessing import RobustScaler’: Importa a classe `RobustScaler` do módulo `preprocessing` para dimensionar características numéricas.
- ‘from sklearn.preprocessing import OneHotEncoder’: Importa a classe `OneHotEncoder` do módulo `preprocessing` para codificar características categóricas.
- ‘from sklearn.pipeline import Pipeline’: Importa a classe `Pipeline` do módulo `pipeline` para construir pipelines de transformação de dados.
- ‘from sklearn.compose import ColumnTransformer: Importa a classe `ColumnTransformer` do módulo `compose` para aplicar diferentes transformações a diferentes colunas do conjunto de dados.

2. Pipeline para Dados Numéricos:

- ‘num_pipeline’: Cria um pipeline para dados numéricos com duas etapas:
- ‘SimpleImputer’: Preenche valores ausentes com a mediana usando a estratégia definida como ‘median’.
- ‘RobustScaler’: Escala as características numéricas usando o RobustScaler para lidar com outliers.

3. ColumnTransformer para Pré-Processamento Geral:

- `preprocessed_pipeline`: Cria um `ColumnTransformer` para aplicar o pipeline `num_pipeline` apenas às variáveis que não serão deletadas (‘variavel_nao_deletada’). Isso garante que apenas as variáveis que serão mantidas após a remoção das variáveis altamente correlacionadas sejam pré-processadas.

Basicamente, a propositura de utilização desse código irá definir um pipeline de pré-processamento que preenche valores ausentes com a mediana e dimensiona características numéricas usando RobustScaler, aplicando essas transformações apenas às variáveis que não serão deletadas após a identificação das variáveis altamente correlacionadas.

Agora, você tem disponíveis os conjuntos de dados necessários para alimentar os algoritmos de aprendizado de máquina. O Código 28 faz o treinamento direto de 7 modelos de aprendizado de máquina bastante corriqueiros, sendo eles: logistic regression; KNN; Árvores de Decisão; Support Vector Machine; Random Forest; XGBoost (método de ensemble) e o Naive Bayes.

Treinamento dos modelos de aprendizado de máquina

Código 28 - Código utilizado para treinamento dos modelos de machine learning.

```
#CÓDIGO 28
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB

# Selecionar algoritmos
models = []
models.append(('LR', LogisticRegression(solver='liblinear')))
    # For small datasets, 'liblinear' is a good choice
models.append(('KNN', KNeighborsClassifier(n_neighbors=5)))
models.append(('CART', DecisionTreeClassifier()))
models.append(('SVM', SVC(kernel='linear')))
models.append(('RF',   RandomForestClassifier(random_state=0)))      #
random_state para algoritmos estocásticos
models.append(('XGB', GradientBoostingClassifier(random_state=0)))
models.append(('NaiveBayes', GaussianNB()))

# Avaliar cada modelo com validação cruzada (stratified 10-fold cross-
validation)
acc = []
precision = []
roc_auc = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
     acc_results = cross_val_score(model, X_train, y_train.ravel(),
cv=kfold, scoring='accuracy')
    acc.append(acc_results)
      precision_results = cross_val_score(model, X_train, y_train.
ravel(), cv=kfold, scoring='precision')
    precision.append(precision_results)
```

```
   roc_auc_results = cross_val_score(model, X_train, y_train.ravel(),
cv=kfold, scoring='roc_auc')
    roc_auc.append(roc_auc_results)
    names.append(name)
    print(f'{name} Accuracy: {acc_results.mean():.2f} ({acc_results.
std():.2f})')
         print(f'{name}  Precision:  {precision_results.mean():.2f}
({precision_results.std():.2f})')
     print(f'{name} ROC_AUC: {roc_auc_results.mean():.2f} ({roc_auc_
results.std():.2f})')
    print('------------------------------')
```

O Código 28 é uma demonstração de como avaliar o desempenho de vários algoritmos de classificação utilizando validação cruzada. Vamos discutir a proposta geral, linha por linha:

1. Importações: O código importa as bibliotecas necessárias do scikit-learn para realizar a modelagem e avaliação de algoritmos de machine learning.

2. Definição dos algoritmos: Aqui, vários algoritmos de classificação serão definidos com diferentes parâmetros. Os algoritmos incluem Regressão Logística, K-Vizinhos Mais Próximos, Árvore de Decisão, SVM (Máquinas de Vetores de Suporte), Floresta Aleatória, Gradient Boosting e Naive Bayes.

3. Avaliação com validação cruzada: Para cada modelo definido, o código realiza os seguintes passos:

- Cria um objeto StratifiedKFold que divide o conjunto de dados em 10 partes, preservando a proporção de classes.
- Utiliza o método 'cross_val_score' para calcular a métrica de desempenho (precisão, acurácia e área sob a curva ROC) usando validação cruzada.
- As métricas são armazenadas em listas para posterior análise.
- O desempenho médio e o desvio padrão das métricas (acurácia, precisão e área sob a curva ROC) são impressos para cada modelo.

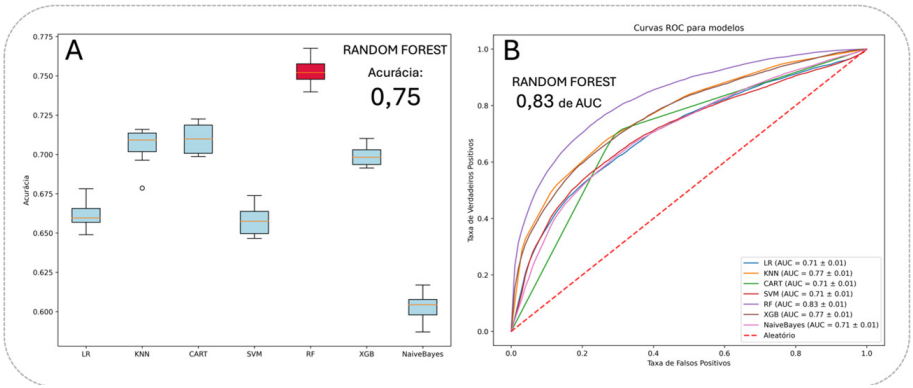
Notaram que o treinamento de algoritmos de aprendizado de máquina é relativamente simples de se fazer usando Python? Agora a parte fundamental é analisar as métricas e, assim, decidir qual é o melhor classificador para reconhecer padrões em dados orgânicos. Caso queira gerar uma representação gráfica que facilita a visualização do melhor modelo, sugiro a utilização do Código 29:

Código 29 – Gráfico para visualização o desempenho dos modelos de machine learning.

```
#CÓDIGO 29
import matplotlib.pyplot as plt
import numpy as np

# Calcula a média da acurácia para cada algoritmo
mean_acc = [np.mean(scores) for scores in acc]

# Encontra o índice do algoritmo com a maior média de acurácia
best_idx = np.argmax(mean_acc)

# Lista de cores para os box plots
colors = ['lightblue' if i != best_idx else 'crimson' for i in
range(len(names))]

# Comparar algoritmos
plt.figure(figsize=(10, 8))
box = plt.boxplot(acc, labels=names, patch_artist=True)

# Aplicar cores aos box plots
for patch, color in zip(box['boxes'], colors):
    patch.set_facecolor(color)

plt.ylabel('Acurácia')
plt.savefig('metricas_boxplot.png', dpi=600, bbox_inches='tight')
plt.show()
```

Realizando uma comparação entre os desempenhos dos modelos de aprendizado de máquina, nota-se que o random forest destacou-se apresentando as melhores métricas de acurácia (com valor de 0.75) (Figura 42-A), precisão e curva-ROC (0.86) (Figura 42-B).

Figura 42 - Desempenho dos algoritmos de aprendizado de máquina utilizados para estudos classificatórios de moléculas orgânicas de fungos e de bactérias.



Como temos um problema de classificação binária, podemos utilizar a curva ROC (Receiver Operating Characteristic), uma ferramenta útil para avaliar o desempenho de um modelo em diferentes pontos de corte (thresholds).

A curva ROC mostra a relação entre a taxa de verdadeiros positivos (sensibilidade) e a taxa de falsos positivos (1 - especificidade) em diferentes pontos de corte. Um modelo com uma curva ROC mais próxima do canto superior esquerdo do gráfico tem um desempenho melhor, pois alcança uma alta sensibilidade com uma baixa taxa de falsos positivos.

A área sob a curva ROC (AUC) é uma medida resumida do desempenho geral do modelo. Quanto maior a AUC, melhor o desempenho do modelo em discriminar entre as classes positiva e negativa. Uma AUC de 0.5 indica um desempenho aleatório, enquanto uma AUC de 1.0 indica um desempenho perfeito. No nosso caso, o modelo de random forest apresentou uma AUC de 0,83, destacando-se por apresentar métrica melhor do que os demais.

Percebam que foi inserida uma linha diagonal no gráfico, ela representa o desempenho aleatório, onde a sensibilidade é igual à especificidade. Portanto, um modelo com uma curva ROC abaixo da linha diagonal é pior do que um modelo aleatório. Por outro lado, uma curva ROC acima da linha diagonal indica um desempenho melhor do que o aleatório. No nosso estudo, todos os modelos foram melhores do que chutes aleatórios para classificação das estruturas orgânicas.

MÉTRICAS DE DESEMPENHO

Foram apresentados na discussão de curva ROC os conceitos de sensibilidade, especificidade, acurácia. Mas exatamente o que são essas métricas?

Uma vez que você nota que existe um algoritmo que se destaca na classificação, vamos considerá-lo como o melhor modelo apresentado para aquela situação. No nosso caso, selecionaremos o random forest. Caso você queira treiná-lo isoladamente, utilize o Código 30:

Código 30 - Individualização do melhor modelo de machine learning: Random Forest.

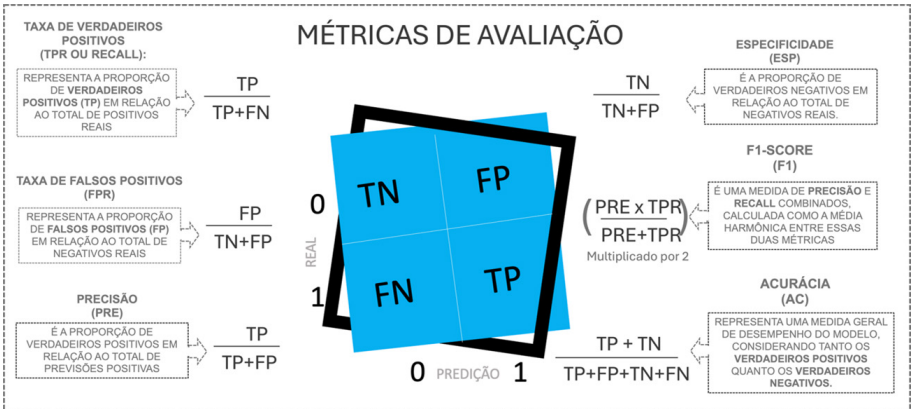
```
#CÓDIGO 30:
rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train.ravel())
predictions_rf = rf.predict(X_test)

from sklearn.metrics import ConfusionMatrixDisplay
confusion_rf = confusion_matrix(y_test, predictions_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=confusion_rf,
display_labels=['Fungos', 'Bactérias'])

# Avaliação da qualidade do modelo
print(f"Accuracy score: {accuracy_score(y_test, predictions_
rf):.2f}")
print()

print("Classification report:")
print(classification_report(y_test, predictions_rf))

print("Confusion matrix:")
disp_rf.plot()
print()
```

Além do treinamento do modelo de random forest, será construída uma ferramenta de avaliação das métricas do modelo: a matriz de confusão. Somado a isso, será emitido um "classification report", um relatório de sistematização das métricas do modelo.

A Figura 43 sistematiza as principais métricas que utilizaremos nesse livro. Você tem que ter disponível apenas uma ferramenta: a matriz de confusão. Apesar do código fornecer um "relatório" chamado de "classification report" vale a pena se aprofundar um pouco no entendimento de cada um

desses pontos. É o que iremos fazer com a matriz de confusão gerada a partir do modelo de random forest.

Figura 43 - Métricas de avaliação de modelos de aprendizado de máquina utilizando matriz de confusão.

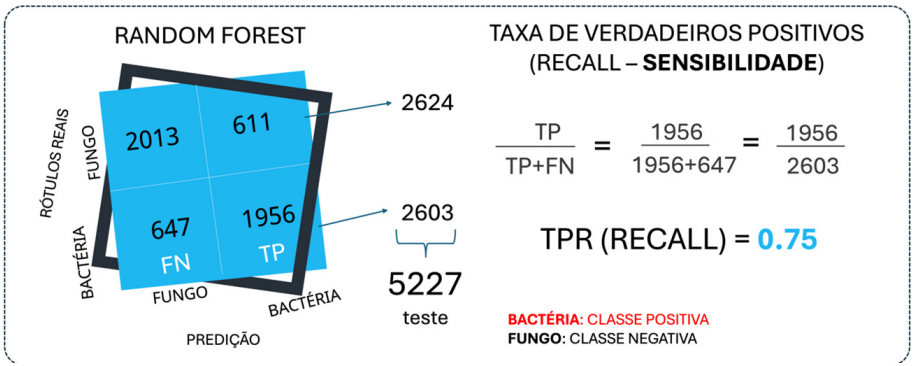


Agora, de posse das fórmulas necessárias para se calcular as métricas associadas ao melhor modelo de machine learning, vamos explorar de maneira racional cada uma das métricas e tirar informações valiosas para nosso problema proposto. O modelo tem capturado os padrões de cada grupo de molécula? Ou será que fungos e bactérias (considerando o espaço químico desse conjunto de dados) apresentam moléculas tão similares que não é possível separá-las? Tais respostas podemos encontrar a partir do conjunto de testagem do modelo fornecido. Ou seja, o dataframe inteiro de moléculas apresentava 26.134 moléculas, divididas igualmente entre estruturas oriundas de fungos e bactérias. Porém, realizamos a divisão desse conjunto de dados entre treinamento e teste, considerando a proporção de 80/20. Ou seja, 80% dos dados foram utilizados para treinar os modelos, e o restante, 20%, para testá-los. Dessa forma, a matriz de confusão é gerada com base nesses 20% das instâncias que serviram para testagem, ou seja, 5.227 moléculas. Dentre essas, 2.624 de fungos e 2.603 de bactérias.

Geralmente na linguagem de machine learning, estabelece-se que uma das classes é dita “positiva” e a outra, por conseguinte, “negativa”. Mas esse rótulo não quer dizer que uma classe seja melhor ou pior do que a outra, é utilizada apenas como uma classificação binária genérica, e por este motivo, as fórmulas contemplam o valor de positivo/negativo. Não importa quem seja considerado como positivo ou negativo. Veja o exemplo abaixo.

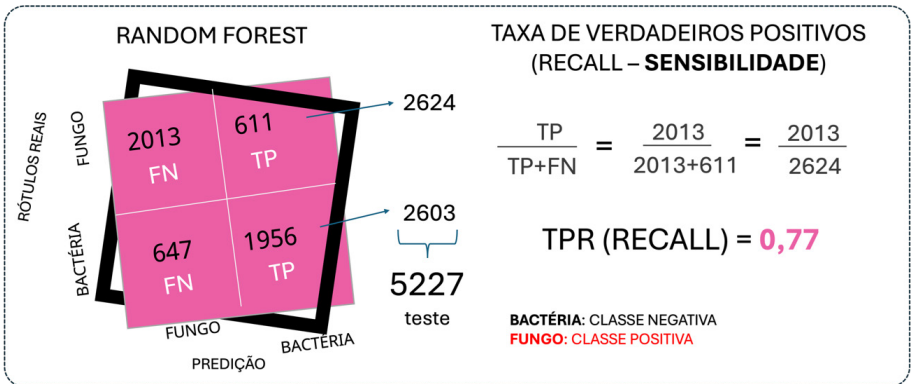
Considerando que a classe bactérias seja positiva, encontra-se o valor de 0,75 para a sensibilidade do modelo, uma métrica que leva em consideração todas as instâncias da classe bactéria, ou seja, 2.603, e computa quantas moléculas foram classificadas corretamente pelo modelo. Assim, o modelo de random forest apontou que 1.956 estruturas eram de bactérias, acertando, portanto, 75% dos casos. Os cômputos podem ser analisados com mais cuidado na Figura 44.

Figura 44 - Cálculos de sensibilidade (ou recall) considerando a classe bactéria como sendo positiva.



Por outro lado, caso considere a classe fungo como sendo o grupo positivo, os cálculos são exatamente os mesmos, porém considerando a linha associada ao fungo (a linha de cima da matriz de confusão) como sendo positiva. Logo, das 2.624 moléculas de origem fúngica, 2.013 foram classificadas corretamente pelo modelo random forest, gerando, assim, uma sensibilidade de 0,77. Você pode acompanhar os cálculos na **Figura 45**.

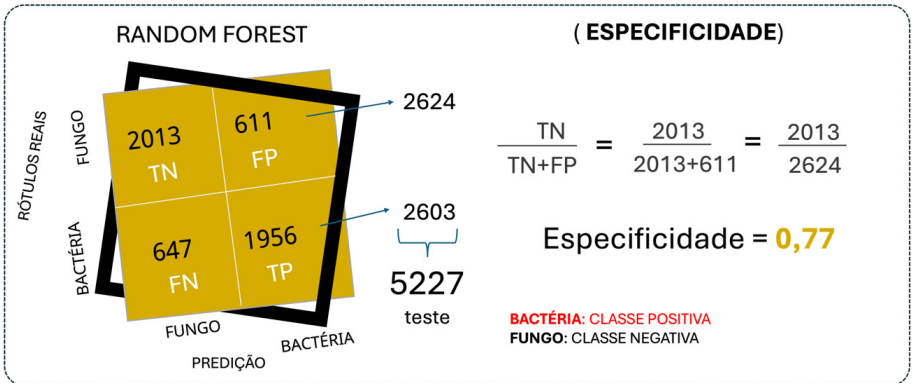
Figura 45 - Cálculos de sensibilidade (ou recall) considerando a classe fungo como sendo positiva.



Porém, consideramos sempre, em nossos problemas futuros, que a linha de baixo da matriz de confusão será considerada a classe positiva (em classificações binárias), quando compararmos a quantidade de instâncias que foram corretamente associadas dentro dessa classe positiva, iremos nos referir à sensibilidade do modelo. Ou seja, a sensibilidade do modelo de random forest é de 0,75.

Isso é feito corriqueiramente porque existe uma outra métrica, chamada de especificidade, que irá explorar a classe negativa da matriz de confusão. Ou seja, irá ser encontrado o mesmo valor se tivéssemos considerado a classe fungo como positiva. Por este motivo, não precisamos ficar alternando as classes entre positiva e negativa, pois existem métricas que irão considerar essa situação, veja o cálculo associado à especificidade do modelo na Figura 46.

Figura 46 - Cálculo de especificidade do modelo de random forest usando matriz de confusão.



Note que existem 4 valores específicos nessa matriz de confusão binária:

- **TN** (True Negative) ou verdadeiros negativos – são moléculas pertencentes à classe de fungos (classe negativa) que foram corretamente classificadas como pertencentes à classe negativa. No nosso caso, a quantidade de instâncias classificadas como TN é igual a 2.013.
- **FP** (False Positive) ou falsos positivos – são instâncias pertencentes à classe dos fungos (classe negativa) que foram erroneamente classificadas como bactérias (que são classe positiva). Há 611 moléculas que estão nesse grupo.

- **FN** (False Negative) ou falsos negativos – neste caso, são estruturas moleculares que originalmente são de bactérias (classe positiva), mas que o modelo teve confusão no reconhecimento de padrões, classificando-as, erroneamente, como sendo de fungos (classe negativa). No nosso contexto, há 647 moléculas nesse grupo.
- **TP** (True Positive) ou verdadeiros positivos, é o grupo de moléculas que são realmente da classe positiva (bactérias) e que foram classificadas assim. No nosso contexto, há 1.956 moléculas nesse grupo.

Voltando a discutir o caso da especificidade do modelo, note que o valor obtido para essa métrica foi de 0,77. Ou seja, o restante, os 0,23 são associados à taxa de falsos positivos, que em associado à taxa de verdadeiros positivos é construída a curva ROC mostrada na Figura 42-B.

Note que os valores FN (falso negativo) e TP (verdadeiro positivo) são os valores da linha de baixo da matriz de confusão, pois se relacionam com a classe positiva. E os demais, FP (falso positivo) e TN (verdadeiro negativo) estarão na primeira linha porque dizem respeito sempre à classe negativa.

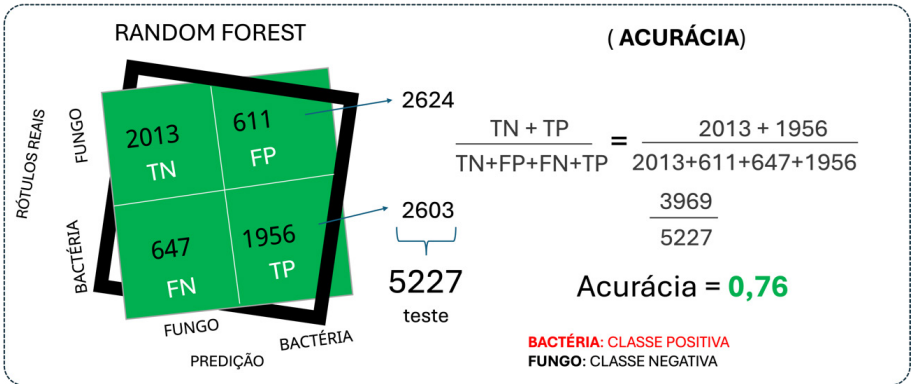
E a respeito da **precisão** do modelo estabelecido? Ou seja, existem 5.227 moléculas de testes que estão distribuídas nas duas classes. O modelo tem sido preciso em suas classificações? Analisando a segunda coluna da matriz de confusão (somando FP e TP) temos o valor de 2.567. Ou seja, o modelo disse que existem 2.567 moléculas oriundas de bactérias, mas ele acertou 1.956. ou seja, o modelo foi preciso em 76% dos casos. Existem 611 estruturas que causam confusão na classificação do modelo, podem ser moléculas de fungos que apresentam padrões similares às de bactérias, algo totalmente plausível.

Se quiséssemos verificar a precisão para a classe dos fungos, utilizaríamos as informações da primeira coluna da matriz de confusão (somando os valores de TN e FN), o que englobaria 2.660 moléculas. Desse montante, o modelo acerta a classificação de 2.013, tendo uma precisão de 76% também. Ou seja, o modelo apresenta a mesma precisão para ambas as classes.

E sobre a acurácia do modelo? Foi a métrica utilizada para criar os gráficos de boxplot da Figura 42-A. Ela busca relacionar as classificações corretas no contexto geral. Ou seja, quantas moléculas da classe negativa (fungos) realmente foram classificadas como fungos? A resposta é 2.013.

Do mesmo modo, quantas moléculas da classe positiva foram corretamente classificadas? 1.956. Logo, 3.969 acertos foram feitos em um conjunto de teste de 5.227 instâncias. Logo, a acurácia do modelo de random forest para o nosso problema de reconhecimento de moléculas de fungos e bactérias foi de 0,76, ou seja, classificou corretamente 76% das moléculas. Os cálculos associados à acurácia estão contemplados na Figura 47.

Figura 47 - Cálculo de acurácia do modelo de random forest utilizando matriz de confusão.



Na nossa busca de tentar encontrar um algoritmo de aprendizado de máquina que pudesse reconhecer padrões assertivos e separar moléculas de fungos das moléculas de bactérias, percebemos que das 5.227 moléculas que usamos para testar o modelo, 1.258 causaram confusão no processo classificatório. Isso nos dá indícios de que, aproximadamente, no espaço químico explorado, 24% das moléculas de fungos e bactérias apresentam similaridade e não podem, portanto, ser separadas. **Mas e se a quantidade de descritores fosse aumentada? Será que poderíamos ter um modelo que conseguisse captar mais detalhes nas moléculas e conseguir métricas melhores?** Vamos reconstruir o processo de aprendizado de máquina, de modo que calcularemos, usando o pacote RDKIT, mais de 200 descritores, e assim, verificaremos o comportamento dos modelos perante um conjunto maior de features.

Aumento das features (descritores químicos)

Logo, para se calcular descritores químicos usando o RDKIT basta utilizar o Código 31. A depender da sua máquina e da quantidade de amostras disponíveis no dataframe, pode demorar um certo tempo! Mas aguarde os cálculos, eles valerão a pena!

Código 31 - Cálculo dos descritores químicos gerados pela biblioteca RDKIT.

```
#CÓDIGO 31
## pip install rdkit

def calc_descriptors(df):
    from rdkit import Chem
    from rdkit.Chem import Descriptors
    from rdkit.ML.Descriptors import MoleculeDescriptors

    descriptors_list = [x[0] for x in Descriptors._descList]
    for desc in descriptors_list:
        names = [desc]
      calc = MoleculeDescriptors.MolecularDescriptorCalculator(names)
    df[desc] = df["Smiles"].apply(lambda x: calc.CalcDescriptors(Chem.
MolFromSmiles(x))[0])
    return df

df_208descritores = calc_descriptors(df)
```

Utilize o Código 31 para calcular descritores moleculares para compostos químicos representados por SMILES (Simplified Molecular Input Line Entry System) no seu DataFrame pandas. O que esse código irá fazer:

Instalação da biblioteca RDKit:

Antes de executar o código, é necessário instalar a biblioteca RDKit, que é uma coleção de software de química e informática química.

A linha ## pip install rdkit é um comentário indicando que você deve usar o gerenciador de pacotes pip para instalar o RDKit. Se você estiver executando o código em um ambiente onde o RDKit não está instalado, pode descomentar esta linha para instalar o RDKit antes de prosseguir.

Definição da função calc_descriptors:

Esta função recebe um DataFrame pandas (df) como entrada.

A função utiliza a biblioteca RDKit para calcular descritores moleculares para os compostos representados pela coluna “Smiles” do DataFrame.

Primeiro, a função importa os módulos necessários do RDKit, como Chem, Descriptors e MoleculeDescriptors.

Em seguida, a função cria uma lista de nomes de descritores usando Descriptors._descList, que contém uma lista de descritores moleculares disponíveis no RDKit.

Para cada descritor na lista de descritores, a função calcula o valor do descritor para cada molécula representada pelo SMILES e adiciona uma nova coluna ao DataFrame com o nome do descritor e os valores calculados.

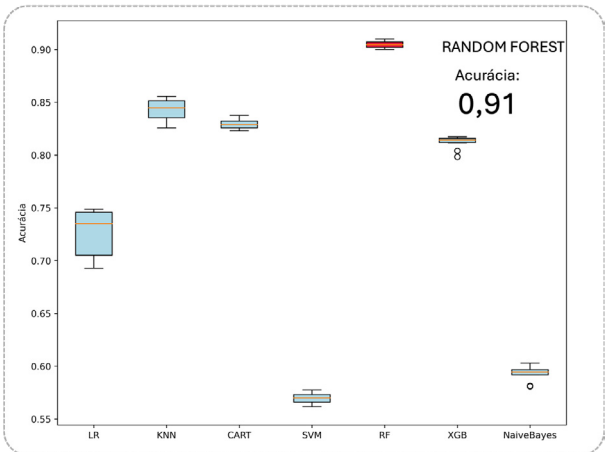
Finalmente, a função retorna o DataFrame com as novas colunas de descritores moleculares adicionadas.

Chamada da função calc_descriptors:

A função calc_descriptors é chamada passando o DataFrame df como argumento. Isso calcula os descritores moleculares para as moléculas representadas pelos SMILES no DataFrame e retorna um novo DataFrame (df_208descritores) com as colunas de descritores moleculares adicionadas.

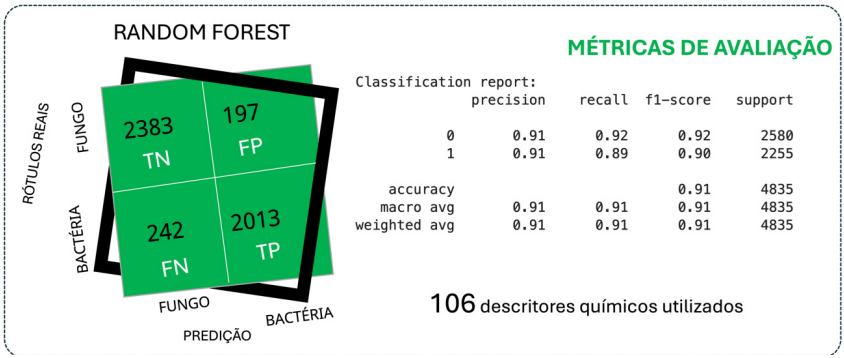
A quantidade de descritores químicos utilizados para o reconhecimento de padrões de moléculas oriundas de fungos e bactérias é algo extremamente relevante. Percebam que com a utilização de cinco descritores, o melhor modelo capaz de separar as classes com maior acuracidade foi o random forest, com métrica de 0,75. Ao disponibilizar mais descritores químicos, novos padrões foram reconhecidos, e o modelo de random forest aumentou sua robustez classificatória, agora chegando a 91% de acerto nas moléculas de fungos e bactérias. Ressalta-se que a mesma estratégia foi utilizada, de obter apenas descritores que apresentavam baixa correlação. Assim, 208 descritores foram calculados usando a biblioteca RDKIT, mas por filtragem e seleção apenas dos não correlacionados, obteve-se um conjunto químico com 106 descritores. Veja o desempenho de todos os modelos na Figura 48 utilizando essa nova fonte de dados.

Figura 48 - Desempenho dos modelos de aprendizado de máquina utilizando 106 descritores químicos.



Assim, a matriz de confusão gerada para este novo conjunto de dados, agora com 106 descritores, pode ser utilizada para um entendimento didático das novas métricas estabelecidas para solução do problema proposto. A Figura 49 faz a sistematização das métricas de avaliação para o novo modelo.

Figura 49 - Métricas de avaliação e matriz de confusão para o novo modelo treinado utilizando 106 descritores químicos no conjunto de treinamento e teste.



Uma observação extremamente importante que deve ser considerada é que a quantidade de amostras utilizadas para separação dos dados de treino e teste foi mudada devido às características dos descritores no momento da normalização dos dados. Quando aplicado o código 10, algumas variáveis apresentam valores que tendem ao infinito, e isso apresentou um erro nos algoritmos de aprendizado de máquina. Visando contornar essa situação, as amostras que apresentavam essas características foram removidas, resultando, portanto, em um conjunto de treinamento menor, mas que apresentou uma métrica mais elevada. Para solução de situações como essas, você pode se inspirar no Código 32.

Código 32 – Código para remoção de variáveis com erros, geralmente associadas a valores infinitos.

```
#CÓDIGO 32
def remove_infinity_and_large_values(X, y, threshold=1e6):
    mask = np.isfinite(X).all(axis=1)  # Selecionar linhas com valores finitos
    mask &= np.all(np.abs(X) < threshold, axis=1)  # Selecionar linhas com valores menores que o threshold em todas as colunas
     mask &= np.all(X != -np.inf, axis=1)  # Selecionar linhas sem valores infinitos negativos
     mask &= np.all(X != np.inf, axis=1)  # Selecionar linhas sem valores infinitos positivos
    return X[mask, :], y[mask]
```

```
df_novo,y_train_novo = remove_infinity_and_large_values(preprocessed_
data, y)

print(len(df_novo))
print(len(y_train_novo))

df_processado = pd.DataFrame(X_train, columns=variavel_nao_deletada)
df_ processado [‘Class’] = y_train
```

Use o Código 32 quando você tiver problemas no treinamento dos algoritmos, sinalizando que existem valores infinitos. Essa é uma função chamada ‘remove_infinity_and_large_values’, que tem a finalidade de remover linhas de um conjunto de dados onde há valores infinitos ou muito grandes. O que esse código fará?

1 – Definição da função: A função recebe três argumentos: ‘X’, que representa os dados a serem processados, ‘y’, que representa os rótulos ou classes associadas aos dados, e ‘threshold’, um valor limite para identificar valores muito grandes. Por padrão, o limite é definido como ‘1e6’ (ou seja, 1 milhão).

2 - Máscara para valores finitos: A primeira linha cria uma máscara booleana (‘mask’) que identifica as linhas em ‘X’ que contêm apenas valores finitos. Isso é feito usando ‘np.isfinite(X).all(axis=1)’, que verifica se todos os valores em cada linha de ‘X’ são finitos. Isso é útil para eliminar linhas que contenham valores NaN (não são números) ou valores infinitos.

3 - Máscara para valores menores que o limite: A segunda linha adiciona à máscara a condição de que todos os valores em cada linha de ‘X’ devem ser menores que o ‘threshold’. Isso é feito usando ‘np.all(np.abs(X) < threshold, axis=1)’, que verifica se todos os valores absolutos em cada linha de ‘X’ são menores que o limite especificado.

4 - Máscara para valores infinitos negativos: A terceira linha adiciona à máscara a condição de que não pode haver valores infinitos negativos em nenhuma linha de ‘X’. Isso é feito usando ‘np.all(X != -np.inf, axis=1)’, que verifica se não há valores ‘-np.inf’ (infinito negativo) em cada linha de `X`.

5 - Máscara para valores infinitos positivos: A quarta linha adiciona à máscara a condição de que não pode haver valores infinitos positivos em nenhuma linha de ‘X’. Isso é feito usando ‘np.all(X != np.inf, axis=1)’, que verifica se não há valores ‘np.inf’ (infinito positivo) em cada linha de ‘X’.

6 - Retorno dos dados processados: Finalmente, a função retorna os dados filtrados, representados por ‘X[mask, :]’, ou seja, todas as linhas de ‘X’ que satisfazem as condições impostas pelas máscaras, e os rótulos correspondentes ‘y[mask]’.

7 - Aplicação da função e criação de um DataFrame: O código depois aplica a função ‘remove_infinity_and_large_values’ aos dados e aos rótulos ‘y’, armazenando o resultado nas variáveis ‘df_novo e ‘y_train_novo’.

Conclusões sobre o problema classificatório binário.

O objetivo principal do nosso estudo era treinar algoritmos de aprendizado de máquina para identificar padrões em informações moleculares que distinguem entre duas classes de estruturas químicas advindas de fungos e bactérias. Inicialmente, treinamos modelos com apenas 5 variáveis, o que resultou em métricas de desempenho baixas, sugerindo dificuldade em separar as moléculas em suas respectivas classes.

Entretanto, ao adotarmos um novo cenário, no qual calculamos novos descritores químicos e os incorporamos como dados de treinamento e teste, observamos resultados promissores. Das 4.835 moléculas utilizadas no conjunto de teste, apenas 439 apresentaram desafios ao poder classificatório do algoritmo Random Forest, resultando em uma acurácia de 91% nas classificações moleculares.

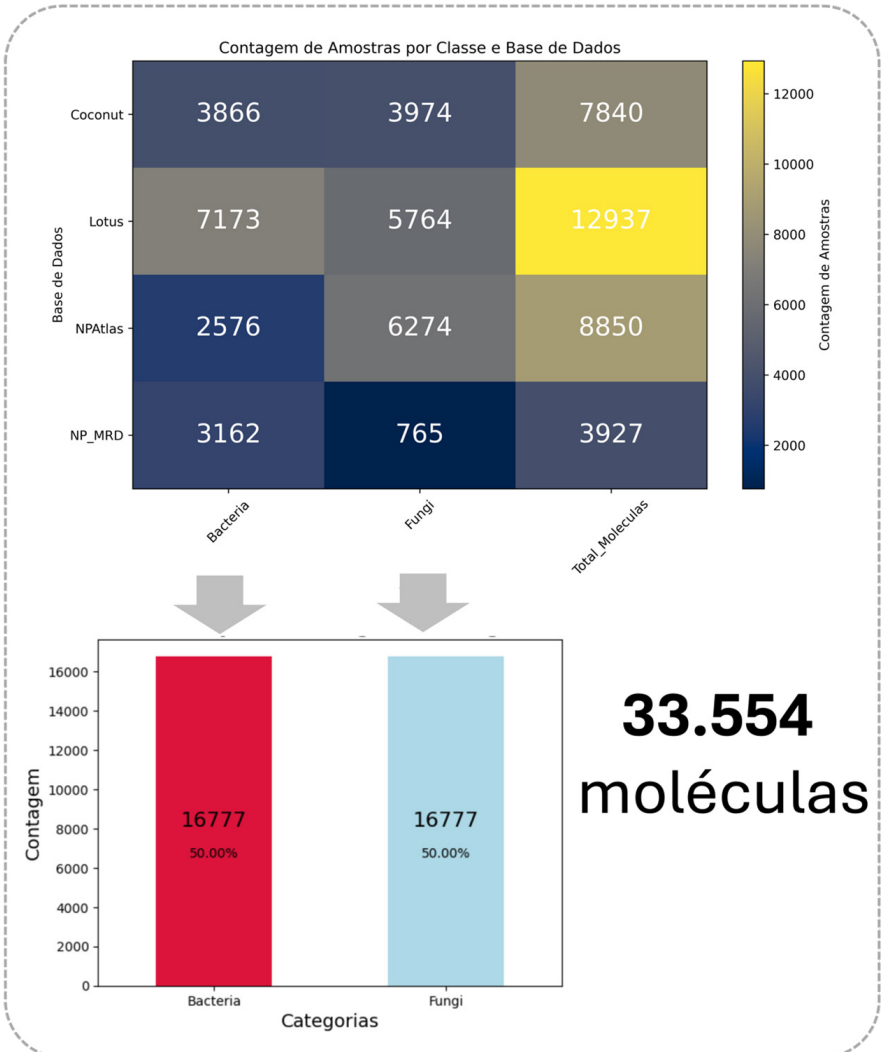
Esses resultados indicam que, ao aumentarmos o número de descritores químicos considerados, especialmente para 106 descritores, o algoritmo Random Forest é capaz de classificar corretamente 91% dos dados. Isso sugere que, do ponto de vista químico, uma parcela significativa das moléculas de bactérias e fungos apresenta diferenças estruturais distintas. Essas descobertas destacam a importância da fonte natural dessas moléculas na seleção e triagem de candidatos a fármacos ou agentes moduladores de alvos biomacromoleculares.

O que acha de darmos continuidade à exploração racional de produtos naturais? Note que até o momento utilizamos dados apenas de uma base de dados, a plataforma NPAtlas. E se compilássemos um novo conjunto de dados, agora contendo mais informações, oriundo de bases de dados distintas? Ampliando o nosso espaço químico, inserindo novos tipos de moléculas

(também de bactérias e fungos), será que os modelos teriam desempenhos parecidos na classificação entre moléculas de fungos e bactérias? Ou teriam tido “sorte” de encontrar um conjunto de dados “fácil” de separar.

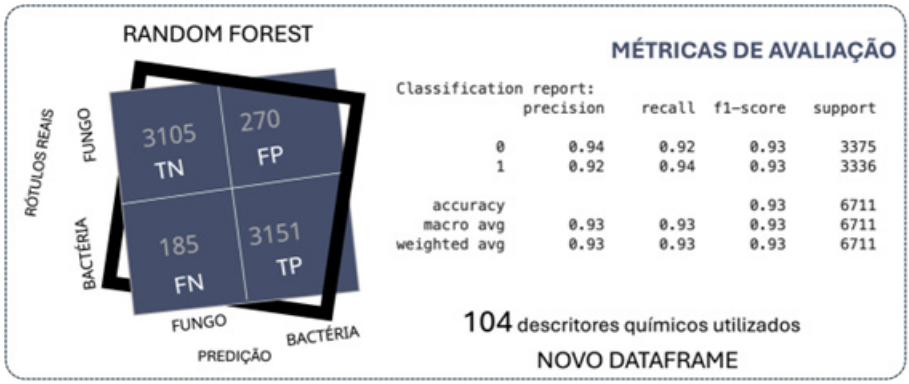
Com vistas a explorar ainda mais a diversidade química, compilamos um novo conjunto de dados, agora usando outras bases de dados. A diversidade dos dados no novo espaço químico pode ser visualizada na Figura 50.

Figura 50 - Distribuição das amostras moleculares de acordo com novos bancos de dados.



Utilizando os mesmos procedimentos para treinamento dos modelos, evidencia-se novamente o algoritmo de Random Forest, novamente apresentando excelentes métricas, por exemplo, a de acurácia de 93% no reconhecimento de padrões entre as duas classes de moléculas. Visualize na Figura 51, a matriz de confusão e o relatório de classificação das métricas para tal modelo.

Figura 51 - Métricas de avaliação e matriz de confusão para o novo conjunto de dados utilizando 104 descritores químicos no conjunto de treinamento e teste.



A análise dos resultados revelou que, dos modelos treinados, o Random Forest obteve o melhor desempenho, alcançando uma acurácia superior a 90% em ambos os conjuntos de dados. Este sucesso do Random Forest sugere sua eficácia na captura e generalização de padrões moleculares complexos presentes nos dados (usando 104 descritores químicos). Além disso, a consistência nos resultados entre diferentes plataformas indica uma robustez dos modelos, demonstrando sua capacidade de lidar com a heterogeneidade nos dados e manter um bom desempenho mesmo diante de variações nos ambientes de origem. Essa consistência é um indicativo positivo da capacidade dos algoritmos de generalizar padrões complexos e fornecer resultados confiáveis em diferentes contextos, o que fortalece a confiança em sua aplicabilidade e relevância para uma variedade de cenários que envolvam produtos naturais.

NOVOS DESAFIOS. A CLASSIFICAÇÃO MULTICLASSE

Diante do sucesso notável do modelo Random Forest na distinção entre moléculas de fungos e bactérias, surge uma empolgante oportunidade de expandir os horizontes e explorar novos desafios. Por que não estender essa capacidade de reconhecimento de padrões para um cenário mais complexo e desafiador? Considerando a eficácia do Random Forest na separação dessas duas classes distintas, podemos nos perguntar: será que esse modelo poderia ser treinado para reconhecer padrões em uma gama mais ampla de moléculas, agora incluindo também as de plantas?

Ao buscar essa nova fronteira, não apenas expandimos nossos conhecimentos sobre os limites e capacidades desses modelos de aprendizado de máquina, mas também abrimos portas para aplicações mais amplas e buscas por respostas úteis no nosso estudo de produtos naturais. Imagine a possibilidade de identificar e classificar moléculas de plantas, fungos e bactérias com base em características moleculares distintivas, abrindo caminho para avanços significativos em áreas da metabolômica.

Com o conhecimento adquirido e a disponibilidade dos códigos necessários, surge uma oportunidade empolgante para expandir nossas discussões além das classificações binárias. Vamos agora explorar classificações multiclasses, que envolvem a análise e classificação de dados em três ou mais categorias distintas.

Este novo desafio nos levará a explorar padrões mais complexos e nuances sutis nos dados, expandindo assim nossa compreensão e habilidades em análise de dados e aprendizado de máquina. Ao enfrentarmos a tarefa de classificar em múltiplas categorias, seremos desafiados a desenvolver modelos mais sofisticados e robustos, capazes de lidar com a diversidade e a complexidade dos dados.

Ao abraçarmos essa nova abordagem, não apenas ampliamos nosso conjunto de habilidades e conhecimentos, mas também abrimos portas para uma gama ainda maior de aplicações práticas e descobertas científicas. Vamos aproveitar essa oportunidade para explorar os desafios e as possibilidades que surgem com as classificações multiclasses, e assim avançar em nossa jornada rumo à excelência em análise de dados e aprendizado de máquina.

Classificações em Múltiplas Categorias

Separação de moléculas de plantas, fungos e bactérias.

Nas abordagens computacionais anteriores, analisamos um conjunto de dados contendo moléculas de fungos e bactérias. Treinamos diversos algoritmos de aprendizado de máquina para responder à pergunta: “Existem diferenças significativas entre moléculas de fungos e bactérias?” Descobrimos que o modelo Random Forest foi capaz de distinguir essas duas classes com uma precisão superior a 90%.

Agora, ampliando nosso desafio, nos perguntamos: E se adicionarmos moléculas de plantas ao conjunto de dados? Será que o algoritmo manterá o desempenho? Conseguiremos métricas tão interessantes em uma classificação multiclasse? A inclusão de uma nova classe aumenta a complexidade do modelo, já que novos padrões precisam ser identificados.

Para explorar esses dados e treinar os modelos, utilizamos os mesmos descritores químicos calculados pela biblioteca RDKit. É importante ressaltar que, embora estejamos lidando com uma classificação multiclasse, o conceito básico é semelhante ao da classificação binária. Agora, concentramos nossas discussões na interpretação da matriz de confusão gerada para essa nova situação.

OBTENÇÃO DOS DADOS

Utilize o link para download do conjunto de dados moleculares:
https://drive.google.com/drive/folders/1C1yN0d10C_SRreQT8SE95RGDBkc6qL1i?usp=share_link

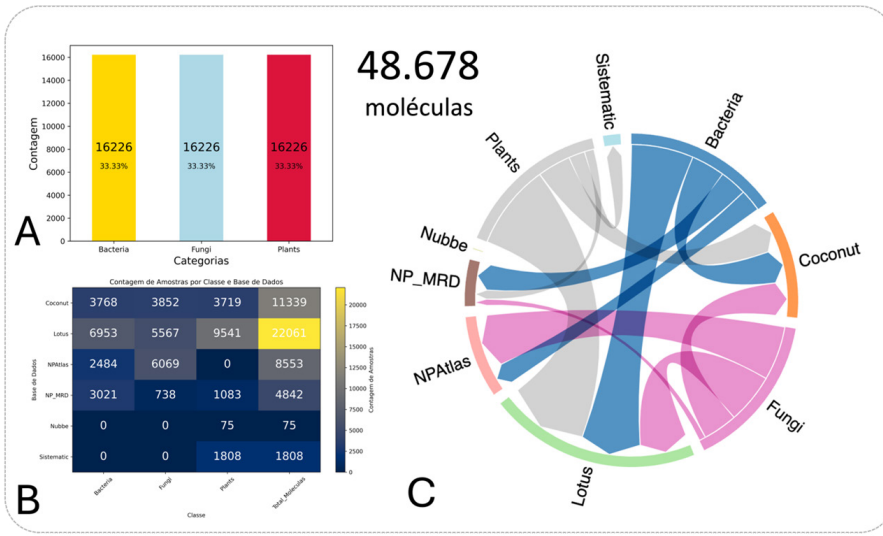
CASO VOCÊ QUEIRA REALIZAR APENAS O PROCESSO DE APRENDIZADO DE MÁQUINA, USE O LINK 1, PARA OBTER O CONJUNTO DE DADOS JÁ LIMPO. CASO CONTRÁRIO, VEJA A DISCUSSÃO PARA PREPARAR OS DADOS ADEQUADAMENTE PARA UM BOM TREINAMENTO DOS MODELOS...

Código 33 – Importação do dataset multiclasse, com 48678 moléculas de plantas, fungos e bactérias.

```
#CÓDIGO 33
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
df = pd.read_csv("dataframe_multiclasse_48678.csv")
```

Os códigos utilizados para processamento dos dados, para uma classificação mutilclasse, segue os mesmos princípios da classificação binária. O conjunto de dados para essa análise multiclasse foram obtidos de bases de dados moleculares bastante consolidadas, como a Lotus6, Coconut7, NPAtlas8, NP_MRD9, entre outras, como mostrada na Figura 52.

Figura 52 - Distribuição entre as classes (A). Quantificação das classes distribuídas pelas bases de dados (B). Gráfico de relações entre classes e base de dados.



6 https://lotus.naturalproducts.net
7 https://coconut.naturalproducts.net
8 https://www.npatlas.org
9 https://np-mrd.org

Após o carregamento do dataframe, você pode realizar uma análise exploratória dos dados seguindo os mesmos procedimentos discutidos nos capítulos anteriores. Inicialmente, fazemos uma transformação das classes, que estão em string, para números. Por que fazemos essa prática? É por capricho? Não. A maioria dos modelos de Machine Learning (especialmente os do scikit-learn, redes neurais, etc.) foi construída sobre operações matemáticas (matrizes, vetores, derivadas).

Strings como “planta” ou “fungo” não têm significado matemático. Já números como 0 e 1 podem ser usados diretamente em cálculos.

Exemplo:

- “ativo” → não dá pra somar, multiplicar, calcular distância
- 1 → dá pra usar em qualquer operação matemática

O Código 34, disponibilizado abaixo, é mais profissional, neste exemplo, é apresentado um fluxo completo de preparação de dados e treinamento de modelos de aprendizado de máquina para um problema de classificação multiclasse. O objetivo é transformar dados brutos em uma representação adequada, selecionar as variáveis mais relevantes e comparar diferentes algoritmos de classificação. Analise com cuidado o algoritmo abaixo, ele é bem completo!

Código 34 - Código para treinamento dos modelos de machine learning visando a separação multiclasse de moléculas de acordo com sua origem.

```
#CÓDIGO 34
from sklearn.preprocessing import LabelEncoder, RobustScaler, label_
binarize
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from       sklearn.ensemble       import       RandomForestClassifier,
GradientBoostingClassifier, ExtraTreesClassifier, AdaBoostClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
# ======================
# LabelEncoder
# ======================
le = LabelEncoder()
y = le.fit_transform(df_desc[“Type”])

# ======================
# Split
# ======================
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# ======================
# Undersampling (apenas treino)
# ======================
rus = RandomUnderSampler(random_state=42)
X_train_res, y_train_res = rus.fit_resample(X_train, y_train)

# ======================
# Scaling
# ======================
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train_res)
X_test_scaled = scaler.transform(X_test)

# ======================
# Converter para DataFrame
# ======================
X_train_scaled = pd.DataFrame(
    X_train_scaled,
    columns=X_train_res.columns,
    index=X_train_res.index
)

X_test_scaled = pd.DataFrame(
    X_test_scaled,
    columns=X_test.columns,
    index=X_test.index
)

# ======================
# Feature Selection (Top 20)
# ======================
```

```
selector = SelectKBest(score_func=f_classif, k=200)

X_train_sel = selector.fit_transform(X_train_scaled, y_train_res)
X_test_sel = selector.transform(X_test_scaled)

mask = selector.get_support()
selected_features = X_train_scaled.columns[mask]

# ======================
# DataFrame final
# ======================
X_train2 = pd.DataFrame(
    X_train_sel,
    columns=selected_features,
    index=X_train_scaled.index
)

X_test2 = pd.DataFrame(
    X_test_sel,
    columns=selected_features,
    index=X_test_scaled.index
)

# ======================
# (Opcional) Binarização para AUC multiclass
# ======================
classes = sorted(set(y_train_res))
y_test_bin = label_binarize(y_test, classes=classes)

# ======================
# Treinar e avaliar modelos
# ======================
modelos = {
    # Lineares
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Ridge Classifier": RidgeClassifier(),

    # Naive Bayes
    "Gaussian NB": GaussianNB(),
    "Bernoulli NB": BernoulliNB(),

    # Árvores
    "Decision Tree": DecisionTreeClassifier(),

    # Ensemble (muito fortes 🔥)
    "Random Forest": RandomForestClassifier(),
    "Extra Trees": ExtraTreesClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "AdaBoost": AdaBoostClassifier(),
```

```
# Vetores de suporte
    "SVM (RBF)": SVC(probability=True),
    "Linear SVM": LinearSVC(),

    # Vizinhos
    "KNN": KNeighborsClassifier(),

    # Estatístico clássico
    "LDA": LinearDiscriminantAnalysis()
}

resultados = {}

for nome, modelo in modelos.items():
    try:
        modelo.fit(X_train2, y_train_res)

        y_pred = modelo.predict(X_test2)

        # Probabilidade (se disponível)
        y_prob = None
        if hasattr(modelo, "predict_proba"):
            y_prob = modelo.predict_proba(X_test2)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='macro')

        resultados[nome] = {
            "Accuracy": acc,
            "F1": f1
        }

    except Exception as e:
        print(f"{nome} falhou: {e}")
```

Codificação das classes (Label Encoding)

Inicialmente, a variável alvo (Type) é convertida de valores categóricos (por exemplo, "planta", "fungo", "bactéria") para valores numéricos inteiros. Isso é feito com o LabelEncoder, que transforma cada classe em um número (por exemplo: 0, 1, 2).

É importante destacar que essa codificação não implica ordem ou hierarquia entre as classes, sendo apenas uma representação numérica necessária para os algoritmos de machine learning.

Separação em treino e teste

- O conjunto de dados é dividido em duas partes:
- **Treinamento (80%):** usado para treinar os modelos
- **Teste (20%)**: usado para avaliar o desempenho

A divisão é feita com train_test_split, utilizando o parâmetro stratify=y, que garante que a proporção das classes seja mantida em ambos os conjuntos. Isso é fundamental em problemas multiclasse.

Balanceamento das classes (Undersampling)

- Em muitos conjuntos de dados, algumas classes aparecem mais frequentemente que outras. Isso pode enviesar os modelos.
- Para corrigir esse problema, é aplicado o método **Random Under Sampling**, que reduz o número de amostras das classes majoritárias no conjunto de treino, equilibrando as classes.

Importante: esse processo é aplicado apenas no conjunto de treinamento, preservando o conjunto de teste original para uma avaliação realista.

Normalização dos dados

- Os dados são então escalados utilizando o RobustScaler, que é menos sensível a outliers do que outros métodos como o StandardScaler.
- Esse passo é importante porque muitos algoritmos (como SVM e KNN) são sensíveis à escala das variáveis.
- O scaler é ajustado (fit) apenas nos dados de treino
- Em seguida, é aplicado (transform) tanto no treino quanto no teste

Conversão para DataFrame

- Após o escalonamento, os dados são convertidos novamente para DataFrame, preservando:
- nomes das variáveis (colunas)
- índices das amostras
- Isso facilita a rastreabilidade e interpretação dos resultados, especialmente na etapa de seleção de variáveis.

Seleção de variáveis (Feature Selection)

- Para reduzir a dimensionalidade e melhorar o desempenho dos modelos, é utilizada a técnica SelectKBest com o teste estatístico f_classif (ANOVA).

- São selecionadas as 200 variáveis mais relevantes (E AQUI VOCÊS PODEM DECIDIR SE ESCOLHEM MENOS. PORQUE QUANTO MAIS VARIÁVEIS, MAIS TEMPO LEVA PARA SER TREINADO).
- O método avalia a relação entre cada variável e a classe alvo
- Esse processo ajuda a:
- reduzir ruído
- diminuir custo computacional
- melhorar a generalização dos modelos

Preparação final dos dados

- Após a seleção, os dados são reorganizados em novos DataFrames (X_train2 e X_test2) contendo apenas as variáveis selecionadas.
- Além disso, opcionalmente, a variável alvo do conjunto de teste é binarizada (label_binarize) para possibilitar cálculos de métricas como AUC em problemas multiclasse.

Treinamento dos modelos

- Diversos algoritmos de classificação são avaliados, incluindo:
- Modelos lineares (Logistic Regression, Ridge)
- Naive Bayes
- Árvores de decisão
- Métodos ensemble (Random Forest, Gradient Boosting, Extra Trees, AdaBoost)
- Máquinas de vetor de suporte (SVM)
- KNN
- LDA

Essa diversidade é importante porque diferentes algoritmos podem ter desempenhos distintos dependendo da estrutura dos dados.

Avaliação dos modelos

- Para cada modelo, são calculadas duas métricas principais:
- **Accuracy (Acurácia):** proporção de acertos totais
- **F1-score (macro):** média do F1 entre as classes, tratando todas igualmente

- O uso do F1-score macro é especialmente importante em problemas com múltiplas classes, pois evita que classes majoritárias dominem a métrica.

Tratamento de erros

- O treinamento de cada modelo é envolvido em um bloco try/except, garantindo que, caso algum algoritmo falhe, o restante do processo continue normalmente.

Notem que esse pipeline representa uma abordagem robusta e bem estruturada para problemas de classificação em quimioinformática (ou áreas similares), incluindo:

- tratamento de desbalanceamento
- normalização adequada
- seleção de variáveis
- comparação de múltiplos modelos
- Esse tipo de fluxo é essencial para garantir resultados confiáveis, reprodutíveis e cientificamente consistentes.

A partir dele você consegue treinar qualquer conjunto de dados fazendo as adaptações necessárias ao seu caso.

A Tabela 7 apresenta o ranqueamento dos modelos avaliados no processo de treinamento para a classificação multiclasse. Observa-se que o algoritmo Extra Trees obteve o melhor desempenho, alcançando uma acurácia superior a 90% na identificação de padrões associados a moléculas de plantas, fungos e bactérias.

Diante desse resultado, o modelo pode ser considerado altamente eficiente para a nossa tarefa proposta. Assim, uma vez treinado, é possível salvá-lo para uso posterior, evitando a necessidade de repetir todo o processo de treinamento – etapa que pode ser computacionalmente custosa e demorada. Dessa forma, o modelo previamente treinado pode ser simplesmente carregado e aplicado na classificação de novas moléculas, tornando o fluxo de análise mais ágil e eficiente. Vou mostrar para vocês como fazer isso.

Tabela 7 - Desempenho dos modelos de classificação

Modelo	Accuracy	F1-score
Extra Trees	0.901820	0.901848
Random Forest	0.897810	0.897809
Decision Tree	0.812481	0.812436
KNN	0.804359	0.804371
Gradient Boosting	0.780508	0.780583
LDA	0.729721	0.731361
Ridge Classifier	0.728076	0.727612
AdaBoost	0.655392	0.656204
Logistic Regression	0.647373	0.645544
Linear SVM	0.590316	0.589766
Bernoulli NB	0.503547	0.502478
Gaussian NB	0.376272	0.263221
SVM (RBF)	0.363319	0.231095

Explorando o melhor modelo para classificação multiclasse

Dessa forma, selecionamos o modelo Extra Trees, podemos treinar apenas ele para que possamos salvá-lo. Observe o Código 35 como isso deve ser feito:

Código 35 - Código para utilizar do algoritmo Extra Trees Classifier, utilizado pois apresentou o melhor desempenho na comparação com os demais.

```
#CÓDIGO 35
from sklearn.ensemble import ExtraTreesClassifier
modelo = ExtraTreesClassifier(
    n_estimators=200,
    random_state=42,
    class_weight='balanced'
)
modelo.fit(X_train2, y_train_res)
```

Assim, o modelo desejado já foi treinado e você pode explorar um pouco das métricas a ele associadas, sobretudo utilizando a validação cruzada para testar a robustez do seu modelo, verifique no Código 36.

Código 36 – Treinamento do modelo Extra Trees utilizando validação cruzada.

```
#CÓDIGO 36
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report
target_names = [“planta”, “fungo”, “bactéria”]

y_pred_cv = cross_val_predict(modelo, X_train2, y_train_res, cv=10)

print(classification_report(
    y_train_res,
    y_pred_cv,
    target_names=target_names
))
```

Você terá o resultado abaixo como um resumo das métricas do modelo:

Tabela 8 - Relatório de classificação do modelo Extra Trees

Classe	Precision	Recall	F1-score	Support
Planta	0.89	0.90	0.90	12946
Fungo	0.89	0.88	0.88	12946
Bactéria	0.92	0.92	0.92	12946
Accuracy	—	—	0.90	38838
Macro Avg	0.90	0.90	0.90	38838
Weighted Avg	0.90	0.90	0.90	38838

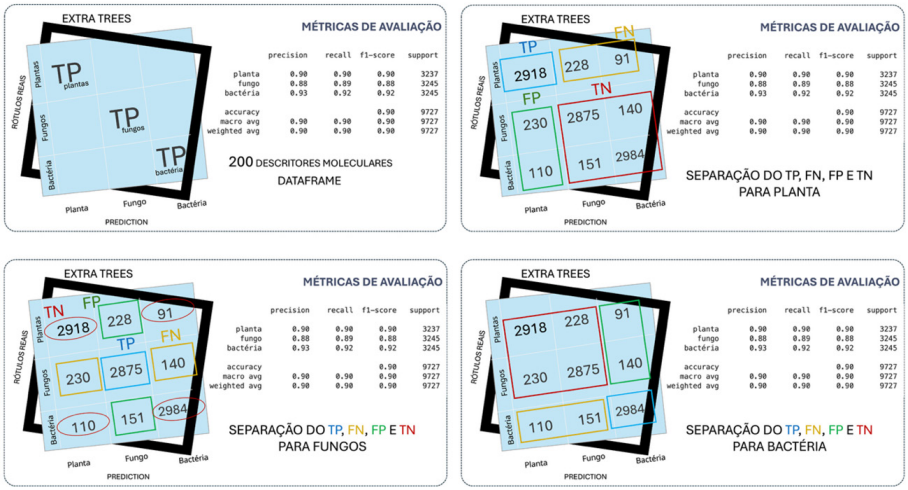
A Tabela 8 é o relatório de desempenho do modelo Extra Trees na tarefa de classificação multiclasse. As métricas precision, recall e F1-score são apresentadas para cada classe (planta, fungo e bactéria), juntamente com o número de amostras (support). Observa-se um desempenho equilibrado entre as classes, com destaque para a classe “bactéria”, que apresentou os maiores valores em todas as métricas. A acurácia global do modelo foi de 90%, indicando alta capacidade preditiva desse modelo.

Utilizando o mesmo conceito, segundo o Código 37, é possível avaliar o desempenho do modelo para dados que ele não viu, ou seja, utilizando os dados de testagem.

Código 37 - Análise de desempenho do modelo selecionado.

```
#CÓDIGO 37
from sklearn.metrics import classification_report
target_names = ["planta", "fungo", "bactéria"]
y_pred = modelo.predict(X_test2)
print(classification_report(y_test,    y_pred,    target_names=target_
names))
```

Figura 53 - Identificação dos grupos de falsos positivos, falsos negativos, verdadeiros negativos e verdadeiros positivos utilizados para multiclasses.



Classificação Multiclasse e Interpretação da Matriz de Confusão

Em problemas de classificação multiclasse, o objetivo é atribuir cada amostra a uma entre três ou mais categorias. No contexto apresentado, o modelo foi treinado para classificar moléculas em três classes distintas: planta, fungo e bactéria.

Uma das principais ferramentas para avaliar o desempenho de modelos multiclasse é a matriz de confusão. Nessa matriz, as linhas representam os rótulos reais e as colunas representam as predições do modelo. Os valores na diagonal principal correspondem às classificações corretas, enquanto os demais representam erros.

Abordagem One-vs-Rest (Um contra Todos)

- Para calcular métricas como precisão, recall e especificidade em cenários multiclasse, utiliza-se frequentemente a abordagem one--vs-rest (OvR). Nessa estratégia, cada classe é analisada individualmente como se fosse um problema binário:
- A classe de interesse é considerada positiva
- Todas as demais classes são agrupadas como negativas

Exemplo: Classe “Planta”

- Ao analisar a classe planta:
- **TP (True Positive):** amostras de plantas corretamente classificadas como plantas
- **FN (False Negative):** plantas classificadas como fungos ou bactérias
- **FP (False Positive):** fungos ou bactérias classificados incorretamente como plantas
- **TN (True Negative):** fungos e bactérias corretamente classificados
- Essa decomposição permite aplicar diretamente as métricas clássicas:
- Precisão → qualidade das predições positivas
- Recall → capacidade de encontrar todos os positivos
- Especificidade → capacidade de rejeitar negativos

Generalização para as demais classes

O mesmo raciocínio é aplicado para fungos e bactérias. A cada nova análise:

- A classe em foco passa a ser a “positiva”
- As demais são tratadas como “negativas”

Isso gera diferentes valores de **TP**, **FP**, **FN** e **TN** para cada classe, mesmo utilizando a mesma matriz de confusão.

Interpretação dos resultados

Observa-se que os valores na diagonal (por exemplo, 2918, 2875 e 2984) representam os acertos do modelo para cada classe. Já os valores fora

da diagonal indicam os tipos de erro, permitindo identificar onde o modelo confunde classes (por exemplo, plantas sendo classificadas como fungos).

A partir dessa análise, é possível:

- Avaliar o desempenho individual por classe
- Identificar padrões de erro
- Ajustar o modelo ou os dados para melhorar a classificação

A classificação multiclasse exige uma análise mais detalhada do que problemas binários. A utilização da matriz de confusão aliada à abordagem one-vs-rest permite decompor o problema e aplicar métricas tradicionais de forma consistente, proporcionando uma avaliação completa e interpretável do modelo que pode resolver o problema proposto.

Agora que o problema de classificação multiclasse foi compreendido e o modelo de melhor desempenho já foi devidamente treinado, não há necessidade de repetir todo o processo de treinamento. Para otimizar tempo e recursos computacionais, o modelo pode ser salvo e reutilizado sempre que houver a necessidade de classificar novas moléculas. Para isso, siga o Código 38 para salvar seu modelo já treinado:

Código 38 - Código para salvamento dos dados e modelo já treinado.

```
#CÓDIGO 38
import joblib
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(modelo, 'model.pkl')
joblib.dump(selector, 'selector.pkl')
joblib.dump(X_train_scaled.columns.tolist(), "colunas.pkl")
joblib.dump(le, "label_encoder.pkl")
```

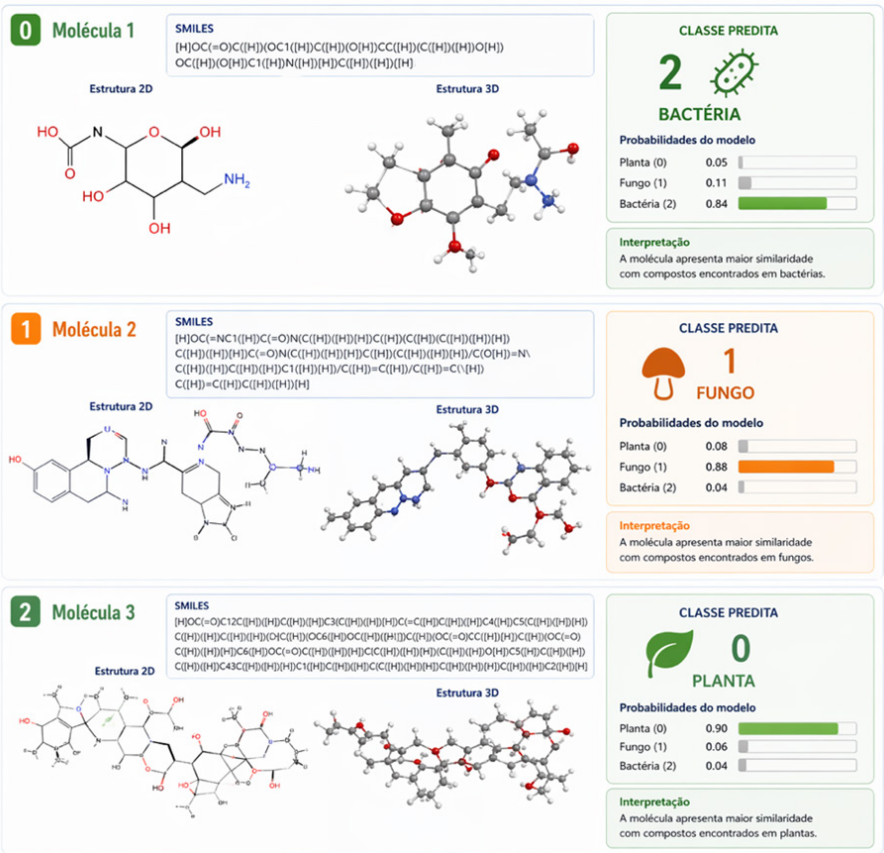
Em seguida, deve-se construir um conjunto de dados contendo os SMILES das moléculas que serão avaliadas. O Código 39 mostra a construção de um dataset com três amostras nunca vistas pelo modelo, além de fazer a chamada dos dados salvos e entrega de um dataframe com os SMILES e a classificação de cada molécula.

Código 39 - Construção de um pequeno dataset para testagem dos modelos.

```
#CÓDIGO 39
import pandas as pd
smiles   =   ["[H]OC(=O)C([H])(OC1([H])C([H])(O[H])C([H])(C([H])([H])
O[H])OC([H])(O[H])C1([H])N([H])[H])C([H])([H])[H]", "[H]OC(=NC1([H])
C(=O)N(C([H])([H])[H])C([H])(C([H])(C([H])([H])[H])C([H])([H])[H])
C(=O)N(C([H])([H])[H])C([H])(C([H])([H])[H])/C(O[H])=N\C([H])([H])
C([H])([H])C1([H])[H])/C([H])=C([H])/C([H])=C(\[H])C([H])=C([H])
C([H])([H])[H]",    "[H]OC(=O)C12C([H])([H])C([H])([H])C3(C([H])([H])
[H])C(=C([H])C([H])([H])C4([H])C5(C([H])([H])[H])C([H])([H])C([H])
([H])C([H])(OC6([H])OC([H])([H])C([H])(OC(=O)C([H])([H])[H])C([H])
(OC(=O)C([H])([H])[H])C6([H])OC(=O)C([H])([H])[H])C(C([H])([H])
[H])(C([H])([H])O[H])C5([H])C([H])([H])C([H])([H])C43C([H])([H])
[H])C1([H])C([H])([H])C(C([H])([H])[H])(C([H])([H])[H])C([H])([H])
C2([H])[H]"]

df_teste = pd.DataFrame({'Smiles': smiles})
import joblib
import numpy as np
from descriptors import calc_descriptors

# 1. descritores
X_new = calc_descriptors(df_teste)

# 2. alinhar colunas
cols = joblib.load('colunas.pkl')
X_new = X_new.reindex(columns=cols, fill_value=0)

# 3. robustez
X_new = X_new.replace([np.inf, -np.inf], 0)
X_new = X_new.fillna(0).astype(float)

# 4. scaler
scaler = joblib.load('scaler.pkl')
X_new_scaled = scaler.transform(X_new)

# 5. selector
selector = joblib.load('selector.pkl')
X_new_sel = selector.transform(X_new_scaled)

# 6. modelo
model = joblib.load('model.pkl')
pred = model.predict(X_new_sel)

# 7. transformar para string
le = joblib.load("label_encoder.pkl")
pred_str = le.inverse_transform(pred)

# 8. salvar resultado
df_teste['predicao'] = pred_str
df_teste
```

A Figura 54 apresenta um infográfico com os resultados obtidos pelo modelo Extra Trees, o qual alcançou aproximadamente 90% de acurácia na classificação de moléculas provenientes de plantas, fungos e bactérias. Considerando as três moléculas submetidas à predição, observa-se que a molécula de índice 0 foi classificada como de origem bacteriana; a de índice 1 apresentou características associadas a fungos; enquanto a molécula de índice 2 exibiu padrões moleculares compatíveis com plantas.

Figura 54 - Predição de moléculas-teste propostas pelo usuário. Moléculas que não estavam no conjunto de treino e nem de testes.



Se você chegou até esta seção do livro, já possui os conhecimentos necessários para aplicar seus próprios dados e propor soluções utilizando aprendizado de máquina baseado em descritores químicos.

No próximo estudo de caso, o objetivo será desenvolver um modelo capaz de classificar moléculas quanto ao seu potencial de atuação contra

diferentes arbovírus, especificamente dengue, zika e chikungunya. Esse tipo de abordagem é extremamente relevante para nós que exploramos produtos naturais dentro do contexto da descoberta de fármacos, pois permite analisar rapidamente novas estruturas químicas e identificar aquelas com maior probabilidade de atividade biológica.

Com um modelo devidamente treinado e validado, e apresentando métricas satisfatórias, torna-se possível utilizá-lo para testar novas moléculas, auxiliando na triagem inicial de candidatos promissores para esses alvos terapêuticos.

CASO DE ESTUDO: TREINAMENTO DE MODELO DE MACHINE LEARNING VISANDO IDENTIFICAR MOLÉCULAS CONTRA DENGUE, ZIKA OU CHIKUNGUNYA

Neste ponto, é fundamental destacar o papel central das plataformas moleculares na obtenção dos conjuntos de dados utilizados no treinamento de modelos de aprendizado de máquina. Essas plataformas constituem a base de todo o processo, uma vez que disponibilizam informações estruturadas, confiáveis e em larga escala sobre compostos químicos e suas atividades biológicas – sem as quais o desenvolvimento de modelos preditivos seria inviável.

Para a condução deste estudo de caso, os dados foram obtidos a partir da plataforma Cortellis[10]. No interior dessa base, foram realizadas buscas direcionadas às doenças dengue, zika e chikungunya. Em seguida, aplicou-se um critério de filtragem com o objetivo de selecionar apenas pequenas moléculas, considerando a faixa de massa molecular até 750 Daltons, garantindo maior consistência e relevância para o treinamento desse modelo que pudesse solucionar o nosso problema: reconhecer padrões de moléculas associadas à essas três arboviroses. Assim, ao testar uma nova molécula, podemos averiguar se ela tem mais semelhança com alguma dessas classes e assim, delimitar nosso estudo em busca de novos fármacos.

Sobre os dados do modelo.

Após a exploração da plataforma Cortellis, com foco na obtenção de moléculas associadas às três classes de interesse — dengue, zika e chikungunya —, foi possível consolidar um conjunto de dados adequado para o início do cálculo de descritores moleculares.

Entretanto, observa-se que o dataset é desbalanceado, uma vez que a classe dengue apresenta maior representatividade em relação às demais. A distribuição das moléculas está apresentada a seguir:

- Dengue: 487

10 https://access.clarivate.com/login?app=cortellis

- Zika: 250
- Chikungunya: 186

Após a obtenção dos datasets contendo moléculas associadas às doenças de interesse (dengue, zika e chikungunya), torna-se necessário realizar um processo de limpeza, padronização e integração dos dados. Essa etapa é fundamental para garantir a qualidade das informações que serão utilizadas nos modelos de aprendizado de máquina.

Para isso, utilizamos a biblioteca RDKit, amplamente empregada em quimioinformática para manipulação de estruturas químicas, juntamente com o pandas, responsável pela organização dos dados em tabelas. Para isso, você pode utilizar o Código 40.

Código 40 - Código para classificação multiclasse de moléculas aplicadas contra dengue, zika e chikungunya.

```
#CÓDIGO 40
import pandas as pd
from rdkit import Chem
import openpyxl
# =========================
# 1. Função de limpeza
# =========================
from rdkit import Chem

from rdkit import Chem
from rdkit import RDLogger

#  silencia mensagens do RDKit
RDLogger.DisableLog('rdApp.*')

def process_smiles(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None

    try:
        Chem.SanitizeMol(mol)
        return Chem.MolToSmiles(mol)
    except:
        return None

# =========================
# 2. Carregar datasets
# =========================
df_dengue = pd.read_excel("dengue.xlsx")
```

```
df_zika = pd.read_excel("zika.xlsx")
df_chik = pd.read_excel("chikungunya.xlsx")


# =========================
# 3. Limpar SMILES
# =========================
def clean_df(df):
    df["smiles_clean"] = df["Smiles"].apply(process_smiles)
    df = df.dropna(subset=["smiles_clean"])
    df = df.drop_duplicates(subset=["smiles_clean"])
    return df

df_dengue = clean_df(df_dengue)
df_zika = clean_df(df_zika)
df_chik = clean_df(df_chik)


# =========================
# 4. Organizar por tamanho
# =========================
datasets = [
    ("dengue", df_dengue),
    ("zika", df_zika),
    ("chik", df_chik),
]

datasets = sorted(datasets, key=lambda x: len(x[1]))


# =========================
# 5. Remover duplicados ENTRE datasets
# =========================
final_data = {}

used_smiles = set()

for name, df in datasets:
    df_clean = df[~df["smiles_clean"].isin(used_smiles)].copy()
    used_smiles.update(df_clean["smiles_clean"])
    final_data[name] = df_clean


# =========================
# 6. Adicionar rótulos
# =========================
final_data["dengue"]["label"] = "dengue"
final_data["zika"]["label"] = "zika"
final_data["chik"]["label"] = "chikungunya"
```

```
# =========================
# 7. Juntar tudo
# =========================
df_all = pd.concat(final_data.values(), ignore_index=True)


# =========================
# 8. Conferir
# =========================
print(df_all[“label”].value_counts())
```

Parte 1 do código 17: Limpeza e Padronização de SMILES

Inicialmente, definimos uma função responsável por processar os SMILES:

- Converte a string SMILES em uma molécula
- Verifica se a estrutura é válida
- Realiza a sanitização química
- Retorna uma versão padronizada do SMILES
- Caso a molécula seja inválida ou apresente erro, ela é descartada.

Por que isso é importante? SMILES inválidos ou inconsistentes podem comprometer completamente o cálculo de descritores moleculares e, consequentemente, o desempenho do modelo.

Parte 2 do código 17: Carregamento dos Datasets

Os dados são carregados a partir de arquivos Excel, cada um representando uma classe:

- Dengue
- Zika
- Chikungunya

Cada arquivo contém uma coluna chamada “Smiles”, que armazena as estruturas moleculares.

Parte 3 do código 17: Remoção de Dados Problemáticos

Em seguida, aplicamos um processo de limpeza em cada dataset:

- Remoção de moléculas inválidas
- Exclusão de valores nulos
- Eliminação de duplicatas dentro de cada classe

Essa etapa garante que cada molécula seja única e quimicamente válida.

Parte 4 do código 17: Organização Estratégica dos Dados

Os datasets são organizados com base no número de moléculas, do menor para o maior.

Por que isso foi feito? Isso permite priorizar as classes menores durante a remoção de duplicatas entre datasets, evitando que classes com menos dados sejam prejudicadas.

Parte 5 do código 17: Remoção de Duplicatas Entre Classes

Um ponto crítico deste processo é garantir que uma mesma molécula não apareça em mais de uma classe.

Para isso:

- Criamos um conjunto (set) para armazenar SMILES já utilizados
- Percorremos os datasets em ordem crescente de tamanho
- Removemos moléculas que já apareceram em classes anteriores

Resultado: Cada molécula passa a pertencer exclusivamente a uma única classe.

Parte 6 do código 17: Rotulagem dos Dados

Após a limpeza, adicionamos uma coluna chamada “label”, que identifica a qual classe cada molécula pertence:

- “dengue”
- “zika”
- “chikungunya”

Essa informação será essencial para o treinamento supervisionado dos modelos.

Parte 7 do código 17: Integração dos Datasets

Todos os dados são então combinados em um único DataFrame, formando o dataset final que será utilizado nas próximas etapas.

Parte 8 do código 17: Verificação Final

Por fim, realizamos uma contagem das classes para verificar a distribuição dos dados:

print(df_all[“label”].value_counts())

Essa etapa permite identificar possíveis desbalanceamentos, que podem impactar o desempenho do modelo.

O algoritmo apresentado anteriormente tem como objetivo consolidar o conjunto de dados, garantindo que as moléculas estejam limpas, padronizadas e sem duplicações entre as classes. No entanto, mesmo após esse processamento, o dataset ainda permanece desbalanceado.

Com os dados devidamente organizados, o próximo passo consiste no cálculo dos descritores moleculares, que serão utilizados como variáveis de entrada para os modelos de aprendizado de máquina. Para isso, utilizaremos o script descriptors.py, previamente desenvolvido e apresentado em exemplos anteriores.

A execução desse script é simples e pode ser realizada conforme ilustrado no Código 41, bastando realizar sua chamada sobre o dataset consolidado.

Código 41 - Código para chama externa do algoritmo de cálculo de descritores moleculares.

```
#CÓDIGO 41
from descriptors import calc_descriptors
df_desc = calc_descriptors(df_all)
df_desc = df_desc.dropna()
df_desc.reset_index(drop=True, inplace=True)
```

Sequencialmente, precisamos selecionar apenas os descritores que serão utilizados para treinamento do modelo, ou seja, apenas as colunas que apresentam valores numéricos que podem participar dos cálculos dos modelos. Com isso, rodamos o Código 42:

Código 42 - Seleção das colunas numéricas que serão utilizadas pelo treinamento do modelo.

```
#CÓDIGO 42
X = df_desc.select_dtypes(include='number')
X = X.drop(columns=['Entry Number'])
```

A partir daqui, temos o dataset ideal para processamentos necessários antes de inseri-los nos treinamentos dos modelos. O Código 43 serve como um pipeline consolidado para que você possa treinar todos os modelos no qual os descritores moleculares previamente calculados são utilizados para treinar e avaliar diferentes algoritmos de classificação. O objetivo é construir modelos capazes de distinguir moléculas associadas às classes dengue, zika e chikungunya.

Código 43 - Pipeline para estudos de comparação entre diferentes modelos de machine learning treinados para separar moléculas aplicadas à dengue, zika e chikungynia

```
#CÓDIGO 43
from imblearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.feature_selection import SelectKBest, f_classif
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import LabelEncoder, RobustScaler, label_
binarize
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from       sklearn.ensemble       import       RandomForestClassifier,
GradientBoostingClassifier, ExtraTreesClassifier, AdaBoostClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# ======================
# LabelEncoder
# ======================
le = LabelEncoder()
y = le.fit_transform(df_desc["label"])

# ======================
# Modelos
# ======================
SEED = 42

modelos = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_
state=SEED),
    "Ridge Classifier": RidgeClassifier(),
    "Gaussian NB": GaussianNB(),
    "Bernoulli NB": BernoulliNB(),
    "Decision Tree": DecisionTreeClassifier(random_state=SEED),
    "Random Forest": RandomForestClassifier(random_state=SEED),
    "Extra Trees": ExtraTreesClassifier(random_state=SEED),
   "Gradient Boosting": GradientBoostingClassifier(random_state=SEED),
    "AdaBoost": AdaBoostClassifier(random_state=SEED),
    "SVM (RBF)": SVC(probability=True),
    "Linear SVM": LinearSVC(),
```

```
    "KNN": KNeighborsClassifier(),
    "LDA": LinearDiscriminantAnalysis()
}

# ======================
# CV
# ======================
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# ======================
# Avaliação
# ======================
resultados = {}

for nome, modelo in modelos.items():
    try:
        pipe = Pipeline([
            ("sampler", RandomUnderSampler(random_state=42)),
            ("scaler", RobustScaler()),
            ("selector", SelectKBest(score_func=f_classif, k=200)),
            ("model", modelo)
        ])

        scores = cross_validate(
            pipe,
            X,
            y,
            cv=cv,
            scoring=["accuracy", "f1_macro"],
            n_jobs=-1
        )

        resultados[nome] = {
            "Accuracy_mean": scores["test_accuracy"].mean(),
            "Accuracy_std": scores["test_accuracy"].std(),
            "F1_mean": scores["test_f1_macro"].mean(),
            "F1_std": scores["test_f1_macro"].std()
        }

    except Exception as e:
        print(f"{nome} falhou: {e}")
```

Descrição metodológica do pipeline de aprendizado de máquina

Inicialmente, os rótulos das classes são convertidos em valores numéricos por meio do LabelEncoder, permitindo sua utilização pelos algoritmos

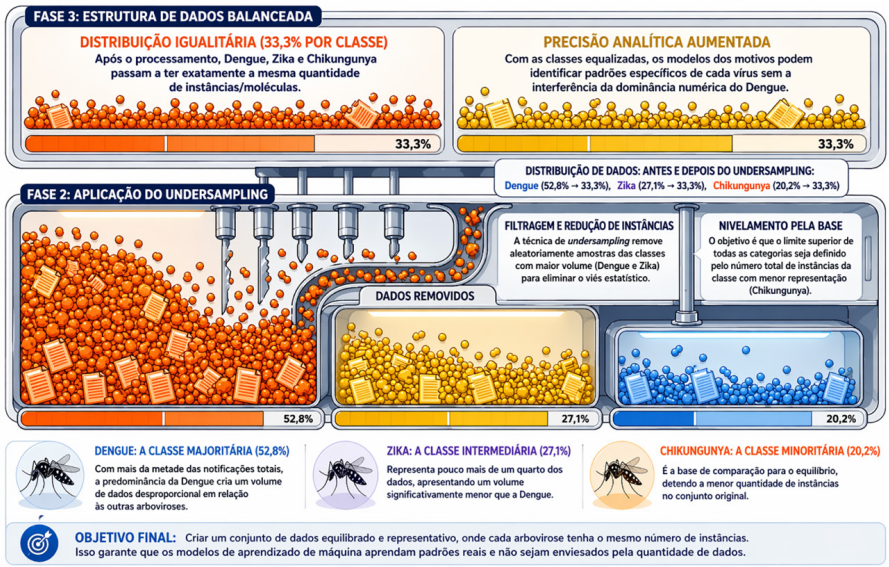
de aprendizado de máquina. Em seguida, define-se um conjunto diversificado de modelos de classificação, incluindo abordagens lineares, baseadas em árvores, métodos de ensemble, algoritmos probabilísticos e técnicas baseadas em distância.

Para garantir uma avaliação robusta e evitar viés decorrente de uma única divisão dos dados, emprega-se a técnica de validação cruzada estratificada por meio do StratifiedKFold. Esse método divide o conjunto de dados em múltiplos subconjuntos (folds), preservando a proporção das classes em cada divisão, o que é particularmente importante em cenários com desbalanceamento.

O processo de modelagem é estruturado utilizando um pipeline do Pipeline, o que garante que todas as etapas de pré-processamento sejam aplicadas de forma consistente e sem vazamento de dados. Dentro desse pipeline, são realizadas as seguintes etapas:

- **Balanceamento de classes:** Utiliza-se o RandomUnderSampler para reduzir o número de amostras das classes majoritárias, promovendo equilíbrio entre as classes apenas nos dados de treino de cada fold, como demonstrado na **Erro! Fonte de referência não encontrada..**

Figura 55 - Esquema representativo e didático do funcionamento e importância da técnica de undersampling.



- **Escalonamento dos dados:** Aplica-se o RobustScaler, que é robusto à presença de outliers, pois utiliza a mediana e o intervalo interquartil para normalização.
- **Seleção de características:** Emprega-se o SelectKBest com o teste estatístico ANOVA (*f_classif*), selecionando as 200 características mais relevantes para o problema de classificação.

A avaliação dos modelos é realizada por meio da função cross_validate, que executa o pipeline completo em cada fold da validação cruzada. São utilizadas como métricas a acurácia (*accuracy*) e o F1-score macro (f1_macro), sendo esta última especialmente relevante em cenários com classes desbalanceadas, pois considera igualmente o desempenho em todas as classes.

Por fim, para cada modelo, são calculadas a média e o desvio padrão das métricas ao longo dos folds, fornecendo uma estimativa robusta tanto do desempenho quanto da estabilidade do modelo. Esses resultados permitem a comparação criteriosa entre diferentes algoritmos, auxiliando na seleção do modelo mais adequado para o problema estudado.

Tabela 9 - Resultados do treinamento dos modelos de machine learning para separação entre as classes de dengue, zika e chikungunya.

Ranking	Modelo	Accuracy (média)	Accuracy (desvio)	F1-score (média)
1º	**Extra Trees**	**0.7902**	0.0443	0.7828
2º	Random Forest	0.7467	0.0424	0.7383
3º	Gradient Boosting	0.7380	0.0542	0.7297
4º	Linear SVM	0.7315	0.0418	0.7205
5º	LDA	0.7304	0.0350	0.7186
6º	Ridge Classifier	0.7283	0.0271	0.7163
7º	Logistic Regression	0.7065	0.0466	0.6937
8º	Decision Tree	0.6380	0.0432	0.6207
9º	KNN	0.6228	0.0563	0.6135
10º	AdaBoost	0.6130	0.0647	0.5950
11º	Gaussian NB	0.5467	0.0456	0.5459
12º	Bernoulli NB	0.5315	0.0499	0.5083

Discutimos anteriormente que no desenvolvimento de modelos de aprendizado de máquina, uma etapa fundamental consiste na comparação entre diferentes algoritmos para identificar aquele que melhor se adapta ao problema em estudo. No código 43 foi apresentado essa comparação, a qual foi realizada de forma sistemática utilizando validação cruzada estratificada (Stratified K-Fold Cross Validation) aplicada sobre todo o conjunto de dados disponível, e não apenas sobre uma divisão prévia em treino e teste.

Inicialmente, a variável alvo (y) é codificada numericamente por meio do LabelEncoder, permitindo que os algoritmos de classificação possam processar corretamente as classes. Em seguida, define-se um conjunto diversificado de modelos, incluindo desde métodos lineares, como Regressão Logística, até algoritmos mais complexos, como Random Forest, Gradient Boosting e Support Vector Machines (SVM). Essa diversidade é importante para garantir uma análise abrangente do espaço de soluções.

Para garantir uma avaliação robusta, utiliza-se o método StratifiedK-Fold, que divide os dados em 10 subconjuntos (folds), preservando a proporção das classes em cada divisão. Esse cuidado é essencial, especialmente em problemas com possível desbalanceamento de classes.

A Tabela 9 mostra o ranking dos modelos treinados. Mais uma vez temos notado que para problemas envolvendo descritores moleculares, os modelos de árvores têm funcionado muito bem, conseguindo identificar os padrões associados a essas moléculas. Assim, selecionaremos o modelo Extra Trees Classifier (através do Código 44) para ser o modelo preditivo para reconhecer padrões de estruturas moleculares que tem sido estudas para diferentes arboviroses.

Código 44 - Pipeline para construção do modelo Extra Trees Classifier, selecionado por ter melhor desempenho na classificação multiclasse de moléculas aplicadas à dengue, zika e chikungunya.

```
from sklearn.model_selection import cross_validate, StratifiedKFold

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

pipe = Pipeline([
    ("sampler", RandomUnderSampler(random_state=42)),
    ("scaler", RobustScaler()),
    ("selector", SelectKBest(score_func=f_classif, k=200)),
    ("model", ExtraTreesClassifier(
        n_estimators=200,
```

```
        random_state=42,
        class_weight='balanced'
    ))
])

scores = cross_validate(
    pipe,
    X,
    y,
    cv=cv,
    scoring=["accuracy", "f1_macro"]
)

print("Accuracy:", scores["test_accuracy"].mean())
print("F1:", scores["test_f1_macro"].mean())
```

Agora, é possível realizar a separação dos dados em conjuntos de treino e teste, permitindo o re-treinamento do modelo utilizando apenas os dados de treino (X_train) e, posteriormente, sua avaliação com dados inéditos (X_test), ou seja, dados que não foram utilizados durante o processo de aprendizado.

Para isso, utiliza-se o Código 45, que permite obter o relatório de classificação (*classification report*) referente à solução do problema multiclasse.

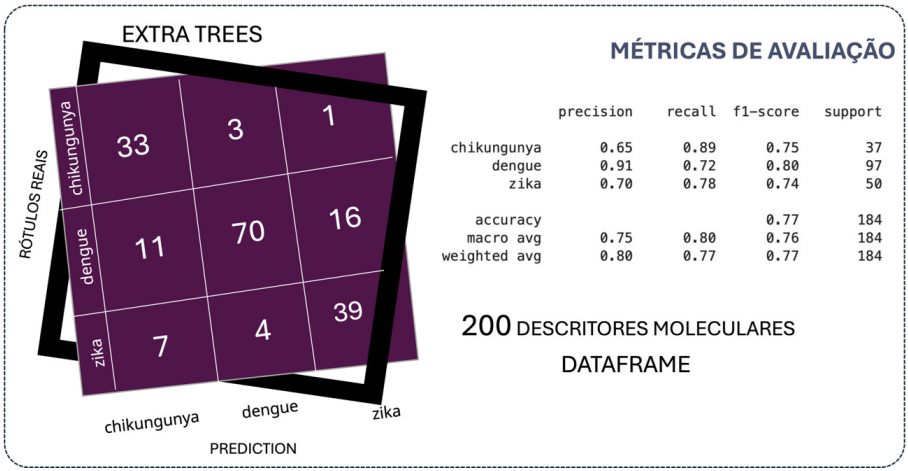
Vale destacar que o modelo apresenta uma acurácia superior a 77%, o que pode ser considerado um bom desempenho, especialmente levando em conta que o conjunto de dados utilizado é relativamente pequeno quando comparado ao conjunto previamente empregado na classificação de plantas, fungos e bactérias.

Espera-se que, com o avanço de novos estudos voltados a esses alvos, novas moléculas possam ser incorporadas ao conjunto de dados. Esse aumento na base de dados tende a tornar o modelo mais robusto, aprimorando sua capacidade de reconhecer padrões e melhorar o desempenho na distinção entre as três classes analisadas.

Código 45 - Classification report para o modelo de Extra Trees Classifier.

```
#CÓDIGO 45
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)

print(classification_report(y_test, y_pred, target_names=le.classes_)
```

Figura 56 – Métricas de desempenho do modelo Extra Trees Classifier para moléculas aplicadas à arboviroses.



O modelo apresentou melhor desempenho na classificação de moléculas associadas à dengue, possivelmente devido à maior quantidade de amostras disponíveis para essa classe. No entanto, observa-se que o recall para chikungunya é superior, indicando maior capacidade do modelo em identificar corretamente essa classe quando presente. Além disso, a matriz de confusão revela uma tendência de confusão entre dengue e zika, sugerindo similaridade nos padrões moleculares dessas classes. Ainda assim, com acurácia global de aproximadamente 77% e desempenho consistente entre as métri-

cas, o modelo pode ser considerado válido, embora melhorias possam ser alcançadas com o aumento e balanceamento do conjunto de dados.

Para finalizar a exploração dos dados, é possível utilizar os Código 38 e 39 para salvar o modelo treinado e aplicá-lo na análise de novas estruturas moleculares. No contexto de produtos naturais, essa abordagem permite investigar se moléculas recém-isoladas ou obtidas por processos de desreplicação apresentam características semelhantes às já associadas às três arboviroses estudadas.

Dessa forma, mesmo com limitações inerentes ao tamanho do conjunto de dados, o modelo desenvolvido já se mostra uma ferramenta útil para triagem inicial de moléculas com potencial atividade biológica.

CONSIDERAÇÕES FINAIS SOBRE A OBRA

A aplicação da inteligência artificial no estudo de moléculas orgânicas tem transformado profundamente a forma como fazemos ciência. Hoje, somos capazes de acessar, processar e interpretar volumes de dados que, há pouco mais de uma década, na época do meu mestrado, por exemplo, eram praticamente inalcançáveis. O avanço do poder computacional, aliado à expansão e ao refinamento dos bancos de dados químicos, abriu novas possibilidades para a descoberta e o desenvolvimento de moléculas com potencial biológico.

Nesse novo contexto, vocês devem ter notado que a Química Orgânica não apenas evoluiu, nós fomos personagens principais em reinventá-la, aceitando encabeçar projetos inovadores, modernos e ousados. Esta obra é um reflexo desse momento de transição, em que ferramentas computacionais e conhecimento químico caminham lado a lado, ampliando nossa capacidade de compreender e explorar a complexidade molecular.

Mais do que apresentar conceitos e algoritmos, este livro busca despertar o interesse pelo universo da quimioinformática – um campo ainda em plena expansão, repleto de desafios e oportunidades. Há muito a ser descoberto, e cada pesquisador pode contribuir de forma única para esse avanço.

Se há uma mensagem final a ser deixada, é esta: você não precisa esperar por grandes estruturas ou bases de dados consolidadas para começar. É possível construir seus próprios conjuntos de dados, formular suas próprias hipóteses e explorar caminhos promissores a partir de suas próprias perguntas.

A ciência avança justamente assim – com curiosidade, iniciativa e a disposição de investigar o desconhecido. Desejo sucesso na exploração dos seus dados orgânicos!

REFERÊNCIAS

BROWN, Frank K. Chemoinformatics: What is it and how does it impact drug discovery? **Annual Reports in Medicinal Chemistry**, v. 33, p. 375–384, 1998.

CAPECCHI, Alice; REYMOND, Jean-Louis. Classifying natural products from plants, fungi or bacteria using the COCONUT database and machine learning. **Journal of Cheminformatics**, v. 13, n. 1, p. 82, 2021. DOI: https://doi.org/10.1186/s13321-021-00559-3

CHEN, Hongming *et al.* The rise of deep learning in drug discovery. **Drug Discovery Today**, 2018. DOI: https://doi.org/10.1016/j.drudis.2018.01.039

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. **The Elements of Statistical Learning.** 2. ed. Springer, 2009. DOI: https://doi.org/10.1007/978-0-387-84858-7

JAMES, Gareth *et al.* **An Introduction to Statistical Learning.** Springer, 2013. DOI: https://doi.org/10.1007/978-1-4614-7138-7

LANDRUM, Greg. RDKit: Open-source cheminformatics. Disponível em: https://www.rdkit.org

LEACH, Andrew R.; GILLET, Valerie J. **An Introduction to Chemoinformatics.** Dordrecht: Springer, 2007. DOI: https://doi.org/10.1007/978-1-4020-6291-9

LIPINSKI, Christopher A. Lead- and drug-like compounds: the rule-of-five revolution. **Drug Discovery Today: Technologies**, 2004. DOI: https://doi.org/10.1016/j.ddtec.2004.11.007

MAYR, Andreas *et al.* Deep learning as a tool for chemists. **Chemical Science**, 2018.

MURPHY, Kevin P. **Machine Learning:** A Probabilistic Perspective. MIT Press, 2012.

NEWMAN, David J.; CRAGG, Gordon M. Natural Products as Sources of New Drugs over the Nearly Four Decades. **Journal of Natural Products**, 2020. DOI: https://doi.org/10.1021/acs.jnatprod.9b01285

SCHNEIDER, Gisbert. Automating drug discovery. **Nature Reviews Drug Discovery**, 2018. DOI: https://doi.org/10.1038/nrd.2017.232

VARNEK, Alexandre; TROPSHA, Alexander (Eds.). **Chemoinformatics Approaches to Virtual Screening**. Royal Society of Chemistry, 2008.

VIEIRA, Rafael; SOUSA, Kally Alves de; CASTRO-GAMBOA, Ian. LUMIOS – Label Using Machine in Organic Samples: A software for dereplication, molecular docking, and combined machine and deep learning. **Expert Systems with Applications**, v. 248, 123447, 2024. DOI: https://doi.org/10.1016/j.eswa.2024.123447

VIEIRA, Rafael; SOUSA, Kally Alves de; SILVA, Givaldo Souza da; SILVA, Dulce Helena Siqueira; CASTRO-GAMBOA, Ian. CHEIC: Chemical Image Classificator. An intelligent system for identification of volatile compounds with potential for respiratory diseases using deep learning. **Expert Systems with Applications**, v. 234, 121178, 2023. DOI: https://doi.org/10.1016/j.eswa.2023.121178

Sobre o Autor

Rafael é um pesquisador apaixonado pela interseção entre a química e a computação. Graduado em Licenciatura em Química pela Unesp de Araraquara, aprofundou-se nesse campo durante seus estudos de pós-graduação na mesma instituição, obtendo, ao longo desse período, os títulos de mestrado e doutorado.

Seu trabalho esteve focado em pesquisas inovadoras relacionadas a produtos naturais, explorando sua química e potencial terapêutico.

Sua trajetória acadêmica se destacou pela aplicação de abordagens computacionais avançadas, incorporando técnicas de aprendizado de máquina, modelagem molecular e análise de dados. Esses recursos ampliaram sua compreensão sobre os produtos naturais bioativos e possibilitaram novos estudos para desenvolvimento de fármacos.

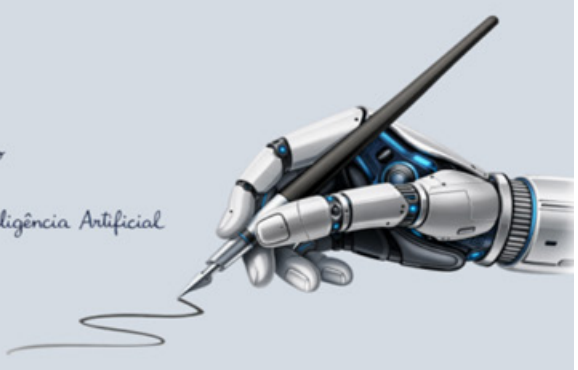
Atualmente, Rafael Vieira é professor de Química no Instituto Federal de Rondônia e continua a expandir o conhecimento humano contribuindo para soluções inovadoras em química medicinal que promovem a saúde e o bem-estar.

Este livro foi escrito como parte de sua linha de pesquisa, visando incentivar novos pesquisadores a explorarem seus dados utilizando ferramentas computacionais e técnicas avançadas de análise.

Rafael Vieira

Formação

- Graduado em Química
- Graduado em Ciência de Dados
- Graduado em Ciência da Computação
- Especialista em Ciência de Dados e Inteligência Artificial
- Mestre em Química Orgânica
- Doutor em Química Orgânica



ÍNDICE REMISSIVO

A

B

C

D

E

F

H

M

O

P

Q

R

T

U

V

AYA EDITORA
2026