CITHA | Inovação e Sustentabilidade na Amazônia

2025

INTELIGÊNCIA ARTIFICIAL

APRENDIZAGEM DE MÁQUINA

(modelos preditivos e suas aplicações práticas em Python)

FOTO DE TARA WINSTEAD

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Inteligência artificial [livro eletrônico] : aprendizagem de máquina (modelos preditivos e suas aplicações práticas em Python) / Eduardo Palhares Júnior...[et al.] ; coordenação Tiago Francisco Andrade Diocesano...[et al.]. -- 1. ed. -- Manaus, AM : Ed. dos Autores, 2025.
PDF

Outros autores: Wenndisson da Silva Souza, Raimundo Fagner Costa, Alexandre Lopes Martiniano, Nivaldo Rodrigues e Silva.
Outros coordenadores: Jaidson Brandão da Costa, Elvican dos Santos Silva, Martinho Correia Barros, Adelino Maia Galvão Filho.
Bibliografia.
ISBN 978-65-01-74472-8

1. Aprendizagem de máquina 2. Ciência da computação 3. Inteligência artificial 4. Python (Linguagem de programação para computadores) 5. Tecnologia I. Júnior, Eduardo Palhares. II. Souza, Wenndisson da Silva. III. Costa, Raimundo Fagner. IV. Martiniano, Alexandre Lopes. V. Silva, Nivaldo Rodrigues e. VI. Diocesano, Tiago Francisco Andrade.

25-308727.0 CDD-006.3

Índices para catálogo sistemático:

1. Inteligência artificial 006.3

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

DOI: 10.5281/zenodo.15333838

Expediente do IFAM

Reitor
Jaime Cavalcante Alves

Pró-Reitor de Administração
Fábio Teixeira Lima

Pró-Reitor de Gestão de Pessoas
Leandro Amorim Damasceno

Pró-Reitora de Ensino
Rosângela Santos da Silva

Pró-Reitora de Extensão
Maria Francisca Morais de Lima

Pró-Reitor de Pesquisa, Pós-Graduação e Inovação
Paulo Henrique Rocha Aride

Diretor Geral do Campus Manaus Distrito Industrial
Nivaldo Rodrigues e Silva

Expediente do Projeto CITHA

Gestores
Nivaldo Rodrigues e Silva
Samirames da Silva Fleury
Alyson de Jesus dos Santos
Maria Cassiana Andrade Braga
Adanilton Rabelo de Andrade

Coordenadores
Tiago Francisco Andrade Diocesano
Jaidson Brandão da Costa
Elcivan dos Santos Silva
Martinho Correia Barros
Adelino Maia Galvão Filho

Expediente de Produção

Autor(es)
Eduardo Palhares Jr.
Wenndisson da Silva Souza
Raimundo Fagner Costa
Alexandre Lopes Martiniano
Nivaldo Rodrigues e Silva

Avaliação Pedagógica
Samirames da Silva Fleury

Diagramadores
Wenndisson da Silva Souza
Eduardo Palhares Jr.
Fabio Serra Ribeiro Couto

Revisores de Texto
Alexandre Lopes Martiniano
Alyson de Jesus dos Santos

Inteligência Artificial — Aprendizagem de Máquina *(modelos preditivos e suas aplicações práticas em Python)*

Autores
Eduardo Palhares Jr.
Wenndisson da Silva Souza
Raimundo Fagner Costa
Alexandre Lopes Martiniano
Nivaldo Rodrigues e Silva

Prefácio por Nivaldo Rodrigues e Silva

Revisão
Alexandre Lopes Martiniano
Alyson de Jesus dos Santos

1ª Edição

Manaus - AM
2025

Prefácio

Vivemos em uma era impulsionada por dados e decisões inteligentes, e o Machine Learning ou Aprendizado de Máquina está no centro dessa transformação. Esta tecnologia tem revolucionado diversos setores ao permitir que sistemas aprendam com os dados e tomem decisões com mínima intervenção humana. Este e-book foi desenvolvido para guiar o leitor por uma jornada estruturada e acessível no universo do Machine Learning, desde os fundamentos teóricos até aplicações práticas em problemas reais.

O conteúdo foi cuidadosamente planejado para atender tanto iniciantes quanto profissionais que desejam aprofundar seus conhecimentos. No primeiro módulo, são apresentados os conceitos essenciais do Machine Learning, incluindo os tipos de aprendizado, as etapas de um projeto e as principais bibliotecas da linguagem Python utilizadas na área. O segundo módulo aprofunda-se em algoritmos supervisionados e não supervisionados, trazendo exemplos práticos que facilitam a compreensão e a aplicação das técnicas.

No terceiro módulo, abordamos a preparação de dados, a avaliação de modelos e a importância da escolha adequada de métricas de desempenho. Já o módulo final propõe um estudo de caso aplicado à área ambiental, demonstrando como modelos de Machine Learning podem contribuir para a previsão de padrões climáticos e a tomada de decisões sustentáveis.

Mais do que um guia técnico, este e-book busca despertar no leitor o senso crítico e a curiosidade para explorar o potencial do Machine Learning. Ao aplicar os conhecimentos adquiridos, o leitor será capaz de transformar dados em soluções inteligentes, contribuindo para um mundo mais eficiente, inovador e sustentável. Que esta leitura inspire descobertas, estimule o pensamento analítico e fortaleça sua jornada no fascinante campo do Aprendizado de Máquina.

Projeto de Capacitação e Interiorização em Tecnologias Habilitadoras na Amazônia - CITHA

O projeto CITHA surge com o objetivo de fortalecer a economia da Amazônia por meio do incentivo ao empreendedorismo local e do desenvolvimento sustentável. Sua proposta é capacitar profissionais e impulsionar a criação de startups voltadas para a bioeconomia, além de apoiar cooperativas locais na melhoria de seus processos produtivos. A implementação de tecnologias inovadoras é uma das estratégias centrais do projeto, visando oferecer soluções eficientes que atendam às necessidades regionais, como a otimização dos recursos naturais e a melhoria da infraestrutura local.

Ao longo de sua execução, o projeto se compromete a integrar os diversos stakeholders, como governos, empresas, ONGs e comunidades, por meio da capacitação da mão de obra local. O objetivo é formar um capital intelectual qualificado, capaz de apoiar uma governança eficiente, promover a inovação e assegurar a sustentabilidade. O CITHA dedica-se à criação de processos internos que incentivem o desenvolvimento de novos métodos e tecnologias, adaptáveis às particularidades do território amazônico.

Em síntese, o projeto CITHA visa criar um ciclo de desenvolvimento que não só incentive o empreendedorismo, mas também promova a modernização das estruturas locais, elevando a qualidade de vida das populações da Amazônia. Focado em áreas como bioeconomia, inovação e transferência de tecnologia, o projeto busca estabelecer um ecossistema mais forte e autossustentável, capaz de responder eficientemente às demandas do mercado e da sociedade.

Lista de Expressões para Enriquecimento de Conteúdo

Este material foi cuidadosamente estruturado para apoiar sua jornada de aprendizado. Ao longo dos capítulos, você encontrará diversas chamadas sinalizadas por ícones especiais, que ajudarão a destacar pontos-chave e enriquecer sua compreensão. Durante a diagramação, esses ícones serão inseridos conforme as indicações dos autores, guiando você para diferentes tipos de conteúdo e atividades que potencializam seu estudo.

Fique Alerta!

Destaque para conceitos, expressões e trechos fundamentais que merecem sua atenção especial para a compreensão do conteúdo.

Iniciando o diálogo...

Espaço para reflexão crítica. Aqui você será convidado(a) a problematizar os temas abordados, relacionando-os com sua experiência e buscando conexões relevantes para aprofundar seu aprendizado.

Conhecendo um pouco mais!

Indicação de fontes complementares, como livros, entrevistas, vídeos, aplicativos, links e outros recursos para ampliar seu conhecimento sobre o tema.

Caso Prático

Aplicação direta do conteúdo em exemplos concretos, para facilitar a fixação e demonstrar a utilidade do que foi aprendido.

Copie e Teste!

Trechos de código prontos para serem copiados e executados, para que você possa experimentar, validar e explorar na prática os conceitos estudados.

Sumário

Capítulo 1

Aprendizagem Supervisionada

1.1 O que é Machine Learning?

Iniciando o diálogo...

Vivemos em um momento em que a inteligência artificial (IA) e o aprendizado de máquina (machine learning) estão cada vez mais presentes nas nossas vidas. Desde aplicativos de redes sociais que recomendam novos amigos, até plataformas de streaming que sugerem filmes e séries com base nas nossas preferências, percebemos como os computadores podem "adivinhar" de maneira surpreendente o que nos interessa. Mas o que está por trás dessas ferramentas tão inteligentes? E por que vale a pena entender esse universo agora? De forma simples, machine learning é a capacidade de um computador aprender automaticamente, sem que seja necessário programar cada passo em detalhes (GÉRON, 2023). Em vez de escrever um código dizendo "faça isso, depois aquilo", fornecemos exemplos de dados para que a máquina "aprenda" um padrão ou uma regra geral.

Segundo Mitchell (1997), o aprendizado de máquina consiste em sistemas que melhoram seu desempenho a partir da experiência. Para entender como isso funciona, imagine que você queira ensinar um computador a reconhecer se uma imagem é de um gato ou de um cachorro. Em vez de programar cada característica (por exemplo, "patas mais compridas", "orelhas pontudas"), você treina o modelo com diversas fotos rotuladas. A máquina, aos poucos, descobre sozinha quais detalhes são mais importantes para diferenciá-las. Quanto mais exemplos ela analisa, melhor fica na tarefa, um processo muito parecido com o aprendizado humano!

Fique Alerta!

Um dos nomes mais reconhecidos na área de IA, o professor Andrew Ng, explora essa ideia no curso *Machine Learning Specialization*, da Universidade de Stanford, disponível na plataforma Coursera. Fica a sugestão para quem deseja aprofundar seus estudos nessa área.

1.1.1 Por que ensinar máquinas a aprender?

Para lidar com grandes volumes de dados

Hoje em dia, geramos dados o tempo todo: quando navegamos na internet, quando medimos a temperatura de um ambiente, ou até mesmo ao usar redes sociais. O volume de

informações é enorme e cresce exponencialmente. Sem a ajuda de algoritmos de machine learning, seria impossível processar, analisar e extrair conclusões de maneira rápida e eficiente. Logo, ensinar máquinas a aprender nos ajuda a encontrar padrões e tomar decisões com base em dados que, sozinhos, não daríamos conta de analisar.

Para personalização e praticidade

Muitos serviços de streaming de música ou vídeo utilizam aprendizado de máquina para sugerir conteúdos de acordo com o nosso gosto. Esse tipo de personalização surge porque sistemas inteligentes analisam quais filmes assistimos, que músicas ouvimos e em que momento trocamos de faixa, para oferecer recomendações cada vez mais certeiras. Desse modo, ensinar máquinas a aprender cria experiências mais práticas e personalizadas, tanto para quem consome conteúdos quanto para quem os produz.

Para inovação em diversos setores

Desde a agricultura até a área da saúde, o aprendizado de máquina traz possibilidades de inovação. Na agricultura, por exemplo, é possível prever pragas ou identificar o melhor momento para irrigar, baseando-se em estatísticas de um grande conjunto de dados. Já na medicina, algoritmos podem auxiliar médicos a detectar precocemente doenças em exames de imagem, contribuindo para diagnósticos mais rápidos e assertivos. Para alunos do Ensino Médio, isso significa compreender desde cedo como a tecnologia pode ser usada para resolver problemas reais e melhorar a qualidade de vida das pessoas.

1.1.2 Uma nova forma de aprender (e ensinar)

O machine learning desperta interesse não só por suas aplicações práticas, mas pela lógica que ele segue. Em vez de dizer explicitamente como resolver uma questão, confiamos em dados e padrões. Esse método é bastante alinhado ao pensamento científico: levantamos hipóteses, testamos com exemplos e checamos se o modelo previu corretamente ou não, esse ciclo de tentativa e erro não só estimula a curiosidade, mas nos ensina a ter uma mente investigativa e metódica.

É muito importante e de valor imensurável compreender essas ideias agora. Saber como trabalhar com dados e algoritmos abre portas em várias carreiras, desde áreas tecnológicas como Ciência de Dados e Engenharia de Software, até campos em que a tomada de decisões baseada em análise de dados é fundamental, como Economia, Biologia e afins.

1.1.3 Conectando teoria e prática

Uma das características mais legais de machine learning é que podemos ver resultados na prática rapidamente. Há ferramentas e bibliotecas em Python, por exemplo, que permitem criar modelos simples para classificar imagens, textos ou até prever valores em tabelas. Ao testar pequenos projetos, como analisar por que algumas músicas recebem mais "likes" do que outras, vocês começam a ter uma noção real de como funciona o ciclo de coleta de dados, treinamento de algoritmo e avaliação de resultados.

Esses experimentos são estimulantes porque mostram o lado prático do que se estuda nas aulas de Matemática, Estatística e até mesmo de Programação. É aí que enxergamos a importância dos conceitos de média, variância, probabilidade e lógica: sem eles, é difícil ajustar um modelo de forma correta.

1.1.4 Por que isso tudo importa?

Entender por que ensinar máquinas a aprender não é só uma curiosidade científica; é preparar-se para o futuro. As transformações tecnológicas nos convidam a ser mais criativos, analíticos e colaborativos. Quanto antes compreendermos esses processos, melhor conseguiremos aplicá-los de maneira ética e construtiva, resolvendo problemas e contribuindo para o desenvolvimento da nossa sociedade.

Vivemos em uma era em que as inovações tecnológicas fazem parte do nosso cotidiano. Aprender machine learning é como revelar os bastidores de muitas dessas tecnologias, desde as recomendações que recebemos online até previsões de clima e trânsito, tudo isso impulsionado por algoritmos treinados com dados. E o mais interessante: vocês podem ser os próximos a desenvolver soluções criativas, responsáveis e transformadoras!

Pense um pouco, na Amazônia, o aprendizado supervisionado pode ser empregado para Monitoramento de espécies, usando fotos ou áudios gravados na floresta, o modelo aprende a reconhecer animais específicos e alertar quando uma espécie rara é detectada; Agricultura de precisão: A partir de variáveis como tipo de solo, umidade e histórico de safra, prevê-se a quantidade de fertilizante ideal, evitando o desperdício; Prevenção de desmatamento: Com imagens de satélite rotuladas, o computador consegue identificar áreas de risco para desmatamento ilegal, ajudando órgãos de fiscalização; E outra infinidade de possibilidades, descubra!

1.2 Aprendizagem Supervisionada

Iniciando o diálogo...

Você já imaginou como um computador consegue “adivinhar” qual animal aparece em uma foto ou prever se vai chover amanhã? No aprendizado supervisionado, fornecemos exemplos rotulados para uma máquina, para que ela possa descobrir padrões e aprender a tomar decisões ou fazer previsões a partir de novos dados (RUSSELL; NORVIG, 2003). É uma das áreas mais conhecidas e baseia-se em técnicas que permitem que sistemas computacionais evoluam com a experiência. E se quisermos que um programa identifique se uma folha pertence a uma planta nativa da Amazônia ou não? Precisamos primeiro mostrar várias folhas, dizendo “Esta é de uma planta amazônica; esta outra não é”. Assim, o modelo “aprende” como reconhecer essas características.

Figura 1.1: Imagens retiradas da Wikimedia Commons: vitória-régia amazônica e do cerrado arnica do campo.

1.2.1 Como funciona o aprendizado supervisionado?

O paradigma supervisionado, de acordo com Mitchell (1997), envolve treinar modelos com exemplos rotulados para aprender funções de mapeamento. Para entendermos melhor como funciona o processo de aprendizado supervisionado, podemos visualizar um ciclo composto por quatro etapas principais, conforme ilustrado na figura a seguir:

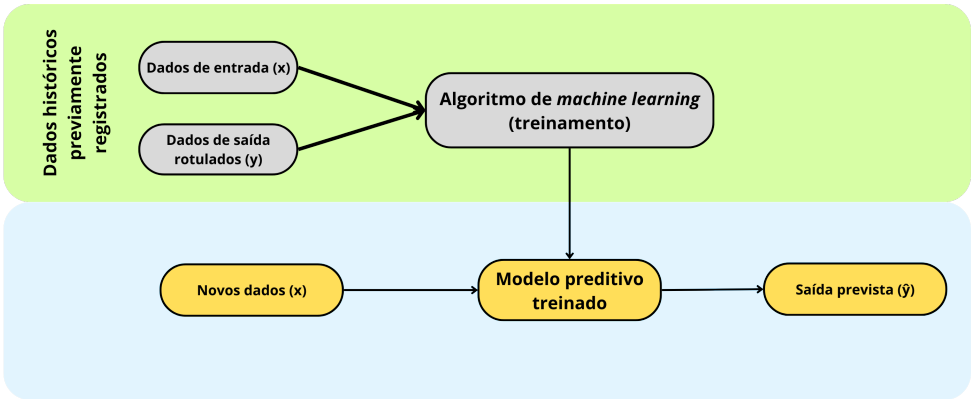


Figura 1.2: Ciclo do Aprendizado Supervisionado.

- **Coleta e Preparação dos Dados (x, y):** O processo começa com a coleta de dados históricos compostos por pares de entrada (x) e saída (y). As entradas podem ser variáveis como imagens, textos, medições ou registros diversos (por exemplo, temperatura e umidade), enquanto as saídas são os rótulos ou respostas corretas (como "choveu ou não"). Esses pares (x, y) formam a base sobre a qual o modelo vai aprender.

- **Treinamento do Modelo com um Algoritmo de Machine Learning:** Os pares (x, y) são enviados para um algoritmo de aprendizado, que busca identificar padrões e relações entre as entradas e suas respectivas saídas. Esse processo é conhecido como treinamento e envolve o ajuste de parâmetros internos do modelo, com o objetivo de minimizar os erros entre as previsões feitas e os rótulos reais. Nesta etapa, também é importante avaliar se o modelo está bem generalizado (aprendeu os padrões de forma eficaz) ou se sofre de overfitting (decorou os exemplos) ou underfitting (não captou os padrões).

- **Geração do Modelo Treinado:** Após o treinamento, temos um modelo ajustado, pronto para ser testado em novos dados. Esse modelo é, essencialmente, uma representação da relação aprendida entre entradas e saídas, e será usado para fazer previsões.

- **Predição com Novos Dados (ŷ):** Quando fornecemos uma nova entrada (x), agora sem o rótulo conhecido, o modelo treinado utiliza o que aprendeu para estimar uma saída (chamada de ŷ, ou y-hat). Se o modelo foi bem treinado, essa previsão tende a ser bastante próxima da resposta correta, permitindo a aplicação prática em diversos contextos, como reconhecimento de padrões, diagnósticos, previsões meteorológicas e muito mais.

Dessa forma, o ciclo se completa toda vez que ajustamos um modelo com dados rotulados e, posteriormente, utilizamos esse modelo para predizer rótulos desconhecidos. Esse mesmo fluxo pode ser repetido quando surgem novos dados ou quando desejamos melhorar ainda mais a qualidade das previsões. Suponha que queremos ensinar o computador a distinguir peixes-boi de outros mamíferos aquáticos por fotos. Precisamos:

- **Dados de Treinamento:** Conjunto de imagens de peixes-boi (rotuladas como "peixe-boi") e de outros animais (rotuladas como "outros mamíferos").
- **Treinamento:** O modelo analisa as imagens, compara com o rótulo ("peixe-boi" ou "outro") e ajusta seus parâmetros.
- **Predição:** Ao receber uma nova foto, o modelo "decide" se vê um peixe-boi ou não, com base nos padrões que aprendeu.

Assim, mesmo sem entrar em detalhes matemáticos, compreendemos que o aprendizado surge da exposição a exemplos corretos. Quanto melhor a qualidade dos dados, mais eficaz é a classificação.

1.2.2 Classificação e Regressão

No aprendizado supervisionado, há duas tarefas básicas:

- **Classificação:** quando o objetivo é colocar cada entrada em uma categoria. Exemplo: dizer se a imagem de um drone mostra um trecho de floresta alagada ou em área seca.
- **Regressão:** quando o objetivo é prever um valor numérico. Exemplo: estimar quanta chuva cairá em uma região amazônica, analisando dados climáticos como umidade e temperatura.

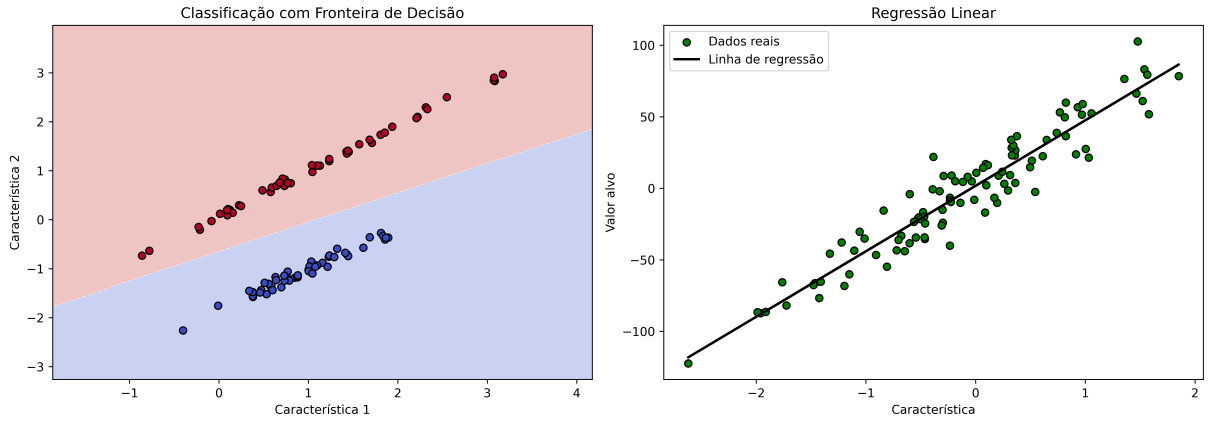


Figura 1.3: Gráfico didático para reforçar os conceitos de Classificação e de Regressão.

Esses dois formatos abrangem boa parte das aplicações na área. Por exemplo, prever se uma pessoa está "apta" ou "não apta" a receber um medicamento é classificação; estimar a altura de uma árvore com base em sua circunferência é regressão.

Independentemente do tipo de problema, seja de classificação ou de regressão, existe um passo fundamental para garantir que o modelo funcione de forma confiável: a separação dos dados em conjuntos de treino e teste. Essa prática é essencial para avaliar o desempenho do modelo em situações reais, evitando o temido Overfitting e o Underfitting como veremos a seguir.

Fique Alerta!

Overfitting (superajuste): é quando o modelo decora detalhes específicos do conjunto de treinamento e falha ao prever corretamente fora desses exemplos. **Underfitting (subajuste):** é quando o modelo não aprende os padrões o suficiente e obtém baixa precisão, mesmo nos dados de treino. **Generalização:** é o objetivo principal é equilibrar aprendizado nos dados conhecidos e bom desempenho em dados novos e desconhecidos.

1.2.3 Por que separar os dados?

Separar os dados em conjuntos de treino e teste é uma prática essencial no desenvolvimento de modelos de regressão e de aprendizado de máquina em geral. Essa divisão permite que o modelo aprenda com uma parte dos dados (conjunto de treino) e seja avaliado com outra parte que ele nunca viu antes (conjunto de testes). Isso ajuda a verificar se o modelo realmente está aprendendo a identificar padrões e não apenas memorizando os dados fornecidos.

Quando um modelo é testado nos mesmos dados com os quais foi treinado, os resultados podem ser enganosamente bons. Isso porque ele pode simplesmente estar reproduzindo os exemplos memorizados, sem ser capaz de generalizar para novas situações. Separar os dados garante que o desempenho avaliado reflita a capacidade do modelo de lidar com dados do "mundo real", ou seja, dados novos que não estavam disponíveis durante o processo de aprendizado.

Além disso, essa abordagem ajuda a evitar o chamado overfitting, que ocorre quando o modelo se ajusta demais aos dados de treino e perde a capacidade de fazer boas previsões em novos casos. O conjunto de teste, portanto, serve como um "termômetro" para verificar se o modelo está robusto e equilibrado.

Em geral, costuma-se usar de 70% a 80% dos dados para o treinamento e o restante para o teste. Em projetos mais complexos, pode-se ainda usar um terceiro subconjunto chamado de validação, destinado ao ajuste de hiperparâmetros antes da avaliação final com o conjunto de teste. Esse cuidado todo contribui para o desenvolvimento de modelos mais confiáveis e eficazes.

O aprendizado supervisionado é como ensinar um estudante com respostas corretas para que, no futuro, ele possa resolver novas questões sozinho. É a base de muitas aplicações de inteligência artificial e ciência de dados. No próximo módulo, veremos como funcionam os algoritmos e as técnicas que tornam esse aprendizado possível, bem como os cuidados na coleta e preparação dos dados.

1.2.4 Aplicando seus conhecimentos

1. Explique, com suas palavras, a diferença entre classificação e regressão.

2. Dê um exemplo prático de como aplicar aprendizado supervisionado para resolver um problema da sua comunidade.

3. Por que é importante separar dados de treino e dados de teste?

1.3 Regressão Linear

Iniciando o diálogo...

Olá, estudante! Chegamos a um tópico muito importante, a Regressão Linear. De forma bem simplificada, podemos dizer que a regressão linear é um método usado para descobrir como variáveis se relacionam e, com isso, prever valores futuros. No dia a dia, ela aparece em diversas áreas: da economia para prever o preço dos itens da cesta básica, na agricultura para estimar a produção de grãos, ou ainda no campo ambiental, para estimar a qualidade do ar, mudanças na temperatura, períodos de chuvas, etc.

1.3.1 O que é Regressão Linear?

A Regressão Linear é uma técnica estatística e computacional usada para entender e quantificar a relação entre variáveis (MITCHELL, 1997), sendo uma das técnicas fundamentais em modelagem preditiva, como discutido por Géron (2023). Nesse exemplo, estamos interessados em prever a quantidade de frutos produzidos por hectare com base em uma única variável: a quantidade de chuva registrada em determinado período.

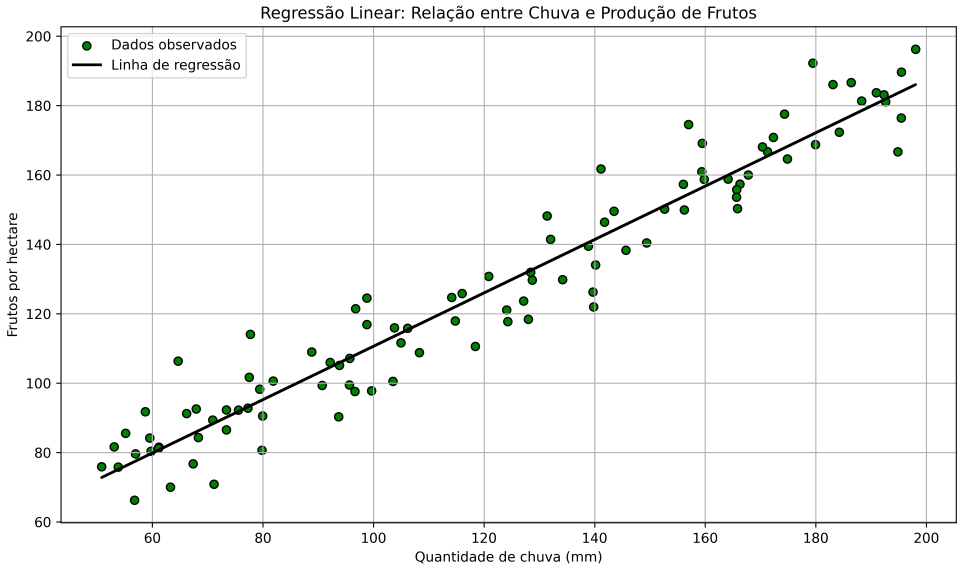


Figura 1.4: Regressão Linear: Relação entre Chuva e Produção de Frutos.

A variável independente (no eixo X) é a quantidade de chuva (em milímetros). A variável dependente (no eixo Y) é a produção de frutos por hectare, é o que queremos estimar. Cada ponto verde no gráfico representa uma observação real: uma combinação de quantidade de chuva e respectiva produção observada. A linha preta traçada é o resultado do modelo de regressão linear, ela indica a tendência geral da relação entre as variáveis. Podemos interpretar essa linha como uma estimativa média: quanto mais chuva, maior tende a ser a produção de frutos. Porém, a produção também sofre influência de outros fatores (como solo, fertilizantes, pragas etc.), o que explica por que os pontos não estão exatamente sobre a linha (isso é o ruído dos dados). Este é um ótimo exemplo de como o machine learning pode ser aplicado de forma prática: ao entender padrões em dados históricos, conseguimos fazer previsões e tomar

decisões mais informadas, neste caso, ajudar agricultores a planejar suas safras com base em fatores ambientais.

Regressão Linear Simples

Quando há apenas uma variável explicativa (x) para prever y, chamamos de regressão linear simples. A fórmula que define a linha (reta) de melhor ajuste é:

$$\hat{y} = \theta_0 + \theta_1 x$$

onde:

- $\hat{y}$ é o valor previsto,
- x é a variável independente,
- θ_0 é o coeficiente que chamamos de intercepto ou termo de bias,
- θ_1 é o coeficiente angular (a inclinação da reta).

Regressão Linear Múltipla

Se tivermos mais de uma variável independente por exemplo, chuva (mm), insolação (horas de sol) e fertilizante (em kg), o modelo se torna:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Vejam! É o mesmo princípio, mas agora com várias variáveis explicativas (mais de uma entrada).

1.3.2 Para que serve?

O principal uso é prever um valor contínuo. Seja prever a nota de um aluno a partir de horas de estudo e exercícios resolvidos, prever o volume de chuva a partir de padrões climáticos ou prever a produtividade agrícola em determinada região. A regressão linear "aprende" com dados já conhecidos (históricos), ajusta uma reta (ou hiperplano) que melhor representa a relação entre x e y e, depois, usa essa reta para prever valores futuros.

1.3.3 Como Funciona na Prática?

Para entender como o modelo encontra a reta de melhor ajuste, é importante conhecer três componentes centrais: a hipótese (ou função de predição), a função de custo e o processo de otimização, conhecido como descida do gradiente.

Hipótese ou Função de Predição

A hipótese (também chamada de função de predição) é a equação citada antes:

$$\hat{y} = \theta_0 + \theta_1 x$$

No caso de uma só variável, é como traçar uma reta que melhor se encaixe aos pontos em um gráfico onde o eixo x é a variável preditora e o eixo y é o valor observado. É como o modelo enxerga os dados e tenta prever resultados a partir disso. A hipótese é uma das peças-chave da regressão linear (e de muitos algoritmos de aprendizado de máquina). Ela representa a função que o modelo usa para fazer previsões, ou seja, é o "chute educado" que o modelo dá com base nos dados de entrada.

Por que ela é importante?

- É a fórmula usada para prever novos valores.
- Serve como base para calcular o erro (função de custo).
- É ajustada repetidamente durante o treinamento para melhorar a performance do modelo.

Função de Custo

A função de custo é uma fórmula matemática usada para medir o erro entre as previsões do modelo e os valores reais dos dados. Em outras palavras, ela diz ao modelo o quanto ele está errando e o objetivo do treinamento é minimizar esse erro. Vamos imaginar que temos uma função de predição, nossa famosa reta, e queremos saber o quão boa ela está. Para isso, usamos a função de custo, que compara os valores previstos com os valores reais de um conjunto de dados. Para medir se a reta está "boa" ou não, definimos uma função de custo, como o Erro Quadrático Médio (MSE - Mean Squared Error). Na forma simples, podemos escrever:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

onde:

- m é o número total de exemplos;
- $y^{(i)}$ é o valor real do exemplo i;
- $\hat{y}^{(i)}$ é o valor previsto.
- $J(\theta_0, \theta_1)$ é o valor da função de custo (o erro total)

O objetivo é minimizar J. Se esse valor é pequeno, significa que as previsões do modelo ficam próximas dos valores reais. Se o valor é grande, algo está errado: ou a reta não está bem ajustada, ou há muitos "ruídos" nos dados.

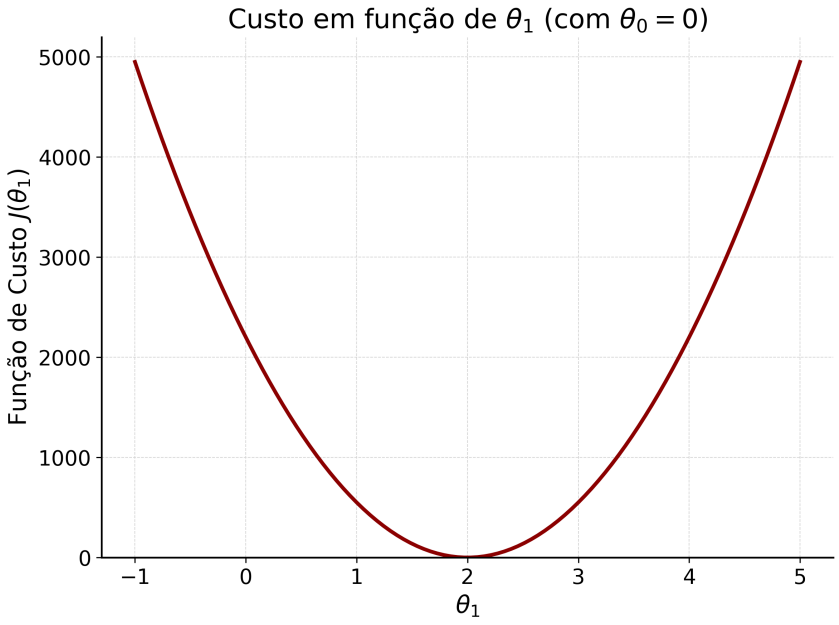


Figura 1.5: Custo em função de θ_1 (com $\theta_0 = 0$).

Otimização (Descida do Gradiente)

Para chegar aos coeficientes θ_0 e θ_1 que minimizam a função de custo, um método frequentemente usado é o Gradiente Descendente. De modo geral, ele faz várias iterações, ajustando θ_0 e θ_1 passo a passo. A seguir, apresentamos a formulação matemática básica da atualização dos parâmetros com o algoritmo de gradiente descendente:

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} ((\hat{y}^{(i)} - y^{(i)}) \cdot x^{(i)})$$

A cada passo, o algoritmo calcula o quanto alterar cada θ para tornar o custo menor, até "convergir" para valores que representem um mínimo, esperamos que seja o mínimo global ou algo próximo. Imagine a função de custo como um vale ou uma montanha com um fundo (mínimo). O objetivo é chegar até o ponto mais baixo, onde o erro é mínimo. A descida do gradiente funciona como se estivéssemos descendo esse terreno, dando passos na direção mais inclinada, ou seja, onde o custo diminui mais rapidamente, como pode ser exemplificado no gráfico abaixo.

Entretanto, precisamos ficar atentos na Taxa de Aprendizado (α), cuidado com o tamanho do passo, pois se for muito pequena, a descida é lenta, mas se for muito grande, o modelo pode "pular"o mínimo e nunca convergir.

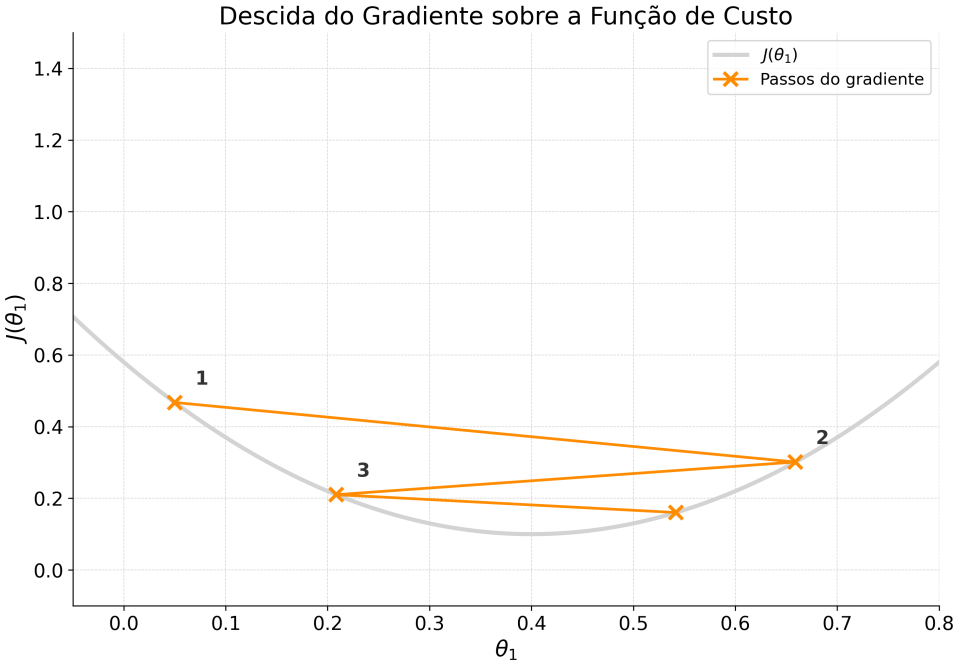


Figura 1.6: Descida do Gradiente sobre a Função de Custo.

Fique Alerta!

A função de predição gera os valores estimados. **A função de custo** mede o erro dessas previsões. **A descida do gradiente** ajusta os parâmetros para reduzir esse erro, é o mecanismo que faz o modelo aprender com os erros.

Exemplo prático: Produção Agrícola e Sustentabilidade
Imagine que você vive em uma comunidade amazônica e quer entender como a quantidade de chuva (variável x) influencia a produção de açaí por hectare (variável y). Para isso, você coleta dados reais de cinco meses consecutivos.

Mês	Chuva (mm)	Produção de Açaí (kg)
1	120	55
2	80	32
3	150	63
4	90	40
5	110	52

Tabela 1.1: Dados de chuva e produção de açaí.

Com esses dados, aplicamos um modelo de regressão linear simples, que estima a relação entre as variáveis. O resultado é uma equação que nos permite prever a produção com base na quantidade de chuva. Para efeitos didáticos, digamos que após o treinamento, o modelo encontrou os seguintes valores: θ_0 (intercepto): aproximadamente 2.76 θ_1 (inclinação): aproximadamente 0.43

Ou seja, o modelo estima que a cada 1 mm a mais de chuva, a produção média de açaí aumenta em cerca de 0,43 kg por hectare. Se quisermos prever a produção em um mês com 130 mm de chuva, basta substituir o valor na equação:

$$y = 2,76 + 0,43 \cdot 130 = 58,66kg$$

Abaixo, visualizamos os pontos coletados e a reta de regressão ajustada, que mostra a tendência da relação entre chuva e produtividade.

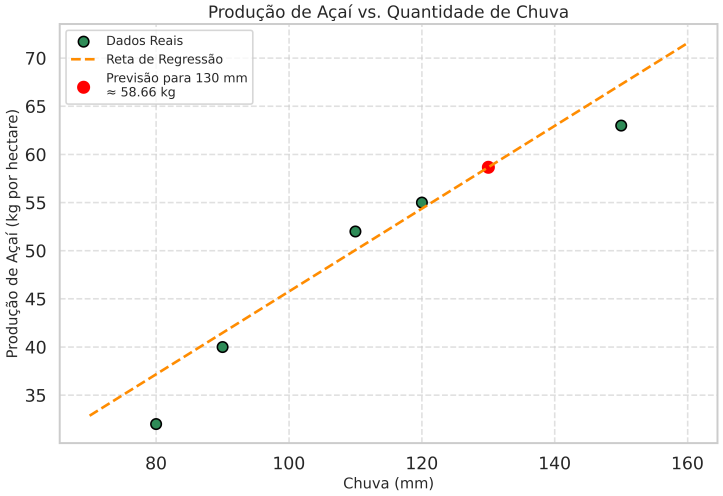


Figura 1.7: Produção de Açaí vs. Quantidade de Chuva.

Além de ser uma ferramenta de tomada de decisão, esse tipo de análise promove o uso consciente dos recursos naturais, respeitando os ritmos da floresta e garantindo maior segurança econômica para comunidades ribeirinhas e agricultores familiares. Esse tipo de modelo pode ajudar cooperativas agrícolas e gestores regionais a planejar melhor o estoque, a logística de transporte e até o preço de venda em diferentes épocas. Em termos de sustentabilidade,

quando sabemos os períodos de maior produção, podemos equilibrar o uso de recursos (água, fertilizantes) sem desperdício.

Fique Alerta!

- **Correlação vs. Causalidade:** O fato de as variáveis estarem bem relacionadas não quer dizer que uma seja a única causa da outra. Em nosso exemplo, embora a chuva afete a produção, outros fatores (fertilizantes, qualidade do solo, pragas etc.) podem impactar.
- **Outliers:** É comum, na Amazônia, ocorrerem fenômenos climáticos extremos. Um mês excepcionalmente chuvoso ou seco pode distorcer o modelo. Precisamos observar esses pontos "fora da curva" com cuidado.
- **Dados insuficientes:** Ter apenas alguns dados pode tornar a regressão linear menos confiável. É recomendado reunir mais informações ao longo do tempo para a robustez do modelo.

Conhecendo um pouco mais!

Estude bibliotecas em Python, o pacote `scikit-learn` (especialmente `LinearRegression`) oferece ferramentas prontas para regressão linear, facilitando a análise de dados reais. Material em inglês disponível em: `https://scikit-learn.org/stable/user_guide.html`

Caso Prático

A seguir, temos um exemplo de implementação manual de regressão linear em Python, utilizando o algoritmo de descida do gradiente para ajustar os parâmetros do modelo. Neste caso, queremos entender como a quantidade de horas de sol por dia (variável independente X) influencia a produção de frutas (variável dependente Y).

Copie e Teste!

```python
import numpy as np
import matplotlib.pyplot as plt

# ------------------------------
# Funções auxiliares
# ------------------------------

def prever(x, intercepto, inclinacao):
    """Calcula a predição baseada na equação da reta."""
    return intercepto + inclinacao * x

def custo(intercepto, inclinacao, X, Y):
    """Calcula o erro quadrático médio (função de custo)."""
    y_previsto = prever(X, intercepto, inclinacao)
    return np.mean((y_previsto - Y) ** 2) / 2
```

```
def plotar_regressao(X, Y, intercepto, inclinacao):
    """Plota os dados e a reta de regressão."""
    plt.figure(figsize=(8, 5))
    plt.scatter(X, Y, color='green', label='Dados reais')
    plt.plot(X, prever(X, intercepto, inclinacao), color='orange',
    label='Reta de regressão')
    plt.xlabel('Horas de Sol por Dia')
    plt.ylabel('Produção de Frutas (kg)')
    plt.title('Regressão Linear: Produção vs Horas de Sol')
    plt.legend()
    plt.grid(True)
    plt.show()

def plotar_custo(historia_custo):
    """Plota a evolução da função de custo."""
    plt.figure(figsize=(8, 5))
    plt.plot(historia_custo, color='blue')
    plt.xlabel('Épocas')
    plt.ylabel('Custo')
    plt.title('Convergência da Função de Custo')
    plt.grid(True)
    plt.show()

# ------------------------------
# Dados de entrada
# ------------------------------

# Exemplo de dados: Horas de sol (X) e Produção de frutas (Y)
X = np.array([4, 5, 6, 7, 8], dtype=float)
Y = np.array([40, 45, 54, 56, 62], dtype=float)

# Hiperparâmetros
taxa_aprendizado = 0.01   # taxa de aprendizado
epocas = 10  # número de iterações

# Inicialização dos parâmetros
intercepto = 0.0
inclinacao = 0.0
historia_custo = []

# ------------------------------
# Gradiente Descendente
# ------------------------------

for _ in range(epocas):
    y_previsto = prever(X, intercepto, inclinacao)
    erro = y_previsto - Y
    grad0 = np.mean(erro)
    grad1 = np.mean(erro * X)
    intercepto -= taxa_aprendizado * grad0
```

```
    inclinacao -= taxa_aprendizado * grad1
    historia_custo.append(custo(intercepto, inclinacao, X, Y))

# ------------------------------
# Resultados
# ------------------------------

print(f"Intercepto (theta0): {intercepto:.2f} \n\tEste valor
   representa o ponto onde a reta cruza o eixo y.")
print(f"\nInclinação (theta1): {inclinacao:.2f} \n\tA inclinação
   da reta indica a taxa de variação da produção de frutas\n\tem
   relação às horas de sol.")
print(f"\nCusto final: {historia_custo[-1]:.2f} \n\tEste é o valor
    da função de custo final, que mede o erro médio \n\tentre as
   previsões e os valores reais.")

# Previsão para 6.5 horas de sol
x_teste = 6.5
y_previsto = prever(x_teste, intercepto, inclinacao)
print(f"\n\tOu seja, para {x_teste} horas de sol, a produção
   estimada é {y_previsto:.2f} kg\n\tIsso mostra como a nossa
   equação de regressão prevê a produção de frutas\n\tcom base nas
    horas de sol.")

# ------------------------------
# Visualizações
# ------------------------------

plotar_regressao(X, Y, intercepto, inclinacao)
plotar_custo(historia_custo)
```

Resultado Esperado

```
Intercepto (theta0): 1.38
Este valor representa o ponto onde a reta cruza o eixo y.
Inclinação (theta1): 8.13
A inclinação da reta indica a taxa de variação da produção
de frutas em relação às horas de sol.
Custo final: 8.56
Este é o valor da função de custo final, que mede o erro
médio entre as previsões e os valores reais.
Ou seja, para 6.5 horas de sol, a produção estimada é 54.23
kg. Isso mostra como a nossa equação de regressão prevê a
produção de frutas com base nas horas de sol.
```

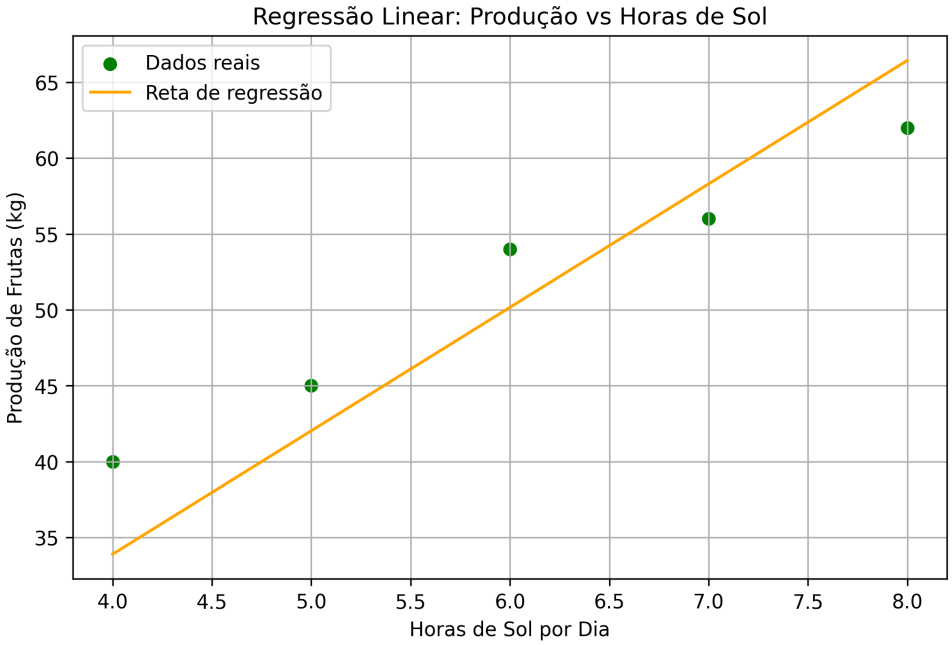


Figura 1.8: Gráfico de Regressão Linear: Produção de Frutas vs. Horas de Sol por Dia.

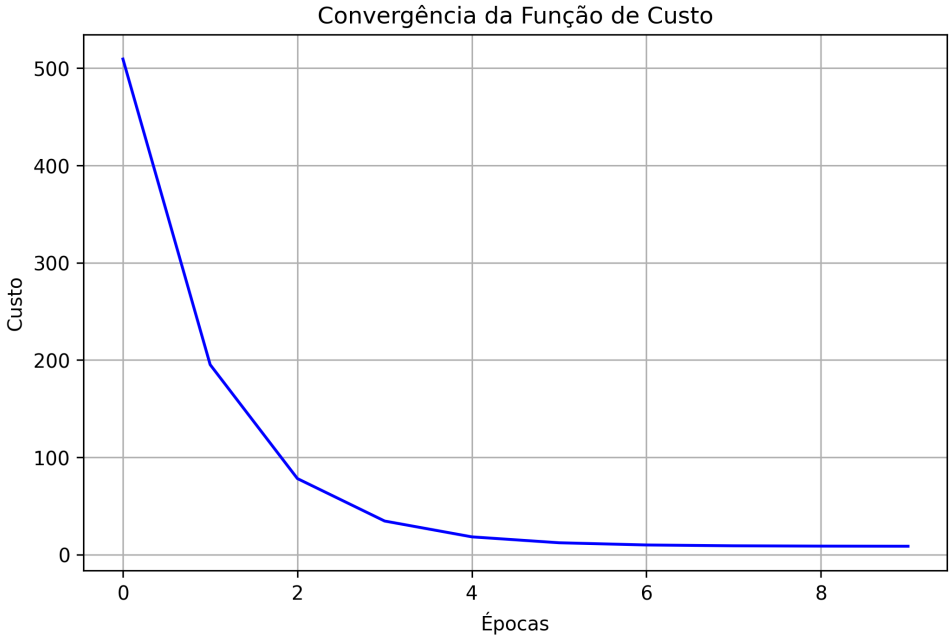


Figura 1.9: Gráfico da Convergência da Função de Custo ao longo das épocas de treinamento.

Fique Alerta!

Em aplicações reais, você pode querer usar pandas para carregar dados de um CSV, matplotlib para visualizar gráficos e, claro, scikit-learn para não precisar implementar tudo do zero.

1.3.4 Múltiplas Variáveis

Quando lidamos com dados ambientais, é comum termos diversas variáveis envolvidas na predição, como:

- Índice pluviométrico;
- Umidade do ar;
- Temperatura média;
- Tipo de solo;
- E outras características.

A regressão linear múltipla pode auxiliar na compreensão de como cada um desses fatores impacta a produção de frutos ou o desenvolvimento de uma espécie. Além disso, pode apoiar políticas de conservação ambiental, pois se consegue prever como mudanças em alguns desses fatores (como aumento da temperatura) afetariam a biodiversidade local.

Exemplo de Regressão Múltipla

Ao adicionar mais variáveis $(x_1, x_2, x_3, \dots)$, a função de predição fica:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

O processo de aprendizado continua usando uma função de custo e métodos de otimização parecidos. A diferença é que, agora, ajustamos muitos coeficientes em vez de apenas 2 (θ_0 e θ_1).

Em muitos contextos ambientais, especialmente na Amazônia, a produção agrícola ou o desenvolvimento de uma espécie não depende de um único fator. Pelo contrário, é comum lidarmos com diversas variáveis interagindo ao mesmo tempo, como o índice pluviométrico, a umidade do ar, a temperatura média, o tipo de solo, entre outras características ambientais. Para compreender como esses fatores, em conjunto, afetam um determinado resultado, utilizamos a regressão linear múltipla.

Essa técnica permite analisar o impacto de várias variáveis independentes sobre uma variável dependente. Por exemplo, podemos estudar como a combinação de chuvas, temperatura e solo influencia a produção de frutos por hectare. A equação de predição, nesse caso, é ampliada: ao invés de termos apenas um coeficiente (como na regressão simples), passamos a ter vários, um para cada variável explicativa. O modelo aprende os pesos de cada fator com base nos dados históricos, buscando minimizar a diferença entre as previsões e os valores reais.

A estrutura matemática continua semelhante: usamos uma função de custo (geralmente o erro quadrático médio) e métodos de otimização, como o gradiente descendente, para ajustar os coeficientes. A principal diferença está na complexidade: agora estamos ajustando um vetor de parâmetros $(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$, o que nos permite modelar relações mais realistas e abrangentes.

Suponha que queiramos prever a produção de açaí com base em três fatores: quantidade de chuva (x_1), temperatura média (x_2) e tipo de solo (x_3, representado numericamente). A equação da regressão múltipla teria a forma:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Com os dados certos, o modelo pode dizer, por exemplo, o quanto a produção esperada varia a cada grau de aumento na temperatura, ou como diferentes tipos de solo afetam o rendimento.

A regressão linear múltipla é uma técnica poderosa e acessível, especialmente útil quando lidamos com múltiplos fatores que influenciam um resultado. Entre seus principais benefícios, podemos destacar:

- **Simplicidade e interpretabilidade:** Mesmo com várias variáveis envolvidas, o modelo mantém uma estrutura linear, permitindo que cada coeficiente indique diretamente o impacto de uma variável específica sobre o resultado.
- **Eficiência computacional:** O processo de treinamento é relativamente rápido, funcionando bem mesmo com conjuntos de dados de tamanho moderado.
- **Base conceitual sólida:** A regressão múltipla serve como fundamento para métodos mais sofisticados de aprendizado de máquina e modelagem preditiva.

Por outro lado, é importante considerar algumas limitações:

- **Suposição de linearidade:** O modelo parte do princípio de que a relação entre as variáveis é linear. Quando isso não ocorre, o ajuste pode ser inadequado.
- **Multicolinearidade:** A presença de variáveis independentes altamente correlacionadas pode comprometer a estabilidade dos coeficientes e dificultar a interpretação dos resultados.
- **Dependência da qualidade dos dados:** A presença de ruído ou a escassez de dados pode prejudicar o desempenho do modelo e sua capacidade de generalização.

1.3.5 Próximos Passos

Após explorar a regressão linear como uma ferramenta essencial para entender relações entre variáveis e fazer previsões baseadas em dados, é hora de avançar para novas possibilidades de aprendizado e aplicação prática.

Aumentando seu conhecimento Agora que você já domina os fundamentos, vale explorar recursos que ajudam a melhorar o desempenho e a generalização dos modelos:

- **Regularização:** Técnicas como Ridge e Lasso são muito úteis para evitar o sobreajuste (quando o modelo se ajusta demais aos dados de treino e perde capacidade de prever novos dados). Elas funcionam adicionando um ”freio”à complexidade do modelo, equilibrando precisão e simplicidade.
- **Modelos não lineares:** Em muitos casos, a relação entre as variáveis não é perfeitamente linear. Métodos como regressão polinomial, árvores de decisão ou florestas aleatórias podem capturar essas variações com mais flexibilidade, ampliando as possibilidades de análise.

Projetos Práticos Colocar a teoria em prática é essencial para consolidar o aprendizado. Aqui vão algumas ideias:

- **Use dados abertos:** Acesse portais como dados.gov.br, escolha um conjunto de dados de interesse (educação, meio ambiente, agricultura etc.) e aplique regressão linear para responder a uma pergunta prática. Crie gráficos, interprete os coeficientes e compartilhe os resultados com colegas ou em sala de aula.
- **Faça um projeto orientado:** Comece pequeno. Escolha um problema claro, como prever o consumo de energia com base na temperatura. Baixe os dados, limpe-os, visualize, modele, teste e apresente, cada passo pode ser acompanhado com orientação, quase como "pegar pela mão".
- **Colete dados locais:** Uma abordagem simples e engajadora é levantar dados da própria comunidade ou escola. Por exemplo, medir a temperatura média e o consumo de energia em diferentes dias, ou a quantidade de chuva e o número de visitantes em um parque. Em seguida, aplique os conceitos de regressão para descobrir se há relação entre as variáveis.

Esses passos podem não apenas reforçar seu entendimento, mas também gerar insights úteis para a comunidade, conectar teoria com realidade e abrir caminhos para projetos mais avançados no futuro. Fechamos aqui nossa seção sobre Regressão Linear!

Ficou curioso(a)? Então continue a explorar. A regressão linear é apenas uma peça em um mosaico de técnicas de análise de dados. Não se esqueça de que sua aplicação prática depende de dados de qualidade, conhecimento do problema e reflexão ética ao lidar com os resultados. Em breve você descobrirá novas técnicas que se unem a este conhecimento para formar uma base sólida em Machine Learning.

Boas descobertas e até a próxima!

1.4 Vetorização

Iniciando o diálogo...

Ao começar seus estudos em Machine Learning, você logo percebe que os algoritmos realizam muitas operações matemáticas ao mesmo tempo. Imagine, por exemplo, que você tem 100 mil pontos de dados e deseja calcular a soma de todos eles. Se formos somar cada valor um a um, isso seria extremamente demorado. Mas se utilizarmos a vetorização, é como se fizéssemos essa soma "toda de uma vez", aproveitando a força do hardware para processar várias operações em paralelo.

1.4.1 O que é Vetorização?

Vetorização é a prática de substituir estruturas de laços de repetição na programação (por exemplo, `for` ou `while`) por operações em blocos completos de dados. Em vez de processar cada elemento de um vetor ou matriz passo a passo, você executa uma instrução que manipula todo o conjunto de dados de uma só vez.

Copie e Teste!

```
# Forma não vetorizada

vetor = [1, 2, 3, 4, 5]
soma = 0
for i in range(len(vetor)):
    soma += vetor[i]
print("Soma (não vetorizada):", soma)
```

Copie e Teste!

```
# Forma vetorizada

import numpy as np

vetor = np.array([1, 2, 3, 4, 5])
soma = np.sum(vetor)

print("Soma (vetorizada):", soma)
```

No segundo caso acima, a soma de todos os elementos ocorre internamente de maneira otimizada, sem que você precise escrever um laço manualmente.

1.4.2 Por que eliminar laços de repetição?

Figura 1.10: Imagens criadas com o DALL-E, representando de um único "core" trabalhando em sequência, e de vários pintores trabalhando em paralelo, ilustrando o conceito de paralelismo com eficiência e divisão de tarefas.

A vetorização costuma ser muito mais rápida que o uso de laços (`for`, `while`) tradicionais. Isso acontece porque bibliotecas como o NumPy em Python são escritas em linguagens de baixo nível (como C ou C++), que conseguem usar instruções vetoriais do processador. Estas

instruções permitem que várias operações sejam feitas em paralelo, aproveitando ao máximo os núcleos (cores) do processador e a GPU (Unidade de Processamento Gráfico), que é muito boa para lidar com grandes volumes de dados de uma vez.

- **O que é core?** Um core é como um "trabalhador" dentro do processador. Os processadores modernos têm vários cores (dual-core = 2, quad-core = 4, etc.). Se você tem tarefas independentes, cada core pode trabalhar em paralelo, acelerando o processo.
- **E a GPU?** A GPU (Graphics Processing Unit) é uma unidade especializada em realizar milhares de operações simples ao mesmo tempo. Inicialmente usada para gráficos (por exemplo, em jogos), hoje ela também é usada em ciência de dados, IA e processamento de matrizes, por ser absurdamente boa nisso.

Uma analogia que podemos pensar é numa linha de produção, Imagine que você precisa pintar 1000 carrinhos de brinquedo, comparando as abordagens:

- **Laço de repetição:** 1 pintor pinta os 1000 carrinhos um por um.
- **Vetorização:** 100 pintores pintam 10 carrinhos cada ao mesmo tempo, ou seja, mais trabalhadores (cores / GPU) é igual a menos tempo.

Legibilidade do código

Códigos com muitos laços tendem a ficar grandes, difíceis de entender e manter. Vetorização deixa o código mais direto.

Código com laço aninhado (um `for` dentro de `for`, dentro de outro `for`)

```
# Multiplicando duas matrizes manualmente
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
resultado = [[0, 0], [0, 0]]

for i in range(2):
    for j in range(2):
        for k in range(2): # <- terceiro nível de loop!
            resultado[i][j] += A[i][k] * B[k][j]
```

Código vetorizado com NumPy

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
resultado = np.dot(A, B)
```

Fique Alerta!

O que é laço aninhado? É um laço dentro de outro. Cada nível de aninhamento aumenta a complexidade. Isso é comum em manipulações de matrizes ou listas multidimensionais.

Menos erros

Quando usamos laços, é comum errar no índice, especialmente com o famoso erro *off-by-one* (erro de 1 unidade a mais ou a menos).

Exemplo de erro off-by-one

```
vetor = [10, 20, 30, 40, 50]
# ERRO: começa em 1 e vai até len(vetor) = 5 -> índice 5 não existe!
for i in range(1, len(vetor)+1):
    print(vetor[i]) # IndexError!
```

Qual seria o "certo"

```
vetor = [10, 20, 30, 40, 50]
for i in range(len(vetor)):
    print(vetor[i]) # Tudo certo aqui
```

Fique Alerta!

Usando vetorização, você evita esse tipo de erro porque não precisa mexer diretamente com índices, olhe abaixo!

```
import numpy as np
vetor = np.array([10, 20, 30, 40, 50])
print(vetor) # imprime tudo, simples assim
```

1.4.3 Onde a vetorização aparece no dia a dia do Machine Learning?

A vetorização é fundamental em tarefas de aprendizado de máquina, especialmente quando lidamos com grandes volumes de dados e modelos com muitos parâmetros, como:

Cálculo de Funções de Custo

Ao treinar um modelo de regressão linear, por exemplo, trabalhamos com milhares (ou até milhões) de amostras. Em vez de calcular a previsão h(x) para cada exemplo individualmente usando um laço, podemos fazer isso de uma só vez com uma multiplicação de matrizes.

Fique Alerta!

Isso permite calcular todas as previsões em uma única operação vetorizada, muito mais eficiente e rápida.

Atualização de Parâmetros com Gradient Descent

Em algoritmos de otimização como o gradiente descendente, os pesos do modelo (parâmetros θ) são atualizados com base no erro da previsão. Essa atualização pode ser escrita de forma vetorizada, o que elimina a necessidade de laços para cada parâmetro. Ou seja, em vez de atualizar cada peso individualmente com for, usamos uma expressão como `theta = theta - alpha * grad`, onde `grad` é o gradiente vetorizado.

Propagação em Redes Neurais

Cada camada de uma rede neural aplica operações matemáticas, principalmente multiplicações de matrizes seguidas de funções de ativação. Como essas redes frequentemente têm milhares ou milhões de parâmetros, usar laços aninhados para computar as ativações seria inviável. Com vetorização, a saída de uma camada inteira é calculada de uma só vez, tornando o treinamento e a inferência muito mais rápidos.

1.4.4 Vetorização na Prática

A vetorização vai muito além de "remover laços"; ela é uma estratégia essencial para escrever código mais limpo, rápido e eficiente, especialmente em aplicações de Machine Learning. Praticamente todas as etapas do aprendizado de máquina se beneficiam disso: desde o cálculo de funções de custo, passando pela atualização de pesos com gradient descent, até as operações entre camadas em redes neurais.

Essa abordagem elimina laços explícitos, reduz a chance de erros e permite que cálculos sejam distribuídos de forma paralela por múltiplos núcleos da CPU e até pela GPU graças às bibliotecas otimizadas como o NumPy, que usam instruções de baixo nível (como SSE e AVX) para acelerar tudo. O resultado é um código mais curto, mais legível e muito mais rápido.

Copie e Teste!

```python
import numpy as np
import time

# Criando um vetor com 10 milhões de elementos
a = np.random.rand(10_000_000)
b = np.random.rand(10_000_000)

# Forma com for
start = time.time()
resultado = np.zeros_like(a)
for i in range(len(a)):
    resultado[i] = a[i] + b[i]
print("Tempo com for:", time.time() - start)

# Forma vetorizada
start = time.time()
resultado = a + b
print("Tempo vetorizado:", time.time() - start)
```

Resultado Esperado

```
Tempo com for: 4.461456298828125
Tempo vetorizado: 0.019437074661254883
```

Mesmo em máquinas comuns, o ganho pode ser de 10x ou mais. Dominar a vetorização não só torna seu código mais robusto, como também melhora sua compreensão prática de álgebra linear e potencializa sua capacidade de construir soluções escaláveis em Machine Learning.

Mas usar loops não é mais simples às vezes?

Em alguns casos muito específicos, um laço simples pode até parecer intuitivo. Porém, quando os dados crescem, mesmo um simples `for` se torna um gargalo significativo de desempenho. Além disso, escrever `C = A + B` para somar vetores normalmente é mais legível do que laços extensos.

Vetorização resolve tudo na programação?

Não. Vetorização é muito útil para operações matemáticas bem estruturadas (somas, multiplicações, funções de ativação em redes neurais, etc.). Entretanto, em tarefas lógicas muito específicas (como `if/else` encadeados ou fluxos complexos de dados), nem sempre é possível eliminar completamente todos os loops. O segredo é encontrar onde a vetorização faz sentido.

1.4.5 Como a vetorização afeta meu dia a dia como estudante de Machine Learning?

Quando você for implementar regressão linear para aprender a programar o gradiente descendente, por exemplo, a vetorização permitirá que você aplique as fórmulas de erro, gradiente e atualização de parâmetros de modo muito mais rápido. Em projetos maiores, como redes neurais, as bibliotecas (TensorFlow, PyTorch etc.) fazem a maior parte da vetorização automaticamente. Mas entender o conceito ajuda a escrever código mais limpo e explorar certos truques de otimização.

Fique Alerta!

Ao aplicar vetorização, utilizamos melhor os recursos do hardware, como CPUs e GPUs, o que resulta em ganhos significativos de desempenho. No entanto, é essencial ter atenção às dimensões das matrizes, erros de `shape` são comuns, especialmente para quem está começando. Como o código vetorizado tende a ser mais compacto, depurar pode ser desafiador; por isso, é uma boa prática inspecionar os shapes com `A.shape` e visualizar pequenos trechos dos dados, como `A[:10]`, para entender o que está acontecendo.

Dominar a vetorização não apenas melhorará seu entendimento prático de álgebra linear aplicada ao ML, mas também ampliará sua capacidade de criar soluções mais eficientes e escaláveis. A vetorização sendo um dos pilares da computação científica moderna, é essencial em Machine Learning. Sempre que você vir uma repetição que processa elementos um a um, investigue se é possível trocar por uma forma vetorizada. Quanto maior o volume de dados, maior será o ganho. Agora que você já sabe o que é e por que faz diferença, fica muito mais fácil reconhecer onde aplicar essa técnica no seu dia a dia de estudos e projetos de aprendizado de máquina!

Conhecendo um pouco mais!

A documentação oficial do NumPy é uma excelente fonte para explorar mais sobre vetorização, operações com arrays, funções matemáticas e muito mais. Lá temos exemplos práticos, explicações detalhadas e guias úteis para todos os níveis do iniciante ao avançado, veja em `https://numpy.org/doc/`

1.5 Engenharia de características

Iniciando o diálogo...

Você já se perguntou por que certos modelos de Inteligência Artificial conseguem ser mais precisos do que outros, mesmo usando algoritmos parecidos? Um dos segredos está na etapa de Engenharia de Características, também chamada de *Feature Engineering*. É nessa fase que transformamos dados brutos em atributos (características) mais relevantes para o modelo, ampliando suas chances de sucesso.

1.5.1 O que é Engenharia de Características?

A Engenharia de Características é o processo de selecionar, criar e transformar dados em variáveis (ou características) que melhor representem o fenômeno que se deseja modelar. Imagine que você tem uma tabela com informações sobre clima, produtividade agrícola ou comportamento de consumidores. Nem sempre os dados vêm "prontos" para serem usados em algoritmos de Machine Learning. Muitas vezes, precisamos:

- Limpar os dados e ajustar *outliers* (pontos "fora da curva"), valores ausentes, etc.;
- Criar colunas a partir das já existentes (por exemplo, calcular a "taxa de crescimento" ou a "variação" entre períodos);
- Reformatar variáveis categóricas (como nomes de cidades) em códigos numéricos.
- Normalizar ou padronizar valores, para que fiquem em escalas comparáveis.

A engenharia de características é uma etapa essencial dentro do fluxo de Ciência de Dados, ela faz o "meio de campo" entre a coleta dos dados (seja de fontes brutas ou bases prontas) e a construção de modelos de Machine Learning. É nesse ponto que os dados são transformados, combinados e representados de forma que os algoritmos possam aprender com eles de maneira mais eficaz. Sem uma boa engenharia de características, mesmo os melhores modelos não performam bem.

Fique Alerta!

A qualidade das características costuma ser até mais importante do que o algoritmo em si. Um bom conjunto de atributos pode fazer toda a diferença no desempenho final de um modelo preditivo.

1.5.2 Principais Técnicas de Engenharia de Características

Transformação de Variáveis Categóricas

Em problemas de aprendizado de máquina, é comum lidarmos com variáveis categóricas, ou seja, informações que representam categorias em vez de números, como "cidade", "tipo de solo", "cultura agrícola", entre outros. No entanto, a maioria dos algoritmos de machine learning exige que os dados estejam em formato numérico. Por isso, é necessário transformar essas categorias em números de forma que preserve seu significado, mas sem induzir o modelo ao erro.

One-Hot Encoding: Codificação Binária por Colunas Essa técnica cria uma coluna para cada categoria existente. O valor será 1 se aquela categoria estiver presente na observação, e 0 caso contrário. Suponha que temos a variável ”cidade”, com os seguintes valores:

cidade
Manaus
Tefé
Coari

Após aplicar One-Hot Encoding, o conjunto será:

cidade_Manaus	cidade_Tefé	cidade_Coari
1	0	0
0	1	0
0	0	1

Essa abordagem é bastante segura, pois não introduz relações artificiais entre as categorias.

Label Encoding: Codificação por Rótulos Neste método, cada categoria é substituída por um número inteiro. É uma técnica simples, mas deve ser usada com cuidado. Suponha as cidades de Manaus, Tefé, Coari, Lábrea, São Gabriel da Cachoeira e Manacapuru, com Label Encoding poderia ser representado assim:

cidade	cidade_encoded
Manaus	1
Tefé	2
Coari	3
Lábrea	4
São Gabriel da Cachoeira	5
Manacapuru	6

Apesar de parecer eficiente, essa representação pode induzir o modelo a interpretar que existe uma ordem entre as cidades como se “Coari” fosse maior ou mais importante que “Tefé” ou “Manaus”. Em muitos contextos, essa suposição não faz sentido e pode comprometer a performance do modelo. Sobre a escolha entre One-Hot e Label Encoding depende do tipo de dado e do algoritmo que será utilizado. Para variáveis categóricas sem relação de ordem, o One-Hot Encoding é geralmente mais seguro. Já o Label Encoding pode ser útil com algoritmos baseados em árvores (Tree-based models), desde que as relações implícitas não causem viés.

Criação de Variáveis Sintéticas

Além de interpretar o dado como ele vem, é comum criarmos atributos que resumem relações importantes. Por exemplo, a densidade de focos de calor em determinada área, que seria o número de focos dividido pela área total. Ou, no contexto agrícola, a variação de temperatura ao longo de um mês. Essas variáveis “sintéticas” (ou “derivadas”) frequentemente ajudam o modelo a encontrar padrões que não estavam claros nos dados originais.

Normalização e Padronização: Equações Matemáticas

Em muitos cenários, as variáveis do nosso conjunto de dados apresentam escalas muito distintas (por exemplo, "pluviosidade" em milímetros e "renda per capita" em reais). Isso pode dificultar o trabalho de algoritmos baseados em distâncias ou que são sensíveis a magnitudes diferentes. Para contornar esse problema, aplicamos algumas transformações de escala, como normalização ou padronização.

Min-Max Normalization A normalização Min-Max (ou feature scaling) reescala os valores para ficarem dentro de um intervalo, comumente entre 0 e 1. A equação padrão é:

$$X_{\text{min-max}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Onde:

- X é o valor original de uma determinada observação;
- X_{min} e X_{max} são o valor mínimo e máximo dessa variável no conjunto de dados;
- O resultado $X_{\text{min-max}}$ fica compreendido entre 0 e 1.

Mean Normalization (Normalização pela Média) Uma variação menos comum, mas ainda utilizada, é a mean normalization, em que subtraímos a média da coluna e dividimos pela amplitude $(X_{\text{max}} - X_{\text{min}})$:

$$X_{\text{mean-norm}} = \frac{X - \bar{X}}{X_{\text{max}} - X_{\text{min}}}$$

Onde $\bar{X}$ é a média de todos os valores da variável. Dessa forma, deslocamos o valor para que a média da série fique em 0 e a amplitude final fique em torno de 1.

Z-Score Standardization (Padronização) A padronização Z-Score reescala os dados para terem média igual a 0 e desvio-padrão igual a 1. A equação que demonstra isso é:

$$X_{\text{z-score}} = \frac{X - \mu}{\sigma}$$

Onde:

- μ é a média da variável;
- σ é o desvio-padrão da variável.

1.5.3 Por que padronizar os Dados?

Quando trabalhamos com variáveis numéricas muito diferentes entre si, por exemplo, "chuva (mm)" variando entre 0 e 300, e "temperatura (°C)" entre 15 e 35, essas diferenças de escala podem afetar negativamente o desempenho de muitos algoritmos de aprendizado de máquina. A padronização resolve esse problema ao transformar os dados para que tenham média zero e desvio padrão igual a um. Isso faz com que todas as variáveis passem a ter peso semelhante, evitando que uma variável com valores maiores domine a análise.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Dados originais
X = np.array([[10], [20], [30], [40], [50]])

# Min-Max Normalization
scaler_minmax = MinMaxScaler()
X_minmax = scaler_minmax.fit_transform(X)

# Mean Normalization (manual)
X_mean = X.mean()
X_min = X.min()
X_max = X.max()
X_meannorm = (X - X_mean) / (X_max - X_min)

# Z-Score Standardization
scaler_zscore = StandardScaler()
X_zscore = scaler_zscore.fit_transform(X)

# Plotando os resultados
plt.figure(figsize=(10, 6))
plt.plot(X, np.zeros_like(X), 'o', label='Original', markersize
   =10)
plt.plot(X, X_minmax, 'o-', label='Min-Max [0, 1]')
plt.plot(X, X_meannorm, 's--', label='Mean Normalization')
plt.plot(X, X_zscore, 'd-.', label='Z-Score')
plt.title('Comparação de Técnicas de Normalização/Padronização')
plt.xlabel('Valor Original')
plt.ylabel('Valor Transformado')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Resultado Esperado

Valores Originais	Min-Max	Mean Norm	Z-Score
10	0.00	-0.50	-1.41
15	0.25	-0.25	-0.71
20	0.50	0.00	0.00
25	0.75	0.25	0.71
30	1.00	0.50	1.41

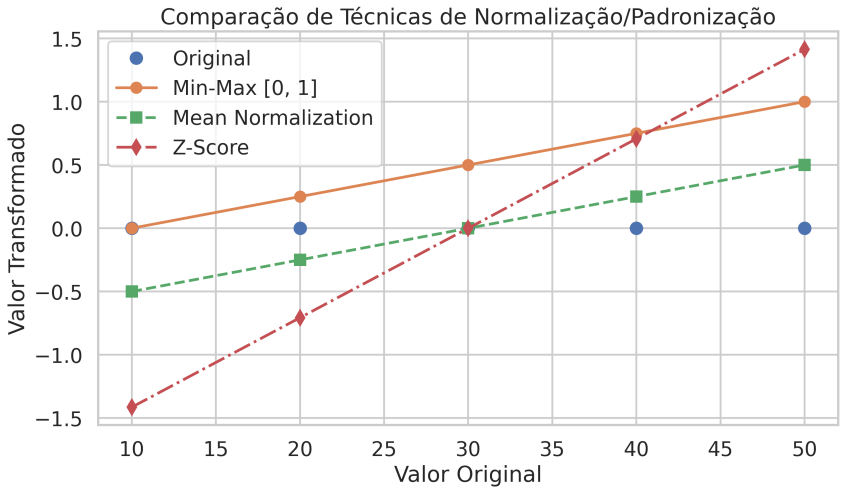


Figura 1.11: Gráfico comparando os resultados das técnicas de Normalização Min-Max, Normalização pela Média e Padronização Z-Score.

Nesse gráfico estamos comparando três técnicas comuns de normalização e padronização aplicadas a um conjunto simples de valores:

- **Min-Max Normalization**, onde transforma os dados para o intervalo [0, 1], porém é sensível a outliers;
- **Mean Normalization** que centraliza os dados em torno da média e os escala pela amplitude (máximo - mínimo); e
- **Z-Score** que transforma os dados em uma distribuição com média 0 e desvio padrão 1. É bastante usada em algoritmos baseados em distância, como k-NN e SVM.

1.5.4 Isso importa tanto para alguns algoritmos?

k-NN - k-Nearest Neighbors

Esse algoritmo decide a qual grupo um novo dado pertence com base nos dados mais próximos (vizinhos) dele. Como a proximidade é calculada usando distância (geralmente Euclidiana), variáveis em escalas diferentes distorcem essa medida. A padronização garante que todas as variáveis contribuam de forma justa para o cálculo da distância.

SVM - Support Vector Machine

O SVM constrói um "limite" (chamado hiperplano) para separar diferentes categorias. Para definir esse limite corretamente, ele também depende de cálculos de distância entre pontos e margens. Sem padronização, esse limite pode ser enviesado em direção à variável dominante.

Redes Neurais

Redes neurais ajustam seus pesos com base em gradientes (variações dos erros). Se uma variável tem uma escala muito maior que as outras, ela pode gerar gradientes muito mais intensos, fazendo com que o modelo aprenda mais rápido nessa variável e ignore as outras. Isso prejudica o equilíbrio do aprendizado.

Fique Alerta!

A padronização é um passo simples, mas essencial em muitos cenários. Ela ajuda a:

- Melhorar a eficiência do treinamento
- Garantir justiça entre as variáveis
- Evitar que o modelo seja influenciado por escalas artificiais

Em modelos mais simples, como árvores de decisão, a padronização pode não ser necessária, mas para modelos baseados em distância ou gradiente, como k-NN, SVM e redes neurais, ela faz toda a diferença.

1.5.5 Redução de Dimensionalidade

Em projetos de ciência de dados, é comum trabalharmos com conjuntos que possuem dezenas, ou até centenas de variáveis. No entanto, nem todas essas variáveis contribuem igualmente para o desempenho do modelo. Ter muitas dimensões pode aumentar a complexidade, dificultar a visualização e até causar o chamado "problema da maldição da dimensionalidade", onde o modelo começa a se perder em ruídos em vez de padrões úteis.

Uma das técnicas mais utilizadas para enfrentar esse desafio é o PCA (Principal Component Analysis) ou Análise de Componentes Principais. Essa técnica estatística transforma o conjunto de variáveis originais em um novo conjunto de variáveis chamadas componentes principais, que preservam o máximo possível da variância (informação) original, mas em menos dimensões. Vamos imaginar um conjunto com 4 variáveis que medem características florais, inspirado no famoso dataset Iris:

Amostra	Largura_Sépala	Comprimen_Sépala	Largura_Pétala	Comprimen_Pétala
1	3.1	5.4	1.5	4.5
2	3.5	5.9	1.8	4.8

Atenção neste momento, aplicando PCA, podemos transformar estas 4 variáveis em apenas 2 componentes principais:

Amostra	PCA_1	PCA_2
1	2.51	-0.42
2	2.90	-0.35

Essas duas novas variáveis (PCA_1 e PCA_2) carregam a maior parte da informação que estava distribuída nas 4 variáveis originais, facilitando a visualização em um gráfico 2D e reduzindo o custo computacional do modelo, sem perder muito da performance. Abaixo temos um exemplo prático de redução de dimensionalidade usando o famoso conjunto de dados Iris. O dataset original contém 4 variáveis numéricas: comprimento e largura das sépalas e pétalas. Utilizando o método PCA (Principal Component Analysis), conseguimos reduzir estas 4 variáveis para apenas 2 componentes principais, mantendo boa parte da variabilidade dos dados.

Copie e Teste!

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Carrega o dataset Iris
iris = load_iris()
X = iris.data
y = iris.target
features = iris.feature_names

# Criação do DataFrame original
df_original = pd.DataFrame(X, columns=features)

# Aplica PCA para reduzir de 4 para 2 dimensões
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
df_pca = pd.DataFrame(X_pca, columns=["PCA_1", "PCA_2"])
df_pca["target"] = y

# Retorna os 5 primeiros valores dos dataframes original e
   transformado
df_original_head = df_original.head()
df_pca_head = df_pca.head()

# Criar gráfico para visualizar
plt.figure(figsize=(8, 6))

for label in np.unique(y):
    plt.scatter(X_pca[y == label, 0], X_pca[y == label, 1], label=
   iris.target_names[label])

plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.title("Iris Dataset com PCA (2 Componentes Principais)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("pca_iris_plot.png")

print(df_original_head)

print(df_pca_head)
```

Resultado Esperado

```
sepal length(cm)   sepal width(cm)   petal length(cm)
       5.1               3.5                1.4
       4.9               3.0                1.4
       4.7               3.2                1.3
       4.6               3.1                1.5
       5.0               3.6                1.4

petal width(cm)       PCA_1        PCA_2    target
      0.2           -2.684126    0.319397      0
      0.2           -2.714142   -0.177001      0
      0.2           -2.888991   -0.144949      0
      0.2           -2.745343   -0.318299      0
      0.2           -2.728717    0.326755      0
```

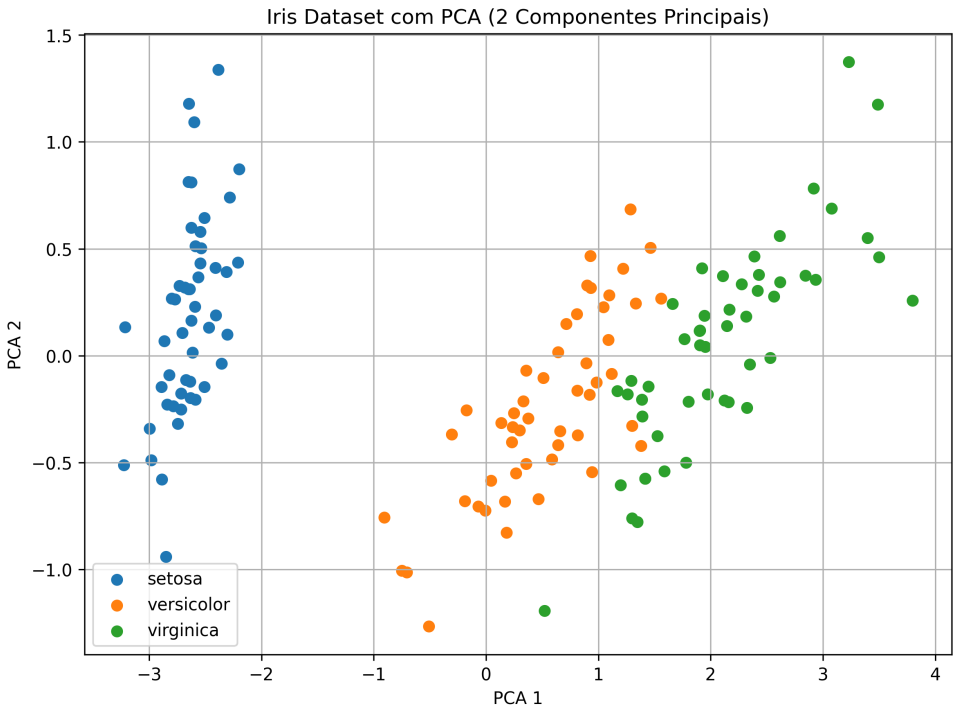


Figura 1.12: Visualização do dataset Iris projetado nos dois primeiros componentes principais.

Como podemos observar no gráfico, após a aplicação do PCA, os dados que originalmente existiam em um espaço de quatro dimensões agora podem ser visualizados em um plano 2D. O mais importante é notar que os três tipos de flores (setosa, versicolor e virginica) formam agrupamentos visualmente distintos. A espécie setosa, em particular, é facilmente separada das outras duas.

Isso demonstra o poder da Análise de Componentes Principais: reduzimos a dimensionalidade pela metade (de 4 para 2 variáveis), mas os dois novos componentes (PCA_1 e PCA_2) conseguiram reter a informação mais relevante que diferencia as classes. Com essa representação simplificada, um modelo de machine learning pode atingir alta performance de forma mais eficiente e com menor risco de overfitting.

1.5.6 Como Construir Boas Características?

A engenharia de características é um dos passos mais importantes no desenvolvimento de modelos preditivos. Trata-se de transformar dados brutos em representações úteis que melhorem o desempenho dos algoritmos. Um bom modelo com dados ruins terá resultados ruins. Já um modelo simples com boas características pode ter ótimo desempenho.

Etapas Fundamentais

1. **Observação e Limpeza de Dados:** Antes de tudo, é preciso verificar a qualidade dos dados: há valores ausentes (`null`)? Existem registros duplicados? Há valores fora do esperado (*outliers*)? Limpar essas inconsistências é essencial para garantir que o modelo aprenda corretamente.

2. **Exploração dos Dados:** Aqui você começa a entender os dados. Observe estatísticas descritivas, distribuições, gráficos de dispersão e correlações. Essa etapa revela padrões, tendências e relações escondidas entre as variáveis.

3. **Criação e Transformação de Características:** Esta é a etapa mais criativa! Consiste em construir novas variáveis que representem melhor o fenômeno. Tais como:

   - Converter uma data em variáveis como dia da semana, mês, ou estação do ano;
   - Agrupar faixas etárias em categorias como “jovem”, “adulto”, “idoso”.
   - Calcular uma nova coluna como `renda_per_capita = renda_familiar / número_de_pessoas`.

4. **Teste e Validação:** Após criar características, é hora de testá-las. Use modelos simples (como regressão linear ou árvores de decisão) para verificar se elas realmente ajudam na previsão. Remova o que não contribui.

Exemplo Prático

Imagine que temos um conjunto de dados sobre o consumo de energia elétrica de residências na região amazônica, com colunas como:

data	temperatura	umidade	consumo (kWh)
2023-07-15	33	80	130
2023-12-10	31	85	110

Podemos fazer as seguintes melhorias, essas transformações podem ajudar o modelo a perceber que o consumo aumenta em meses mais quentes, ou que há padrões sazonais.

- Criar nova coluna: `mês = data.month`
- Criar outra coluna: `estação = 'verão' ou 'inverno'` com base no mês
- Transformar a temperatura em categorias: alta, média, baixa
- Padronizar as variáveis numéricas para que fiquem na mesma escala

Copie e Teste!

```
import pandas as pd
import numpy as np

# Simulação de um pequeno dataset
data = pd.DataFrame({
    'data': pd.to_datetime(['2023-07-15', '2023-12-10', '
   2023-03-05', '2023-09-21', '2023-01-11']),
    'temperatura': [33, 31, 29, 34, 28],
    'umidade': [80, 85, 75, 82, 78],
    'consumo_kwh': [130, 110, 95, 140, 100]
})

# Extração de informações de data
data['mes'] = data['data'].dt.month
data['dia_da_semana'] = data['data'].dt.dayofweek

# Classificação de estação do ano
def identificar_estacao(mes):
    if mes in [12, 1, 2]:
        return 'verao'
    elif mes in [3, 4, 5]:
        return 'outono'
    elif mes in [6, 7, 8]:
        return 'inverno'
    else:
        return 'primavera'

data['estacao'] = data['mes'].apply(identificar_estacao)

# Criação de categoria para temperatura
data['temp_categoria'] = pd.cut(data['temperatura'], bins=[0, 30,
   32, np.inf], labels=['baixa', 'media', 'alta'])

# Padronização (Z-score) das variáveis numéricas
data[['temperatura_z', 'umidade_z']] = data[['temperatura', '
   umidade']].apply(lambda x: (x - x.mean()) / x.std())

data[['data', 'temperatura', 'umidade', 'consumo_kwh', 'mes', '
   dia_da_semana', 'estacao', 'temp_categoria', 'temperatura_z', '
   umidade_z']]
```

O que foi feito?

- Extração de informações temporais: mês e dia da semana a partir da data;
- Criação de nova variável categórica: estação do ano baseada no mês;
- Agrupamento de temperaturas em faixas: baixa, média e alta;

- Padronização (Z-score): transformação das variáveis temperatura e umidade para que tenham média 0 e desvio padrão 1, o que é útil em modelos que dependem de distância ou gradientes.

A Engenharia de Características é fundamental para elevar o desempenho de modelos de Machine Learning, pois fornece uma visão mais rica dos dados. Em muitos casos, gastar tempo aprimorando e criando atributos relevantes traz melhorias maiores do que simplesmente trocar de algoritmo.

Fique Alerta!

Embora seja tentador criar dezenas de colunas, tome cuidado para não gerar "ruído" ou superdimensionar o problema (o que pode levar à "maldição da dimensionalidade"). Use métricas de validação para verificar se os atributos criados estão, de fato, melhorando seu modelo.

1.5.7 Aplicando seus conhecimentos

1. Você possui uma coluna que indica o tipo de solo de uma plantação. Transforme essas categorias usando Label Encoding. Qual seria o risco de o modelo interpretar o valor 3 como "maior" que o 1? Como resolver isso usando One-Hot Encoding?

Copie e Teste!

```
import pandas as pd

# Dataset de exemplo
dados = pd.DataFrame({
    "Área": ["A1", "A2", "A3", "A4", "A5"],
    "Solo": ["argiloso", "arenoso", "argiloso", "siltoso", "
   arenoso"]
})

# Dicionário para codificação
mapeamento_solo = {
    "argiloso": 1,
    "arenoso": 2,
    "siltoso": 3
}

# Aplicar label encoding
dados["Solo_Encoded"] = dados["Solo"].map(mapeamento_solo)

print(dados)
```

2. Considere um pequeno dataset com dados hipotéticos de clima e temperatura média. O que acontece se os dados tiverem valores extremos (ex.: uma semana com 100 mm de chuva)? Qual técnica parece menos sensível a essas variações? Se você fosse usar esse dataset para prever o crescimento de uma planta, por que normalizar ajudaria?

Copie e Teste!

```
import pandas as pd
import numpy as np

# Novo dataset
df = pd.DataFrame({
    "ChuvaSemanal": [10, 25, 40, 35, 20],
    "TemperaturaMedia": [24.5, 27.0, 29.5, 28.0, 26.0]
})

# Min-Max Normalization
min_max = (df - df.min()) / (df.max() - df.min())

# Mean Normalization
mean_norm = (df - df.mean()) / (df.max() - df.min())

# Z-Score Standardization
z_score = (df - df.mean()) / df.std()

# Exibir resultados
print("=== Dados Originais ===\n", df)
print("\n=== Min-Max Normalization ===\n", min_max)
print("\n=== Mean Normalization ===\n", mean_norm)
print("\n=== Z-Score Standardization ===\n", z_score)
```

1.6 Taxa de aprendizagem

Iniciando o diálogo...

Imagine que você está em uma trilha na Floresta Amazônica e precisa seguir um mapa até chegar ao seu destino. Se caminhar rápido demais, pode se perder; se caminhar muito devagar, pode demorar muito para chegar. A taxa de aprendizado (ou learning rate, geralmente representada pela letra grega α) tem um papel semelhante em modelos de Machine Learning que usam métodos de otimização, como o gradient descent. Ela controla o "passo" dado na busca pela minimização do erro.

1.6.1 Conceituando a Taxa de Aprendizado

No contexto de algoritmos de Machine Learning que utilizam métodos de gradiente (*gradient-based methods*), a taxa de aprendizado (α) diz respeito a quanto ajustamos os parâmetros do modelo a cada iteração de treinamento. Em termos mais práticos:

- α **pequeno:** O "passo" de ajuste dos parâmetros é curto. O aprendizado avança devagar, mas em geral tende a ser mais estável (com menor risco de "pular" o ótimo);
- α **grande:** O "passo" de ajuste é maior. O aprendizado pode convergir rapidamente, mas corre o risco de ultrapassar o mínimo e "oscilar", sem chegar a uma convergência adequada.

Em uma linguagem mais simples, é como decidir quão rápido você vai ajustar suas respostas em um teste de múltipla escolha com tempo limitado. Ajustes pequenos (cautelosos) podem garantir mais precisão, porém podem requerer muitas tentativas; ajustes grandes podem levá-lo rapidamente a uma resposta, mas o risco de erro é maior.

1.6.2 Por que a Taxa de Aprendizado é Importante?

Encontrar o valor de α que seja "equilibrado" para o seu problema. Se α for muito pequena, o treino poderá levar muitas iterações e, dependendo do contexto, até tornar o processo inviável. Se for muito grande, o modelo pode não convergir, ou seja, não "aprende" de fato, pois fica "saltando" em torno do mínimo.

Conhecendo um pouco mais!

A taxa de aprendizado costuma ser ajustada de forma empírica ou por meio de técnicas como *grid search* e *random search*. Em alguns modelos avançados, a taxa de aprendizado varia durante o treinamento (ex.: *learning rate decay*, *adaptive methods*).

Em cenários como previsão de chuvas na região amazônica, acertar a taxa de aprendizado significa conseguir um modelo estável, que faça previsões climáticas sem consumir recursos computacionais em excesso ou subaproveitar dados.

1.6.3 Visão Matemática Simplificada

Considere um modelo com parâmetros θ (por exemplo, coeficientes de uma reta, pesos de uma rede neural etc.). Temos uma função de custo $J(\theta)$ que mede o quanto o modelo está errando. O procedimento de descida de gradiente (*gradient descent*) atualiza cada parâmetro θ_j assim:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Onde:

- α é a taxa de aprendizado;
- $\frac{\partial}{\partial \theta_j} J(\theta)$ é o gradiente (a "inclinação") da função de custo em relação ao parâmetro θ_j.

No início, podemos imaginar θ como uma posição qualquer na "trilha" (superfície de erro). A cada passo do gradiente, ajustamos θ na direção que faz $J(\theta)$ diminuir, mas o quanto mexemos depende diretamente de α.

1.6.4 Escolhendo o Tamanho do Passo

Suponha que uma cooperativa de produtores de açaí no interior do Amazonas deseje prever quanto colher em cada safra, considerando fatores como quantidade de chuva, temperatura e histórico de vendas. Eles estão treinando um modelo de regressão simples:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Onde:

- $\hat{y}$ representa a produtividade estimada;

- θ_i são os parâmetros a serem aprendidos;
- x_i são as variáveis (chuva, temperatura, entre outras).

Se a taxa de aprendizado for muito grande, o algoritmo pode saltar de uma previsão de safra para outra, nunca "afinando" a estimativa corretamente. Se a taxa for muito pequena, o modelo levará um tempo enorme para chegar em resultados úteis e, no ritmo de uma safra, isso pode ser tarde demais para o planejamento.

1.6.5 Ajustando na Prática

Normalmente, começamos com um valor α em torno de 0,01 ou 0,001 (dependendo da escala dos dados) e observamos se o erro (função de custo) diminui suavemente. Podemos monitorar o erro ao longo das iterações, em alguns cenários estabelecer um número máximo de iterações ou um critério de convergência, como $\Delta J(\theta)$ menor que um valor mínimo.

- Se o erro diminui, mas de forma muito lenta, α pode estar pequena demais.
- Se o erro oscila ou aumenta, α pode estar grande demais.

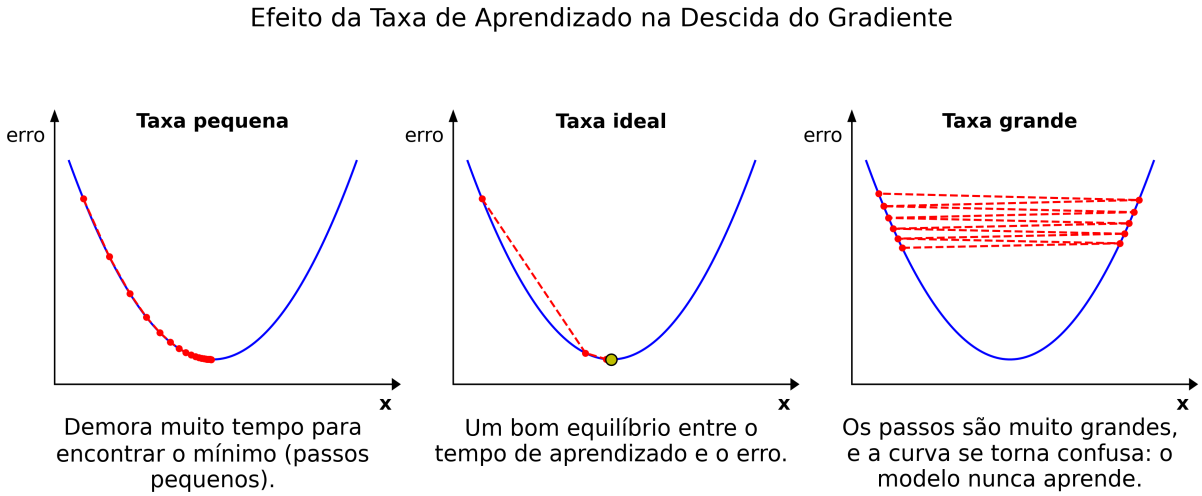


Figura 1.13: Comparação do comportamento da função de custo com diferentes taxas de aprendizado.

Conhecendo um pouco mais!

Ferramentas como TensorFlow, PyTorch ou bibliotecas como scikit-learn facilitam o ajuste do learning rate. Em tutoriais avançados, são comuns estratégias de *learning rate scheduling* que reduzem a taxa de aprendizado à medida que o erro converge.

1.6.6 Devo escolher sempre?

A escolha de uma boa taxa de aprendizado não é apenas uma questão de "otimizar o código", também é fundamental ter previsões seguras e rápidas, sejam em quaisquer áreas. Ajustar adequadamente a taxa de aprendizado permite menor gasto computacional, evitando milhões de iterações desnecessárias, como ocorre quando α é minúscula, assim reduz tempo e consumo de energia.

Fique Alerta!

Lembre-se de que nenhum modelo é isolado; cada escolha de α envolve contexto. Se os dados do seu problema forem muito grandes ou muito variáveis (como dados de monitoramento de chuva, umidade e temperatura em diferentes regiões do Amazonas), poderá ser necessário um ajuste dinâmico ou algoritmos de otimização mais avançados (como Adam, RMSProp, etc.).

A taxa de aprendizado é um dos hiperparâmetros mais importantes em algoritmos de Machine Learning baseados em gradiente. Apesar de parecer "apenas um número", determina, em muitos casos, se o seu modelo chegará a uma solução aceitável ou se ficará "preso" em tentativas frustradas.

Em termos práticos, mantenha sempre um olhar investigativo e faça experimentos gradativos com diferentes valores de α. Monitore a função de custo e avalie o equilíbrio entre velocidade e estabilidade de convergência. Nesse processo, lembre-se de considerar o contexto, dados e recursos computacionais podem ser limitados ou distribuídos em localidades com pouca infraestrutura, o que torna o ajuste adequado da taxa de aprendizado ainda mais determinante para o sucesso do projeto.

1.6.7 Aplicando seus conhecimentos

Neste exercício, você vai simular um pequeno projeto de regressão linear usando dados fictícios. A proposta é compreender, de forma prática, como a taxa de aprendizado (α) influencia o comportamento do algoritmo de gradiente descendente.

Etapas

1. **Crie ou baixe um conjunto simples de dados:** Neste exemplo: número de milímetros de chuva por semana (variável X) e quantidade de produção de açaí em kg (variável Y). Um total de 10 a 20 amostras já é suficiente para este experimento.

2. **Implemente um modelo de regressão linear:** Use uma implementação manual (com gradiente descendente), como já vimos anteriormente, e insira diferentes valores de taxa de aprendizado (α), como: $\alpha = 0.1$, $\alpha = 0.01$, $\alpha = 0.0001$.

3. **Monitore o número de iterações necessárias para convergir:** Observe se o custo diminui de forma estável ou se há oscilações nos valores. Anote o número de épocas (iterações) necessárias até o custo se estabilizar.

Para cada valor de α testado, plote a curva do custo (erro) em função do número de iterações. Isso vai te permitir visualizar claramente o impacto da taxa de aprendizado:

- Um valor alto de α pode fazer o custo oscilar ou até divergir.
- Um valor muito pequeno faz a curva descer lentamente, tornando o treinamento mais demorado.
- Um valor adequado mostra uma descida suave e consistente até a convergência.

Compare os gráficos lado a lado para entender visualmente os efeitos. Essa análise ajuda a desenvolver intuição sobre o processo de otimização.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt

# Dados fictícios: Chuva (mm) vs Produção de Açaí (kg)
X = np.array([120, 80, 150, 90, 110], dtype=float)
Y = np.array([55, 32, 63, 40, 52], dtype=float)

# Normaliza os dados para melhorar o desempenho do gradiente
   descendente
X_norm = (X - X.mean()) / X.std()

# Parâmetros
alphas = [0.1, 0.01, 0.0001]
epochs = 100
m = len(X)

# Função de custo
def compute_cost(theta0, theta1, X, Y):
    predictions = theta0 + theta1 * X
    return ((predictions - Y) ** 2).mean() / 2

# Gradiente descendente
def gradient_descent(X, Y, alpha, epochs):
    theta0, theta1 = 0.0, 0.0
    costs = []
    for _ in range(epochs):
        predictions = theta0 + theta1 * X
        errors = predictions - Y
        grad0 = errors.mean()
        grad1 = (errors * X).mean()
        theta0 -= alpha * grad0
        theta1 -= alpha * grad1
        costs.append(compute_cost(theta0, theta1, X, Y))
    return theta0, theta1, costs

# Plot das curvas de custo
plt.figure(figsize=(10, 5))
for alpha in alphas:
    theta0, theta1, costs = gradient_descent(X_norm, Y, alpha,
   epochs)
    plt.plot(range(epochs), costs, label=f'α = {alpha}')

plt.title('Curva de Custo por Época')
plt.xlabel('Iterações')
plt.ylabel('Custo')
plt.legend()
plt.grid(True)
plt.tight_layout()
```

```
plt.show()

# Visualização das retas de regressão ajustadas
x_range = np.linspace(X_norm.min(), X_norm.max(), 100)
plt.figure(figsize=(10, 5))
for alpha in alphas:
    theta0, theta1, _ = gradient_descent(X_norm, Y, alpha, epochs)
    y_pred = theta0 + theta1 * x_range
    plt.plot(x_range, y_pred, label=f'α = {alpha}')

# Dados reais (normalizados)
plt.scatter(X_norm, Y, color='black', label='Dados reais')
plt.title('Retas de Regressão com Diferentes α')
plt.xlabel('Chuva (normalizada)')
plt.ylabel('Produção de Açaí (kg)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

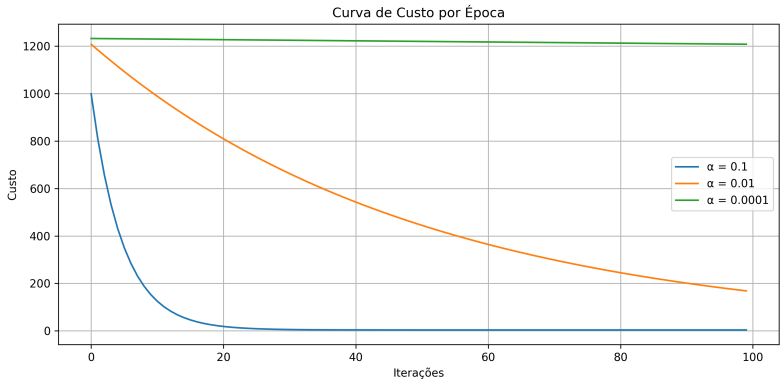


Figura 1.14: Comparação da convergência da função de custo para diferentes taxas de aprendizado (α).

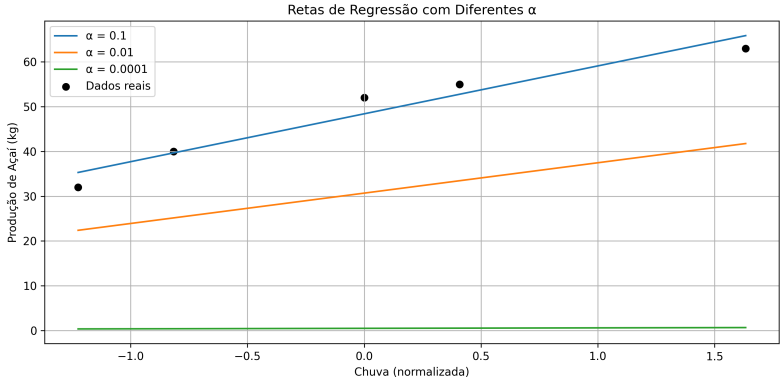


Figura 1.15: Comparação das retas de regressão ajustadas com diferentes taxas de aprendizado (α).

Agora responda...

1. Qual foi o comportamento da curva de custo para cada valor de α (0.1, 0.01 e 0.0001)?
2. Qual valor de α convergiu mais rapidamente? Houve algum valor que causou instabilidade ou oscilação? Por quê?
3. O que aconteceu com o valor de α muito pequeno? E o muito grande?
4. As retas finais de regressão ficaram muito diferentes entre si? Por quê, mesmo com taxas de aprendizado distintas?
5. Experimente e verifique caso aumentarmos o número de épocas para $\alpha = 0.0001$, você acha que a reta final seria similar às outras? Justifique.

1.7 Avaliação do modelo de regressão

Iniciando o diálogo...

Você já se perguntou como avaliar se um modelo de regressão que prevê valores numéricos, como temperatura, quantidade de chuva ou até mesmo vendas de uma loja está de fato desempenhando bem? Nesta seção, vamos conhecer duas métricas importantes para medir a qualidade das previsões de um modelo de regressão: o RMSE (*Root Mean Squared Error*) e o R^2 (*Coeficiente de Determinação*).

1.7.1 Por que avaliar um modelo de regressão?

Construímos modelos para prever valores numéricos com base em diversas variáveis (também chamadas de features ou atributos). Por exemplo, podemos prever:

- A precipitação média em uma região da Amazônia no próximo mês, dada a quantidade de chuva nos meses anteriores;
- O volume de vendas de uma cooperativa local, com base em dados de estoque e procura;
- A temperatura média diária, considerando medições históricas.

Contudo, nem sempre um modelo está "bom" apenas porque parece plausível. Precisamos de medidas que nos mostrem, numericamente, se as previsões estão próximas dos valores reais. É aí que entram as métricas de avaliação, como o RMSE e o R^2.

1.7.2 Erro Médio Quadrático (MSE) e Raiz do Erro Médio Quadrático (RMSE)

Para entender o RMSE, vamos primeiro conhecer o Erro Médio Quadrático (MSE – *Mean Squared Error*), que é definido da seguinte forma:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

onde:

- y_i é o valor real (observado).
- $\hat{y}_i$ é o valor previsto pelo modelo.
- n é a quantidade total de observações (amostras).

O RMSE (*Root Mean Squared Error*) nada mais é do que a raiz quadrada desse valor:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

O que significa o RMSE?

Se o RMSE for zero, o modelo prevê exatamente o valor real para todas as amostras. Quanto menor o RMSE, mais próximo o modelo tende a estar dos valores reais. A interpretação numérica do RMSE é intuitiva: se você obtém RMSE = 2, significa que, em média, seu modelo "erra" cerca de 2 unidades em suas previsões (dependendo de qual grandeza se está medindo).

Fique Alerta!

Se as unidades reais forem, por exemplo, em milímetros de chuva, o RMSE também será expresso em milímetros. É importante verificar se esse valor faz sentido no contexto do problema. Um RMSE de 2 mm de chuva pode ser excelente se estivermos falando de um total de 10 mm, mas pode ser irrelevante se a média de chuva for 300 mm.

1.7.3 Coeficiente de Determinação (R^2)

Enquanto o RMSE mede quão distantes estão as previsões do modelo em relação aos valores observados, o R^2 (R quadrado) avalia quão bem o modelo explica a variação dos dados. A fórmula resumida do R^2 é:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

onde:

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ é a soma dos erros quadráticos (igual à parte de cima do MSE).
- $\sum_{i=1}^{n}(y_i - \bar{y})^2$ é a soma dos quadrados da diferença entre cada valor real e a média $\bar{y}$.

Como interpretar o R^2?

Em termos simples, o R^2 indica qual fração da variação total dos dados está sendo "capturada" pelo modelo.

- **R^2 próximo de 1:** o modelo explica muito bem a variação dos dados.
- **R^2 próximo de 0:** o modelo não explica praticamente nada; é quase como fazer previsões aleatórias.
- **R^2 negativo:** pode acontecer em situações em que o modelo está prevendo pior do que uma simples média dos valores.

Caso Prático

Estimativa de Precipitação
Vamos imaginar um cenário em que desejamos prever a precipitação (em milímetros) em uma região da Amazônia, utilizando informações como umidade e temperatura dos últimos dias. A seguir, temos um exemplo com valores reais observados e valores preditos por um modelo de regressão. Vamos calcular duas métricas importantes para avaliação: o RMSE (erro quadrático médio da raiz) e o R^2 (coeficiente de determinação).

Copie e Teste!

```python
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

# Valores reais de precipitação (em mm)
y_real = np.array([12, 10, 25, 18, 30, 28])

# Valores previstos pelo modelo
y_pred = np.array([14, 9, 20, 16, 31, 27])

# Cálculo do MSE e RMSE
mse = mean_squared_error(y_real, y_pred)
rmse = np.sqrt(mse)

# Cálculo do coeficiente de determinação (R²)
r2 = r2_score(y_real, y_pred)

# Impressão dos resultados
print("Valores reais de precipitação:", y_real)
print("Valores previstos pelo modelo:", y_pred)
print(f"MSE = {mse:.2f}")
print(f"RMSE = {rmse:.2f}")
print(f"R² = {r2:.2f}")
```

Resultado Esperado

```
Valores reais de precipitação: [12 10 25 18 30 28]
Valores previstos pelo modelo: [14 9 20 16 31 27]
MSE = 6.00
RMSE = 2.45
R² = 0.90
```

O **RMSE** indica o desvio médio das previsões em relação aos valores reais, se o RMSE for 2.45, por exemplo, significa que o modelo erra, em média, 2.45 mm de precipitação por previsão. **R^2** mede o quanto da variação dos dados reais é explicada pelo modelo. Neste caso, um valor próximo de 1 indica que o modelo explica bem os dados; um valor próximo de 0 indica baixo poder explicativo.

Fique Alerta!

Você pode alterar os valores de `y_real` e `y_pred` para simular diferentes cenários.

1.7.4 Dicas para melhorar sua regressão

- **Compare diferentes algoritmos**, ou seja, não se limite à regressão linear. Experimente outras abordagens como árvores de decisão, Random Forest, k-NN ou modelos baseados em redes neurais. Cada um pode ter desempenho diferente dependendo dos dados;
- **Ajuste de hiperparâmetros faz toda diferença**, pois parâmetros como taxa de aprendizado, número de árvores ou profundidade do modelo podem impactar diretamente o RMSE e o R^2. Pequenas mudanças podem levar a grandes melhorias;
- **Normalize seus dados sempre que necessário**, modelos baseados em distância ou gradiente (como SVM, k-NN e redes neurais) se beneficiam muito de normalização ou padronização. Isso ajuda o algoritmo a tratar todas as variáveis de forma equilibrada;
- **Considere sempre o contexto**, veja que métricas como RMSE e R^2 são importantes, mas o contexto real é o que define a qualidade de um modelo.

Pense sempre no contexto, um erro médio de 2 mm de precipitação pode ser irrelevante, mas um erro de 20 mm pode ser decisivo para áreas com risco de enchente ou seca.

1.8 Classificação

De acordo com Mitchell (1997), a classificação visa categorizar dados de entrada em rótulos discretos, utilizando aprendizado supervisionado. É uma das tarefas centrais em Ciência de Dados e Aprendizado de Máquina. Ela aparece sempre que precisamos, a partir de um conjunto de exemplos, reconhecer padrões e atribuir "rótulos" a novos casos.

É o que acontece quando um aplicativo de mensagens filtra automaticamente spam, quando um sistema de previsão do tempo sinaliza "chuva" ou "sol" e até quando uma plataforma de streaming sugere se determinado vídeo pode ser "de seu interesse" ou não. Nesta seção, vamos explorar por que problemas de classificação são tão relevantes e de que forma eles nos ajudam a criar soluções práticas para desafios do mundo real.

1.8.1 Por que estudar classificação?

Muitas vezes, quando lidamos com dados, queremos fazer previsões. Porém, existem diferentes tipos de previsões. Em alguns casos, desejamos estimar valores numéricos contínuos, como o preço de uma casa ou a temperatura do dia seguinte, e chamamos esse tipo de problema de regressão. Em outros casos, contudo, nossa meta é tomar decisões do tipo "sim ou não", "positivo ou negativo", "compra ou não compra". Esse segundo tipo de problema recebe o nome de classificação.

Suponha que você queira criar um sistema capaz de detectar fraudes em transações bancárias. Seria inviável prever numericamente "quão fraudulento" é cada pagamento. O que precisamos saber é simplesmente se a transação é suspeita ou legítima. Esse formato de resposta categórica "fraude" ou "não fraude" é justamente o foco dos problemas de classificação.

Da mesma forma, imagine uma rede social que precisa identificar se o conteúdo de uma postagem está dentro das regras de uso ou se infringe alguma política de segurança. Novamente, temos um problema em que a resposta não é um valor numérico, mas a escolha de uma categoria: "permitido" ou "proibido". São incontáveis as aplicações em que a principal ação é classificar algo como pertencendo a uma categoria ou a outra (ou até a várias categorias possíveis).

1.8.2 Reconhecendo padrões e tomando decisões

O principal objetivo dos algoritmos de classificação é aprender a separar corretamente os diferentes tipos de objetos (ou acontecimentos) que analisamos. De modo geral, lidamos com duas grandes etapas:

- **Treinamento (aprendizado):** Nesta etapa, oferecemos ao algoritmo um conjunto de exemplos com rótulos conhecidos. O modelo de classificação tenta capturar padrões que justifiquem por que cada exemplo foi marcado com cada rótulo.
- **Previsão (generalização):** Depois que o modelo foi treinado, ele passa a "prever" o rótulo de novos exemplos, ou seja, de exemplos que nunca foram vistos antes. Com base na experiência acumulada no treinamento, o algoritmo decide se o novo exemplo é ou não é aquele rótulo.

Para entender a importância disso, imagine um agricultor que monitora sua lavoura por meio de imagens de drones. A cada voo, são geradas dezenas ou centenas de imagens. Verificar manualmente quais áreas apresentam sinais de pragas, deficiência de nutrientes ou estresse hídrico seria extremamente trabalhoso, e muitas vezes exigiria a ajuda de um agrônomo. Agora, imagine um modelo treinado para classificar automaticamente essas imagens e apontar regiões críticas. Isso poupa tempo, reduz custos e permite uma intervenção mais rápida e precisa.

Esse mesmo princípio se aplica a diversas outras áreas: monitoramento de safras, recomendação de práticas de cultivo, classificação de tipos de solo, previsão de produtividade, entre outros. Em todos esses casos, o desafio é basicamente o mesmo: identificar a qual categoria uma nova observação pertence, com o apoio de modelos que aprenderam com dados anteriores.

Os algoritmos de classificação trazem diversos benefícios, como a automação de tarefas repetitivas, escalabilidade para lidar com grandes volumes de dados, tomada de decisão mais rápida e precisa, além de maior padronização nas análises. No entanto, também apresentam desafios significativos, como a necessidade de dados de boa qualidade e bem rotulados, problemas com desequilíbrio entre classes, dificuldades de generalização quando o cenário muda, e a complexidade na interpretação dos resultados em modelos mais avançados. Esses fatores devem ser cuidadosamente considerados ao aplicar classificação em problemas reais.

1.8.3 Diferentes abordagens para classificação

Existem várias abordagens para resolver problemas de classificação, cada uma com características próprias. Classificadores lineares, como a Regressão Logística, utilizam uma fronteira de decisão simples para separar as classes. Árvores de decisão e Florestas Aleatórias fazem divisões sucessivas nos dados até definir um rótulo. Métodos baseados em vizinhança, como o k-NN, classificam novos exemplos com base na similaridade com casos anteriores. Já as redes neurais, inspiradas no cérebro humano, são capazes de identificar padrões complexos. A

escolha da técnica ideal depende do problema em questão, da quantidade de dados disponível, da necessidade de interpretar os resultados e dos recursos computacionais disponíveis.

1.8.4 A importância de entender a motivação antes da técnica

Muitas vezes, quando começamos a estudar algoritmos de aprendizagem de máquina, a tendência é mergulhar direto nas fórmulas e no código. Embora essa parte seja, de fato, muito importante, entender a motivação por trás da classificação nos ajuda a manter o foco em resolver problemas concretos e a perceber as possibilidades e limitações das ferramentas que vamos aprender.

Ao entendermos por que a classificação é tão usada e qual impacto isso tem no nosso dia a dia, é mais fácil apreciar a relevância das técnicas que veremos adiante, como a Regressão Logística, e por que é essencial aprender a avaliar corretamente nossos modelos por meio de métricas adequadas (acurácia, precisão, recall, F1, etc.). Isso tudo servirá para termos uma visão mais completa de como um sistema de classificação deve ser planejado, construído e analisado, sempre com a consciência de que ele existe para resolver um problema real.

Classificar é muito mais do que simplesmente agrupar objetos em categorias: trata-se de um processo de decisão fundamentado em dados. O estudo da classificação nos aproxima do mundo real, pois é por meio dela que reconhecemos padrões, tomamos atitudes preventivas (como bloquear fraudes) e geramos conveniência (recomendar filmes, músicas e textos de interesse). É fundamental em inúmeros setores da sociedade, unindo conhecimentos de diferentes áreas e reflete uma necessidade humana fundamental, a de distinguir e categorizar.

1.8.5 Aplicando seus conhecimentos

Veja abaixo um exemplo prático de classificação usando o famoso dataset Iris, amplamente utilizado em tarefas de aprendizado supervisionado. Utilizaremos o algoritmo k-NN para prever a espécie da flor com base nas medidas de suas pétalas e sépalas. Neste exemplo,

a) Carregamos o dataset, que contém medidas de flores de três espécies diferentes;

b) Dividimos os dados em conjunto de treino (70%) e teste (30%) para avaliar a performance do modelo em dados não vistos;

c) Padronizamos as variáveis, pois o algoritmo k-NN é sensível à escala dos dados; e

d) Treinamos um classificador k-NN com hiperparâmetro k=3, ou seja, o modelo considera os 3 vizinhos mais próximos para classificar uma nova amostra.

Copie e Teste!

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Carregando o dataset Iris
iris = load_iris()
X = iris.data[:, 2:4] # Usando apenas comprimento e largura da
```

```
   pétala (features 2 e 3)
y = iris.target
target_names = iris.target_names

# 2. Dividindo em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.3, random_state=42)

# 3. Padronização
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Treinamento do modelo k-NN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

# 5. Previsões e avaliação
y_pred = knn.predict(X_test_scaled)

print("Acurácia:", accuracy_score(y_test, y_pred))
print("\nRelatório de Classificação:\n")
print(classification_report(y_test, y_pred, target_names=
   target_names))

# 6. Visualização dos dados
plt.figure(figsize=(8, 6))

# Plotando os pontos de teste com cores de acordo com o rótulo
   previsto
for i, target_name in enumerate(target_names):
    plt.scatter(
        X_test_scaled[y_pred == i][:, 0],
        X_test_scaled[y_pred == i][:, 1],
        label=f'Predito: {target_name}',
        edgecolor='black',
        alpha=0.6
    )

plt.xlabel('Comprimento da Pétala (padronizado)')
plt.ylabel('Largura da Pétala (padronizado)')
plt.title('Classificação das flores do Iris dataset (k-NN)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Resultado Esperado

```
Acurácia: 1.0

Relatório de Classificação:
              precision    recall  f1-score   support
setosa           1.00        1.00      1.00        19
versicolor       1.00        1.00      1.00        13
virginica        1.00        1.00      1.00        13

accuracy                             1.00        45
macro avg        1.00        1.00      1.00        45
weighted avg     1.00        1.00      1.00        45
```

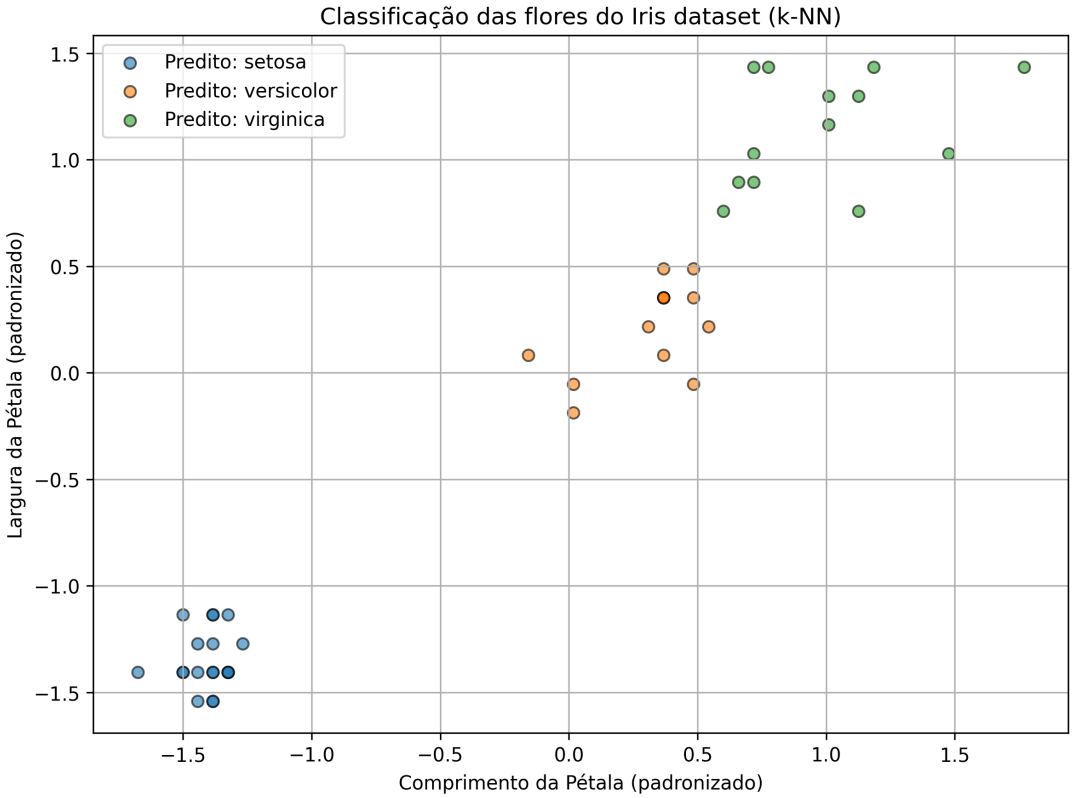


Figura 1.16: Visualização da classificação do dataset Iris utilizando o algoritmo k-NN.

No gráfico acima, cada ponto representa uma flor da base de teste, com a cor indicando a classe prevista pelo modelo. Essa visualização permite observar de forma intuitiva como o classificador está separando os dados com base nas características fornecidas, revelando se há sobreposição entre classes ou se a fronteira de decisão está bem definida.

1.9 Regressão logística

Iniciando o diálogo...

A Regressão Logística é uma técnica de aprendizado de máquina voltada para tarefas de classificação. Embora o nome "regressão" possa causar alguma confusão, na prática, a regressão logística é usada para prever categorias (por exemplo, "sim" ou "não", "positivo" ou "negativo", "desmatamento" ou "não desmatamento"). Ela difere da regressão linear porque, em vez de gerar valores contínuos como resposta, trabalha com probabilidades associadas a cada classe.

1.9.1 Por que utilizar a Regressão Logística?

Por que não usar simplesmente Regressão Linear para classificação? A regressão linear gera valores contínuos que podem ficar abaixo de 0 ou acima de 1, o que não faz sentido para interpretar probabilidades que devem estar sempre entre 0 e 1. Além disso, a regressão linear pode ter resultados extremos ou inconsistentes para dados onde a variável-alvo é claramente categórica. Já a regressão logística resolve essa limitação ao aplicar a função sigmoide (ou logística) sobre uma combinação linear das variáveis de entrada. Dessa forma, a saída resultante fica sempre no intervalo (0, 1), interpretável como probabilidade.

1.9.2 Modelo Matemático e a Função Sigmoide

A função sigmoide é definida como:

$$\text{sigm}(z) = \frac{1}{1 + e^{-z}}$$

Em que z é uma combinação linear das variáveis de entrada (features). Se tivermos um conjunto de variáveis $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ e $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ pesos, podemos escrever z como:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

onde w_0 é o termo de bias (ou intercepto do modelo) e cada w_i é o peso associado a cada variável de entrada x_i.

Aplicando a função sigmoide, temos:

$$h_w(\mathbf{x}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_n x_n)}}$$

Essa saída $h_w(\mathbf{x})$ pode ser interpretada como a probabilidade de a observação pertencer à classe "1" (ex.: risco de desmatamento). Automaticamente, a probabilidade de pertencer à classe "0" seria $1 - h_w(\mathbf{x})$.

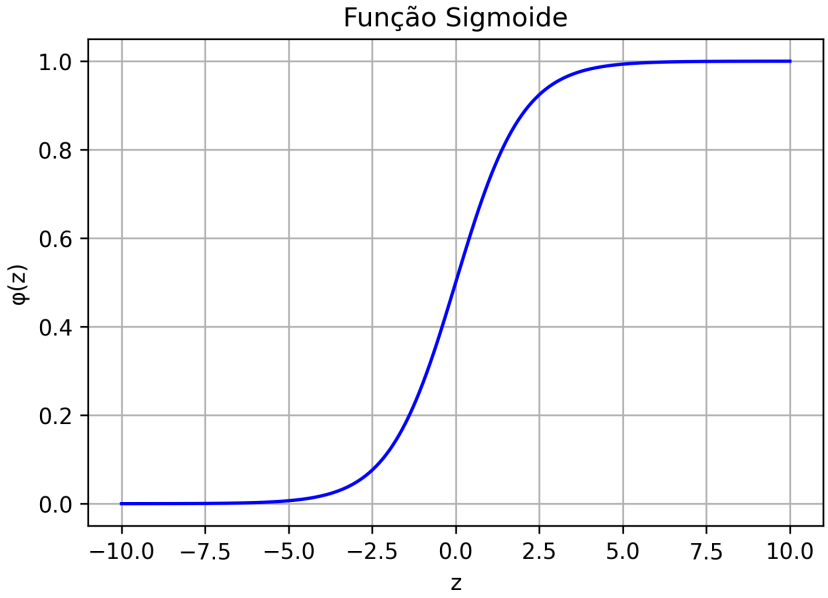


Figura 1.17: Função Sigmoide.

Para entender isso de forma intuitiva, pense que z faz uma soma ponderada das características (dados de entrada). Se essa soma for grande e positiva, a função sigmoide tende a valores próximos de 1 (interpretamos como "alta probabilidade de classe 1"). Se a soma for muito negativa, a sigmoide se aproxima de 0 (probabilidade de classe 0). Por exemplo, considere um modelo simples para predizer a probabilidade de desmatamento em uma determinada área:

$$z = w_0 + w_1 \times (\text{Área de floresta}) + w_2 \times (\text{Distância de rodovias})$$

Se a área de floresta for muito grande (e o peso w_1 for positivo), z tende a aumentar, indicando maior probabilidade de desmatamento. Se a distância de rodovias for muito pequena (e w_2 for negativo), significa que quanto mais próximo de rodovias, maior chance de desmatamento, logo z também tende a subir nesse caso, apontando para classe 1. É claro que a importância de cada variável depende dos pesos aprendidos automaticamente pelo modelo durante o processo de treinamento.

1.9.3 Função de Custo (Cost Function)

Para medir o desempenho do modelo, utilizamos a Função de Custo de Regressão Logística, também conhecida como *Loss Function* baseada em "cross-entropy". Em vez de medir erros quadráticos (como na regressão linear), a regressão logística utiliza a seguinte forma:

$$J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_w(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h_w(\mathbf{x}^{(i)}))]$$

onde:

- m é o número de exemplos de treinamento.
- $y^{(i)}$ é o valor real da classe do i-ésimo exemplo (0 ou 1).
- $h_w(\mathbf{x}^{(i)})$ é a probabilidade prevista para a classe 1.

Essa função de custo penaliza fortemente quando o modelo tem alta certeza (probabilidade próxima de 1 ou 0) mas erra a classe. É isso que faz a regressão logística ser robusta para classificação binária.

1.9.4 Otimização por Gradiente

Para encontrar os melhores pesos $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, utiliza-se geralmente o Gradiente Descendente (*Gradient Descent*). A ideia básica é:

1. Iniciar os pesos com valores aleatórios ou zeros.
2. Calcular a função de custo $J(\mathbf{w})$ e os gradientes parciais em relação a cada w_i.
3. Atualizar cada peso, movendo-se na direção contrária ao gradiente (que é onde o custo diminui).
4. Repetir até que o custo não apresente redução significativa ou alcance um número de iterações pré-determinado.

Matematicamente, cada atualização se parece com:

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

em que α é a taxa de aprendizado (*learning rate*), responsável pelo "tamanho do passo" que damos em cada atualização.

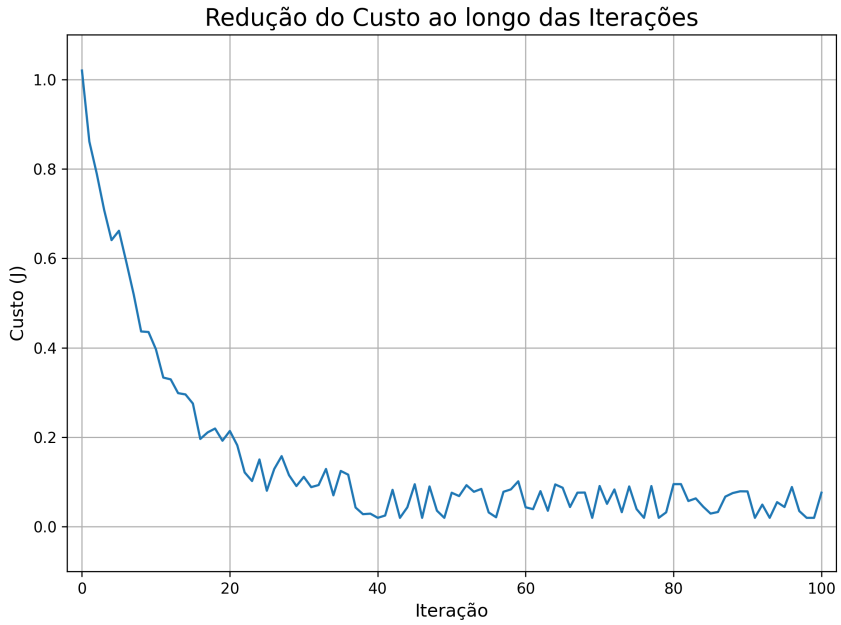


Figura 1.18: Redução do Custo ao longo das Iterações.

1.9.5 Aplicando seus conhecimentos

Abaixo, mostramos um pequeno exemplo prático em Python usando a biblioteca scikit-learn, que oferece a implementação de regressão logística. Suponha que temos um conjunto de dados fictício que relaciona algumas características ambientais e o rótulo de "desmatamento" (1) ou "não desmatamento" (0).

Copie e Teste!

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Exemplo fictício
# Vamos criar um DataFrame com variáveis simuladas:
data = {
    'area_floresta': [10, 50, 80, 30, 60, 90, 45, 85, 20, 70],
    'dist_rodovias': [1, 3, 5, 2, 4, 8, 6, 7, 2, 9],
    'desmatamento':  [0, 1, 1, 0, 1, 1, 1, 1, 0, 1]
}
df = pd.DataFrame(data)

# Separe variáveis de entrada (X) e o alvo (y)
X = df[['area_floresta', 'dist_rodovias']]
y = df['desmatamento']

# Dividindo em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.2, random_state=42)

# Criando o modelo de Regressão Logística
modelo = LogisticRegression()
modelo.fit(X_train, y_train)

# Previsões e acurácia
y_pred = modelo.predict(X_test)
acuracia = modelo.score(X_test, y_test)

print("Previsões no conjunto de teste:", y_pred)
print("Acurácia do modelo:", acuracia)
```

Resultado Esperado

```
Previsões no conjunto de teste: [0 1]
Acurácia do modelo: 1.0
```

Neste exemplo simples:

- Criamos dados fictícios para área de floresta (em hectares), distância de rodovias e rótulos (desmatamento ou não).
- Dividimos os dados em conjunto de treino (80%) e teste (20%).
- Treinamos uma regressão logística e avaliamos a acurácia no conjunto de teste.

Naturalmente, em aplicações reais, o conjunto de dados seria muito maior e com mais variáveis (chuva, uso do solo, tipo de vegetação etc.).

1.9.6 Fronteira de Decisão

A fronteira de decisão é a linha (em problemas 2D) ou superfície (em problemas com mais dimensões) que separa as regiões de previsão para cada classe. Por exemplo, se $\mathbf{x}$ estiver de um lado da fronteira, o modelo atribui a classe 0; se estiver do outro lado, classe 1. Na regressão logística, essa fronteira é definida quando:

$$h_w(\mathbf{x}) = 0.5 \iff \frac{1}{1+e^{-z}} = 0.5 \iff z = 0$$

Ou seja, a condição $\mathbf{w} \cdot \mathbf{x} = 0$ (somando o termo bias) define geometricamente a fronteira. Apenas olhar para os números de acurácia ou as previsões finais pode não ser suficiente para entender como um modelo realmente está se comportando. Por isso, uma forma poderosa de interpretar algoritmos de classificação é visualizar como ele separa as diferentes classes no espaço das variáveis.

A seguir, aproveitando o exemplo anterior de regressão logística aplicado à previsão de desmatamento com base em características ambientais (área de floresta e distância até rodovias), vamos construir um gráfico que mostra a fronteira de decisão aprendida pelo modelo.

Copie e Teste!

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,
   classification_report

# Dados fictícios
data = {
    'area_floresta': [10, 50, 80, 30, 60, 90, 45, 85, 20, 70],
    'dist_rodovias': [1, 3, 5, 2, 4, 8, 6, 7, 2, 9],
    'desmatamento':  [0, 1, 1, 0, 1, 1, 1, 1, 0, 1]
}
df = pd.DataFrame(data)

# Separação de variáveis
X = df[['area_floresta', 'dist_rodovias']]
y = df['desmatamento']

# Padronização
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
   test_size=0.2, random_state=42)
```

```
# Modelo
modelo = LogisticRegression()
modelo.fit(X_train, y_train)

# Previsões
y_pred = modelo.predict(X_test)

# Malha de pontos para o contorno
h = .02  # Passo da malha
x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() + 1
y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Previsões na malha
Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Gráfico
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu, alpha=0.6)

# Pontos reais
scatter = plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y,
   edgecolor='k', cmap=plt.cm.bwr, s=100)
plt.title("Fronteira de decisão da Regressão Logística")
plt.xlabel("Área de floresta (padronizada)")
plt.ylabel("Distância até rodovia (padronizada)")
plt.legend(*scatter.legend_elements(), title="Desmatamento")
plt.grid(True)
plt.tight_layout()
plt.show()
```

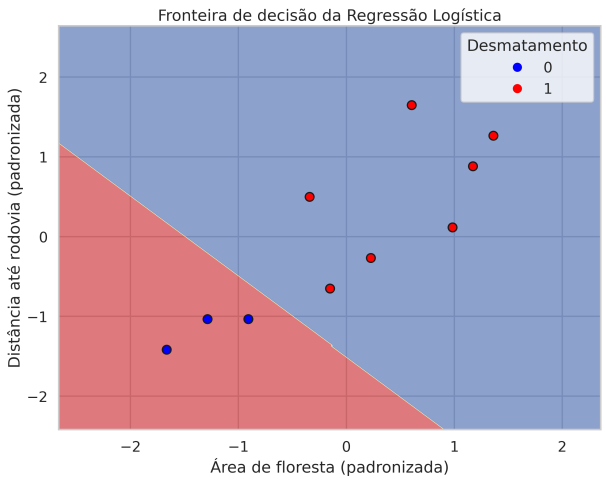


Figura 1.19: Visualização da fronteira de decisão do modelo de Regressão Logística para o problema de previsão de desmatamento.

Fique Alerta!

A regressão logística transforma uma combinação linear das variáveis de entrada em uma saída probabilística pela função sigmoide. A função de custo usada é baseada em "cross-entropy", adequada para classificação. O treinamento do modelo ajusta os pesos por meio do Gradiente Descendente ou outros métodos de otimização. A fronteira de decisão aparece onde a probabilidade calculada é 0,5, correspondente a $z = 0$.

A regressão logística é um pilar fundamental para problemas de classificação binária, seja para prever probabilidade de desmatamento, risco de doenças ou qualquer outro cenário de decisão entre duas classes. Ela mescla a simplicidade do modelo linear com a capacidade de produzir uma saída probabilística entre 0 e 1. Ao longo de estudos mais avançados, você verá variações da regressão logística (multiclasse, regularizações distintas etc.) que expandem ainda mais seu campo de aplicação. Mas o mais importante é entender sua base: função sigmoide, função de custo apropriada e o ajuste dos parâmetros via métodos de otimização.

1.10 Avaliação do modelo de classificação

A avaliação de modelos de classificação é parte essencial do desenvolvimento de soluções de Machine Learning, pois fornece uma visão abrangente de quão bem o modelo está distinguindo as classes de interesse.

1.10.1 Por que avaliar um modelo de classificação?

Imagine que queremos desenvolver um sistema para identificar, em imagens de satélite, se uma determinada área da Amazônia está sendo impactada pelo desmatamento (classe positiva) ou permanece intacta (classe negativa). Ao criar esse modelo, precisamos saber se ele realmente acerta na maior parte das vezes e, principalmente, se ele é confiável para informar políticas de proteção ambiental. É possível que um modelo apresente resultados otimistas em alguns casos pontuais e não seja realmente efetivo em um cenário mais amplo. Em situações em que a classe "desmatamento" ocorre com frequência relativamente baixa (ou seja, a maioria das áreas ainda está preservada), confiar apenas em uma métrica como a acurácia pode levar a conclusões equivocadas. Por isso, precisamos de métricas complementares, como precisão, recall, F1 Score e a análise da Curva ROC e AUC, que nos oferecem uma visão mais rica sobre os erros e acertos.

1.10.2 Matriz de confusão

Para interpretar corretamente as métricas de avaliação de um modelo de classificação, é fundamental começar pela matriz de confusão. Essa matriz nada mais é do que uma tabela que compara as previsões do modelo com os valores reais observados. Em um cenário de classificação binária, por exemplo, classificar áreas como "desmatadas" (positivo) ou "não desmatadas" (negativo), a matriz de confusão organiza os resultados em quatro categorias:

- **TP (Verdadeiro Positivo):** é quando o modelo previu que a área está desmatada, e ela realmente está.
- **TN (Verdadeiro Negativo):** é quando o modelo previu que a área não está desmatada, e isso é verdade.

- **FP (Falso Positivo)**: é quando o modelo previu desmatamento, mas a área não estava desmatada de fato.
- **FN (Falso Negativo)**: é quando o modelo previu que não houve desmatamento, mas na realidade houve.

Essa estrutura simples permite calcular métricas importantes como precisão, sensibilidade (recall), especificidade e acurácia, fornecendo uma visão mais completa da performance do modelo, especialmente quando há desequilíbrio entre as classes. Para compreender melhor as métricas de avaliação, vamos começar pela matriz de confusão, uma tabela que compara as previsões de um modelo aos valores reais. Em um problema binário, digamos, "área desmatada" (positivo) vs. "área não desmatada" (negativo), a matriz de confusão assume a forma:

	Predição Positivo	**Predição Negativo**
Real Positivo	TP (Verdadeiro Positivo)	FN (Falso Negativo)
Real Negativo	FP (Falso Positivo)	TN (Verdadeiro Negativo)

Agora vamos aproveitar o exemplo anterior de previsão de desmatamento com regressão logística e adicionar matriz de confusão.

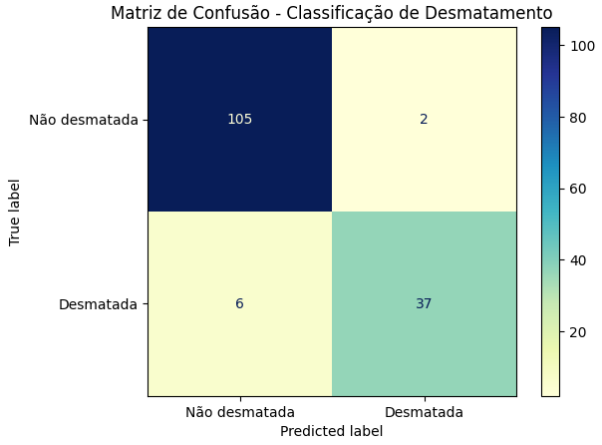
Copie e Teste!

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
   classification_report, ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt

# Dados simulados
np.random.seed(42)
n = 500
area_floresta = np.random.randint(10, 100, n)
dist_rodovias = np.random.randint(1, 20, n)
desmatamento = ((area_floresta < 60) & (dist_rodovias < 10)).
   astype(int)

df = pd.DataFrame({
    'area_floresta': area_floresta,
    'dist_rodovias': dist_rodovias,
    'desmatamento': desmatamento
})

# Separar variáveis
X = df[['area_floresta', 'dist_rodovias']]
y = df['desmatamento']

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
    test_size=0.3, random_state=42)

# Treinar o modelo
modelo = LogisticRegression()
modelo.fit(X_train, y_train)

# Previsões
y_pred = modelo.predict(X_test)

# Métricas
print("Relatório de Classificação:")
print(classification_report(y_test, y_pred, digits=2))

# Matriz de Confusão
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels
    =["Não desmatada", "Desmatada"])

# Visualização
plt.figure(figsize=(6,4))
disp.plot(cmap="YlGnBu")
plt.title("Matriz de Confusão - Classificação de Desmatamento")
plt.grid(False)
plt.show()
```

Resultado Esperado

```
Relatório de Classificação:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       107
           1       0.95      0.86      0.90        43

    accuracy                           0.95       150
   macro avg       0.95      0.92      0.93       150
weighted avg       0.95      0.95      0.95       150
```



Acurácia

A primeira métrica que normalmente vemos é a acurácia, definida como:

$$\text{Acurácia} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Significa a proporção total de acertos entre todas as previsões feitas. Ela é intuitiva, mas pode ser enganosa se as classes estiverem desbalanceadas. Suponha que 90% do território analisado não apresente sinais de desmatamento; um modelo que "chuta" sempre "não desmatado" poderá atingir 90% de acurácia sem de fato conseguir identificar as áreas em risco.

Se o conjunto de dados fosse altamente desbalanceado, por exemplo, contendo pouquíssimos casos de desmatamento em meio a muitas áreas preservadas, a acurácia poderia se mostrar elevada por motivos equivocados como "prever sempre preservado". Por isso, exploraremos mais métricas.

Precisão

A precisão (*precision*) responde à pergunta: Dentre as áreas que o modelo classificou como desmatadas, qual a proporção de vezes em que ele estava certo?

$$\text{Precisão} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Quando a precisão é alta, significa que o modelo raramente dá "alarme falso". Em um contexto amazônico, isso seria bom para evitar mobilizar fiscais ou drones para áreas que, na realidade, estariam intactas.

Recall

O recall (sensibilidade) avalia: Dentre todas as áreas que de fato estão desmatadas, quantas foram identificadas corretamente pelo modelo?

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Quando o recall é alto, significa que o modelo dificilmente deixa passar um caso real de desmatamento. Num cenário de preservação, esse aspecto pode ser crítico, pois FN (falso negativo) tende a ser muito prejudicial: deixamos de detectar um problema real. Um recall muito baixo pode indicar que muitos desmatamentos estão passando despercebidos (FN alto). Uma precisão muito baixa pode indicar que estamos tendo muitos alarmes falsos (FP alto), o que pode levar a custo excessivo de fiscalização.

F1 Score

Para conciliar a importância tanto da precisão quanto do recall, a métrica F1 Score é frequentemente usada. Ela é a média harmônica dessas duas medidas:

$$F1 = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}}$$

O F1 Score é especialmente útil em bases desbalanceadas, pois evita que o modelo foque apenas em um dos aspectos (por exemplo, ter precisão altíssima às custas de um recall péssimo,

ou vice-versa). Ele condensa em um único valor a noção de equilíbrio entre os acertos positivos corretos e a abrangência em capturá-los. Se seu problema exige um certo compromisso entre não perder exemplos positivos e não rotular negativos como positivos, o F1 Score pode ser um bom indicador de desempenho geral.

Curva ROC

Quando temos um problema de classificação binária, o modelo frequentemente retorna não apenas uma classe (0 ou 1), mas uma probabilidade associada a cada classe. Podemos então variar um limiar (*threshold*) para decidir quando essa probabilidade deve ser tratada como "positivo" ou "negativo". A Curva ROC (*Receiver Operating Characteristic*) é um gráfico em que o Eixo Y é o *True Positive Rate* (TPR), também chamado de *Recall*, e o Eixo X é o *False Positive Rate* (FPR). Para cada limiar, obtemos um par (FPR, TPR). Ao plotar todos os pares, teremos uma curva que começa em (0,0) e termina em (1,1), mostrando o comportamento do modelo sob diversos níveis de "exigência" para prever positivo.

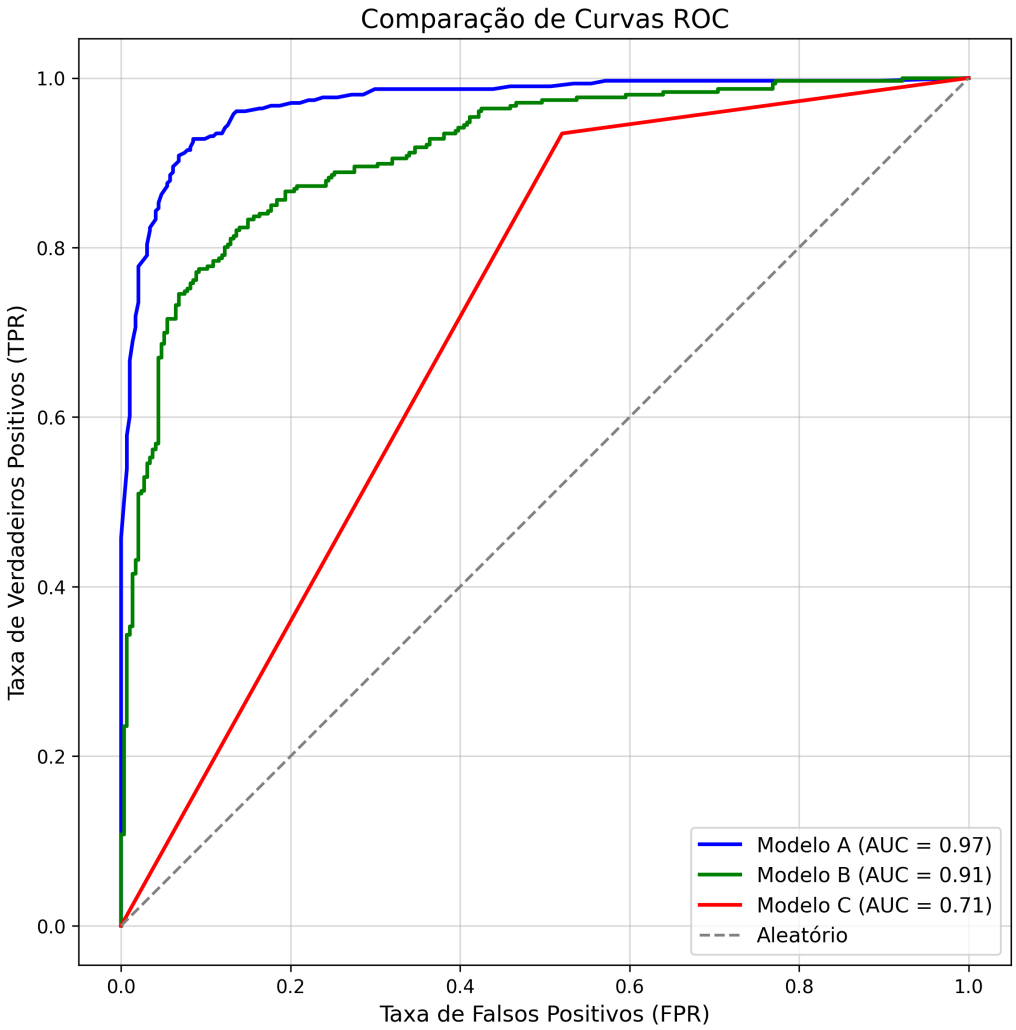


Figura 1.20: Comparação de Curvas ROC de Três Modelos.

AUC

A métrica AUC (*Area Under the ROC Curve*) resume o desempenho do modelo em termos de separação entre classes. Seu valor varia de 0 a 1, e quanto mais próximo de 1, melhor a capacidade do modelo em distinguir corretamente entre as classes positiva e negativa. De forma geral, valores acima de 0,7 já indicam uma separação satisfatória.

Copie e Teste!

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Probabilidades preditas para a classe positiva
y_pred_proba = modelo.predict_proba(X_test)[:, 1]
# Cálculo da curva ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Cálculo da AUC
auc_value = roc_auc_score(y_test, y_pred_proba)
print("AUC:", auc_value)
# Plot da Curva ROC
plt.plot(fpr, tpr, label=f"AUC = {auc_value:.2f}")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray") # linha de
    referência
plt.title("Curva ROC")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend()
plt.grid(True)
plt.show()
```

Resultado Esperado

```
AUC: 0.99
```

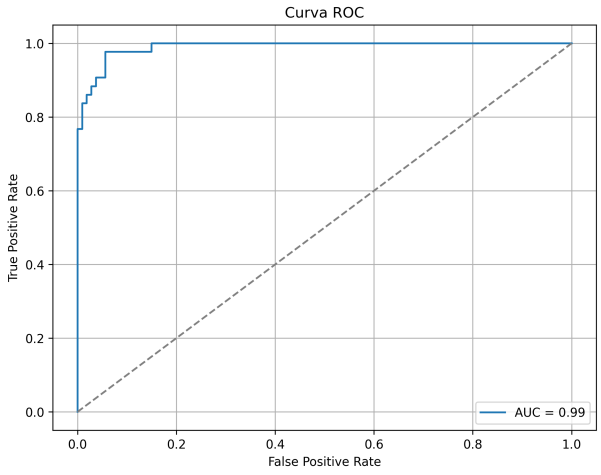


Figura 1.21: Gráfico da Curva ROC (Receiver Operating Characteristic), que mede a capacidade do modelo de distinguir entre as classes positiva e negativa.

1.11 Overfitting, viés e variância

Esta seção aborda um dos temas centrais em algoritmos de classificação e, de modo geral, em aprendizado de máquina: *overfitting* (ou sobreajuste), *bias* (ou viés) e variância. Essas noções são fundamentais para que possamos desenvolver modelos mais robustos e confiáveis em diferentes contextos, seja para prever se um aluno vai passar em uma prova, classificar imagens de áreas desmatadas na Amazônia ou mesmo identificar emoções em textos.

1.11.1 O que é Overfitting?

O fenômeno de overfitting ocorre quando o modelo se ajusta demasiadamente ao ruído dos dados de treinamento, como discutido por Goodfellow, Bengio e Courville (2016). Podemos até chamar de overfitting o fenômeno em que o modelo "decora" os dados de treinamento ao invés de aprender realmente os padrões gerais que podem ser aplicados em dados novos (de teste ou do "mundo real"). Imagine que você esteja estudando para uma prova apenas memorizando todas as perguntas de exercícios anteriores, sem realmente compreender a matéria. Se a prova tiver qualquer pergunta ligeiramente diferente, você terá dificuldade para responder. Esse é o tipo de problema que também pode acontecer com algoritmos de aprendizado de máquina.

O modelo apresenta erro (ou perda) muito baixo no conjunto de treinamento, mas um erro consideravelmente mais alto no conjunto de validação/teste. O desempenho se degrada bastante quando surgem dados que o modelo nunca tinha "visto" antes.

Imagine que você construa um classificador de imagens para distinguir entre áreas de floresta amazônica saudável e áreas degradadas. Se, ao treinar, você usar apenas fotos muito específicas (com mesma iluminação, mesma época do ano) e fizer o modelo "decorar" cada detalhe das fotos, ele pode se sair muito bem nessas imagens. Porém, ao receber uma foto de outro dia, diferente iluminação ou estação chuvosa, o classificador não saberá o que fazer. Essa é uma situação típica de overfitting.

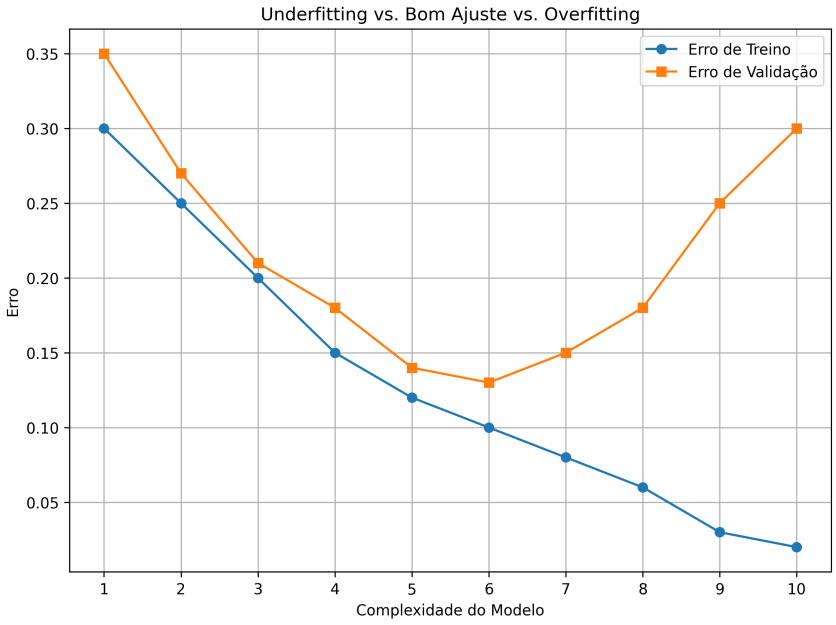


Figura 1.22: Underfitting vs. Bom Ajuste vs. Overfitting.

1.11.2 Bias e Variância

Para compreender melhor por que acontece o overfitting, precisamos entender dois conceitos: bias (viés) e variância.

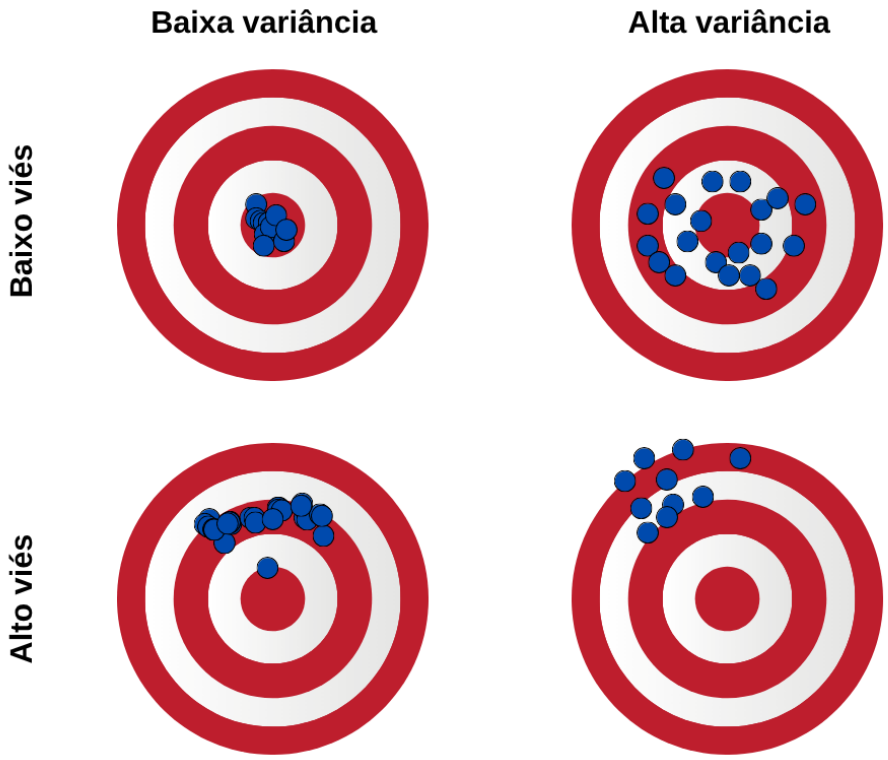


Figura 1.23: Ilustração do trade-off entre viés e variância.

Bias (Viés)

O viés de um modelo refere-se à sua capacidade de representar corretamente a relação entre as variáveis de entrada (features) e a variável de saída (rótulo ou classe). Modelos com alto viés geralmente são modelos muito simples ou com suposições fortes demais, por exemplo, supor que a relação entre as variáveis seja sempre linear. Isso tende a gerar *underfitting* (subajuste), pois o modelo não consegue capturar a complexidade real do problema. **Exemplo de alto viés:** Um classificador que sempre responde que qualquer imagem é "floresta saudável", independentemente do que realmente esteja na foto. Ele simplificou demais o problema e, consequentemente, terá baixo desempenho, mesmo que seja muito rápido de treinar.

Variância

A variância de um modelo se relaciona ao quanto esse modelo muda ao ser treinado em diferentes subconjuntos de dados. Modelos com alta variância geralmente aprendem os detalhes e ruídos do conjunto de treinamento de forma muito forte, ficando muito "sensíveis" a pequenas alterações. Esse comportamento leva ao *overfitting*, o modelo "decora" o conjunto de treinamento e não consegue generalizar. **Exemplo de alta variância:** Um classificador que distingue floresta de áreas degradadas com tanta precisão e tanto detalhamento específico do conjunto de treino, como padrões de pixels ou pequenos ruídos da foto que não consegue

repetir esse desempenho em novas imagens. No treino, ele é quase "perfeito"; no teste, falha consideravelmente.

Equilíbrio entre Viés e Variância

No aprendizado de máquina, buscamos equilibrar bias e variância de modo a obter um modelo que generalize bem. A ideia é não cair nos extremos:

- Alto viés (modelo muito simples, que subestima a complexidade do problema);
- Alta variância (modelo muito complexo, que decora o conjunto de treinamento e não generaliza).

Em termos práticos, quando treinamos um modelo de classificação, sempre monitoramos como ele se comporta tanto nos dados de treino quanto em dados de validação ou teste. Se o erro de treino estiver muito baixo, mas o erro de validação/teste for alto, suspeitamos de overfitting. Se o erro em ambos for alto, provavelmente estamos diante de underfitting.

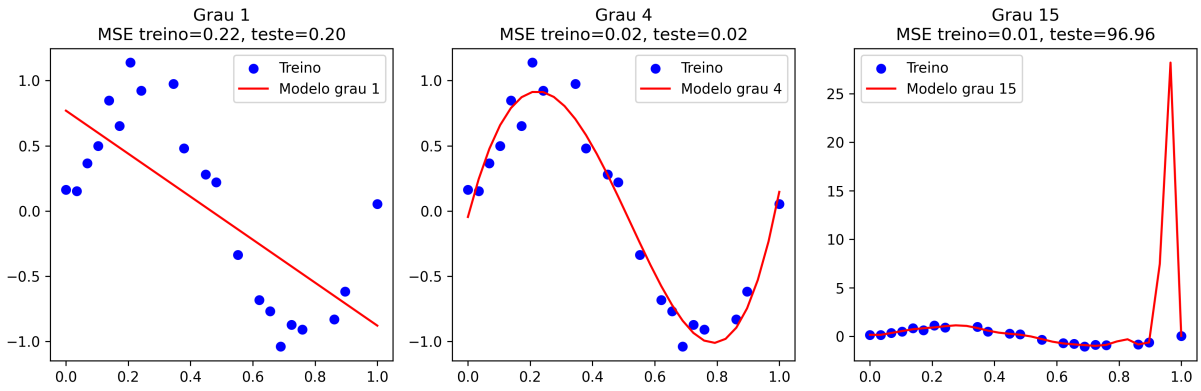


Figura 1.24: Comparação de modelos polinomiais de diferentes graus.

Veja, o modelo de grau 1 subestima a complexidade da relação (underfitting). O modelo de grau 4 faz um bom compromisso entre ajustar os dados e manter uma boa generalização. Agora, o modelo de grau 15 ajusta demais os dados de treino e falha em prever corretamente novos exemplos, caracterizando o overfitting.

1.11.3 Ilustração Matemática Simplificada

Considere uma função de custo $J(\theta)$ que mede o desempenho do modelo, por exemplo, a entropia cruzada na regressão logística ou o erro quadrático médio em outros contextos. Quando o modelo tem alto viés, é como se tivéssemos poucos parâmetros ou fossemos muito restritivos na forma da função: não conseguimos "encaixar" bem a solução para os dados. O erro tende a ser alto, tanto em treino quanto em teste. Quando o modelo tem alta variância, significa que temos muitos parâmetros e uma forma muito "flexível". Consegue-se um erro baixo em treino, porém há uma grande chance de o erro crescer em teste.

Formalmente, o erro esperado de um modelo pode ser aproximado pela soma de:

$$\text{Erro} = \text{Viés}^2 + \text{Variância} + \text{Ruído intrínseco dos dados}$$

Nosso objetivo é escolher um modelo (ou ajustar seus hiperparâmetros) que minimize essa soma.

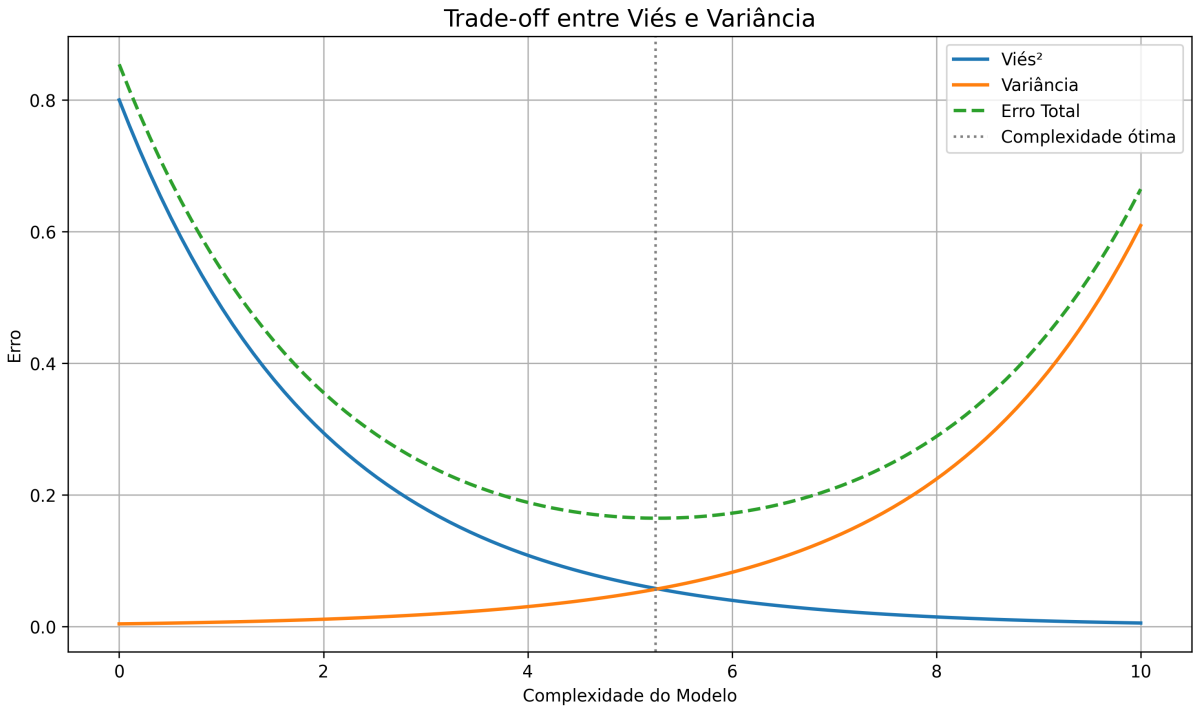


Figura 1.25: Trade-off entre Viés e Variância.

1.11.4 Como Prevenir ou Reduzir Overfitting

A seguir, listamos algumas técnicas comuns para lidar com overfitting:

- **Mais Dados:** Sempre que possível, colete mais exemplos de treino. Se o conjunto de dados for pequeno, o modelo fica mais suscetível a memorizar ruídos;
- **Regularização:** A Regressão Logística (e outros modelos como Redes Neurais) frequentemente usa termos de regularização (L2, L1) para evitar que os parâmetros cresçam demais. Falaremos mais sobre Regularização L2 na próxima seção.
- **Early Stopping:** Em algumas abordagens, especialmente em redes neurais, podemos interromper o treinamento assim que o erro em validação começa a aumentar, mesmo que o erro em treino continue diminuindo.
- **Cross-Validation:** Dividir repetidamente o conjunto de dados em diferentes subconjuntos de treino e validação ajuda a avaliar a estabilidade do modelo. Se o modelo funciona consistentemente bem em várias partições, há menor risco de overfitting.
- **Data Augmentation:** Em problemas de imagens, por exemplo, podemos gerar novas imagens aplicando rotações, inversões, recortes etc., para aumentar a variedade do conjunto de dados. Em problemas de texto, pequenas modificações ou sinônimos podem enriquecer o conjunto de exemplos.
- **Simplicidade do Modelo:** Às vezes, reduzir a complexidade do modelo, com menos camadas ou menos parâmetros ajuda a controlar a variância.

Fique Alerta!

Em resumo, *overfitting* ocorre quando o modelo aprende demais os detalhes, inclusive ruídos dos dados de treino, prejudicando o desempenho em novos dados. **Viés** está associado ao quão próximo (ou quão simples) o modelo é em relação à função real; alto viés implica subajuste (*underfitting*). **Variância** reflete a sensibilidade do modelo a mudanças nos dados de treino; alta variância implica *overfitting*.

Ao longo do processo de modelagem, devemos sempre buscar um equilíbrio entre viés e variância, visando a melhor capacidade de generalização possível. Para isso, recorremos a estratégias como regularização, coleta de mais dados, cross-validation e outras.

1.12 Regularização L2

Iniciando o diálogo...

Técnicas como a regularização L2 ajudam a mitigar o risco de overfitting e melhorar a generalização do modelo (Ng, 2018). Quando treinamos um modelo de classificação, como a Regressão Logística, é comum que desejemos "perceber" o quanto cada variável (também chamada de feature) influencia na previsão. Se o modelo se torna muito complexo, por exemplo, dando pesos extremamente altos a algumas variáveis ele pode acabar "decorando" ruídos ou padrões muito específicos dos dados de treinamento, em vez de aprender os padrões reais. Esse fenômeno é chamado de *overfitting*.

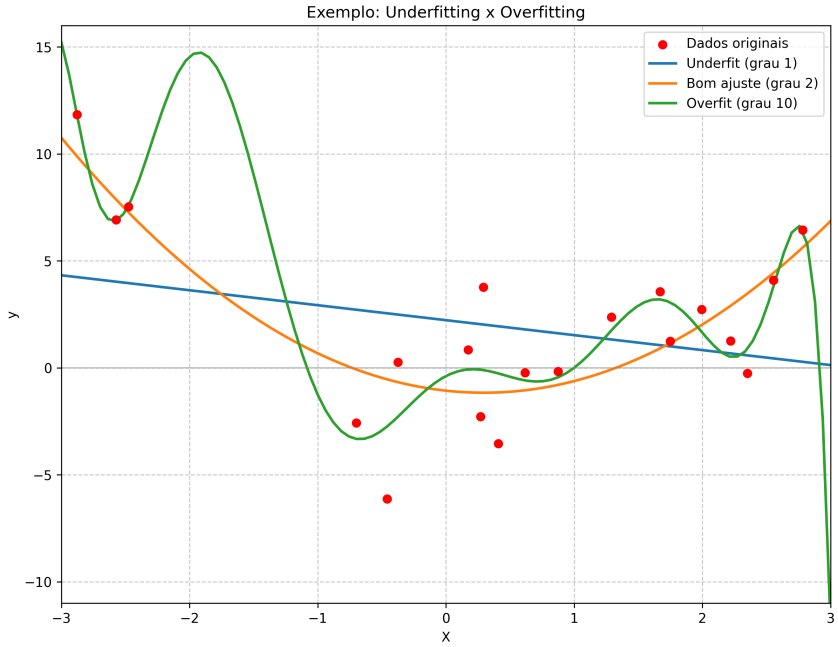


Figura 1.26: Exemplo de Underfitting (modelo muito simples), Bom Ajuste e Overfitting (modelo muito complexo).

A Regularização surge como uma maneira de "segurar a mão" do modelo, impedindo que ele atribua pesos exageradamente grandes. Há diversas formas de regularizar um modelo,

mas aqui o foco será na Regularização L2, também conhecida como *Ridge Regularization*.

Para ilustrar, imagine que você está construindo um modelo para classificar se determinada região amazônica está em processo de desmatamento ou não. O modelo recebe fatores ambientais (chuvas, temperatura, imagens de satélite) e socioeconômicos (índice de desenvolvimento humano, crescimento populacional etc.). Se certos fatores tiverem pesos extremamente altos, o modelo pode estar "superajustado" a um conjunto pequeno de exemplos específicos, e não necessariamente estar aprendendo algo geral sobre desmatamento. Com a Regularização L2, esses pesos são contidos, penalizando valores muito grandes, ajudando o modelo a ser mais robusto e generalizar melhor.

1.12.1 Definição Matemática da Regularização L2

Na Regressão Logística simples (sem regularização), a função de custo (ou *loss function*) pode ser escrita da seguinte forma:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h_\theta(\mathbf{x}^{(i)}))]$$

onde:

- m é o número de exemplos de treinamento
- $y^{(i)}$ é o rótulo real do i-ésimo exemplo (0 ou 1 no caso binário)
- $\mathbf{x}^{(i)}$ é o vetor de features
- $h_\theta(\mathbf{x}^{(i)})$ é a hipótese do modelo: $\sigma(\theta^T\mathbf{x}^{(i)})$, com σ sendo a função sigmoide.

A Regularização L2 adiciona um termo de penalidade ao custo, que empurra os coeficientes θ_j para valores menores. A função de custo com essa penalização passa a ser:

$$J_{reg}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h_\theta(\mathbf{x}^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Note que a soma do termo de regularização não inclui θ_0 (o termo independente ou viés) na maior parte das implementações, pois não desejamos penalizar esse termo.

- λ é o **hiperparâmetro** de regularização, que controla a intensidade da penalização
- Se λ for muito grande, o modelo ficará "excessivamente suave" pode gerar *underfitting*
- Se λ for muito pequeno, o efeito de regularização será insignificante e o modelo pode "voltar" a sofrer *overfitting*

Esse termo adicional $\frac{\lambda}{2m}\sum\theta_j^2$ "puxa" todos os coeficientes θ_j para mais perto de zero, reduzindo a complexidade do modelo e, por consequência, seu potencial de sobreajuste.

1.12.2 Gradiente e Atualização dos Pesos

Sem regularização, o gradiente descendente ajusta θ_j segundo:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

onde α é a taxa de aprendizado (*learning rate*). Com Regularização L2, a derivada da função de custo ganha um termo adicional, resultando em:

$$\theta_j := \theta_j - \alpha \left(\frac{\partial}{\partial \theta_j} J(\theta) + \frac{\lambda}{m} \theta_j \right), \quad \text{para } j \geq 1$$

Observe que o gradiente passa a ter esse acréscimo $\frac{\lambda}{m}\theta_j$. Em outras palavras, a cada passo do gradiente, é como se "empurrássemos" θ_j levemente em direção a zero, de modo a evitar valores extremos.

1.12.3 Por que a Regularização L2 ajuda a reduzir o overfitting?

Modelos que não são regularizados podem "inflar" coeficientes para obter uma classificação perfeita em cenários muito específicos do conjunto de treinamento. Ao penalizar pesos grandes, a regularização "doutrina" o modelo a somente elevar muito um peso se isso for realmente necessário para o aprendizado, caso contrário, ele "prefere" manter os coeficientes mais conservadores, próximos de zero. Na prática, isso faz com que o modelo seja mais "suave": se uma única variável X_k não for realmente tão determinante, seu peso não crescerá descontroladamente. Assim, o modelo pode se generalizar melhor para novos dados.

Podemos usar o conceito amazônico para exemplificar: se existirem diversos sinais ambientais e socioeconômicos que apontam para desmatamento, a regularização faz o modelo levar todos em conta de forma balanceada. Sem a regularização, poderia acontecer de algum indicador pontual, por exemplo, uma variação climática específica de certo ano assumir um peso exagerado, o que não se sustentaria ao longo do tempo.

Caso Prático

Regularização L2 na Previsão de Risco de Desmatamento
Vamos explorar uma aplicação simples de Regressão Logística com Regularização L2 em um problema fictício de classificação binária. Suponha que tenhamos dados ambientais de áreas de preservação e queiramos prever se há alto risco de desmatamento (1) ou baixo risco (0), com base em variáveis como índice de chuva, densidade populacional e quantidade de árvores por hectare.

Copie e Teste!

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score

# Gerando dados simulados
```

```
np.random.seed(42)
n = 50
chuva = np.random.normal(loc=200, scale=60, size=n)
densidade_pop = np.random.normal(loc=80, scale=30, size=n)
arvores_hectare = np.random.normal(loc=1000, scale=200, size=n)

# Definindo o risco com base em regra fictícia
risco = (chuva < 150).astype(int) | (densidade_pop > 110).astype(
   int) | (arvores_hectare < 800).astype(int)

# Montando o DataFrame
df = pd.DataFrame({
    'chuva': chuva,
    'densidade_pop': densidade_pop,
    'arvores_hectare': arvores_hectare,
    'risco_desmatamento': risco
})

# Separando X e y
X = df[['chuva', 'densidade_pop', 'arvores_hectare']]
y = df['risco_desmatamento']

# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.25, random_state=42)

# Modelo com Regularização L2
model = LogisticRegression(penalty='l2', C=1.0)
model.fit(X_train, y_train)

# Avaliação
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Resultados
print("Acurácia:", acc)
print("Precisão:", prec)
print("Recall:", rec)
print("F1 Score:", f1)
print("Coeficientes (Pesos):", model.coef_)
```

* Os valores abaixo podem divergir dependendo de quais valores aleatórios o algoritmo gere para a simulação.

Resultado Esperado

```
Acurácia: 0.9230769230769231
Precisão: 1.0
Recall: 0.8888888888888888
F1 Score: 0.9411764705882353
Coeficientes (Pesos): [[-0.0658  0.0336 -0.0109]]
```

1.12.4 O que fazer a seguir

- **Escolha de λ**: na prática, experimentamos diferentes valores de λ, ou diferentes valores de C, no caso do scikit-learn e avaliamos qual equilibra melhor o erro de treinamento e a capacidade de generalização, erro no conjunto de validação ou teste.
- **Verificação de λ**: se a regularização estiver muito forte, isto é, λ muito grande ou C muito pequeno, o modelo pode não capturar as relações necessárias e ter desempenho ruim até mesmo em treinamento. Nesse caso, a estratégia é reduzir λ.
- **Combinando com outras técnicas**: a regularização pode ser combinada com seleção de variáveis, uso de mais dados de treinamento e técnicas de validação cruzada para encontrar um bom ponto de equilíbrio.

Sempre que lidamos com problemas que envolvam múltiplas variáveis ambientais ou demográficas, a Regularização L2 pode ser uma forma confiável de evitar que ruídos específicos, por exemplo, variáveis muito correlacionadas ou picos fora do comum em certo período façam o modelo "se perder" no processo de aprendizagem. A Regularização L2 é uma ferramenta fundamental para a construção de modelos de classificação mais robustos, sobretudo ao lidar com cenários em que há muitas variáveis potenciais e o risco de overfitting é alto. Seu funcionamento matemático baseia-se em adicionar ao custo um termo que penaliza o crescimento excessivo dos coeficientes, trazendo-os em direção a zero.

Fique Alerta!

Regularização L2 não é só "uma fórmula", mas uma filosofia de modelagem que busca aprender padrões de modo controlado, contribuindo para um modelo mais confiável e útil na classificação de problemas reais.

Capítulo 2

Aprendizagem Não-Supervisionada e Sistemas de Recomendação

Iniciando o diálogo...

Você já se perguntou como as plataformas de streaming sugerem filmes ou séries sem que você diga exatamente do que gosta? Ou então, como aplicativos de compras online conseguem recomendar produtos que parecem ter tudo a ver com você, mesmo sem ter tantas informações sobre seu perfil? A resposta está em um ramo da Inteligência Artificial chamado Aprendizado Não Supervisionado, no qual os computadores aprendem padrões e relações a partir dos dados, tudo isso sem receber, de cara, o "rótulo" correto de cada exemplo.

2.1 O que é Aprendizado Não Supervisionado?

De acordo com Mitchell (1997), o aprendizado não supervisionado busca identificar estruturas ocultas em dados sem o uso de rótulos. Para entendermos melhor o não supervisionado, vale relembrar rapidamente o que chamamos de aprendizado supervisionado. Nele, sabemos as respostas corretas (rótulos) para cada exemplo de treino, por exemplo, saber se uma imagem é de um gato ou de um cachorro e usamos esses rótulos para ensinar o computador a classificar ou fazer previsões.

No aprendizado não supervisionado, essa etapa de "ensinar com respostas prontas" não existe. Ou seja, temos apenas os dados, sem classificações ou explicações fornecidas de antemão, e o computador tenta encontrar padrões ou estruturas por conta própria. Esses padrões podem surgir de muitas formas: grupos de pessoas com comportamentos parecidos, textos que tratam de assuntos semelhantes ou até produtos que geralmente são comprados juntos.

Uma analogia simples para o aprendizado não supervisionado é imaginar que você tem uma caixa com peças de LEGO de vários tamanhos e cores, sem qualquer identificação. A tarefa é separar essas peças em grupos parecidos. Você não tem uma etiqueta "vermelho" ou "amarelo" nem mesmo um "conjunto oficial": você descobre, por conta própria, se vai separar por cor, formato ou algum outro critério que faça sentido.

2.1.1 Sistemas de Recomendação

Os Sistemas de Recomendação são uma aplicação muito conhecida do aprendizado não supervisionado (embora possam envolver técnicas variadas). Plataformas como Netflix, YouTube, Amazon e Spotify geram recomendações personalizadas ao agrupar pessoas e conteúdos com comportamentos semelhantes, mesmo que não saibam "exatamente" o que cada pessoa é ou gosta.

- **Netflix e YouTube:** Observam quais filmes/séries/vídeos cada pessoa assiste e criam padrões de preferências. Se você assiste a vários vídeos de culinária, por exemplo, o sistema encontra outros usuários que também veem vídeos de culinária e verificam o que mais eles assistem. A partir daí, surgem recomendações que podem despertar seu interesse.
- **E-commerce como Amazon, Mercado Livre etc.:** Ao analisar o histórico de compras e visualizações, o sistema agrupa clientes com hábitos parecidos. Assim, quando alguém que lembra o seu perfil comprou um determinado produto, você também recebe a indicação.
- **Streaming de Música como Spotify e Deezer:** Comportamentos como "pular uma faixa" ou "ouvir até o final" ajudam o algoritmo a entender quais estilos de música você mais gosta, sugerindo playlists bem específicas.

Fique Alerta!

Embora esses sistemas sejam muito úteis, eles se baseiam na disponibilidade e na qualidade dos dados. Se, por algum motivo, o registro de preferências ou hábitos estiver incompleto, distorcido ou tendencioso, as recomendações podem falhar e até mesmo criar "bolhas" de informação.

2.1.2 Agrupamento (Clustering)

Grande parte dos sistemas de recomendação utiliza a ideia de agrupamento ou *clustering*. O algoritmo recebe dados como:

- Histórico de compras;
- Tipos de produtos visualizados;
- Filmes assistidos;
- Estilo musical preferido.

Em seguida, ele tenta encontrar grupos de usuários com padrões semelhantes. Isso permite associar suas preferências às de outras pessoas que gostam de coisas parecidas, o que acaba gerando sugestões personalizadas.

2.1.3 Relação com o Cotidiano

Além do entretenimento, o aprendizado não supervisionado aparece em muitas outras áreas:

- **Agricultura Inteligente:** Ao monitorar dados de temperatura, umidade e solo, é possível agrupar regiões parecidas e identificar que tipo de cuidado cada área precisa para uma produção mais eficiente.
- **Saúde:** Dados de exames podem ser analisados para encontrar grupos de pacientes com sintomas ou características semelhantes, auxiliando na descoberta de possíveis doenças e tratamentos.
- **Ecologia e Meio Ambiente:** Sensores espalhados em florestas podem coletar informações sobre clima, espécies, índices de poluição, entre outros. A análise não supervisionada agrupa áreas em risco ou com potencial de conservação.
- **Educação:** Plataformas de ensino online agrupam alunos com dificuldades semelhantes, recomendando materiais de estudo específicos ou organizando tutoriais que realmente os ajudem.

Vantagens

- **Encontrar o inesperado:** Sem rótulos pré-definidos, o algoritmo pode revelar padrões que ninguém imaginava existir, trazendo novas descobertas e caminhos de pesquisa.
- **Escalabilidade:** Muitos algoritmos de aprendizado não supervisionado funcionam bem com grandes volumes de dados, o que é fundamental na era digital, onde tudo gera informação.
- **Automatização:** Uma vez configurado, o sistema consegue organizar dados e produzir recomendações continuamente, sem intervenção humana a todo instante.

Desafios

- **Qualidade dos Dados:** Se os dados forem escassos ou não representativos, os padrões podem sair totalmente equivocados. No caso de recomendações, você pode acabar recebendo sugestões irrelevantes.
- **Interpretação:** Identificar que "há grupos de clientes" não basta; é preciso interpretar porque eles se agrupam assim. Em aplicações sérias, como saúde, entender o que cada grupo representa é crucial.
- **Questões Éticas:** Como não há supervisão, o algoritmo pode agrupar pessoas de forma inadequada, reforçar estereótipos ou até mesmo criar discriminações. Em sistemas de recomendação, isso pode amplificar a famosa "bolha de filtros", em que se recebe apenas um tipo de conteúdo.

2.1.4 Por que aprender isso?

Você pode se perguntar: "Ok, mas por que eu preciso entender esse assunto?" Eis alguns pontos:

- **Futuro Profissional:** O domínio de técnicas de inteligência artificial, mesmo que de modo introdutório, é um grande diferencial. O mercado de tecnologia demanda cada vez mais profissionais que entendam de dados.

- **Tomada de Decisão:** Compreender como máquinas identificam padrões ajuda a desenvolver o pensamento crítico. Você passa a questionar as sugestões que recebe: "Será que isso faz sentido para mim?", "De onde vieram esses dados?".
- **Criatividade e Inovação:** Quem sabe você não cria um aplicativo para ajudar seus colegas a escolherem livros ou podcasts? Ou mesmo uma maneira inteligente de reciclar na escola, agrupando materiais por tipo e reaproveitando-os de forma inovadora?
- **Consciência Digital:** Vivemos em uma era em que cada clique gera informação. Saber como essas informações são processadas e recomendadas amplia nossa visão sobre privacidade e segurança de dados.

Conhecendo um pouco mais!

Quer ver de perto como funciona um sistema de recomendação simples? Pesquise por tutoriais de "recomendação de filmes em Python". Há exemplos básicos que podem ser executados até em um navegador, usando plataformas online gratuitas. Para entender melhor a ideia de Agrupamento (*Clustering*), experimente separar objetos de acordo com cor, tamanho ou formato, sem usar nenhum rótulo. Perceba como surgem grupos naturalmente, e como diferentes critérios levam a agrupamentos diferentes.

Prepare-se para um mundo onde as máquinas descobrem sozinhas e nos ajudam a tomar decisões mais inteligentes, seja na hora de escolher o próximo filme para assistir ou de planejar estratégias de cultivo sustentável em grandes fazendas. Boa leitura e bons experimentos!

2.2 Fundamentos de aprendizagem não-supervisionada

2.2.1 Definição e aplicações

A aprendizagem não-supervisionada é uma área da ciência de dados que se dedica a encontrar estruturas ou padrões em conjuntos de dados que não possuem rótulos ou categorias previamente definidas. Em outras palavras, em vez de receber exemplos de "certo" ou "errado" (ou de uma classe específica) para orientar sua análise, o algoritmo trabalha por conta própria para identificar relacionamentos, sem instruções diretas sobre quais respostas ou grupos procurar. Essa abordagem é especialmente útil quando não se tem certeza de como os dados estão organizados ou quando é inviável ou muito caro rotular cada exemplo manualmente.

A seguir, discutiremos de forma simples o conceito de aprendizagem não-supervisionada e algumas de suas aplicações em diferentes contextos. A ideia é oferecer uma visão geral para que você, enquanto estudante de ensino médio, possa compreender o que está por trás dessa metodologia, por que ela é importante e como pode ser aplicada em situações reais.

O que é Aprendizagem Não-Supervisionada?

Para entender melhor, vale a pena fazer uma comparação com a aprendizagem supervisionada. Na abordagem supervisionada, temos dados de entrada (por exemplo, imagens ou descrições de objetos) e o rótulo correspondente (como "gato" ou "cachorro"). Dessa forma, o algoritmo aprende a associação entre os dados de entrada e o rótulo. Já na aprendizagem não-supervisionada, não há rótulos para guiar a busca pelo padrão. O algoritmo recebe apenas o conjunto de dados e a tarefa de analisá-los, procurando maneiras de agrupá-los ou resumir suas características.

Podemos imaginar a aprendizagem não-supervisionada como alguém entrando em um cômodo cheio de caixas e sendo desafiado a organizar tudo sem conhecer o que cada caixa contém. A pessoa vai abrindo as caixas, observando o que há dentro e, a partir dessas observações, agrupa as caixas por similaridade, criando "categorias" próprias. A grande questão é que essas categorias não são definidas de antemão: elas surgem da própria análise dos dados.

Principais Tarefas na Aprendizagem Não-Supervisionada

Embora existam diversas técnicas e objetivos dentro da aprendizagem não-supervisionada, duas tarefas se destacam como as mais conhecidas: agrupamento (*clustering*) e redução de dimensionalidade.

Agrupamento (Clustering) O objetivo é encontrar grupos de elementos que sejam mais similares entre si do que em relação a elementos de outros grupos. Por exemplo, se você tem uma lista de músicas com várias características (ritmo, instrumentos, duração etc.), um algoritmo de agrupamento pode descobrir automaticamente gêneros ou subgêneros baseados nas semelhanças desses atributos, mesmo que as faixas não tenham classificações prévias.

Esse tipo de análise é bastante utilizado em ferramentas de recomendação. Muitas plataformas de filmes e músicas utilizam algoritmos de agrupamento para sugerir conteúdo semelhante, que outros usuários com o mesmo perfil também gostaram.

Redução de Dimensionalidade Em muitos casos, é comum termos um conjunto de dados com centenas ou até milhares de variáveis (também chamadas de "dimensões"). A redução de dimensionalidade visa simplificar esse conjunto de variáveis em um número menor de componentes que ainda representem bem as informações originais.

Essa simplificação ajuda, por exemplo, a visualizar dados complexos em gráficos de duas ou três dimensões, facilitando o entendimento de como eles se relacionam ou se agrupam. Também pode ser útil para preparar os dados para outras tarefas, reduzindo o ruído e otimizando o desempenho de algoritmos mais complexos.

Aplicações Práticas e Exemplos

A aprendizagem não-supervisionada é muito utilizada em diferentes campos e projetos de ciência de dados. Alguns exemplos:

- **Agrupamento de Clientes:** Em mercados e lojas virtuais, profissionais de marketing utilizam esses algoritmos para identificar grupos de clientes com hábitos de consumo parecidos. Isso ajuda a criar campanhas personalizadas, ofertas especiais e até prever tendências de compra.
- **Detecção de Anomalias:** Sistemas de segurança virtual ou detecção de fraudes em cartões de crédito podem monitorar transações e, por meio do agrupamento, perceber quais comportamentos são "normais" e quais são anômalos. Assim, é possível identificar movimentos suspeitos de forma mais ágil, mesmo sem ter exemplos prévios do que seria uma "fraude".
- **Recomendação de Conteúdo:** Em aplicativos de streaming de vídeos ou músicas, os algoritmos analisam o histórico de consumo dos usuários e encontram padrões de preferências musicais ou cinematográficas, sugerindo novos conteúdos que combinem com os gostos identificados.

- **Compressão de Dados e Imagens:** Usando redução de dimensionalidade, é possível comprimir informações, como imagens de alta resolução, em formatos menores, preservando o essencial para reconstruí-las ou analisá-las. Isso diminui a quantidade de espaço de armazenamento necessário e facilita o envio desses arquivos pela internet.
- **Análise de Texto:** Técnicas de agrupamento podem ser aplicadas a coleções de textos (como artigos, postagens em redes sociais ou pesquisas acadêmicas), extraindo tópicos principais e classificando documentos de maneira automática, a partir das palavras que aparecem com maior frequência.

A aprendizagem não-supervisionada é fundamental porque muitas vezes não temos rótulos disponíveis ou não sabemos ao certo como agrupar os dados. Na prática, isso significa que podemos descobrir insights novos e inusitados a partir de conjuntos de dados complexos e diversificados. Com a popularização do acesso a grandes volumes de informações, como registros de sites, plataformas de redes sociais, dados de sensores em cidades inteligentes, entre outros, a capacidade de "aprender sem ser instruído" torna-se cada vez mais valiosa.

É importante enfatizar que a aprendizagem não-supervisionada não substitui outras formas de análise de dados, mas sim as complementam. Muitas vezes, podemos aplicar técnicas não-supervisionadas antes mesmo de usar algoritmos de aprendizagem supervisionada, para entender melhor a estrutura dos dados. Também é possível combinar as duas abordagens em projetos mais complexos, enriquecendo o processo de descoberta de conhecimento.

A partir daqui, convidamos você a refletir: Quais são as áreas do seu cotidiano em que a análise não-supervisionada poderia ser aplicada? Pense em aplicativos, sites ou pesquisas escolares em que faz sentido agrupar informações sem ter um rótulo prévio.

2.3 Agrupamento K-means

2.3.1 Função de custo

Conforme Géron (2023), técnicas de agrupamento como K-means permitem particionar dados em grupos baseados em similaridades, ou seja, estamos lidando com uma técnica de agrupamento cujo objetivo principal é agrupar dados em grupos. Para descobrir os melhores agrupamentos, utilizamos uma medida que indica o quão "bem" os dados estão organizados em torno de cada centroide. Essa medida é conhecida como Função Custo (*cost function*).

De modo geral, a Função Custo de K-means mede o quanto cada ponto do conjunto de dados está distante do centroide de seu grupo. Se os pontos estiverem muito distantes dos respectivos centroides, significa que a partição não está boa. Se, ao contrário, os pontos estiverem próximos de seus centroides, quer dizer que o agrupamento é melhor.

Custo neste contexto, é um valor numérico que indica o "erro" ou a "qualidade" do agrupamento. Quanto menor for esse valor, melhor está o agrupamento, ou seja, os dados estão mais bem "encaixados" nos seus grupos. Nesse método, o custo é calculado com base na soma dos quadrados das distâncias de cada ponto ao seu centroide.

Para tornar as ideias mais claras, vamos usar alguns símbolos matemáticos simples. Suponha que temos um conjunto de dados $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, onde cada $\mathbf{x}_i$ é um ponto (ou um vetor de características) em um espaço $\mathbb{R}^d$.

Queremos dividir esses N pontos em K grupos. Em K-means, cada grupo tem um centroide μ_j, para $j = 1, 2, \ldots, K$. Cada ponto $\mathbf{x}_i$ pertence a exatamente um desses grupos. A Função Custo de K-means, muitas vezes chamada de soma dos erros quadráticos (*Sum of Squared Errors, SSE*), é dada por:

$$J = \sum_{i=1}^{N} ||\mathbf{x}_i - \mu_{c(i)}||^2$$

onde $c(i)$ indica o índice do grupo ao qual o ponto $\mathbf{x}_i$ pertence, ou seja, qual centroide está associado a $\mathbf{x}_i$. A notação $|| \cdot ||^2$ representa a distância euclidiana ao quadrado, ou seja,

$$||\mathbf{x} - \mathbf{y}||^2 = \sum_{k=1}^{d} (x_k - y_k)^2$$

.

Em resumo, a expressão acima soma a distância quadrática de cada ponto até o centroide de seu grupo. A ideia do K-means é encontrar a posição dos centroides μ_j e a atribuição de cada $\mathbf{x}_i$ de modo que esse valor de J seja o menor possível.

Por que Usar Distâncias ao Quadrado?

- **Penalização maior para pontos distantes:** Elevar ao quadrado a distância faz com que pontos mais distantes tenham um peso maior no custo total. Imagine um ponto que esteja muito longe do centroide: ao somar o quadrado de sua distância, esse ponto "puxa" a função custo para cima, incentivando o algoritmo a ajustar o centroide para acomodar pontos fora do centro.
- **Facilidade de cálculo:** Somar distâncias quadráticas facilita derivar fórmulas para atualizar os centroides, como veremos futuramente. Trabalhar com quadrados é matematicamente conveniente, pois torna possível obter expressões exatas para as coordenadas ideais dos centroides.
- **Interpretação geométrica:** Visualmente, se pensarmos em um plano 2D, cada centroide fica no "meio" dos pontos do seu grupo. Minimizar a soma dos quadrados dessas distâncias faz com que cada centroide seja, em essência, a média desses pontos, daí o nome *K-means*, traduzido "K médias".

Na intenção de facilitar o entendimento, é útil relacionar isso à ideia de média aritmética. Se tivermos um conjunto de valores $a_1, a_2, \ldots, a_n$, a média $\bar{a}$ minimiza a soma dos quadrados $\sum(a_i - \bar{a})^2$. No K-means, estamos expandindo essa noção para múltiplas dimensões.

Passo a Passo de Como a Função Custo Entra no Método K-means

Para entender melhor, vamos lembrar do fluxo do K-means:

1. **Inicializar** $\mu_1, \mu_2, \ldots, \mu_K$. Isso pode ser feito de várias formas, por exemplo, escolhendo aleatoriamente K pontos do conjunto de dados.
2. **Atribuir pontos aos centroides:** cada ponto $\mathbf{x}_i$ é associado ao centroide mais próximo, medido pela distância euclidiana.
3. **Recalcular os centroides:** para cada grupo, calcula-se a média dos pontos que foram atribuídos a ele. Essa média se torna o novo centroide.
4. **Calcular a Função Custo:** após cada iteração, atribuir pontos => recalcular centroides, calculamos o valor de J. A cada nova iteração, esperamos que J diminua.

5. **Repetir** até que as mudanças sejam pequenas ou até atingir um número pré-determinado de iterações. Se J parar de diminuir de maneira significativa, dizemos que o algoritmo convergiu.

Fique Alerta!

A Função Custo age como um "termômetro" que avalia se o agrupamento está melhorando, diminuindo o valor de J ou não.

Exemplificando com Um Conjunto de Pontos em 2D

Para tornar isso mais concreto, imagine que temos dez pontos em um plano 2D:

$$\{(1,2),(2,2),(2,3),(3,3),(6,6),(7,7),(8,6),(7,8),(10,1),(12,2)\}$$

Suponha que queiramos dividi-los em $K = 2$ grupos. O que o K-means faz?

- **Inicialização:** Escolha inicial de 2 centroides, digamos $\mu_1 = (1,2)$ e $\mu_2 = (10,1)$, este é um exemplo aleatório.
- **Atribuição:** Cada ponto é associado ao centroide mais próximo. Por exemplo, pontos $(1,2)$, $(2,2)$, $(2,3)$, $(3,3)$ provavelmente ficarão com μ_1. Já $(10,1)$, $(12,2)$ ficarão com μ_2. Os pontos $(6,6)$, $(7,7)$, $(8,6)$, $(7,8)$ dependerão da distância a cada centroide.
- **Recalcular os centroides:** Depois da atribuição, calcula-se a média das coordenadas de cada grupo para obter os novos centroides.
- **Calcular o custo:** $J = \sum_{i=1}^{10} ||\mathbf{x}_i - \mu_{c(i)}||^2$. Se esse valor ainda estiver "alto", sinal de que há espaço para melhorar.
- **Repetir:** Depois de algumas iterações, teremos um agrupamento que tende a "estabilizar". Nesse ponto, dizemos que o algoritmo convergiu.

Em cada etapa, o algoritmo tenta diminuir a soma dos quadrados das distâncias de todos os pontos aos seus centroides. Esse é o "coração" da Função Custo.

Por que a Minimização da Função Custo é Importante?

A qualidade do agrupamento está relacionada ao custo, quanto menor ele for, mais próximos estarão os pontos dentro de cada grupo, tornando a "identidade" de cada agrupamento mais clara. A estabilidade da partição também é importante, pois um custo muito alto pode indicar a necessidade de escolher outro valor de K ou de verificar a presença de ruídos excessivos nos dados. Em contextos educativos, é útil destacar que os centroides representam a "média" de cada grupo, e que minimizar o custo garante que essas médias sejam de fato representativas.

A Função Custo quantifica o quanto o K-means está "performando bem" ou não. Todo o processo de atribuir pontos a centroides e recalcular as posições deles gira em torno de tentar deixar esse valor o menor possível.

Resumo

A Função Custo em K-means, baseada na soma dos quadrados das distâncias dos pontos aos seus centroides, desempenha um papel central para que o algoritmo encontre bons agrupamentos. Seu caráter matemático do uso de distâncias ao quadrado, e sua interpretação geométrica, centroides como médias, tornam o método tanto intuitivo como eficiente no processamento de grandes conjuntos de dados.

2.3.2 Atualização dos centroides

Nesta seção, vamos explorar o segundo passo fundamental do método k-means: a atualização dos centroides. Esse momento é crucial para o algoritmo, pois, após atribuirmos cada ponto de dados a um determinado grupo (cluster), precisamos ajustar a posição de cada centroide de modo que ele represente da forma mais fiel possível todos os pontos que fazem parte daquele grupo. A ideia central aqui é relativamente simples: o novo centroide de um grupo é obtido calculando a média aritmética das posições de todos os pontos que foram atribuídos a esse grupo.

Relembrando o contexto do método k-means

O método k-means se baseia em dois passos que se repetem várias vezes:

1. **Atribuição:** para cada ponto do conjunto de dados, determinamos a qual centroide ele está mais próximo. Esse ponto, então, "entra" no grupo correspondente a esse centroide.

2. **Atualização:** para cada grupo formado, calculamos o novo centroide, reposicionando-o na média dos pontos que pertencem àquele grupo.

Esses dois passos se repetem até que a posição dos centroides praticamente não se altere mais ou até atingirmos um número predeterminado de iterações. Desta forma, o algoritmo busca organizar os dados em k grupos diferentes, de modo que cada grupo fique relativamente coeso em torno de seu centroide.

Conceito de centroide e intuição geométrica

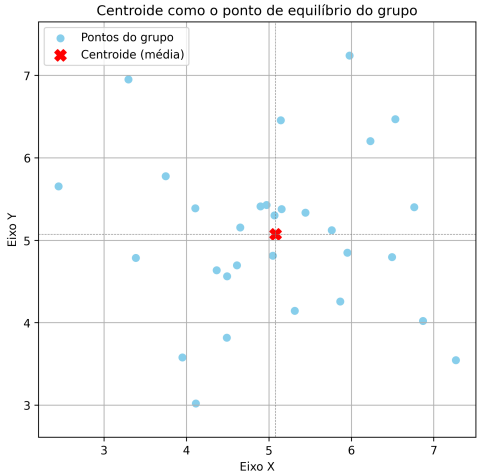


Figura 2.1: Centroide como o ponto de equilíbrio do grupo.

O centroide é, essencialmente, o ponto que melhor representa o "centro" de todos os dados de um grupo. De modo mais intuitivo, se tivermos um conjunto de pontos que caem próximos uns dos outros em um plano, imagine um gráfico de dispersão com várias bolinhas espalhadas, o centroide é aquele ponto em que você "equilibraria" todos eles se fossem de peso igual.

Pense em um grupo de amigos que quer se encontrar em um local central para todos. Se cada pessoa mora em um ponto do mapa, o centroide seria o ponto, o endereço que minimizaria a soma das distâncias de todos. No k-means, a forma mais comum de calcular esse ponto central é fazendo a média das coordenadas de cada pessoa, ou cada ponto de dados.

A fórmula de atualização do centroide

Seja C_j o conjunto de pontos que pertencem ao grupo j, ou seja, todos os pontos que no passo anterior foram mais próximos do centroide μ_j do que dos outros centroides). Então, o novo centroide $\mu_j^{(\text{novo})}$ é dado por:

$$\mu_j^{(\text{novo})} = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$$

Em termos de cada coordenada, isso quer dizer que pegamos todas as posições (coordenadas) dos pontos em C_j, somamos cada uma delas e dividimos pelo número de pontos no grupo. Por exemplo, se estamos trabalhando em duas dimensões coordenadas x e y, e C_j contém n pontos:

$$\mu_j^{(\text{novo})} = \left(\frac{x_1 + x_2 + \cdots + x_n}{n}, \frac{y_1 + y_2 + \cdots + y_n}{n} \right)$$

Em três dimensões, a ideia é a mesma, apenas adicionamos a coordenada z, e assim por diante para mais dimensões.

Por que a média aritmética?

No k-means, queremos minimizar, dentro de cada grupo, as distâncias quadráticas entre cada ponto e seu centroide. A média aritmética é o valor que melhor representa a posição que minimiza essas distâncias quadráticas. Intuitivamente, se a nossa "função custo" mede o quanto cada ponto se afasta do centroide em termos de distância euclidiana ao quadrado, a média é a "melhor resposta" que centraliza os dados nesse sentido.

Para tornar a ideia mais clara, vamos pensar em um exemplo hipotético com um conjunto de pontos em duas dimensões (x, y). Suponha que, depois do passo de atribuição, descobrimos dois grupos:

- **Grupo A** (centroide atual: μ_A) Pontos: $(1, 2), (2, 1), (1, 3)$
- **Grupo B** (centroide atual: μ_B) Pontos: $(5, 4), (6, 5), (5, 5)$

Passo de Atualização para o Grupo A:

- Soma das coordenadas x do Grupo A: $1 + 2 + 1 = 4$
- Soma das coordenadas y do Grupo A: $2 + 1 + 3 = 6$
- Número de pontos no Grupo A: 3
- Novo centroide do Grupo A: $\mu_A^{(\text{novo})} = \left(\frac{4}{3}, \frac{6}{3}\right) = (\frac{4}{3}, 2) \approx (1.33, 2)$

Passo de Atualização para o Grupo B

- Soma das coordenadas x do Grupo B: $5 + 6 + 5 = 16$
- Soma das coordenadas y do Grupo B: $4 + 5 + 5 = 14$
- Número de pontos no Grupo B: 3
- Novo centroide do Grupo B: $\mu_B^{(\text{novo})} = \left(\frac{16}{3}, \frac{14}{3}\right) \approx (5.33, 4.67)$

Perceba que, normalmente, cada novo centroide se desloca um pouco a partir de sua posição anterior para refletir melhor o grupo de pontos que está representando. Esse processo se repete: os pontos são novamente atribuídos aos centroides mais próximos (passo de atribuição), e então recalculamos as médias (passo de atualização). A cada iteração, os centroides vão "caminhando" para posições mais adequadas, até se estabilizarem.

Pode acontecer de um grupo ficar sem pontos atribuídos durante o processo. Nesse caso, a forma mais simples de contornar é recolocar o centroide em alguma posição que faça sentido no contexto, por exemplo, movê-lo para um ponto aleatório ou escolher um ponto mais afastado. Mas essa situação é relativamente incomum quando escolhemos um valor de k adequado e temos um conjunto de dados suficientemente variado.

Em muitos materiais didáticos, vemos a explicação de forma sequencial, onde primeiro atribuímos todos os pontos aos centroides, depois recalculamos todos os centroides de uma vez. Na prática, existem variações onde cada ponto ou cada centroide pode ser atualizado "em tempo real". Entretanto, a ideia fundamental de usar a média como nova posição continua a mesma.

Se você estiver trabalhando com dados em dez, cem ou até mil dimensões, o que é comum em aplicações de Ciência de Dados, a lógica de calcular o centroide como média não muda. Mesmo com muitas dimensões, calculamos a média de cada coordenada separadamente. Embora seja difícil visualizar grupos além de 3 dimensões, a interpretação do centroide como "ponto de equilíbrio" continua válida. Se projetarmos os dados em eixos principais ou reduzirmos a dimensionalidade, ainda vemos que cada centroide tende a ficar no meio do conjunto de pontos que representa.

Resumo

O passo de atualização dos centroides é fundamental para que o método k-means possa agrupar os dados de maneira que cada conjunto fique bem "centralizado". Ao usar a média aritmética como nova posição, garantimos que cada centroide se aproxime da distribuição real dos pontos atribuídos àquele grupo, minimizando a distância dentro do próprio cluster.

Quando voltarmos a falar de todo o processo de k-means, lembre-se de que este passo se alterna com o passo de atribuição. Juntos, eles criam um ciclo de refinamento em que os centroides e a composição dos grupos vão sendo ajustados até atingirmos uma configuração estável ou até alcançarmos o número de iterações que definimos inicialmente.

Em suma, a atualização dos centroides é a essência matemática do k-means que mantém o algoritmo constantemente "buscando" o melhor modo de representar grupos de dados. A beleza desse método está justamente no fato de ser uma ideia simples de média, mas poderosa o suficiente para dar conta de problemas de agrupamento em diferentes áreas, desde imagens até processamento de textos e dados sensoriais.

Fique Alerta!

Tente comparar o k-means com outros métodos de agrupamento, percebendo como diferentes estratégias podem levar a configurações diferentes de grupos. Também vale explorar variações do k-means que levam em conta pesos ou que usam outras métricas de distância em vez da euclidiana.

2.3.3 Escolha do parâmetro k

Uma das decisões mais importantes ao aplicar o método k-means é definir quantos grupos (k) queremos formar. Porém, como saber se devemos criar dois, três, quatro ou vários grupos diferentes? Nessa seção, vamos explorar estratégias que ajudam a escolher o valor de k de modo a capturar, ao mesmo tempo, a estrutura natural dos dados e a simplicidade do modelo.

Por que precisamos escolher k?

O método k-means agrupa os dados em k subconjuntos diferentes, chamados de clusters. Cada cluster fica associado a um centro (o centroide) que, idealmente, deve representar bem os pontos ali agrupados. Se escolhermos um valor muito pequeno de k, podemos acabar "forçando" muitos dados distintos a ficarem em poucos grupos, perdendo detalhes importantes das semelhanças e diferenças entre eles. Por outro lado, se escolhermos um k muito grande, podemos ter um cenário de "superdivisão", ou seja, criamos tantos grupos que eles deixam de ter sentido prático ou de apresentar padrões gerais.

Para entender isso com uma analogia: imagine que estamos organizando livros em prateleiras. Se criarmos poucas categorias, por exemplo, apenas "Ficção" e "Não-ficção"), colocaremos juntos livros muito diferentes e não conseguiremos distinguir mais a fundo os gêneros. Se criarmos muitas categorias, por exemplo, "Terror de autor brasileiro dos anos 2000", "Terror de autor brasileiro dos anos 2010", "Terror de autor estrangeiro dos anos 2000" etc., poderemos ficar sobrecarregados de tantas divisões e perder a noção do conjunto.

O grande desafio é qual seria o número de prateleiras, ou categorias, ideal para dividir nossos livros? No caso do método k-means, qual o número k que melhor equilibra a simplicidade do modelo e a boa representação dos dados?

Critério de Avaliação: Função de Custo (Distância Interna)

Para escolher k, precisamos de algum critério que nos diga como está a qualidade do agrupamento. Geralmente, usamos a função de custo que o próprio k-means tenta minimizar:

$$J(C, \mu) = \sum_{i=1}^{m} ||\mathbf{x}_i - \mu_{c(i)}||^2$$

onde:

- $\mathbf{x}_i$ é o i-ésimo ponto nos dados.
- $c(i)$ indica a qual centroide (ou cluster) o ponto $\mathbf{x}_i$ pertence.
- $\mu_{c(i)}$ é o centroide do cluster associado ao ponto $\mathbf{x}_i$.
- $||\cdot||$ representa a distância euclidiana (normalmente).

Em resumo, essa soma mede o quão "espalhados" ficam os pontos em cada cluster. Agrupamentos bons tendem a ter uma soma de distâncias menor, pois cada ponto está relativamente próximo do seu centroide. No entanto, só olhar para o valor dessa soma não resolve o dilema, pois quanto mais aumentamos k, menor essa soma tende a ficar (com muitos clusters, cada grupo fica bem "apertado" em torno do centroide). Então precisamos de métodos adicionais para fazer uma escolha mais equilibrada.

Método do Cotovelo - Elbow Method

Uma das abordagens mais populares e intuitivas para escolher k é o método do cotovelo (*Elbow Method*). Eis como funciona:

1. **Variar** k: rodamos o k-means para valores de k diferentes (por exemplo, de 1 a 10).
2. **Calcular a soma dos quadrados das distâncias:** obtemos o valor de $J(C, \mu)$ para cada k.
3. **Plotar o resultado:** criamos um gráfico em que o eixo horizontal representa k e o eixo vertical representa o valor da função de custo J.

O comportamento esperado é que, no início quando k é pequeno, a diminuição de J seja bem grande quando aumentamos k (porque estamos passando de um grupo para dois, depois para três etc.). Porém, a partir de certo ponto, essa queda começa a ficar cada vez menor. Imagine o formato de um braço dobrado no cotovelo: o "cotovelo" do gráfico marca aproximadamente um ponto de inflexão. Esse "cotovelo" (ou joelho) costuma indicar um bom valor de k, pois dali em diante o ganho na redução do custo não compensa o aumento de complexidade de adicionar mais clusters.

Na prática, esse "cotovelo" nem sempre é muito nítido; às vezes é sutil. Mesmo assim, o método costuma dar uma boa pista sobre valores de k razoáveis para explorar.

Fique Alerta!

Em muitos cenários de aplicações reais, costumávamos testar alguns valores de k em torno do ponto de inflexão, realizando análises qualitativas e/ou quantitativas para decidir qual deles é mais útil.

Método da Silhueta - Silhouette Method

Outra forma de avaliar a qualidade do agrupamento e ajudar a definir k é a Silhueta (*Silhouette*). A ideia, de maneira simplificada, é: Para cada ponto $\mathbf{x}_i$, calcular:

- **Coesão:** a média das distâncias de $\mathbf{x}_i$ a todos os outros pontos do cluster no qual $\mathbf{x}_i$ está.
- **Separação:** a média das distâncias de $\mathbf{x}_i$ aos pontos do cluster mais próximo, ou seja, o cluster diferente em que ele teria a menor distância média.

A Silhueta s_i de cada ponto $\mathbf{x}_i$ é dada por:

$$s_i = \frac{\text{separação} - \text{coesão}}{\max(\text{separação}, \text{coesão})}$$

Esse valor varia de -1 a +1.

- Valores próximos de +1 indicam que o ponto está bem "encaixado" em seu cluster e distante de outros clusters.
- Valores próximos de 0 indicam que o ponto está próximo da fronteira entre dois clusters.
- Valores negativos indicam que o ponto pode estar "errado" e pertenceria melhor a outro cluster.

Para avaliar todo o conjunto, podemos calcular a média de s_i entre todos os pontos. Essa média nos dá um indicador global de qualidade do agrupamento. Assim como no método do cotovelo, podemos variar k (por exemplo, de 2 até 10) e medir a média das silhuetas. Geralmente, escolhemos o k que maximiza essa média. Se k for muito pequeno, vários pontos tendem a ficar próximos de fronteiras de clusters, e se for muito grande, os clusters podem ficar muito separados ou mal configurados. A silhueta ajuda a encontrar um meio-termo.

Outras Estratégias

Além do *Elbow Method* e da *Silhouette*, existem outras técnicas mais avançadas para escolher k. Duas delas são:

- **Critério da Informação (IC, AIC, BIC)** que são métodos estatísticos que penalizam a complexidade do modelo. Sempre que se aumenta k, a complexidade do modelo cresce; então esses critérios buscam um equilíbrio entre a qualidade do ajuste e a quantidade de parâmetros usados.
- **Método do Salto (Gap Statistic):** Compara a variação interna dos dados (distâncias nos clusters) com o que se esperaria de dados "aleatórios". Assim, podemos identificar se o k escolhido realmente representa um agrupamento melhor do que um cenário "sem padrão".

Intuição Prática

Uma maneira simples de criar uma intuição sobre a escolha de k é observar "manchas" ou "aglomerados" visuais em um gráfico, nos casos em que temos dados em duas dimensões, por exemplo. Nesse caso, o método do "olhômetro" pode funcionar como um primeiro passo: Se você vê claramente 3 grupos bem separados, experimente $k = 3$, mas se vê 2 regiões mais difusas, mas nenhuma outra concentração importante, tente $k = 2$. É claro que, para dados em várias dimensões (acima de 2D ou 3D), essa estratégia de "olhômetro" fica inviável, pois não conseguimos visualizar adequadamente. Ainda assim, a ideia de olhar a "distância interna" dos grupos ou a "silhueta" dos pontos segue sendo valiosa.

Resumo

Na prática, a melhor maneira de escolher k depende do contexto do problema. Sempre verifique se o número de grupos faz sentido dentro do domínio em que os dados foram coletados, por exemplo, biologia, economia, geografia etc. O bom uso do k-means passa por equilibrar conhecimento da área, análise visual quando possível e indicadores numéricos para tomar a decisão sobre quantos clusters realmente são úteis. Com esses conceitos em mente, estaremos mais preparados para aplicar o k-means de forma eficiente e fundamentada, compreendendo em maior profundidade como a escolha de k pode influenciar diretamente a qualidade e a utilidade dos grupos formados.

2.4 Exemplo prático do método k-means

Iniciando o diálogo...

Até agora, vimos os conceitos gerais de agrupamento, a função custo que orienta o processo de minimização da distância entre pontos e centroides, e como escolhemos e atualizamos esses centroides. Também discutimos o desafio de definir o número k de grupos. Nesta seção, vamos colocar tudo isso em prática por meio de um exemplo real de implementação do algoritmo k-means em Python. O principal objetivo é mostrar, de maneira intuitiva, como podemos agrupar um conjunto de dados em diferentes "clusters" sem precisar, necessariamente, saber a que grupo cada dado pertence de antemão. Esse é o poder do aprendizado não supervisionado!

2.4.1 Como faremos?

1. **Gerar um conjunto de dados:** Criaremos pontos de dados artificiais como se fossem coordenadas em um plano.
2. **Aplicar o k-means:** Utilizaremos uma biblioteca Python que facilita o processo, mas também revisaremos a lógica de como a distância entre pontos e centroides é utilizada para formar clusters.
3. **Visualizar os resultados:** É importante ver graficamente como o algoritmo agrupa os pontos e atualiza os centroides ao longo do processo.

Para simplificar, vamos gerar pontos em duas dimensões (x e y). Assim, conseguiremos plotar e enxergar onde cada ponto está. Suponha que cada ponto represente, por exemplo, o tempo de estudo (em horas) e a nota em uma prova de Matemática. Vamos criar quatro grupos distintos de maneira proposital, e depois veremos se o k-means consegue recuperá-los.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Gerando dados artificiais
# "random_state" é usado para manter resultados reproduzíveis
np.random.seed(42)

# Grupo 1: média em (2, 2)
grupo1 = np.random.randn(50, 2) + np.array([2, 2])
# Grupo 2: média em (8, 3)
grupo2 = np.random.randn(50, 2) + np.array([8, 3])
# Grupo 3: média em (4, 7)
grupo3 = np.random.randn(50, 2) + np.array([4, 7])
# Grupo 4: média em (8, 8)
grupo4 = np.random.randn(50, 2) + np.array([8, 8])
# Concatenando todos os grupos em um só array
X = np.vstack([grupo1, grupo2, grupo3, grupo4])
```

```
# Plotando os pontos antes do agrupamento
plt.scatter(X[:, 0], X[:, 1])
plt.title("Dados artificiais (antes do k-means)")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.show()
```

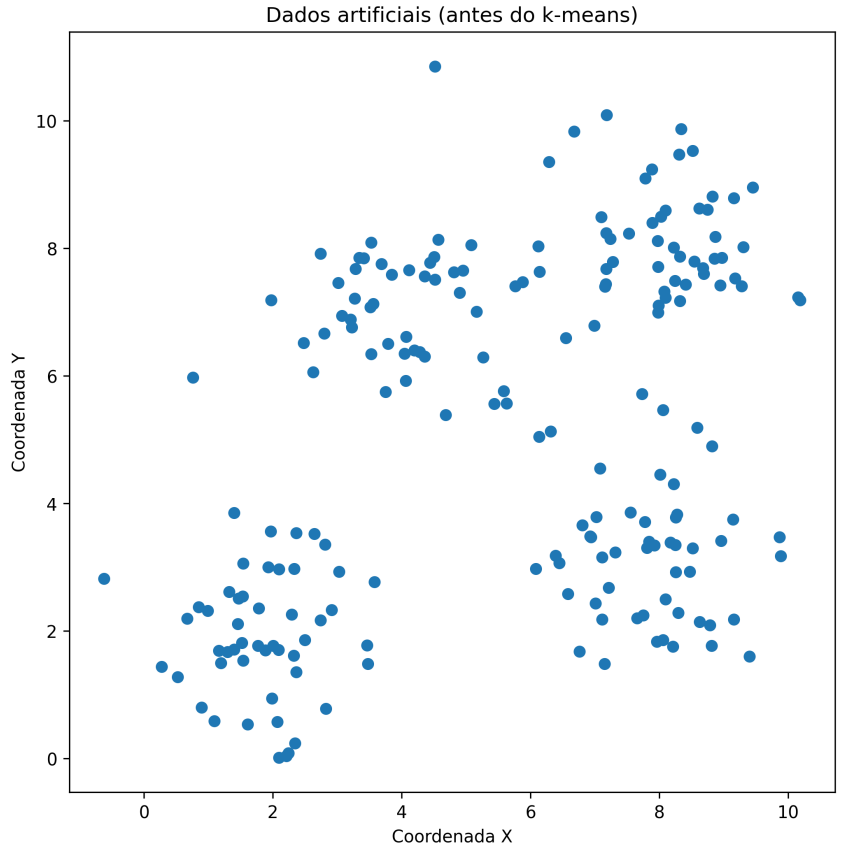


Figura 2.2: Gráfico de dispersão dos 200 pontos de dados artificiais gerados para o experimento.

Criamos 200 pontos, divididos em quatro regiões "centrais", visualmente, esperamos que o método k-means agrupe esses pontos de acordo com sua proximidade.

2.4.2 Executando o k-means

O k-means começa escolhendo k centroides iniciais e, em seguida, ajusta constantemente esses centroides até que o processo de realocação de pontos pare de mudar muito ou até atingir um número máximo de iterações. Cada realocação de pontos visa minimizar a soma das distâncias quadráticas entre cada ponto e o centróide de seu grupo. Vamos escolher $k = 4$ para ver se o algoritmo consegue recuperar as "regiões" que geramos.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Gerando dados artificiais (código omitido, igual ao anterior)
np.random.seed(42)
grupo1 = np.random.randn(50, 2) + np.array([2, 2])
grupo2 = np.random.randn(50, 2) + np.array([8, 3])
grupo3 = np.random.randn(50, 2) + np.array([4, 7])
grupo4 = np.random.randn(50, 2) + np.array([8, 8])
X = np.vstack([grupo1, grupo2, grupo3, grupo4])

# Aplicando k-means para k=4
k = 4
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

# Atribuições finais dos pontos aos clusters
labels = kmeans.labels_

# Coordenadas dos centróides finais
centers = kmeans.cluster_centers_

# Plotando os pontos e os centróides
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200,
   edgecolors='black', c="#40f75c")
plt.title("Agrupamento com k-means (k=4)")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.show()
```

O que está acontecendo no código:

- `KMeans(n_clusters=k, random_state=42)`: indica que vamos agrupar em quatro clusters `n_clusters = 4` e fixamos `random_state` para mantermos resultados iguais em cada execução.
- `kmeans.fit(X)`: executa o algoritmo k-means nos dados X.
- `kmeans.labels_`: para cada ponto, o algoritmo atribui um rótulo (*label*) que indica a qual dos quatro grupos o ponto pertence.
- `kmeans.cluster_centers_`: ao final do ajuste, o k-means retorna a posição de cada centroide.

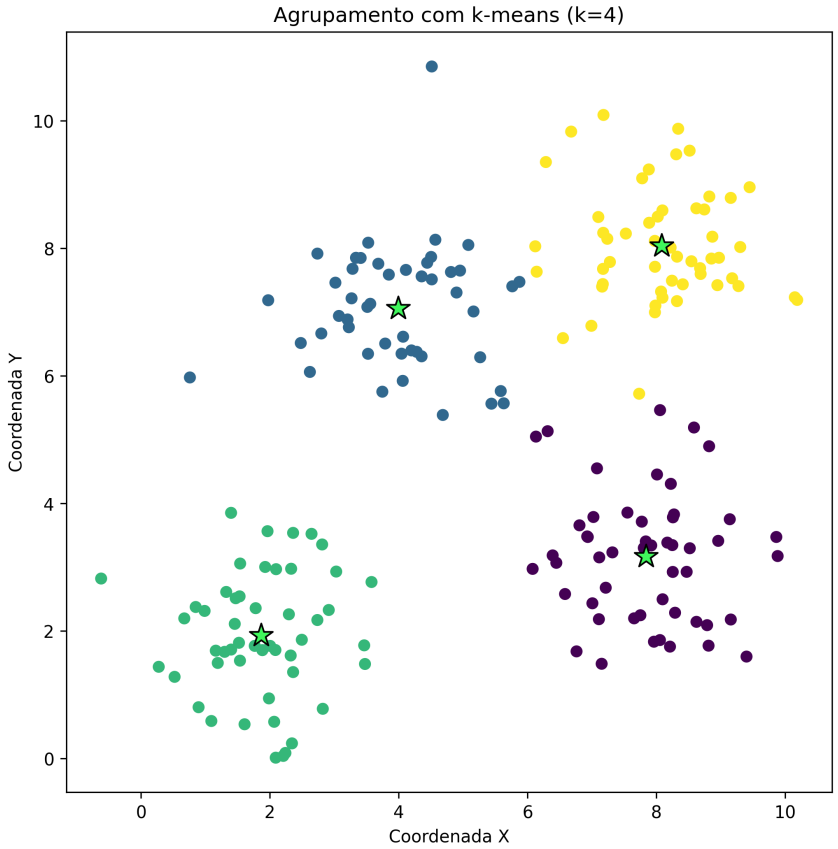


Figura 2.3: Resultado da aplicação do algoritmo k-means com k=4.

No gráfico, usamos cores diferentes (`c=labels`) para distinguir os pontos de cada grupo, e marcamos os centroides com uma estrela usando `marker='*'`. Experimente variar o número de clusters de 2 até 6. Observe como o agrupamento muda e pense em qual cenário faz mais sentido.

2.4.3 Interpretando o resultado

Se o k-means foi bem-sucedido e nosso número $k = 4$ fez sentido, veremos que os pontos se dividiram em quatro regiões, próximas aos "centros" que geramos. Além disso, cada centroide deverá estar aproximadamente na média dos pontos de seu cluster.

- **Função custo:** nesse processo, a soma das distâncias quadráticas entre pontos e seus centroides é minimizada. Ou seja, o algoritmo encontra a forma de agrupar que deixa os pontos "o mais próximo possível" de seus respectivos centroides.
- **Atualização dos centroides:** a cada iteração, o centroide se move para a média dos pontos que pertencem àquele cluster.
- **Escolha do parâmetro k:** nós "acertamos" $k = 4$ porque geramos quatro grupos. Na prática, podemos usar métodos de avaliação, como a "curva do cotovelo" ou o "silhouette score", para escolher k.

2.4.4 Conexão com o aprendizado de máquina

Esse exemplo mostra como o k-means pode ser aplicado em problemas simples, mas a ideia geral pode ser levada para cenários muito mais complexos: segmentação de clientes, análise de imagens, compressão de dados etc. A intuição que queremos reforçar aqui é:

- Cada ponto "pertence" ao centroide mais próximo.
- Os centroides se realocam até "ficarem bons" em representar seus grupos.
- O processo tende a convergir rapidamente, embora, em casos mais avançados, seja preciso retestar com diferentes pontos iniciais ou usar métodos mais sofisticados.

2.4.5 Resumo

- O k-means é simples e eficiente para agrupar dados em k grupos diferentes.
- Ele depende fortemente da escolha de k. É comum testar vários valores antes de decidir o melhor.
- Visualizar o resultado ajuda a entender melhor a lógica de "aproximar pontos a um centroide".

2.5 Análise dos componentes principais (PCA)

Iniciando o diálogo...

Imagine que você tem um conjunto de dados com várias características ou *features*, a análise de componentes principais (PCA) é amplamente utilizada para reduzir dimensionalidade preservando a variância dos dados (Goodfellow; Bengio; Courville, 2016). Por exemplo, vamos considerar um estudo que tenta relacionar hábitos de estudo, horas de sono, tipos de alimentação e o desempenho de diferentes estudantes em uma prova. Cada estudante poderia ser descrito por muitas variáveis, intensidade do estudo, qualidade do sono, número de horas dormidas, variedade da dieta, idade, entre outras. Quando analisamos um conjunto de dados grande desse modo, dizemos que temos um problema de "alta dimensionalidade", pois cada variável representa uma dimensão. A princípio, quanto mais dimensões tivermos, mais rica pode ser a descrição dos nossos dados. Entretanto, surge um desafio: fica difícil, ou mesmo impossível, analisar e visualizar todas essas dimensões ao mesmo tempo. É aqui que entra o conceito de redução de dimensionalidade, um conjunto de técnicas que nos permite "condensar" ou "resumir" as informações de um grande número de dimensões em apenas duas, três ou algumas poucas dimensões para facilitar a análise e a visualização, sem perder a maior parte do significado original.

2.5.1 Redução de Dimensionalidade

Para entender o que é reduzir a dimensionalidade, vamos recorrer a um exemplo intuitivo, pense em desenhar pontos em um papel, com duas dimensões (x e y), podemos representar cada ponto num plano, com três dimensões, a representação se torna um pouco mais difícil de visualizar, pois precisaríamos de um ambiente 3D, agora, imagine termos dez dimensões, cem

dimensões ou até milhares de dimensões. Para o cérebro humano, é impossível "enxergar" diretamente essas "dimensões extras".

Entretanto, muitas dessas dimensões podem conter informações semelhantes ou até mesmo redundantes. Em outras palavras, há variáveis que se sobrepõem no que representam, elas podem estar altamente correlacionadas. Isso faz pensar: será que precisamos de todas essas variáveis para ter uma boa visão do que está acontecendo no nosso conjunto de dados?

A ideia de redução de dimensionalidade é pegar um conjunto com muitas dimensões e encontrar uma forma de sintetizá-las em poucas dimensões, aproveitando as correlações, ou seja, os padrões de variabilidade entre as variáveis para "comprimir" a informação. O truque é tentar fazer isso de modo a perder o mínimo de informação possível.

No caso do PCA (Principal Component Analysis), nós procuramos as direções (ou componentes principais) que concentram a maior parte da variação existente no conjunto de dados. Ao projetar os dados nessas poucas direções, conseguimos reter boa parte do que é mais relevante, em termos de variabilidade, e ignorar o que é menos importante ou redundante.

2.5.2 Relação com a Visão Gráfica

Para visualizar com mais clareza, um exemplo comum é pensar em um conjunto de pontos em três dimensões (eixo X, Y e Z). Às vezes, esses pontos estão, essencialmente, distribuídos em torno de um plano. Se esse for o caso, provavelmente não precisamos de três dimensões completas para descrever os pontos, um plano 2D pode ser o suficiente. E, ao "descer" de três para duas dimensões, tornamos a visualização mais fácil, mantendo a maior parte da "estrutura" dos dados.

No ensino médio, costumamos ver gráficos de funções de duas ou três variáveis. Agora, quando falamos de dezenas ou centenas de variáveis, não conseguimos mais usar o mesmo truque de "eixos" para enxergar tudo de uma só vez. O PCA e outras técnicas de redução de dimensionalidade são, portanto, nossa forma de fazer esse "zoom out": mesmo que as variáveis sejam muitas, podemos encontrar uma projeção, ou várias projeções, que facilitem a análise, seja para fins de descrição, para preparar o terreno para outro método de aprendizado de máquina ou para resumo e comunicação dos resultados.

2.5.3 Por que reduzir a dimensionalidade é Importante?

- **Visualização e Exploração:** Em alta dimensionalidade, é difícil criar gráficos intuitivos. Ao projetar dados em poucas dimensões, podemos gerar gráficos mais simples, como diagramas de dispersão (2D) ou representações tridimensionais, facilitando a análise inicial dos dados.

- **Redução de Ruído:** Certas variáveis podem conter muito ruído ou pouca informação relevante. Ao compactar a informação nas direções que de fato explicam a maior variação, muitas vezes conseguimos deixar de lado esse "ruído" e nos concentrar no que realmente importa.

- **Economia de Recursos:** Modelos de aprendizado de máquina sofrem quando têm de lidar com muitas variáveis irrelevantes ou redundantes. Reduzir as dimensões pode ajudar a treinar modelos com maior eficiência, além de evitar um problema conhecido como overfitting, que é quando o modelo se "vicia" nos detalhes específicos do conjunto de treinamento, perdendo capacidade de generalização.

- **Simplicidade e Interpretabilidade:** Ter um conjunto menor de variáveis derivadas (componentes principais) pode ajudar a interpretar melhor as relações existentes nos dados, embora, nesse ponto, seja preciso equilibrar a redução com a possibilidade de perder, nas novas combinações, o sentido prático das variáveis originais.

2.5.4 Formalismo Matemático

Mesmo que o objetivo aqui seja dar uma visão intuitiva, vale a pena dar um passo adiante e citar de forma simples como o PCA faz essa "mágica":

1. **Matriz de Dados:** Começamos organizando nossos dados em uma matriz. Cada linha representa um indivíduo ou uma observação e cada coluna, uma variável.

2. **Centralizar os Dados:** Depois, subtraímos da matriz a média de cada coluna, para que todas fiquem centradas em torno de zero.

3. **Calcular a Covariância:** Em seguida, calculamos a matriz de covariância, que mostra as correlações entre as variáveis (ou uma matriz de correlação, em alguns casos).

4. **Autovalores e Autovetores:** A partir dessa matriz de covariância, encontramos os autovalores e autovetores. Os autovetores são as direções (ou eixos) onde os dados variam mais, enquanto os autovalores medem a importância (quantidade de variância) que cada uma dessas direções captura.

5. **Componentes Principais:** Ordenamos os autovetores pela magnitude de seus autovalores do maior para o menor e, então, escolhemos apenas alguns deles, pois eles retêm a maior parte da variabilidade total dos dados. Cada autovetor selecionado corresponde a uma "direção principal" e projetar nossos dados nessas direções reduz o número de dimensões, mas conserva a maior parte da informação.

2.5.5 Um exemplo intuitivo para fixar

Pense em uma sala de aula em que se queira analisar como os estudantes se agrupam em relação às suas notas em três disciplinas: Matemática, Língua Portuguesa e Ciências. Você obtém três notas por estudante. Isso forma um espaço tridimensional: cada estudante é um ponto nesse espaço, nota de Matemática no eixo X, nota de Língua Portuguesa no eixo Y, nota de Ciências no eixo Z.

Porém, pode acontecer de Matemática e Ciências estarem altamente correlacionadas, porque quem se destaca em uma, muitas vezes se destaca na outra. Nesse caso, não precisamos de duas dimensões separadas para representar essas duas disciplinas de forma independente. Se projetarmos (via PCA) esses dados em duas dimensões, é provável que a primeira componente principal seja algo como o "desempenho geral em Exatas", enquanto a segunda componente possa estar mais associada a aspectos que diferenciam Língua Portuguesa de Matemática e Ciências, por exemplo.

Em outras palavras, passamos de três notas para dois "indicadores" que resumem bem a variação desses estudantes, preservando a maior parte da informação. Visualmente, deixamos de ter um ponto em 3D para ter pontos em 2D, sem perder muito de nossa capacidade de identificar padrões.

2.5.6 Resumo

Vimos como a redução de dimensionalidade surge como uma solução poderosa quando nos deparamos com conjuntos de dados grandes e complexos. A ideia de procurar as direções que mais "explicam" a variação dos dados e, assim, resumir muitas variáveis em poucas, é central no Método de Análise de Componentes Principais (PCA).

A seguir, ao mergulharmos no estudo de covariância, autovalores e autovetores, veremos em detalhes como o PCA faz esse resumo de informação de maneira matemática. Depois, veremos como gerar visualizações, como o scree plot, para decidir quantos componentes principais usar. Por fim, aplicamos o método em um exemplo prático, para consolidar o aprendizado.

Ou seja, se você já se perguntou por que muitas vezes vemos gráficos com apenas duas ou três dimensões quando o conjunto de dados original tem muito mais variáveis, a resposta está na redução de dimensionalidade. O PCA, em particular, é uma das ferramentas mais populares para esse fim, pois alia uma boa intuição de compressão de informação com bases matemáticas sólidas.

Fique Alerta!

Mesmo que a ideia de "jogar fora" algumas dimensões possa parecer assustadora, afinal, poderíamos estar perdendo dados, na prática, as dimensões descartadas tendem a explicar pouca parte da variação total. Portanto, a redução geralmente facilita nossa vida sem prejudicar a análise, desde que verifiquemos quantos componentes principais de fato precisamos para representar adequadamente a informação relevante.

2.6 Covariância, autovalores e autovetores

Iniciando o diálogo...

Nesta seção, vamos explorar alguns conceitos centrais para a Análise de Componentes Principais (PCA). Nosso foco será entender o papel da covariância, bem como o que são autovalores e autovetores, e por que eles são tão importantes na redução de dimensionalidade.

2.6.1 Por que medir a covariância?

Quando trabalhamos com dados em várias dimensões (por exemplo, altura, peso, idade, notas de uma turma etc.), muitas vezes nos interessa descobrir se existe alguma relação entre essas variáveis. Por exemplo, se quisermos analisar a relação entre altura e peso de um grupo de estudantes, podemos querer saber se à medida que a altura aumenta, o peso também tende a aumentar, ou se não há uma relação clara.

A covariância é uma medida que quantifica o grau de variação conjunta entre duas variáveis. Em termos simples, se a covariância entre duas variáveis for **positiva**, significa que elas tendem a crescer ou diminuir juntas, mas se a covariância for **negativa**, significa que quando uma variável cresce, a outra tende a diminuir, agora, se for próxima de **zero**, as variáveis são, sob certo ponto de vista, "independentes" no que se refere à variação linear, ou seja, não crescem ou diminuem juntas de modo consistente.

Fórmula matemática

A covariância entre duas variáveis X e Y pode ser calculada, de forma simplificada, como a média do produto das diferenças em relação às médias de cada variável:

$$\text{Cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})$$

onde:

- X_i e Y_i são os valores observados das variáveis X e Y para o i-ésimo indivíduo.
- $\bar{X}$ e $\bar{Y}$ são as médias amostrais de X e Y, respectivamente.
- n é a quantidade de observações.

Essa expressão pode parecer um pouco abstrata, mas o fundamental é perceber que o produto $(X_i - \bar{X})(Y_i - \bar{Y})$ é positivo quando X e Y "andam juntas", ou seja, ambas estão acima ou ambas estão abaixo de suas médias e é negativo quando elas se desviam em sentidos opostos, uma está acima da média, a outra está abaixo, e vice-versa.

Matriz de Covariância

Quando lidamos com várias variáveis ao mesmo tempo, em vez de calcular a covariância "dupla a dupla" individualmente, é comum organizar os valores em uma matriz de covariância. Suponha que tenhamos um conjunto de dados com d variáveis. A matriz de covariância será uma matriz $d \times d$ (ou seja, d linhas e d colunas), onde cada elemento (i, j) corresponde à covariância entre a variável i e a variável j.

$$\Sigma = \begin{pmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & \cdots & \text{Cov}(X_1, X_d) \\ \text{Cov}(X_2, X_1) & \text{Var}(X_2) & \cdots & \text{Cov}(X_2, X_d) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_d, X_1) & \text{Cov}(X_d, X_2) & \cdots & \text{Var}(X_d) \end{pmatrix}$$

Note que a diagonal principal (onde linha e coluna são iguais) apresenta as variâncias de cada variável (ou seja, $\text{Var}(X_1)$, $\text{Var}(X_2)$, etc.). Além disso, como a covariância é simétrica ($\text{Cov}(X_i, X_j) = \text{Cov}(X_j, X_i)$), a matriz de covariância é também simétrica. Essa matriz Σ é central para o PCA, pois ela captura em um só lugar toda a informação de quanta variação as variáveis possuem individualmente (variâncias) e de como elas se relacionam entre si (covariâncias).

Autovalores e Autovetores

Para compreender o PCA, precisamos entender dois conceitos matemáticos importantes: autovalores, também chamados de valores próprios ou *eigenvalues* e autovetores, vetores próprios ou *eigenvectors*. Em linhas gerais, se temos uma matriz A e um vetor v, então v é um autovetor de A se, ao multiplicarmos A por v, obtemos um múltiplo de v. Matematicamente, isso significa:

$$Av = \lambda v$$

onde λ é o autovalor (ou valor próprio) associado a esse autovetor v. Podemos pensar em λ como o "quanto" esse vetor v é "esticado" ou "comprimido" quando a matriz A atua sobre ele.

No caso da PCA, a matriz A que nos interessa é exatamente a matriz de covariância Σ. Assim, quando encontramos os autovetores e autovalores de Σ, estamos encontrando as direções e as magnitudes de variação máxima dos nossos dados.

De forma intuitiva, cada autovetor da matriz de covariância aponta para uma direção onde os dados variam de maneira específica, geralmente procurando a maior variabilidade possível. Já o autovalor associado indica o "tamanho" dessa variação. Em poucas palavras:

- **Autovetor** = direção de maior variância nos dados.
- **Autovalor** = a quantidade de variância na direção desse autovetor.

Quando realizamos PCA, ordenamos os autovetores de acordo com seus autovalores, do maior para o menor. Os primeiros autovetores, aqueles associados aos maiores autovalores, correspondem às direções onde os dados mais "espalham", de modo que reter essas direções iniciais significa capturar a maior parte da variabilidade dos dados.

Uma forma de visualizar isso é imaginar um conjunto de pontos em duas dimensões, por exemplo, altura e peso. Se enxergarmos que a maior variação dos dados está no sentido "diagonal", por exemplo, do canto inferior esquerdo para o canto superior direito do plano, então essa diagonal será nosso primeiro autovetor. Já o segundo autovetor, perpendicular ao primeiro, será a direção onde ainda existe alguma variação, menor, porém existente.

2.6.2 Por que tudo isso é útil?

Ao compreender covariância, autovalores e autovetores, fica mais claro porque o PCA é capaz de "resumir" os dados em um número menor de dimensões. As direções principais (autovetores) com maiores autovalores permitem concentrar uma porção significativa da informação (variância) do conjunto original, descartando as direções de menor relevância.

2.6.3 Resumo

A covariância mede como duas variáveis variam juntas. A matriz de covariância resume essas relações em todo o conjunto de dados. A partir dela, extraímos autovalores e autovetores, que são fundamentais para entender a estrutura dos dados. Os autovetores indicam as direções principais de variação (novos eixos), enquanto os autovalores mostram quanta variância (informação) há em cada direção. Juntos, eles permitem técnicas como o PCA, que projeta os dados em um espaço de menor dimensão, preservando o máximo possível da informação original.

2.7 Visualizações e scree plot

Iniciando o diálogo...

Quando trabalhamos com dados de diversas variáveis ao mesmo tempo, pode ser desafiador entender "quem está influenciando o quê" e como essas variáveis se relacionam entre si. A Análise de Componentes Principais (PCA) oferece um caminho para lidar com esse tipo de problema, reduzindo a dimensionalidade dos dados de forma a manter, na medida do possível, a informação relevante. Nesta seção, vamos explorar as principais formas de visualização dos dados após a aplicação do PCA, com destaque para um gráfico bastante utilizado chamado *Scree Plot*.

2.7.1 Visualizações com PCA

Quando reduzimos a dimensionalidade do conjunto de dados para 2 ou 3 componentes, passamos a ter a possibilidade de criar gráficos de dispersão (*scatter plots*) que nos permitem enxergar padrões nos dados. Essas representações bidimensionais ou tridimensionais ajudam a responder a perguntas como:

- Existem grupos naturais (agrupamentos) nos dados?
- Como cada ponto (exemplo) se posiciona em relação aos outros quando projetamos em menos dimensões?
- Existe algum padrão visual que sugira correlação ou separação clara entre os exemplos?

Imagine que você tem um conjunto de dados sobre flores, com várias medidas (largura da pétala, largura da sépala, comprimento da pétala, comprimento da sépala, entre outras). Se você aplicar o PCA, muito provavelmente descobrirá que algumas dessas medidas são "parecidas" e capturam basicamente a mesma informação, pense em pétalas e sépalas correlacionadas. Ao calcular as duas primeiras componentes principais, você pode gerar um gráfico 2D onde cada flor é representada por um ponto. Muitas vezes, apenas olhando para esses dois novos eixos (componentes principais), já é possível ver grupos de flores que se separam nitidamente no gráfico.

Passo a passo básico para visualização:

1. Pegar os dados originais e normalizá-los para que cada variável tenha, por exemplo, média 0 e desvio padrão 1, facilitando comparações.
2. Calcular os autovetores e autovalores a partir da matriz de covariância ou correlação dos dados. Esses autovetores serão as direções das componentes principais, e os autovalores indicarão a importância de cada direção. No mesmo momento será escolhido os dois componentes principais e projetar os dados nessas dimensões. Dessa forma, se obtém um novo conjunto de coordenadas que podem ser plotadas.
3. Criar o gráfico de dispersão (2D ou 3D). Interpretar visualmente possíveis agrupamentos, tendências ou padrões.

Copie e Teste!

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Carregando os dados
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names

# Normalização (média 0, desvio padrão 1)
```

```
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Aplicando PCA
pca = PCA(n_components=2) # Duas componentes principais
X_pca = pca.fit_transform(X_normalized)

# Variância explicada
print("Variância explicada por cada componente:", pca.
    explained_variance_ratio_)

# Criar o gráfico
plt.figure(figsize=(8,6))
for i, target_name in enumerate(target_names):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], label=
    target_name)

plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('PCA do Dataset Iris')
plt.legend()
plt.grid(True)
plt.show()
```

Resultado Esperado

```
Variância explicada por cada componente:
[0.72962445 0.22850762]
```

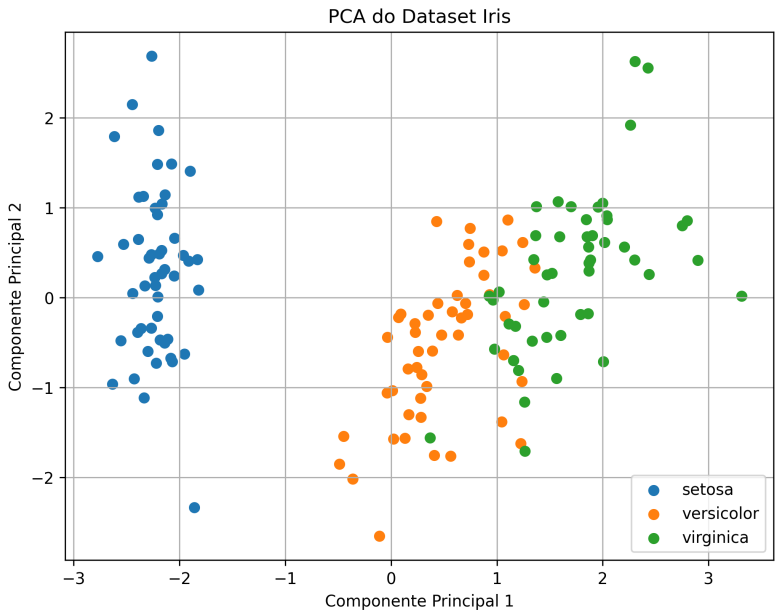


Figura 2.4: PCA do Dataset Iris.

Esse gráfico mostra cada flor como um ponto no plano 2D. Mesmo sem saber quais são as variáveis originais, as espécies se separam visivelmente, especialmente setosa. Isso indica que o PCA capturou bem as diferenças entre as flores, usando apenas duas direções principais.

2.7.2 O que é o Scree Plot?

O *Scree Plot* é um dos principais gráficos usados para decidir quantos componentes principais deveríamos manter para a nossa análise. O nome "scree" vem de "pedregulho" ou "cascalho", pois ele representa visualmente como a "importância", variância explicada, diminui conforme passamos de um componente principal para o próximo.

Como interpretar o Scree Plot

- O eixo horizontal (x) costuma listar os componentes principais em ordem decrescente de importância: PC1, PC2, PC3, e assim por diante.
- O eixo vertical (y) costuma representar a variância explicada ou a proporção da variância explicada por cada componente.
- Cada ponto no gráfico corresponde a um componente. O primeiro ponto (PC1) tende a explicar a maior fatia da variância total; o segundo (PC2) explica a segunda maior fatia, e assim sucessivamente.
- À medida que avançamos pelos componentes, a parcela de variância explicada normalmente diminui. O *Scree Plot* ajuda a identificar visualmente onde ocorre uma queda brusca no "peso" da variância explicada. Uma das práticas comuns é observar o "joelho" do gráfico ("elbow"), que é onde a curva deixa de ter uma queda acentuada e começa a ficar quase horizontal. Geralmente, os componentes principais até esse "joelho" são suficientes para descrever a maior parte da variabilidade dos dados.

Passo a passo para construir o Scree Plot:

1. **Encontrar os autovalores:** Cada componente principal tem um autovalor associado, que indica a variância capturada.
2. **Ordenar os autovalores em ordem decrescente:** Do maior (mais importante) para o menor (menos importante).
3. **Calcular a variância explicada** (ou a porcentagem da variância explicada) por cada componente:
$$\text{Variância Explicada do PC}_i = \frac{\lambda_i}{\sum_j \lambda_j} \times 100\%$$
onde λ_i é o autovalor do componente i.
4. **Plotar os valores** (ou as porcentagens) no eixo vertical em relação ao índice do componente (1º, 2º, 3º...) no eixo horizontal.

O gráfico resultante permite uma visualização clara da contribuição marginal de cada componente. O ponto de inflexão, conhecido como "cotovelo"(*elbow*), é o critério visual usado para decidir quantos componentes reter.

Caso Prático

Visualizando o Scree Plot
Vamos agora aplicar, na prática, o que aprendemos. Imagine um conjunto de dados com informações sobre frutas: peso, doçura, acidez e porcentagem de água. Queremos usar o PCA para entender quantas dimensões realmente precisamos para representar bem esses dados. Com poucas linhas de código, podemos padronizar os dados, aplicar o PCA e visualizar o gráfico de variância explicada por cada componente.

Copie e Teste!

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Dados simulados
dados = pd.DataFrame({
    "peso": [120, 130, 115, 140, 155],
    "doçura": [7.2, 6.8, 7.4, 6.5, 6.9],
    "acidez": [3.1, 3.3, 2.9, 3.5, 3.2],
    "água (%)": [85, 86, 84, 83, 82]
})

# Padronizar os dados
scaler = StandardScaler()
dados_norm = scaler.fit_transform(dados)

# Aplicar PCA
pca = PCA()
pca.fit(dados_norm)

# Variância acumulada
var_acumulada = np.cumsum(pca.explained_variance_ratio_) * 100

# Scree Plot com melhorias
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(var_acumulada)+1), var_acumulada, marker='o'
   , linestyle='--', color='b')
plt.axhline(y=95, color='r', linestyle='--', label='95% da
   variância')
plt.xlabel('Número de Componentes Principais')
plt.ylabel('Variância Explicada Acumulada (%)')
plt.title('Scree Plot - PCA nas características das frutas')
plt.xticks(range(1, len(var_acumulada)+1))
plt.ylim(0, 105)
plt.grid(True)
plt.legend()
plt.tight_layout()
```

```
plt.show()

print("Variância explicada acumulada (%):", var_acumulada)
```

Resultado Esperado

```
Variância explicada(%): [67.9646   95.6828   99.8385  100.0]
```

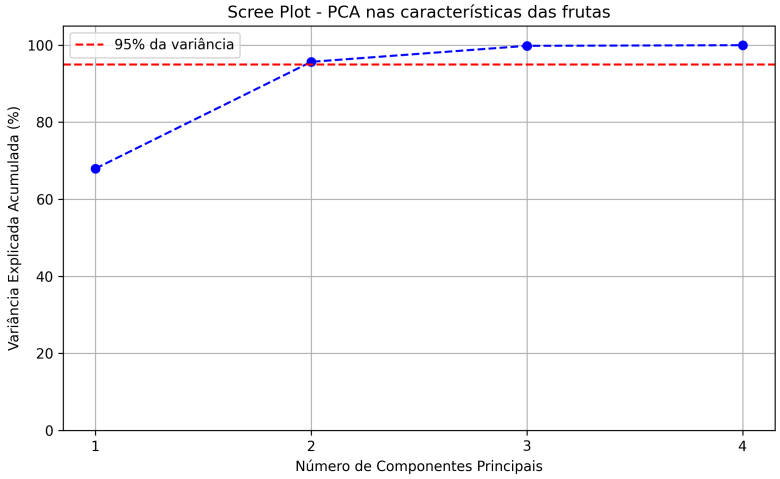


Figura 2.5: Scree Plot - PCA nas características das frutas.

O gráfico mostra o acúmulo da variância explicada conforme aumentamos o número de componentes. Neste exemplo, nossos dois primeiros componentes explicam aproximadamente 95% da variância, isso indicaria que poderíamos reduzir de 4 para apenas 2 dimensões, mantendo quase toda a informação dos dados originais.

2.7.3 Interpretação e Escolha do Número de Componentes

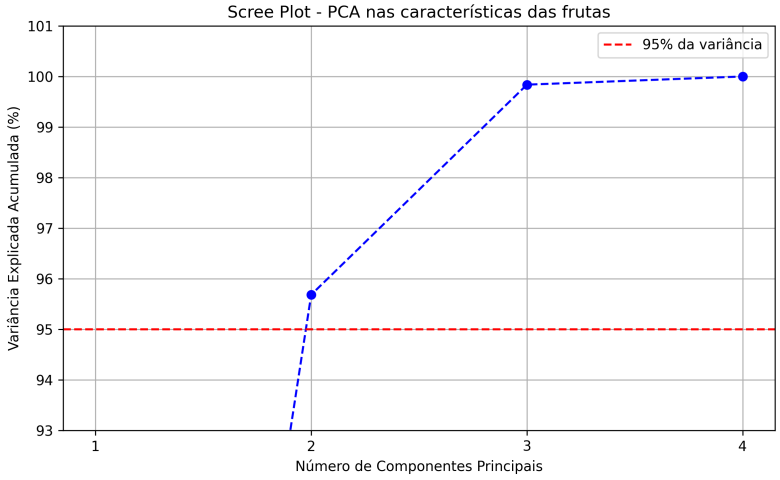


Figura 2.6: Scree Plot - PCA nas características das frutas com destaque no ”joelho”da curva.

Um dos dilemas na aplicação do PCA é determinar quantos componentes manter para reduzir os dados sem perder informação importante. O Scree Plot auxilia bastante nessa decisão, mas não existe uma "fórmula mágica" exata. Algumas orientações práticas:

- **Verifique o "joelho" da curva (elbow method):** Se houver um ponto em que a curva passa a diminuir muito lentamente, esse ponto (ou um pouco antes dele) costuma ser escolhido como o número de componentes adequados, observe a imagem acima com o destaque no "joelho".
- **Soma cumulativa da variância explicada:** Muitas vezes, estabelecemos um patamar de "95% de variância explicada" ou outro valor definido previamente. Nesse caso, somamos as porcentagens da variância explicada componente a componente, até atingir o nível desejado.
- **Análise prática do contexto:** Em algumas situações, podemos manter um número maior de componentes se cada componente tiver um significado interpretável. Em outras, podemos preferir soluções mais enxutas com poucos componentes, para facilitar a visualização ou processamento dos dados. A escolha final, portanto, depende do equilíbrio entre simplificar o máximo possível (menos dimensões) e preservar informação suficiente (variância explicada alta).

2.7.4 Resumo

A Análise de Componentes Principais (PCA) é uma ferramenta poderosa para transformar conjuntos de dados complexos em representações gráficas mais simples e intuitivas. Por meio dela, é possível identificar padrões, relações e agrupamentos que, em um espaço com muitas variáveis, estariam ocultos.

- Um dos principais recursos visuais da PCA é o Scree Plot, que mostra quanta variância cada componente principal consegue explicar. Ele ajuda a decidir de forma objetiva quantos componentes realmente valem a pena manter, indicando onde está o ponto de equilíbrio entre simplificação e perda de informação.
- As visualizações em duas ou três dimensões, feitas com os primeiros componentes, tornam possível "ver" estruturas e agrupamentos que antes estavam escondidos, facilitando a análise e a interpretação dos dados.
- A escolha do número de componentes deve equilibrar critérios matemáticos como a variância explicada ou o ponto de inflexão (joelho na curva) do Scree Plot, com o contexto prático e a clareza na interpretação dos resultados.

Mais do que apenas reduzir variáveis, o PCA nos permite reorganizar os dados de uma forma mais compreensível. Usar essa técnica com consciência envolve entender o papel das componentes principais, interpretar o Scree Plot e transformar os resultados em algo aplicável, mesmo em contextos educacionais como o ensino médio. Assim, seja em áreas como ciências, finanças ou esportes, o PCA se torna uma forma eficaz de revelar padrões e simplificar a complexidade dos dados.

2.8 Detecção de anomalia

Iniciando o diálogo...

Imagine que você trabalha em uma loja virtual e precisa monitorar milhares de transações por dia para identificar possíveis fraudes. Ou pense em um sistema de sensores instalado em uma fábrica, onde cada sensor mede a temperatura, a pressão e outras variáveis que indicam o bom funcionamento das máquinas. Em ambos os casos, queremos identificar rapidamente qualquer ponto "estranho" ou suspeito, para evitar prejuízos ou acidentes. Esse processo de identificar valores fora do comum é o que chamamos de detecção de anomalias. Existem diversas maneiras de se detectar anomalias, mas uma das mais simples e poderosas utiliza a distribuição Gaussiana (também conhecida como distribuição normal). A ideia fundamental é modelar o comportamento "normal" dos dados por meio de uma "função de probabilidade" e, em seguida, verificar quando um novo dado se afasta muito desse comportamento.

2.8.1 A Distribuição Gaussiana

A distribuição Gaussiana é uma das métricas mais importantes na estatística e na ciência de dados. Ela é comumente representada pelo famoso "formato de sino" (a famosa "curva em forma de sino").

Por que "normal"?

A distribuição Gaussiana é chamada de "normal" pois, em muitos casos práticos, grande parte dos fenômenos naturais e sociais tende a apresentar distribuições que se aproximam dessa forma, por exemplo: alturas de pessoas em uma população, erros de medição em experimentos etc. Embora na prática nem todo dado real siga exatamente essa distribuição, ela costuma ser um bom ponto de partida para diversos problemas de análise.

Representação matemática

Na forma mais simples, univariada, para um valor x medido, a distribuição Gaussiana é descrita pela função:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

onde:

- μ é a média dos valores, o centro da distribuição
- σ^2 é a variância que descreve a dispersão em torno da média; o desvio-padrão é σ

Para nós, o importante é entender que essa função nos diz o "quão provável" é observar cada valor de x. Se x estiver muito distante de μ, ou seja, for um valor muito diferente do comum, a probabilidade $p(x)$ será bem pequena. Quando utilizamos a distribuição Gaussiana para a detecção de anomalias, estamos assumindo que os dados "normais" podem ser bem representados por essa forma de distribuição. Assim, valores que tiverem probabilidade muito baixa, ou seja, estiverem "na cauda" da distribuição, podem ser considerados anômalos.

2.8.2 Modelando Dados com Distribuição Gaussiana

Suponha que você tenha um conjunto de dados históricos que acredita serem livres de anomalias, o que chamamos de conjunto de treinamento. Por exemplo, transações que foram confirmadas como legítimas, leituras de sensores sem defeitos, ou outro cenário onde se sabe que os dados representam "comportamento normal". O processo geral é o seguinte:

1. **Estimação dos parâmetros (μ e σ^2):**
    - Calcular a média μ de todos os valores, por exemplo, todos os valores de temperatura medidos
    - Calcular a variância σ^2, ou seja, o quanto os valores variam em torno da média
2. **Cálculo da probabilidade de um ponto (x):**
    - Dado μ e σ^2, aplicamos a fórmula da Gaussiana para encontrar $p(x)$
    - Se $p(x)$ for muito baixo, isso indica que x é bastante improvável, ou seja, potencialmente anômalo
3. **Definição de um limiar (*threshold*):**
    - Precisamos escolher um valor ϵ (*épsilon*) de probabilidade abaixo do qual consideramos que o ponto x é anômalo. Essa escolha pode ser feita de várias maneiras, como analisando a distribuição dos dados de treinamento ou validando em um conjunto de exemplos conhecidos de anomalias.

Caso Univariado

No caso univariado, trabalhamos apenas com um atributo. Por exemplo, apenas a temperatura de um motor. Podemos usar a curva Gaussiana para estimar a probabilidade de cada temperatura ocorrer. Quando uma nova leitura chega, calculamos $p(\text{temperatura})$. Se esse valor for menor que o limiar ϵ, classificamos como anômalo.

Caso Multivariado

Na prática, contudo, muitas vezes temos vários atributos relevantes. Por exemplo, uma transação online tem valor total, localização do comprador, horário do dia, método de pagamento etc. Um único atributo fora da curva não significa necessariamente fraude, mas uma combinação de vários fatores pode sim apontar algo suspeito. Nesse caso, podemos estender o mesmo raciocínio para a distribuição Gaussiana multivariada. Em vez de simplesmente termos média e variância, passamos a ter:

- Um vetor de médias μ, onde cada componente é a média de um atributo.
- Uma matriz de covariância Σ, que não só informa quão dispersos estão os dados em cada atributo, mas também descreve como eles se relacionam entre si.

Matematicamente, a distribuição Gaussiana multivariada de um vetor de atributos $\mathbf{x} \in \mathbb{R}^n$ é dada por:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

onde:

- n é o número de atributos (dimensionalidade do vetor $\mathbf{x}$)
- Σ é a matriz de covariância ($n \times n$), e $|\Sigma|$ é o determinante dela
- Σ^{-1} é a inversa de Σ

Apesar de parecer complexa, a lógica é a mesma, quanto mais distante $\mathbf{x}$ está de μ, levando em conta as correlações entre atributos, menor será a probabilidade $p(\mathbf{x})$.

2.8.3 Escolhendo o limiar épsilon

A escolha do limiar ϵ é um ponto crítico. Se ϵ for muito alto, você acaba sinalizando anomalias com muita facilidade, gerando muitos "falsos positivos", você acha que algo está errado quando na verdade não está. Se ϵ for muito baixo, você pode deixar passar pontos anômalos, gerando "falsos negativos", você não acha nada de errado, mas na verdade há um problema ali. Uma forma prática de ajustar ϵ é:

1. Separar uma parte dos dados conhecidos em um conjunto de validação, onde há exemplos tanto de casos "normais" quanto de casos "anormais".
2. Variar ϵ e medir, em cada valor, a taxa de acerto em classificar os casos do conjunto de validação.
3. Escolher o ϵ que faz um bom equilíbrio entre detectar anomalias e não gerar alarmes em excesso.

2.8.4 Aplicando Modelos Gaussianos na Detecção de Anomalias

1. **Coletar dados representativos do comportamento normal:** Pode ser o histórico de transações legítimas, leituras de sensores com funcionamento correto etc.
2. **Estimar parâmetros (univariados ou multivariados):** Calcular média(s) (μ ou μ) e variância(s) (σ^2) ou covariância (Σ), dependendo de quantos atributos você tiver.
3. **Calcular a função de probabilidade para cada ponto do conjunto de treinamento:**
   - No caso univariado, usar a fórmula da Gaussiana 1D.
   - No caso multivariado, usar a equação generalizada.
4. **Definir e ajustar o limiar ϵ:**
   - Analisar os valores de probabilidade calculados.
   - Se possível, usar um conjunto de validação que contenha exemplos já rotulados como anormais para ajudar a definir ϵ.
5. **Aplicar o modelo em dados novos:**
   - Para cada novo ponto $\mathbf{x}_{\text{novo}}$, calcule $p(\mathbf{x}_{\text{novo}})$
   - Se $p(\mathbf{x}_{\text{novo}}) < \epsilon$, rotule como anômalo.

2.8.5 Limitações

- Se os dados "normais" não forem bem representados por uma Gaussiana, o modelo pode não performar bem.
- Em casos de alta dimensionalidade (muitos atributos), a Gaussiana multivariada pode se tornar complexa de ajustar, principalmente pela dificuldade de calcular a matriz de covariância inversa e por exigir muitos dados para estimar parâmetros de forma confiável.

2.8.6 Resumo

A detecção de anomalias por meio de Modelos Gaussianos é uma técnica eficiente e relativamente simples de implementar. O principal está em:

- Ter dados de referência que representem bem o estado "normal".
- Aprender corretamente parâmetros como média, variância e, em casos multivariados, a matriz de covariância.
- Escolher criteriosamente o limiar para decidir o que é anômalo.

Com esse entendimento, você poderá não apenas aplicar Modelos Gaussianos para detecção de anomalias, mas também compreender quando essa abordagem é adequada (ou não) para determinados problemas. Depois de entender e experimentar esses conceitos, você estará mais preparado para avançar em métodos mais sofisticados, como modelos baseados em Redes Neurais, Máquinas de Vetores de Suporte (SVM) para detecção de anomalias, ou até soluções que combinem diferentes abordagens.

Por ora, é essencial compreender o coração do que são e como funcionam os modelos Gaussianos. Trata-se de uma base sólida que surge constantemente em aplicações de aprendizagem de máquina, desde análises preliminares até a etapa final de tomada de decisão.

2.9 Caso Prático: Exemplo de Detecção de Anomalias no Consumo de Energia em Residências

Imagine que estamos monitorando o consumo de energia elétrica em uma residência ao longo do tempo. Em geral, os valores seguem um padrão relativamente estável, variando conforme o dia da semana e o horário. No entanto, picos incomuns podem indicar problemas como aparelhos defeituosos, uso indevido ou vazamentos de corrente. Nosso objetivo é detectar automaticamente essas situações anômalas com base em dados históricos de consumo.

Variáveis observadas (por hora):

- **Consumo total (kWh):** energia total consumida no intervalo de uma hora.
- **Pico de demanda (kW):** valor mais alto de potência registrada na hora.

Vamos supor que essas duas variáveis sigam distribuições normais durante o funcionamento típico da casa.

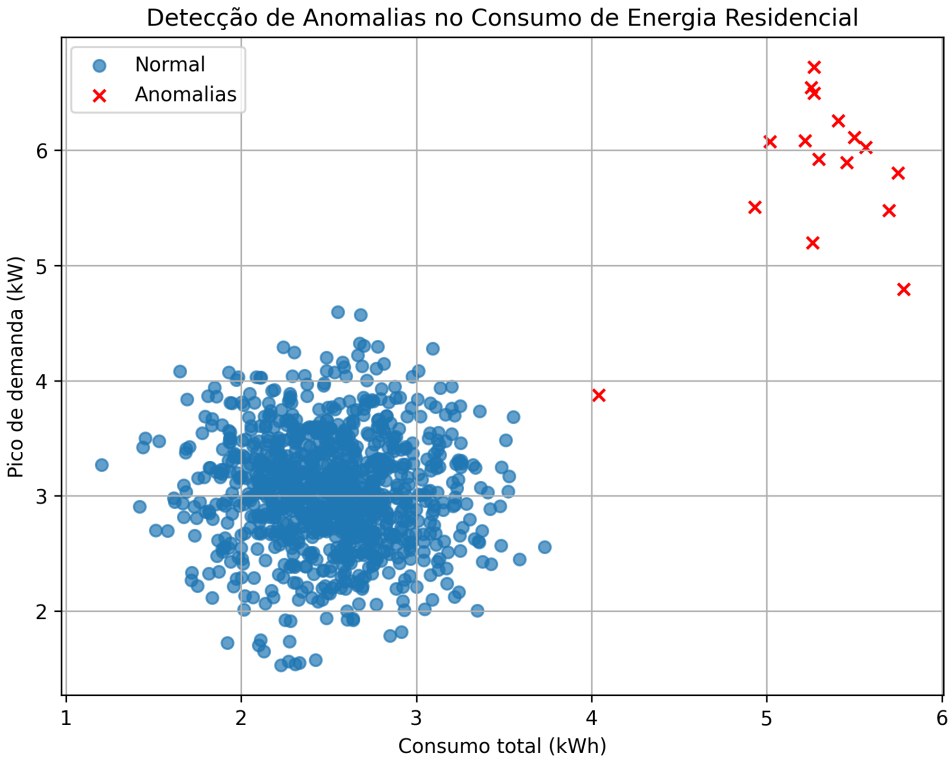
Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt

# Gerando dados normais (comportamento típico da residência)
np.random.seed(42)
# Consumo total em kWh: média 2.5 kWh, desvio padrão 0.4
consumo_normal = np.random.normal(loc=2.5, scale=0.4, size=1000)
# Pico de demanda em kW: média 3.0 kW, desvio padrão 0.5
pico_normal = np.random.normal(loc=3.0, scale=0.5, size=1000)

# Criando dados anômalos (eventos suspeitos)
# Consumo elevado: média 5.5 kWh, desvio padrão 0.3
# Pico de demanda elevado: média 6.0 kW, desvio padrão 0.4
consumo_anomalo = np.random.normal(loc=5.5, scale=0.3, size=15)
pico_anomalo = np.random.normal(loc=6.0, scale=0.4, size=15)

# Unindo os dados
data_normal = np.column_stack((consumo_normal, pico_normal))
data_anomalo = np.column_stack((consumo_anomalo, pico_anomalo))
data_total = np.vstack((data_normal, data_anomalo))

# Estimando média e desvio padrão com base apenas nos dados
   normais
mu_consumo, sigma_consumo = np.mean(consumo_normal), np.std(
   consumo_normal)
mu_pico, sigma_pico = np.mean(pico_normal), np.std(pico_normal)

# Funções de probabilidade
def gaussian_prob(x, mu, sigma):
    return 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-((x - mu)**2)/(2*
   sigma**2))

def calc_joint_prob(x1, x2, mu1, sigma1, mu2, sigma2):
    return gaussian_prob(x1, mu1, sigma1) * gaussian_prob(x2, mu2,
    sigma2)

# Calculando a probabilidade para cada observação
probs = calc_joint_prob(
    data_total[:, 0], # consumo
    data_total[:, 1], # pico
    mu_consumo, sigma_consumo,
    mu_pico, sigma_pico
)

# Definindo limiar
epsilon = 1e-4
anomalias = probs < epsilon
```

```
# Separando os dados
dados_normais = data_total[~anomalias]
dados_anomalias = data_total[anomalias]

# Visualização
plt.figure(figsize=(8, 6))
plt.scatter(dados_normais[:, 0], dados_normais[:, 1], label='
    Normal', alpha=0.7)
plt.scatter(dados_anomalias[:, 0], dados_anomalias[:, 1], color='
    red', label='Anomalias', marker='x')
plt.xlabel('Consumo total (kWh)')
plt.ylabel('Pico de demanda (kW)')
plt.title('Detecção de Anomalias no Consumo de Energia Residencial
    ')
plt.legend()
plt.grid(True)
plt.show()

# Relatório
print("Total de medições:", len(data_total))
print("Total de anomalias detectadas:", np.sum(anomalias))
```

Resultado Esperado

```
Total de medições: 1015
Total de anomalias detectadas: 16
```



O gráfico mostra um plano com duas dimensões: Eixo X: Consumo total de energia na hora (em kWh) e Eixo Y: Pico de demanda registrado na hora (em kW), onde cada ponto representa uma medição de uma hora específica. Os pontos azuis indicam comportamentos considerados normais, ou seja, dentro do padrão histórico da residência. Os pontos vermelhos com ”X”são os que o modelo identificou como anomalias, situações que fogem significativamente do padrão esperado.

Visualmente, podemos perceber que essas anomalias costumam ter valores muito altos de consumo e pico, indicando um possível problema ou evento incomum (como aparelhos funcionando simultaneamente fora do horário usual, por exemplo). Isso significa que, das 1015 horas monitoradas, o modelo identificou 16 medições cujos valores eram tão improváveis segundo a distribuição normal dos dados que foram rotuladas como possíveis anomalias. O objetivo não é afirmar que houve um problema, mas chamar atenção para situações que merecem investigação. Em aplicações reais, essas detecções ajudam equipes técnicas a priorizar inspeções ou enviar alertas aos moradores, evitando desperdícios e aumentando a segurança.

2.10 Sistemas de recomendação

Iniciando o diálogo...

Imagine que você chega a uma livraria virtual e, logo na primeira página, encontra sugestões de livros que parecem ter tudo a ver com seus gostos literários. Ou, em um aplicativo de streaming de música, descobre uma playlist feita sob medida para você. Esses exemplos ilustram o poder dos sistemas de recomendação, que têm como principal objetivo sugerir itens relevantes para usuários, sejam esses itens filmes, músicas, produtos, livros, notícias ou qualquer outro tipo de conteúdo. Segundo Russell e Norvig (2010), sistemas de recomendação são mecanismos inteligentes que buscam maximizar a utilidade percebida pelos usuários. Num mundo com uma quantidade enorme de informação disponível, os sistemas de recomendação atuam como “curadores” digitais. Eles vasculham vastos acervos de dados para apresentar ao usuário aquilo que, segundo o modelo de análise adotado, tem a maior probabilidade de lhe interessar. E, embora por trás dessas recomendações haja algoritmos matemáticos e técnicas de processamento de dados, sua importância e utilidade tornam-se evidentes no dia a dia. Sem esses sistemas, grande parte do conteúdo online permaneceria “escondida”, seria difícil ou demorado descobrir novos itens relevantes sem que algum mecanismo automatizado filtre e ordene as opções.

2.10.1 Por que os Sistemas de Recomendação são Importantes?

- **Redução de Sobrecarga de Informação:** Em ambientes digitais, as opções de escolha podem ser contadas na casa dos milhares ou até milhões. Por exemplo, plataformas de streaming de filmes e séries oferecem catálogos enormes. Diante de tantas possibilidades, decidir o que assistir pode se tornar um desafio. Os sistemas de recomendação aliviam essa “paralisia da escolha”, destacando os itens mais alinhados aos interesses individuais.
- **Melhora na Experiência do Usuário:** Quando as recomendações são bem ajustadas aos gostos de cada pessoa, a interação do usuário com a plataforma se torna mais agradável. Isso não apenas estimula a volta do usuário ao serviço, mas também promove o engajamento contínuo: ele passa a confiar que o sistema oferecerá algo de valor.

- **Personalização:** Este é um pilar dos sistemas de recomendação. Em vez de mostrar o mesmo conteúdo para todos os usuários, esses sistemas aprendem preferências a partir de interações passadas (cliques, visualizações, avaliações, histórico de pesquisa etc.) para entregar sugestões sob medida.
- **Ampliação de Descobertas:** Além de trazer opções "óbvias", como mais filmes de ação para quem curte filmes de ação, os sistemas de recomendação podem ajudar o usuário a explorar novas áreas de interesse. Por exemplo, se você costuma ouvir rock e, de repente, recebe sugestões de bandas de jazz que combinam com o seu perfil, o sistema pode estar incentivando você a descobrir gostos inesperados.

2.10.2 Como os Sistemas de Recomendação funcionam?

Para compreender como funcionam, podemos imaginar que existam, de um lado, **usuários**, pessoas que buscam os produtos, filmes, músicas etc. e, de outro, **itens**, os objetos que queremos recomendar. Em termos mais formais, cada usuário pode ter uma "opinião" ou preferência sobre cada item, a qual frequentemente se representa por meio de avaliações como, por exemplo, notas de 1 a 5 estrelas ou registros de comportamento, como tempo de visualização, cliques, downloads etc. Quando pensamos no conjunto dessas avaliações, podemos dispor os dados em uma matriz de avaliações.

$$\begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{pmatrix}$$

onde:

- m = número de usuários
- n = número de itens
- $r_{u,i}$ = avaliação (explícita ou implícita) do usuário u para o item i

No entanto, na prática, essa matriz costuma ser extremamente esparsa (repleta de "buracos"), pois poucos usuários avaliam todos os itens. Assim, o desafio do sistema de recomendação é prever quais avaliações faltantes podem ser altas, indicando que aquele usuário provavelmente gostará do item.

2.10.3 Tipos de Sistemas de Recomendação

Embora existam diversas abordagens, destacamos algumas categorias principais:

- **Filtragem Colaborativa:** Baseia-se em como diferentes usuários interagem com os itens. A ideia é que, se dois usuários têm gostos semelhantes (por exemplo, deram notas parecidas para vários itens), é provável que um goste de coisas que o outro também gostou. A filtragem colaborativa será explorada em mais detalhes no próximo tópico.
- **Filtragem Baseada em Conteúdo:** Olha para as características ou descrições dos itens. Por exemplo, se um sistema de recomendação de filmes percebe que você frequentemente assiste a comédias românticas com atores específicos, ele vai sugerir outros filmes que tenham descrições (gênero, elenco, sinopse) parecidas.

- **Sistemas Híbridos:** Combinam a filtragem colaborativa e a baseada em conteúdo (ou outras técnicas) para potencializar resultados. Muitas plataformas modernas adotam variações híbridas, pois cada método traz vantagens e limitações específicas. Abordagens de Fatoração Matricial: Ao perceber que a matriz de avaliações pode ser aproximada por fatores latentes como "gosto por ação", "gosto por romance", "gosto por rock" etc., surge a ideia de decompor a matriz em duas ou mais matrizes menores que representem esses fatores. Esse método será detalhado em outro momento, mas é uma das técnicas mais poderosas na filtragem colaborativa.

2.10.4 Dados Explícitos vs. Dados Implícitos

Ao falar em avaliações, podemos ter:

- **Dados Explícitos:** quando o usuário expressa diretamente sua opinião sobre o item, por exemplo, atribuir uma nota de 1 a 5 estrelas.
- **Dados Implícitos:** obtidos de maneira indireta, como cliques, tempo de visualização, quantidade de vezes que um usuário escuta a mesma música, histórico de compras etc.

Em muitos cenários, é mais frequente encontrarmos dados implícitos, por exemplo, nem sempre o usuário clica em um botão de "Curtir", mas seu tempo de permanência assistindo a um vídeo já revela algo sobre seu interesse. Os sistemas de recomendação precisam lidar com esses diferentes tipos de informação para estimar as preferências dos usuários.

2.10.5 Desafios e Cuidados em Sistemas de Recomendação

- **Dados Insuficientes (Cold Start):** Quando um usuário novo entra na plataforma, ainda não há registro de preferências. Da mesma forma, um item recém-adicionado não tem avaliações. Esses casos criam uma situação de "arranque a frio" (*cold start*), em que o sistema tem pouca base para realizar recomendações.
- **Escalabilidade:** Em plataformas populares, temos milhões de usuários e de itens. O processamento necessário para comparar cada usuário com cada item pode se tornar muito custoso. É preciso que os algoritmos sejam eficientes e escaláveis para lidar com grandes volumes de dados.
- **Diversidade e Novidade:** Se o algoritmo apenas sugere mais do mesmo, o usuário pode ficar preso a um conjunto limitado de itens. Em alguns casos, é interessante apresentar itens que, embora não sejam os mais prováveis de agradar, ampliem as possibilidades de descoberta. Essa busca por diversidade e novidade nas recomendações é outro ponto de atenção para o projeto dos sistemas.
- **Questões Éticas e Privacidade:** Para funcionar bem, os sistemas de recomendação frequentemente coletam dados de comportamento dos usuários. É fundamental que esse uso seja transparente e respeite a privacidade. Além disso, recomendações enviesadas podem reforçar estereótipos ou criar bolhas de informação, o que levanta questões éticas sobre a forma como algoritmos influenciam nossas escolhas.

2.10.6 Exemplo Intuitivo

Considere um grupo pequeno de usuários Ana, Bruno e Carlos e alguns filmes, A, B e C. Suponha que eles deram as seguintes notas de 0 a 5 estrelas.

Usuário/Filme	Filme A	Filme B	Filme C
Ana	5	4	?
Bruno	3	?	4
Carlos	?	2	1

Observe que há pontos de interrogação (?) onde o usuário não avaliou o filme. Se, por exemplo, quisermos prever a nota de Ana para o Filme C, podemos analisar os gostos em comum com Bruno e Carlos. Se Ana tem avaliações semelhantes às de Bruno para os filmes A e B, e Bruno deu nota alta para o Filme C, isso sugere que Ana também daria uma nota razoável para o Filme C, é um insight colaborativo.

Em um sistema real, esse processo é feito em larga escala, usando métodos matemáticos como a Fatoração Matricial, ou outras técnicas como média ponderada das notas dos usuários mais similares, sempre com o objetivo de "preencher" as lacunas de forma que cada usuário receba recomendações personalizadas.

2.10.7 Resumo

Os sistemas de recomendação são a espinha dorsal de muitas plataformas populares. Eles ajudam a resolver o problema do excesso de informação, a melhorar a experiência do usuário e a promover descobertas de interesses ocultos. Nesta primeira parte, apresentamos o conceito geral e a relevância desses sistemas no contexto atual, bem como algumas ideias fundamentais sobre como são construídos e quais desafios enfrentam.

Conforme avançarmos, a proposta é equilibrar o formalismo matemático, para entendermos melhor os mecanismos internos dos algoritmos com exemplos e intuições visuais, a fim de tornar o aprendizado mais sólido e relevante. Dessa forma, esperamos que você se familiarize com o funcionamento desses sistemas e possa aplicar conceitos de inteligência artificial em projetos práticos, dentro ou fora da sala de aula.

2.11 Filtragem colaborativa

Iniciando o diálogo...

A recomendação de filmes, músicas, livros e outros produtos em plataformas digitais acontece de forma quase "mágica": basta começar a interagir com um sistema para que ele passe a sugerir itens que combinam com seu gosto. Uma das técnicas que tornam isso possível é a Filtragem Colaborativa. Nosso foco será compreender dois pilares fundamentais da Filtragem Colaborativa: a Matriz Usuário-Item e o cálculo de Similaridade. A primeira nos ajuda a organizar dados sobre como diferentes usuários interagem com diferentes itens. A segunda permite "comparar" usuários entre si, ou itens entre si, usando métricas que expressam quão próximos eles são em termos de preferências ou características. Ao final, estaremos prontos para entender como essa ideia se conecta a algoritmos clássicos de recomendação, como o KNN (*K-Nearest Neighbors*).

2.11.1 O que é a Matriz Usuário-Item?

Imagine que temos um conjunto de usuários, por exemplo, 5 estudantes: Ana, Bruno, Carla, Diogo e Érica, um conjunto de itens, por exemplo, 5 filmes: "Filme A", "Filme B", "Filme C", "Filme D", "Filme E". Uma forma simples de registrar as preferências ou interações desses usuários com os filmes é montar uma tabela em que cada linha representa um usuário e cada coluna representa um filme.

Usuário/Filme	Filme A	Filme B	Filme C	Filme D	Filme E
Ana	5	3	0	4	4
Bruno	4	4	1	3	2
Carla	0	1	5	0	1
Diogo	3	4	2	4	5
Érica	4	2	0	3	4

Cada célula desta matriz pode ser um "score" de avaliação, por exemplo, de 1 a 5 estrelas, a contagem de vezes que o usuário assistiu ao filme ou mesmo um valor binário (0/1) indicando se o usuário consumiu ou não o item. Quando não há informações sobre um item, podemos encontrar uma célula vazia ou com algum código especial, como "NA" ou "?". Em muitos sistemas de recomendação, é comum ter muitos valores vazios, pois cada usuário só interage com uma fração dos itens disponíveis. Essa Matriz Usuário-Item é o ponto de partida para o raciocínio de Filtragem Colaborativa. Ela coleta, de forma condensada, todas as interações ou avaliações que temos de cada usuário em relação a cada item.

2.11.2 Intuição Gráfica

Uma forma de visualizar essa matriz que, no exemplo, tem 5 linhas e 5 colunas é enxergá-la como um plano de pontos ou vetores em um espaço multidimensional. Cada usuário pode ser representado como um vetor de dimensões iguais ao número de itens, no exemplo, cada usuário seria um vetor de 5 dimensões. Ao mesmo tempo, também podemos representar cada item como um vetor de dimensões iguais ao número de usuários, como inversão da matriz. Se fixarmos a perspectiva de usuários como vetores no espaço dos itens, então:

- Ana será o vetor $\mathbf{u}_{\text{Ana}} = (5, 3, 0, 4, 4)$
- Bruno será o vetor $\mathbf{u}_{\text{Bruno}} = (4, 4, 1, 3, 2)$
- ... e assim por diante.

Mesmo sem ser fácil "desenhar" esses pontos num espaço de 5 dimensões, a ideia de usar um espaço matemático para agrupar essas coordenadas ajuda a compreender como dois usuários podem ser mais "próximos", ou seja, mais semelhantes se seus vetores forem parecidos.

2.11.3 Cálculo de Similaridade

Para comparar dois usuários, precisamos de uma métrica de similaridade. Em muitos sistemas, utiliza-se a distância Euclidiana ou o Coeficiente de Correlação de Pearson. No entanto, uma das mais comuns e intuitivas é o Cosseno de Similaridade. O Cosseno de Similaridade entre dois vetores $\mathbf{u}$ e $\mathbf{v}$ é dado por:

$$\text{similaridade}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}$$

onde:

- $\mathbf{u} \cdot \mathbf{v}$ é o **produto escalar** dos dois vetores.
- $||\mathbf{u}||$ é a **norma** do vetor $\mathbf{u}$. Ela mede o "tamanho" do vetor, dada pela raiz quadrada da soma de todos os quadrados das componentes do vetor: $||\mathbf{u}|| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}$.

O valor resultante estará entre -1 e +1. Como normalmente avaliamos preferências, que costumam ser não negativas ou centradas em torno de médias positivas, é frequente que o cosseno resultante fique entre 0 e 1, sendo 1 o caso de vetores exatamente iguais na direção.

A ideia intuitiva é: se dois vetores apontam em direções semelhantes, por exemplo, ambos gostaram dos mesmos filmes com pontuações parecidas, a similaridade pelo cosseno terá valor próximo de 1. Se forem muito diferentes, a similaridade será mais próxima de 0 ou negativa, dependendo dos valores e do tipo de pontuação.

Distância Euclidiana

Outra métrica comum é a distância Euclidiana, definida como:

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \cdots + (u_n - v_n)^2}$$

Nesse caso, quanto menor a distância, mais semelhantes são os usuários ou itens. Apesar de simples, às vezes a distância Euclidiana é afetada por magnitudes das avaliações; por isso, muitas vezes normalizamos os dados antes de usá-la.

Correlação de Pearson

Para dados de preferências, avaliados em estrelas, por exemplo, outro recurso muito útil é a Correlação de Pearson, que mede como duas variáveis variam juntas, independentemente do desvio que cada usuário possa ter em relação à média. Ela é especialmente útil quando cada usuário pode usar escalas de avaliação de forma diferente, por exemplo, alguém que tende a dar pontuações altas para tudo, e outro que tende a avaliar mais rigorosamente.

2.11.4 Por que Precisamos de Similaridade?

A Filtragem Colaborativa baseia-se na hipótese de que usuários semelhantes têm preferências semelhantes. Então, se encontrarmos um grupo de "vizinhos" de Ana (usuários com perfis de avaliação parecidos com o dela), podemos inferir que Ana provavelmente gostará dos itens que esses vizinhos classificaram bem, mesmo que Ana ainda não tenha avaliado ou consumido tais itens.

Da mesma forma, se quisermos recomendar filmes similares a "Filme A", podemos buscar, na Matriz Usuário-Item, itens cuja coluna seja parecida com a coluna de "Filme A". Esses itens similares também devem atrair quem gostou de "Filme A". Na prática, descobrir similares é a base para diversos sistemas de recomendação. Essa etapa acontece com base nos valores da matriz e nas métricas de similaridade que definimos.

2.11.5 Resumo

Até aqui nós vimos:

- **Matriz Usuário-Item**: uma forma de tabular os dados de interação ou avaliação.

- **Vetores de Preferência:** cada usuário ou item pode ser visto como um vetor num espaço em que cada dimensão representa uma categoria de itens ou usuários.
- **Métricas de Similaridade:** o cosseno e a distância Euclidiana são duas maneiras populares de mensurar quão próximos são dois usuários ou dois itens.

No próximo conteúdo, discutiremos como usar essa ideia de similaridade no algoritmo KNN (*K-Nearest Neighbors*) para criar uma Implementação Simples de Filtragem Colaborativa. Você verá como, matematicamente, procuramos os "k" vizinhos mais próximos de um usuário ou item e, a partir disso, geramos recomendações efetivas.

2.12 Implementação Simples com KNN

Iniciando o diálogo...

Aqui nós vamos explorar como o método de *K-Nearest Neighbors (KNN)* pode ser aplicado para criar um sistema de recomendação baseado em Filtragem Colaborativa. A proposta é apresentar o conceito de forma intuitiva, mostrando como ele funciona na prática e discutindo os passos necessários para implementá-lo. Por fim, será fornecida uma visão geral tanto do ponto de vista matemático como de uma perspectiva mais ilustrada, para que o estudante do Ensino Médio possa compreender a lógica fundamental da abordagem.

2.12.1 Visão Geral

O KNN (*K-Nearest Neighbors*, ou "K Vizinhos Mais Próximos") é um método de aprendizado simples que se baseia na ideia de que "exemplos", neste caso, usuários ou itens similares tendem a ter preferências semelhantes. Quando aplicado na Filtragem Colaborativa:

- **K** representa a quantidade de vizinhos que vamos considerar, por exemplo, os usuários mais similares.
- Cada **vizinho** é identificado pela sua semelhança ou distância em relação ao usuário-alvo ou item-alvo, dependendo da abordagem.
- Para fazer a **recomendação** ou prever uma nota (*rating*), levamos em conta as avaliações dadas por esses K vizinhos mais próximos.

Na prática, a implementação de um sistema de recomendação com KNN se desenvolve nos seguintes passos:

1. Coletar dados em uma matriz Usuário-Item, por exemplo, avaliações de filmes, músicas ou produtos.
2. Calcular a similaridade entre o usuário-alvo e todos os outros usuários ou itens.
3. Selecionar os K usuários (ou itens) mais similares de acordo com a medida de similaridade escolhida.
4. Combinar as avaliações desses K vizinhos para estimar a avaliação ou recomendar itens.

2.12.2 Cenário de Recomendação de Filmes - Passo a Passo

Construção da Matriz Usuário-Item

Em um cenário de recomendação de filmes, cada usuário avalia um conjunto de títulos com notas que podem variar, por exemplo, de 1 a 5. Essas notas são organizadas em uma matriz usuário-item, na qual:

- As linhas representam os usuários.
- As colunas representam os itens (filmes).
- Cada célula da matriz contém a nota que um usuário específico deu para um certo filme.

Uma parte simplificada dessa matriz poderia ser assim:

Usuário \ Filme	Filme A	Filme B	Filme C	Filme D
Usuário 1	5	4	0	0
Usuário 2	0	2	5	3
Usuário 3	4	0	4	0
Usuário 4	3	5	5	4

Aqui, "0" representa, por exemplo, a ausência de avaliação, onde o usuário não assistiu ou não avaliou o filme.

Definindo a Similaridade (ou Distância)

Para sabermos se dois usuários são parecidos, calculamos o quão próximas são suas avaliações. Algumas das formas mais comuns de medir essa semelhança incluem:

- **Distância Euclidiana**: Quanto menor a distância entre dois usuários, mais parecidos eles são.
$$d(u,v) = \sqrt{\sum_i (\text{Nota}_{u,i} - \text{Nota}_{v,i})^2}$$
- **Cosseno da Similaridade**: Mede o ângulo entre dois vetores, por exemplo, as avaliações dos usuários. Valores próximos de 1 indicam maior similaridade.
$$\text{similaridade}_{\cos}(u,v) = \frac{\sum_i (\text{Nota}_{u,i} \cdot \text{Nota}_{v,i})}{\sqrt{\sum_i (\text{Nota}_{u,i})^2}\sqrt{\sum_i (\text{Nota}_{v,i})^2}}$$
- **Correlação de Pearson**: Mede a similaridade levando em conta a tendência de cada usuário dar notas altas ou baixas (caso haja variações de perfil).
$$\rho(u,v) = \frac{\sum_i (\text{Nota}_{u,i} - \overline{\text{Nota}}_u)(\text{Nota}_{v,i} - \overline{\text{Nota}}_v)}{\sqrt{\sum_i (\text{Nota}_{u,i} - \overline{\text{Nota}}_u)^2}\sqrt{\sum_i (\text{Nota}_{v,i} - \overline{\text{Nota}}_v)^2}}$$

A escolha da medida de similaridade varia conforme o contexto e a natureza dos dados. A ideia geral, porém, é quanto mais alta a similaridade ou menor a distância, mais "vizinhos" eles são.

Encontrando os K Vizinhos Mais Próximos

Depois de definir a métrica de similaridade/distância, basta:

1. Comparar o usuário-alvo com cada um dos demais usuários para obter todos os valores de similaridade ou distância.
2. Ordenar esses usuários de forma decrescente pela semelhança ou crescente pela distância.
3. Selecionar os K primeiros desta lista, aqueles com maior similaridade ou menor distância.

Se, por exemplo, definirmos K = 3, tomamos os 3 usuários mais próximos do usuário-alvo.

Calculando a Predição de Nota

Para sugerir uma nota e, assim, recomendar ou não um item para o usuário-alvo, reunimos as notas dos K vizinhos apenas para o item que desejamos estimar e fazemos uma média ponderada dessas notas, de acordo com a similaridade de cada um. Assim, vizinhos com maior similaridade têm mais influência na predição. Um modelo simples de média ponderada pode ser:

$$\hat{r}_{u,j} = \frac{\sum_{v \in V} (\text{similaridade}(u, v) \times r_{v,j})}{\sum_{v \in V} \text{similaridade}(u, v)}$$

onde:

- $\hat{r}_{u,j}$ é a predição da nota do usuário u para o item j,
- V é o conjunto dos K vizinhos,
- $r_{v,j}$ é a nota que o vizinho v deu ao item j,
- similaridade(u, v) é o grau de semelhança entre u e v.

Assim, chegamos a uma estimativa de qual nota o usuário daria ao item. Caso essa estimativa seja alta, o item pode ser recomendado.

Caso Prático

Recomendando músicas
Imagine um cenário em que queremos recomendar músicas. Cada pessoa (usuário) pode ser representada como um ponto em um espaço multidimensional, em que cada eixo corresponde a uma preferência musical (por exemplo, “curtiu muito rock”, “curtiu muito pop”, etc.). Um usuário que adora rock e não gosta tanto de pop poderia ficar em uma posição do espaço, enquanto outro que gosta mais de pop estaria em uma posição bem diferente. Para saber se alguém que gosta mais de rock provavelmente gostaria de uma música nova da banda X, basta olhar para pessoas que ocupam posições próximas no “espaço de preferências”, ou seja, as que também gostam muito de rock e pouco de pop e verificar a opinião delas. Se a maioria gostou da música, é bem provável que o usuário-alvo também goste.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA

# 1. Matriz usuário-item com gostos musicais
usuarios = {
    'Alice':   [5, 1, 0, 0],
    'Bruno':   [4, 2, 1, 0],
    'Carla':   [1, 5, 1, 2],
    'Daniela': [0, 4, 3, 3],
    'Eduardo': [1, 0, 4, 5],
    'Fábio':   [2, 1, 4, 4],
    'Usuário-Alvo': [4, 0, 0, 0]
}
nomes = list(usuarios.keys())
matriz = np.array(list(usuarios.values()))

# 2. Calcular similaridades do usuário-alvo com os demais
similaridades = cosine_similarity([matriz[-1]], matriz[:-1])[0]
# 3. Selecionar os K vizinhos mais próximos
K = 3
indices_vizinhos = np.argsort(similaridades)[-K:][::-1]

# 4. Supostas avaliações dos usuários para uma nova música
avaliacoes_nova_musica = np.array([5, 4, 2, 3, 1, 2]) # sem o
   usuário-alvo
avaliacoes_vizinhos = avaliacoes_nova_musica[indices_vizinhos]
similaridades_vizinhos = similaridades[indices_vizinhos]

# 5. Prever a nota do usuário-alvo
nota_prevista = np.dot(avaliacoes_vizinhos, similaridades_vizinhos
   ) / sum(similaridades_vizinhos)

# 6. Redução para 2D com PCA para visualização
pca = PCA(n_components=2)
matriz_2d = pca.fit_transform(matriz)

# 7. Plotar gráfico com destaque dos vizinhos mais próximos
plt.figure(figsize=(10, 6))
for i, nome in enumerate(nomes):
    x, y = matriz_2d[i]
    if i == len(nomes) - 1:
        plt.scatter(x, y, color='red', label='Usuário-Alvo', s
   =100)
    elif i in indices_vizinhos:
        plt.scatter(x, y, color='green', label='Vizinho Próximo'
   if 'Vizinho Próximo' not in plt.gca().get_legend_handles_labels
```

```
    ()[1] else "", s=80)
     else:
         plt.scatter(x, y, color='blue', label='Outros Usuários' if
     'Outros Usuários' not in plt.gca().get_legend_handles_labels()
    [1] else "", s=60)
     plt.text(x + 0.1, y, nome, fontsize=9)

plt.title(f"Espaço de Preferências Musicais (Nota Prevista: {
    nota_prevista:.2f})")
plt.xlabel("Componente Principal 1")
plt.ylabel("Componente Principal 2")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

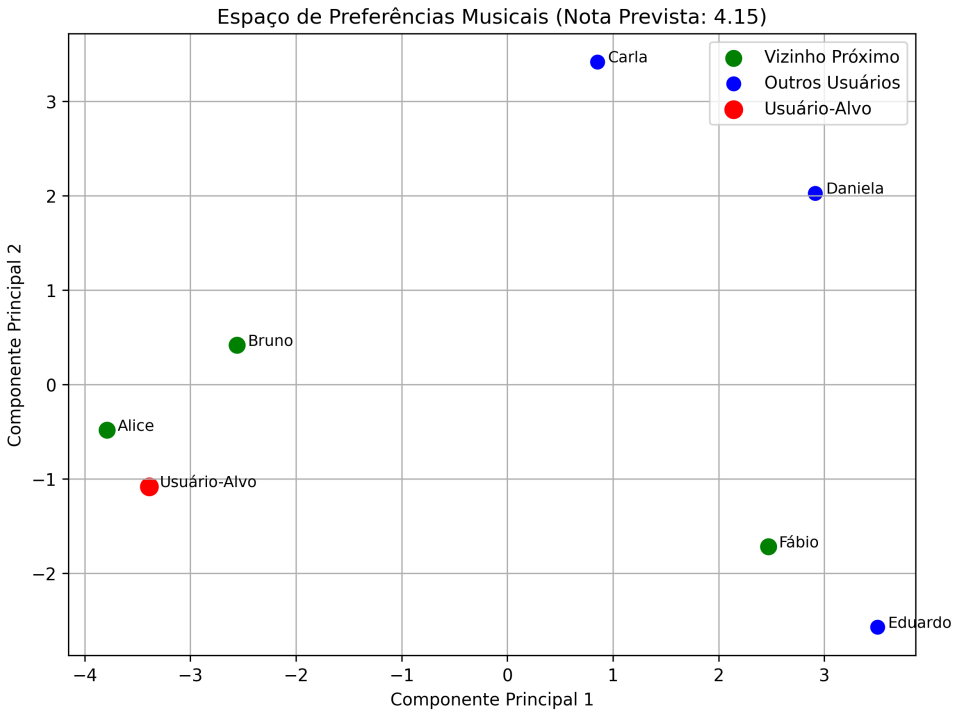


Figura 2.7: Gráfico ilustrando a distribuição das preferências musicais.

Além disso, o gráfico gerado acima mostra os usuários posicionados em um espaço 2D de preferências musicais, criado com PCA (Análise de Componentes Principais) para facilitar a visualização.

- O Usuário-Alvo aparece em vermelho.
- Os usuários mais próximos (vizinhos) aparecem em verde.
- É possível ver que o Usuário-Alvo está mais próximo de pessoas como Alice e Bruno, o que ajuda a justificar a previsão feita.

2.12.3 Resumo

A aplicação do método KNN em sistemas de recomendação é uma forma introdutória de Filtragem Colaborativa, porém bastante didática e intuitiva. Ela nos permite entender os conceitos de similaridade, de próximos vizinhos e de como informações de usuários "parecidos" podem guiar recomendações personalizadas. Com esta base, é possível avançar para métodos mais sofisticados, como filtros baseados em modelos de fatoração de matrizes ou técnicas híbridas. Contudo, o KNN permanece como um dos passos iniciais mais importantes para quem está começando a estudar IA e deseja criar sistemas de recomendação, pois deixa claro o papel das métricas de distância/similaridade e da manipulação de dados na matriz usuário-item.

2.13 Fatoração Matricial

Iniciando o diálogo...

Quando trabalhamos com dados em formato de matriz, por exemplo, notas que estudantes dão a filmes, ou a intensidade de pixels em uma imagem digital, muitas vezes essa matriz é apenas a "superfície" do problema. Por trás desses valores existem características "escondidas" que não aparecem diretamente, mas que influenciam fortemente o comportamento dos dados. Essas características escondidas são chamadas de fatores latentes ou representações latentes.
Imagine que temos uma matriz onde as linhas representam pessoas e as colunas representam filmes, com cada célula indicando a nota que cada pessoa deu a cada filme. Por trás dessas notas, há diversos fatores que influenciam as preferências de cada pessoa: gosto por ação, preferência por filmes de comédia, apreciadores de animação e assim por diante. Essas dimensões, ou fatores, não aparecem explicitamente na matriz de notas, mas são fundamentais para explicar por que diferentes pessoas atribuem diferentes avaliações aos filmes. Ideia: Se conseguíssemos "descobrir" ou "extrair" essas características escondidas e representá-las numericamente, poderíamos descrever cada pessoa, ou cada filme, com base nessas dimensões de gosto. Esse é o cerne do que chamamos de representação latente.

2.13.1 Modelando dados em um espaço de fatores latentes

Uma forma de pensar matematicamente nas representações latentes é supor que cada objeto (pessoa, filme, produto, item de uma base de dados etc.) tenha um conjunto de "pesos" que indicam como ele se distribui ao longo de cada fator. Por exemplo, se definirmos que existem k fatores latentes, podemos associar:

- Um vetor $\mathbf{u}_i \in \mathbb{R}^k$ a cada usuário ou linha i, no caso do nosso exemplo de recomendação de filmes.
- Um vetor $\mathbf{v}_j \in \mathbb{R}^k$ a cada item ou coluna j, no exemplo, cada filme.

Seja M_{ij} o valor que está na célula (i, j) da matriz original, por exemplo, a nota do usuário i para o filme j. Queremos aproximar M_{ij} por meio de uma combinação das características latentes de $\mathbf{u}_i$ e $\mathbf{v}_j$. A forma mais simples de aproximação é:

$$\hat{M}_{ij} = \mathbf{u}_i \cdot \mathbf{v}_j = \sum_{r=1}^{k} u_{i,r} v_{j,r}$$

onde:

- $\hat{M}_{ij}$ é o valor estimado (ou predito) para a célula (i, j).
- $\mathbf{u}_i \cdot \mathbf{v}_j$ representa o produto interno (ou produto escalar) entre os dois vetores.

Nessa representação, cada dimensão r (de 1 a k) corresponde a um fator latente. Por exemplo, se $k = 2$ num caso simples de recomendação de filmes, poderíamos imaginar que o primeiro fator representa "preferência por ação" e o segundo representa "preferência por comédia". Assim, um usuário poderia ter um valor grande na dimensão "ação" e pequeno na "comédia", enquanto outro usuário poderia ter o oposto, e isso se refletiria nos respectivos vetores $\mathbf{u}_i$.

2.13.2 O problema de minimização de erro

Para que as predições $\hat{M}_{ij}$ sejam úteis, queremos que elas fiquem o mais próximo possível dos valores originais M_{ij} observados na matriz. Em problemas práticos, muitos elementos de M podem até estar faltando (por exemplo, em sistemas de recomendação, um usuário não dá nota a todos os filmes), mas a ideia central permanece: ajustar $\mathbf{u}_i$ e $\mathbf{v}_j$ de modo que a diferença entre M_{ij} e $\hat{M}_{ij}$ seja mínima.

Função de custo (erro quadrático)

Uma forma bem comum de mensurar quão boa é a aproximação é novamente usar o erro quadrático médio. Suponha que temos um conjunto de pares (i, j) para os quais os valores M_{ij} são conhecidos. Definimos a função de custo:

$$J(\mathbf{U}, \mathbf{V}) = \sum_{(i,j)\in\Omega} (M_{ij} - \mathbf{u}_i \cdot \mathbf{v}_j)^2$$

onde:

- Ω é o conjunto de todas as posições (i, j) da matriz para as quais os valores são conhecidos (ou seja, não faltantes).
- $\mathbf{U}$ representa todos os vetores de usuários $\mathbf{u}_i$.
- $\mathbf{V}$ representa todos os vetores de itens $\mathbf{v}_j$.

O objetivo é encontrar $\mathbf{U}$ e $\mathbf{V}$ que **minimizem** $J(\mathbf{U}, \mathbf{V})$. Na prática, também se adiciona uma penalização, chamada "regularização", para controlar a complexidade dos vetores e evitar que fiquem "exagerados" e causem *overfitting*. Mas a ideia geral continua: queremos que a diferença entre M_{ij} e $\hat{M}_{ij}$ seja a menor possível.

2.13.3 Por que fatores latentes ajudam a interpretar os dados?

É útil pensar nos fatores latentes como dimensões de um espaço geométrico. Se escolhemos $k = 2$, por exemplo, estamos projetando cada usuário e cada item num plano bidimensional. Assim, podemos imaginar cada usuário como um ponto nesse plano e cada item como outro ponto, de modo que a "semelhança" entre usuário e item seja dada pela distância ou pelo produto interno entre seus vetores.

2.13.4 Interpretação e clusterização

Quando os vetores $\mathbf{u}_i$ são bem ajustados, usuários com gostos parecidos ficam próximos uns dos outros nesse espaço, e o mesmo vale para itens semelhantes, filmes do mesmo gênero, músicas do mesmo estilo etc. Esse agrupamento (*clusterização*) surge naturalmente, mesmo que não tenhamos indicado explicitamente um "rótulo" de gênero ou classificação. É como se o próprio processo de minimização do erro descobrisse:

- Quais filmes são mais parecidos (pois terão vetores $\mathbf{v}_j$ mais semelhantes).
- Quais pessoas têm preferências similares (pois terão vetores $\mathbf{u}_i$ mais próximos).
- E, adicionalmente, como relacionar uma certa pessoa a um determinado filme (por meio do produto interno $\mathbf{u}_i \cdot \mathbf{v}_j$).

2.13.5 Aplicações práticas

A aplicação mais conhecida de fatoração matricial com representações latentes é nos **sistemas de recomendação**. Plataformas de filmes, músicas ou lojas online utilizam técnicas de fatoração matricial para prever as notas que cada usuário daria a cada item não avaliado. Assim, podem recomendar produtos ou conteúdos que provavelmente agradariam a cada usuário.

Em **processamento de imagens**, também podemos ver uma imagem como uma matriz, ou seja, pixels em tons de cinza ou canais de cor. Alguns métodos de compressão decompondo a imagem em fatores que representam padrões locais ou globais. Esses padrões latentes são responsáveis por agrupar regiões semelhantes da imagem e permitem que o arquivo fique menor sem perder muita qualidade. Outro exemplo, é se cada linha for um documento, ou uma frase, e cada coluna for uma palavra, podemos ter uma matriz com o número de ocorrências de cada palavra em cada documento. Fatores latentes podem então representar tópicos ou temas presentes nos textos, ajudando a **análise de textos** ao agrupar documentos com assuntos similares.

2.13.6 O que fazer a seguir?

Em técnicas mais avançadas, especialmente em redes neurais, também se buscam representações latentes, mas de uma forma mais complexa, camadas intermediárias de uma rede, por exemplo. A fatoração matricial é um ponto de partida muito instrutivo para entender como essas "representações internas" podem ser úteis na prática. E aprender a extrair fatores latentes é um passo essencial em muitos sistemas de IA, pois a capacidade de "descobrir" esses fatores equivale a aprender características importantes dos dados de forma automática.

2.13.7 Resumo

A ideia de representações latentes está no cerne de vários problemas de IA. Sempre que temos uma matriz de dados, podemos nos perguntar se há fatores ou dimensões "ocultas" que explicam por que essa matriz se comporta de certa forma. A minimização de erro surge como a maneira prática de fazer com que essas dimensões, armazenadas em $\mathbf{u}_i$ e $\mathbf{v}_j$ aproximem com qualidade os valores originais M_{ij}. Apesar de o raciocínio matemático envolver produtos internos, somas de quadrados e possíveis técnicas de regularização, a intuição é simples: descobrir padrões básicos que expliquem como os dados foram gerados.

Embora pareça "mágico" descobrir fatores latentes capazes de explicar os dados, essa mágica nada mais é do que o resultado de um bom modelo matemático e de um procedimento de otimização robusto. Com exercícios e exemplos práticos, esse processo de descoberta de padrões internos se torna claro e bastante intuitivo, mesmo para quem está começando seus estudos na área.

Fique Alerta!

Fatores latentes são dimensões escondidas que influenciam os valores observados em uma matriz.

2.14 Algoritmo de ALS

A fatoração matricial é uma técnica muito utilizada em sistemas de recomendação, análise de dados e compressão de informações. A ideia central é representar grandes conjuntos de dados na forma de uma matriz R, por exemplo, registros de usuários (linhas) versus produtos ou itens (colunas), de modo que cada célula $R_{u,i}$ guarde informações como avaliações, quantidades ou preferências.

Essas matrizes costumam ser muito grandes e, muitas vezes, esparsas, cheias de células vazias ou desconhecidas. Assim, encontrar padrões diretos pode ser uma tarefa difícil. Para simplificar, recorre-se à ideia de "resumir" cada usuário e cada item em alguns poucos números chamados de fatores latentes. Se tivermos uma matriz R de dimensões $m \times n$ (sendo m o número de usuários e n o número de itens), podemos tentar aproximá-la pelo produto de duas matrizes de menor dimensão:

$$R \approx P \times Q^T$$

onde:

- P é uma matriz de tamanho $m \times k$ (que chamaremos de "matriz de usuários"),
- Q é uma matriz de tamanho $n \times k$ (que chamaremos de "matriz de itens"),
- k é a quantidade de fatores latentes que a gente escolhe.

Cada usuário u agora passa a ser descrito por um "vetor de preferências" de tamanho k, que é uma linha em P. Já cada item i terá um "vetor de características" de tamanho k, que é uma linha em Q. Assim, o $P_u \cdot Q_i^T$ (um produto interno entre o vetor do usuário e o vetor do item) deve se aproximar de $R_{u,i}$, a avaliação real ou outro tipo de dado.

Entretanto, como ajustar esses vetores de forma a minimizar os erros de predição? É nesse contexto que entra o Algoritmo de ALS (*Alternating Least Squares*), que propõe uma estratégia em que ajustamos alternadamente os parâmetros dos usuários e dos itens até chegarmos a uma solução satisfatória.

2.14.1 Alternância

Para compreender por que alternar ajuda, considere um exemplo simples em que temos uma expressão geral para o erro, do tipo:

$$J(P, Q) = \sum_{(u,i)\in\Omega} (R_{u,i} - P_u \cdot Q_i^T)^2$$

onde Ω é o conjunto de pares (u, i) para os quais a avaliação $R_{u,i}$ é conhecida. O grande desafio está no fato de que tanto P como Q aparecem ao mesmo tempo no produto interno $P_u \cdot Q_i^T$. A busca simultânea pelo P ideal e o Q ideal pode ser muito complexa. Então, o Algoritmo de ALS faz o seguinte:

1. Fixa temporariamente os valores de Q, como se já estivessem dados.
2. Ajusta P minimizando o erro J somente em relação aos vetores de P.
3. Em seguida, fixa os valores de P.
4. Ajusta Q, minimizando o mesmo erro, mas agora somente em relação aos vetores de Q.
5. Repete esse processo de forma iterativa, alternando entre ajustes em P e ajustes em Q. Por isso, chama-se *Alternating* (alternado). O "*Least Squares*" (mínimos quadrados) aparece porque, para cada passo, resolvemos um problema de regressão linear minimizando o quadrado do erro de predição.

2.14.2 Passo a Passo do ALS

Inicialização

- **Estimativa inicial de P e Q:** Podemos começar com valores aleatórios pequenos para cada vetor de fator. Por exemplo, inicializar cada elemento de P e Q com valores próximos de zero, de preferência, distribuições aleatórias normais ou uniformes.
- **Parâmetros de controle:** Definimos um número máximo de iterações, chamado de n_{iter} ou outro critério de parada, como uma certa tolerância para o erro. Também escolhemos a dimensão k, isto é, quantos fatores latentes vamos utilizar.

Otimização de P

Com Q fixo, cada linha P_u do P pode ser ajustada através de um problema de regressão linear clássico. Para cada usuário u, ele tem avaliações conhecidas $R_{u,i}$ para alguns itens $i \in I(u)$. Nosso objetivo passa a ser:

$$\min_{P_u} \sum_{i \in I(u)} (R_{u,i} - P_u \cdot Q_i^T)^2 + \lambda ||P_u||^2$$

onde λ é um fator de regularização que ajuda a controlar a magnitude dos vetores. Como se trata de um problema quadrático em P_u (com Q fixo), é possível resolvê-lo com métodos de álgebra linear. Na prática, cada linha P_u pode ser obtida resolvendo um sistema do tipo:

$$P_u \left(\sum_{i \in I(u)} Q_i^T Q_i + \lambda I \right) = \sum_{i \in I(u)} R_{u,i} Q_i$$

Aqui, I é a matriz identidade de dimensão $k \times k$. Note que, para cada usuário, temos um pequeno sistema linear de dimensão k, e como k costuma ser bem menor do que n, isso é computacionalmente viável.

Otimização de Q

Mantemos o P recém-atualizado e, então, repetimos o processo para o Q. Agora, cada linha Q_i é ajustada individualmente resolvendo outro sistema de equações lineares:

$$\min_{Q_i} \sum_{u \in U(i)} (R_{u,i} - P_u \cdot Q_i^T)^2 + \lambda ||Q_i||^2$$

em que $U(i)$ é o conjunto de usuários que avaliaram o item i. Assim, segue-se:

$$Q_i \left(\sum_{u \in U(i)} P_u^T P_u + \lambda I \right) = \sum_{u \in U(i)} R_{u,i} P_u$$

Ao resolver esse sistema, atualizamos a linha Q_i para cada item, garantindo uma aproximação melhor a cada iteração.

Repetição e Critério de Parada

- **Iteração**: Repetimos o par "atualizar $P \rightarrow$ atualizar Q" diversas vezes, e o erro total tende a diminuir.
- **Convergência:** Observa-se que, após um certo número de iterações, as mudanças em P e Q passam a ser mínimas ou o erro não melhora de forma significativa. Esse é o momento de parar o algoritmo.

2.14.3 Entendendo o ALS de um jeito simples

Uma forma legal de entender como o algoritmo ALS funciona é imaginar uma situação parecida com um jogo de "acertar o ponto mais baixo de um terreno". Pense que temos dois grupos: usuários e itens como filmes, músicas e produtos. Cada um deles tem um conjunto de números que representam suas preferências ou características. O que o ALS faz é o seguinte, primeiro, ele fixa as informações dos itens e tenta ajustar os dados dos usuários para que combinem melhor, depois, ele faz o contrário: fixa os dados dos usuários e ajusta os itens e assim vai, alternando entre um e outro, como uma dança.

Visualmente, imagine uma montanha ou uma colina. Queremos achar o ponto mais baixo, onde o erro é o menor possível. Mas, em vez de descer em qualquer direção, só podemos andar primeiro para frente ou para trás, ajustando os usuários, e depois para os lados, ajustando os itens. A cada passo, chegamos um pouco mais perto do ponto mais baixo. Daí esse processo vai se repetindo até que os dois lados, usuários e itens, estejam bem ajustados e a gente tenha uma boa previsão das preferências.

2.14.4 Aplicando seus conhecimentos

Neste exercício, será implementado um algoritmo de fatoração de matrizes utilizando o método ALS. Considerando uma matriz de avaliações R, onde as linhas representam usuários e as colunas representam itens, o algoritmo vai alternar entre a atualização das matrizes de usuários (U) e itens (I). O objetivo é preencher os valores ausentes da matriz de avaliações com uma aproximação das interações reais entre usuários e itens.

O código a seguir apresenta a implementação do ALS, com uma matriz de avaliação inicial que contém alguns valores ausentes. Através da decomposição dessa matriz em dois

fatores latentes, podemos aproximar as avaliações faltantes, realizando uma recomendação baseada nas interações já conhecidas. O treinamento é realizado por várias iterações até que as matrizes convirjam para uma solução que melhor represente as preferências dos usuários em relação aos itens disponíveis.

Copie e Teste!

```
import numpy as np

# 1. Matriz R (4 usuários x 3 itens), com np.nan indicando valores
    ausentes
R = np.array([
    [5, 3, np.nan],
    [4, np.nan, np.nan],
    [1, 1, np.nan],
    [np.nan, 2, 5]
])

num_users, num_items = R.shape
num_factors = 2  # Número de fatores latentes

# 2. Inicialização aleatória das matrizes U e I
np.random.seed(42)
U = np.random.rand(num_users, num_factors)  # matriz de usuários
   (4x2)
I = np.random.rand(num_items, num_factors)  # matriz de itens (3x2
   )

print("Matriz R (original):\n", R)
print("\nMatriz U (inicial):\n", U)
print("\nMatriz I (inicial):\n", I)

# 3. Número de iterações de treinamento
num_iterations = 100  # Definindo o número de iterações de
   treinamento
for iteration in range(num_iterations):
    # Atualiza U
    for u in range(num_users):
        rated_items = ~np.isnan(R[u, :])
        I_rated = I[rated_items]
        R_rated = R[u, rated_items]

        if len(R_rated) == 0:
            continue  # Usuário não avaliou nada

        A = I_rated.T @ I_rated
        V = I_rated.T @ R_rated
        U[u], _, _, _ = np.linalg.lstsq(A, V, rcond=None)

    # Atualiza I
```

```
    for i in range(num_items):
        rated_by = ~np.isnan(R[:, i])
        U_rated = U[rated_by]
        R_rated = R[rated_by, i]

        if len(R_rated) == 0:
        continue  # Item não foi avaliado

        A = U_rated.T @ U_rated
        V = U_rated.T @ R_rated
        I[i], _, _, _ = np.linalg.lstsq(A, V, rcond=None)

# 4. Resultado final após as iterações
print("\nMatriz U final (após treinamento):\n", U)
print("\nMatriz I final (após treinamento):\n", I)

# 5. Reconstrução de R com as matrizes U e I
R_reconstructed = U @ I.T
print("\nMatriz R reconstruída (aproximada):\n", R_reconstructed)
```

Resultado Esperado

```
Matriz R (original):
[[ 5.  3. nan]
[ 4. nan nan]
[ 1.  1. nan]
[nan  2.  5.]]

Matriz U (inicial):
[[0.37454012 0.95071431]
[0.73199394 0.59865848]
[0.15601864 0.15599452]
[0.05808361 0.86617615]]

Matriz I (inicial):
[[0.60111501 0.70807258]
[0.02058449 0.96990985]
[0.83244264 0.21233911]]

Matriz U final (após treinamento):
[[4.79430323 2.99132098]
[2.78711398 3.28303061]
[0.46061453 1.02124799]
[5.51026396 1.94510242]]

Matriz I final (após treinamento):
[[0.60111501 0.70807258]
[0.02058449 0.96990985]
```

```
[0.80685793 0.28481781]]

Matriz R reconstruída (aproximada):
[[5.         3.         4.72030308]
[4.         3.24161506 3.18387062]
[1.         1.         0.6625201 ]
[4.68957607 2.         5.        ]]
```

Este exercício não só ilustra a aplicação prática do algoritmo de fatoração de matrizes, mas também proporciona uma base sólida para entender como sistemas de recomendação modernos são desenvolvidos e ajustados. A seguir, responda as perguntas:

1. Qual seria o impacto de aumentar o número de fatores latentes na variável `num_factors` no desempenho do modelo? Experimente mudar no código e descubra.

2. O que aconteceria se você alterasse o número de iterações `num_iterations` para um valor muito baixo? Como isso afetaria os resultados?

3. Você observou alguma melhoria significativa nas atualizações das matrizes U e I ao longo das iterações? Como você sabe se o modelo está convergindo corretamente?

2.14.5 Resumo

O Algoritmo de ALS é uma peça fundamental no arsenal de técnicas de fatoração matricial, principalmente porque ele oferece uma maneira ordenada e relativamente simples de otimizar fatores latentes para grandes bases de dados. A alternância entre ajustar matrizes de usuários e de itens torna a tarefa mais "quebrável" em etapas, cada uma resolvível com métodos conhecidos de mínimos quadrados.

Compreender a lógica por trás do ALS ajuda a ver como sistemas de recomendação podem personalizar resultados, mesmo em meio a muitos usuários e muitos itens, usando apenas um "pouco" de informação de cada um. As aplicações são várias, indo desde recomendação de filmes, músicas, livros, até soluções de predição em bancos de dados incompletos.

Este método não apenas exemplifica a força de conceitos básicos como regressão linear e projeções em espaços de menor dimensão, mas também mostra como a alternância e a regularização podem ser estratégias inteligentes para lidar com problemas complexos, encontrando um caminho eficiente rumo a soluções robustas e interpretáveis.

2.15 Sistemas Híbridos

Iniciando o diálogo...

Neste conteúdo, vamos explorar como diferentes abordagens podem ser combinadas em sistemas de recomendação, como a filtragem colaborativa e a análise baseada em conteúdo, também chamada de conteúdo-perfil. Também destacaremos a importância de considerar a escalabilidade desses sistemas, especialmente quando lidamos com grandes volumes de dados.

2.15.1 A Motivação para Sistemas Híbridos

Os sistemas de recomendação podem se basear em filtragem colaborativa ou filtragem baseada em conteúdo. Na filtragem colaborativa, procura-se identificar padrões de preferências de grupos de usuários, assumindo que pessoas com gostos parecidos irão gostar de itens semelhantes. Já na abordagem de conteúdo, a ideia é analisar as características do item em si, por exemplo, gênero de um filme, autor de um livro, ingredientes de uma receita e relacioná-las aos interesses do usuário.

No entanto, cada técnica possui suas limitações. A filtragem colaborativa tende a apresentar dificuldades quando há poucos dados disponíveis (o chamado *cold start*), especialmente com novos usuários ou itens recém-adicionados. Já a filtragem baseada em conteúdo pode não refletir adequadamente as preferências coletivas, pois depende fortemente das características e metadados dos itens, o que pode limitar a diversidade das recomendações.

Para superar esses obstáculos, muitos sistemas de recomendação modernos utilizam abordagens híbridas. Um sistema híbrido combina elementos de filtragem colaborativa e filtragem baseada em conteúdo, explorando o melhor de cada método. Em escala real, sobretudo quando lidamos com grandes plataformas, por exemplo, serviços de streaming, grandes lojas virtuais e redes sociais, também surgem questões de escalabilidade: como processar milhões de usuários e itens ao mesmo tempo, mantendo a qualidade das recomendações e o desempenho do sistema?

2.15.2 Combinação de Técnicas e Modelos

A combinação entre diferentes algoritmos de recomendação pode ocorrer de diversas formas. Duas das estratégias mais comuns são:

- **Ponderação (Weighting):** Nesse modelo, cada sistema seja colaborativo, seja baseado em conteúdo, produz uma pontuação de relevância para cada item. Em seguida, o sistema híbrido gera uma recomendação final combinando essas pontuações, geralmente aplicando um peso a cada técnica. Por exemplo:

  $$\text{Pontuação\_Híbrida}(u,i) = \alpha \cdot \text{Pontuação\_Colaborativa}(u,i) + (1-\alpha) \cdot \text{Pontuação\_Conteúdo}(u,i)$$

  em que α é um parâmetro entre 0 e 1 que controla o peso dado a cada abordagem. A ideia é ajustar α para equilibrar a influência de cada método.

- **Regras de Gatilho (Switching ou Mixed):** Aqui, o sistema decide qual técnica de recomendação usar dependendo de certas condições. Por exemplo, se o item é novo no catálogo e não possui avaliações de usuários (dados colaborativos insuficientes), o sistema ativa a recomendação baseada em conteúdo. Se, por outro lado, já há dados suficientes de interações (curtidas, visualizações, avaliações), a filtragem colaborativa toma a dianteira.

2.15.3 Fatoração Matricial como Base de Combinação

A fatoração matricial é uma das técnicas mais eficazes de filtragem colaborativa. A ideia é decompor uma grande matriz de notas ou avaliações em duas matrizes menores:

$$R \approx P \times Q^T$$

onde:

- R é a matriz original de dimensões $M \times N$, em que M é a quantidade de usuários e N é a quantidade de itens.
- P é uma matriz de dimensões $M \times k$, que representa a "preferência" de cada usuário em relação a fatores latentes.
- Q é uma matriz de dimensões $N \times k$, que representa a "caracterização" de cada item em relação a esses mesmos fatores latentes.

Cada linha de P (associada a um usuário) e cada linha de Q (associada a um item) podem ser combinadas para prever a afinidade entre aquele usuário e aquele item. Na prática, é comum que, após treinar o modelo, tenhamos algo como:

$$\hat{r}_{u,i} = p_u \cdot q_i^T$$

onde $\hat{r}_{u,i}$ é a nota estimada do usuário u para o item i. A curva de aprendizado desse processo envolve técnicas como *gradient descent* (descida de gradiente) e regularização, que ajudam a ajustar os parâmetros de P e Q para que a aproximação de R seja a melhor possível.

Em um sistema híbrido, podemos enriquecer esse modelo com atributos de conteúdo, por exemplo, adicionando informações de metadados do item, como gênero, ano, palavras-chave diretamente na representação do item. Isso pode acontecer de várias maneiras, como inserir esses atributos no vetor de fatores de cada item (expandindo q_i) ou criando uma função que combine a pontuação de fatoração matricial com a similaridade de conteúdo.

2.15.4 Exemplificando

Uma forma intuitiva de entender sistemas híbridos é imaginá-los como uma balança de dois pratos. Em um dos lados está a filtragem colaborativa, no outro, a filtragem baseada em conteúdo. Ao construir um sistema híbrido, ajustamos o "peso" de cada abordagem, buscando um ponto de equilíbrio que produza recomendações mais precisas e relevantes.

Esse equilíbrio, no entanto, não é fixo, ele pode mudar com o tempo. Por exemplo, se começarmos a receber muitas novas avaliações de usuários, o lado colaborativo ganha mais peso. Já se os dados sobre os itens forem mais ricos e detalhados, o lado de conteúdo pode se tornar mais influente. O segredo de um bom sistema híbrido está justamente em saber como e quando ajustar essa balança.

Capítulo 3

Aprendizagem por Reforço

Iniciando o diálogo...

A Aprendizagem por Reforço (*Reinforcement Learning*) é um dos ramos mais empolgantes da área de Inteligência Artificial, segundo Russell e Norvig (2010), baseia-se na interação de agentes com ambientes para maximizar recompensas cumulativas. Sua proposta central é ensinar um agente, que pode ser um programa de computador, um robô ou até mesmo um personagem virtual em um jogo a tomar decisões de forma autônoma, aprendendo a partir de interações com o ambiente. Diferentemente de outros métodos de aprendizagem que dependem de dados já rotulados ou de padrões específicos, a Aprendizagem por Reforço destaca-se por trabalhar com a ideia de **tentativa e erro** e pela busca constante de um objetivo, representado por um sinal de **recompensa**.

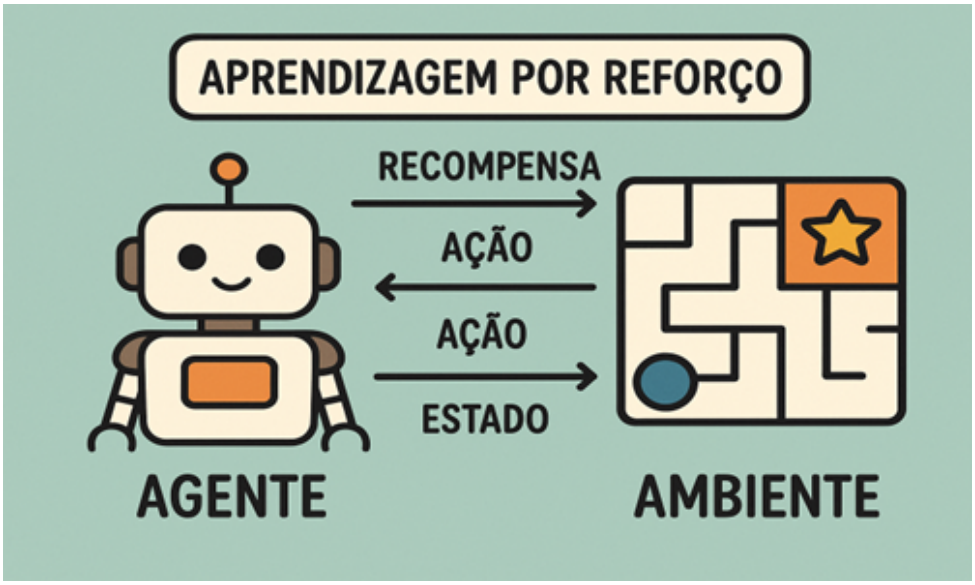


Figura 3.1: Imagem ilustrativa criada com o DALL·E sobre o conceito de Aprendizagem por Reforço, onde um agente robô interage com um ambiente por meio de ações. O agente observa o estado do ambiente, executa uma ação e recebe uma recompensa como feedback, gerado em 15/04/2025.

Para entender a motivação por trás desse tipo de aprendizado, imagine um robô que precisa aprender a andar em linha reta sem cair. Na fase inicial, é provável que ele "tropece"

diversas vezes, mas, com cada tentativa, o robô recebe uma avaliação, seja positiva ou negativa, sobre o seu desempenho. Essa avaliação é traduzida em recompensas, que podem ser pontos acumulados. Ao longo do tempo, ele descobre quais movimentos geram mais recompensas e, gradualmente, aprende a andar de forma mais equilibrada. Assim, a Aprendizagem por Reforço imita um pouco a forma como nós, seres humanos, aprendemos novas habilidades: experimentando, recebendo feedback e ajustando nosso comportamento.

3.1 Diferenças a outros paradigmas de Aprendizagem

No campo da Ciência de Dados, há três grandes paradigmas de Aprendizagem de Máquina (*Machine Learning*): Supervisionada, Não Supervisionada e Por Reforço, onde cada uma atende a objetivos e cenários distintos:

- **Aprendizagem Supervisionada:** O agente (ou algoritmo) aprende a partir de exemplos rotulados, ou seja, dados que já vêm com a resposta correta. É como ter um professor que corrige cada exercício, indicando quais respostas estão certas e quais estão erradas. Esse tipo de aprendizagem é amplamente utilizado em tarefas como a classificação de imagens, por exemplo, identificar se uma imagem contém um gato ou um cachorro e na predição de valores, como estimar o preço de uma casa com base em suas características.

- **Aprendizagem Não Supervisionada:** Aqui, não existem rótulos nos dados, o agente explora os exemplos disponíveis e tenta descobrir padrões ou estruturas por conta própria. É como entregar a um aluno vários exemplos sem dizer o que é cada um, esperando que ele identifique similaridades e diferenças entre eles. Essa abordagem é comum em tarefas como o agrupamento de dados, por exemplo, segmentar clientes com comportamentos semelhantes, e na redução de dimensionalidade, que busca simplificar conjuntos de dados complexos preservando suas informações mais relevantes.

- **Aprendizagem por Reforço:** Na aprendizagem por reforço, não há rótulos explícitos nem uma estrutura de agrupamento a ser descoberta. Em vez disso, o agente interage com um ambiente e aprende com base em um sistema de recompensas e punições. A cada ação tomada, ele recebe uma recompensa, um valor numérico positivo ou negativo dependendo das consequências de sua decisão. O objetivo é aprender uma estratégia que maximize a recompensa acumulada ao longo do tempo. Um exemplo clássico é o treinamento de um agente para jogar xadrez, em que a recompensa vem ao vencer a partida ou evitar situações de xeque-mate.

Uma forma simples de distinguir a Aprendizagem por Reforço dos outros paradigmas é pensar no grau de orientação que o modelo tem. Na Aprendizagem Supervisionada, há muita informação sobre o que é certo e errado em cada tentativa, pois temos exemplos com rótulos. Já na Aprendizagem Não Supervisionada, não há referência para o que é certo ou errado; o algoritmo descobre padrões por conta própria. Na Aprendizagem por Reforço, o feedback não é imediato após cada ação em termos de um "rótulo"; em vez disso, temos apenas sinais de recompensa ou punição que, muitas vezes, só fazem sentido depois de uma série de ações.

3.1.1 Como funciona a Aprendizagem por Reforço?

De forma geral, o processo de aprendizagem por reforço segue um ciclo contínuo de interação entre o agente e o ambiente:

1. **Observação do ambiente:** o agente, seja um robô ou programa, identifica o estado atual em que se encontra.

2. **Tomada de ação:** com base em sua estratégia, conhecida como política, o agente decide qual ação executar naquele momento.

3. **Resposta do ambiente:** após a ação, o ambiente retorna uma nova situação, um novo estado e uma recompensa, que pode ser positiva (indicando um bom desempenho) ou negativa (sinalizando uma má escolha).

4. **Atualização da política:** com base na resposta recebida, o agente ajusta seus parâmetros internos para aumentar suas chances de tomar boas decisões no futuro.

Com o tempo e após muitas tentativas, o agente aprende quais caminhos levam a melhores resultados, ou seja, quais ações maximizam a soma total das recompensas ao longo do tempo. Um ponto crucial da aprendizagem por reforço é a **recompensa atrasada**. Em muitos cenários, a consequência real de uma ação só pode ser avaliada ao final de uma sequência de decisões. Por exemplo, em uma partida de xadrez, o agente só descobre se foi bem-sucedido ao ganhar ou perder o jogo, o que só ocorre no fim. Assim, um dos maiores desafios desse tipo de aprendizagem é atribuir valor às ações intermediárias mesmo quando a recompensa só chega lá na frente.

3.1.2 Onde é aplicado?

A Aprendizagem por Reforço tem uma variedade enorme de aplicações. Na indústria de jogos eletrônicos, é comum usar esses algoritmos para criar adversários virtuais mais inteligentes. Em sistemas de recomendação, é possível usar técnicas de Reforço para sugerir conteúdos ao usuário com base em suas interações, por exemplo, quanto tempo ele assiste a um determinado vídeo ou quantos cliques em um site. Outro exemplo interessante é a robótica, em que robôs aprendem tarefas complexas, como dobrar roupas ou organizar peças, a partir de repetidos experimentos.

Apesar de requerer um número considerável de tentativas e muitas vezes grandes recursos computacionais, a Aprendizagem por Reforço oferece uma maneira poderosa de lidar com problemas em que não há um conjunto de respostas prontas para cada situação. Ela se mostra ideal para cenários dinâmicos, onde há interação contínua com o ambiente e onde errar faz parte natural do processo de aprendizado.

A grande motivação por trás desse campo está na possibilidade de alcançar resultados surpreendentes em problemas que dependem muito da experiência prática, algo que nem sempre está disponível em outros métodos de aprendizado de máquina. Como veremos nos próximos tópicos, o sucesso em Aprendizagem por Reforço depende de noções básicas de probabilidade, estatística, algoritmos e, sobretudo, de uma mentalidade experimental, em que a exploração e a coragem de errar são partes importantes do caminho para o acerto.

3.1.3 Agente e Ambiente

No contexto da aprendizagem por reforço, o mundo é dividido em duas partes fundamentais: agente e ambiente. Em sistemas de decisão sequenciais, Russell e Norvig (2010) apontam que a distinção entre agente e ambiente é fundamental para modelar interações. O **agente** é o ”aluno”do processo de aprendizagem, é ele quem executa ações, observa as consequências dessas ações e ajusta seu comportamento com o objetivo de melhorar seu desempenho

ao longo do tempo. O **ambiente**, por sua vez, representa tudo o que está fora do controle do agente e com o qual ele interage. É o ambiente que responde às ações do agente, fornecendo recompensas ou penalidades e atualizando o estado observado.

Uma forma simples de visualizar essa relação é imaginar o agente como um jogador interagindo com um tabuleiro, que representa o ambiente. A cada jogada, o agente observa o estado atual do tabuleiro, escolhe uma ação e recebe um retorno, que pode ser uma recompensa ou penalidade, além da informação sobre o novo estado resultante. O grande desafio e objetivo do agente é descobrir, por tentativa e erro, uma sequência de ações que leve à maximização da recompensa total ao longo do tempo.

3.1.4 Estado

O estado (*state*) representa a situação em que o agente se encontra em um determinado momento. Pode ser algo bastante simples, como a posição do robô no labirinto. Em aplicações mais complexas, o estado pode englobar diversas variáveis, como velocidade de um carro autônomo, distância até um obstáculo ou até mesmo informações sobre outros agentes em um jogo multijogador.

Em termos formais, cada estado s pertence a um conjunto de estados possíveis, muitas vezes denotado por S. No entanto, para entender intuitivamente:

O estado é a representação da situação atual do ambiente observada pelo agente, é a base sobre a qual ele decide qual ação tomar. Essa representação pode variar bastante dependendo do problema. No caso do labirinto, o estado pode ser simplesmente as coordenadas do robô dentro do mapa. Em um jogo de xadrez, o estado corresponde à configuração completa das peças sobre o tabuleiro, refletindo a posição e o tipo de cada peça. Já em um carro autônomo, o estado é muito mais complexo e pode incluir informações como a posição e velocidade do veículo, a presença de pedestres, sinais de trânsito, obstáculos e até condições climáticas. A qualidade dessa representação é essencial, pois determina o quanto o agente consegue perceber e reagir ao ambiente de forma eficaz.

3.1.5 Ação

A ação (*action*) é o controle que o agente possui sobre o ambiente. É a decisão que ele toma a cada passo. No labirinto, as ações possíveis podem ser "mover para cima", "mover para baixo", "mover para a esquerda" ou "mover para a direita". Em aplicações mais complexas (por exemplo, em robótica), as ações podem envolver girar articulações de um braço mecânico ou alterar a aceleração de um carro autônomo.

Normalmente, em aprendizagem por reforço, cada estado s tem associado um conjunto de possíveis ações $A(s)$. Em um contexto formal, podemos dizer: Se o agente se encontra no estado s, então ele pode escolher uma ação $a \in A(s)$. Em muitos casos simples, assumimos que o conjunto de ações disponíveis é o mesmo para todos os estados, mas isso pode variar de acordo com o problema.

3.1.6 Recompensa

A recompensa é o sinal numérico que indica ao agente o quão boa (ou ruim) foi uma ação em determinado estado, ela é essencial para guiar o processo de aprendizagem. No labirinto, por exemplo, o agente pode receber **+10 pontos** ao alcançar a saída e **-1 ponto** sempre que colidir com uma parede, incentivando caminhos eficientes e evitando erros. Em um jogo de

tabuleiro, a recompensa pode ser mais global: atribuída apenas ao final da partida, como **+1** em caso de vitória, **-1** em caso de derrota e **0 para empates**, exigindo que o agente aprenda estratégias de longo prazo. Já em um sistema de controle de temperatura, a recompensa pode ser contínua, sendo **negativa quando a temperatura se desvia do intervalo ideal** e **positiva quando se mantém estável**, promovendo ajustes constantes. Cada tipo de problema exige uma forma específica de definir recompensas, e a maneira como elas são distribuídas influencia diretamente a velocidade e a qualidade do aprendizado.

Essa sinalização de "bom" ou "ruim" fornecida pela recompensa é o que orienta o processo de aprendizagem do agente. Diferentemente de outros métodos de aprendizagem de máquina, em que trabalhamos com dados rotulados ou não-rotulados, a aprendizagem por reforço conta principalmente com a informação dada pela recompensa para ajustar o comportamento do agente.

3.1.7 Política

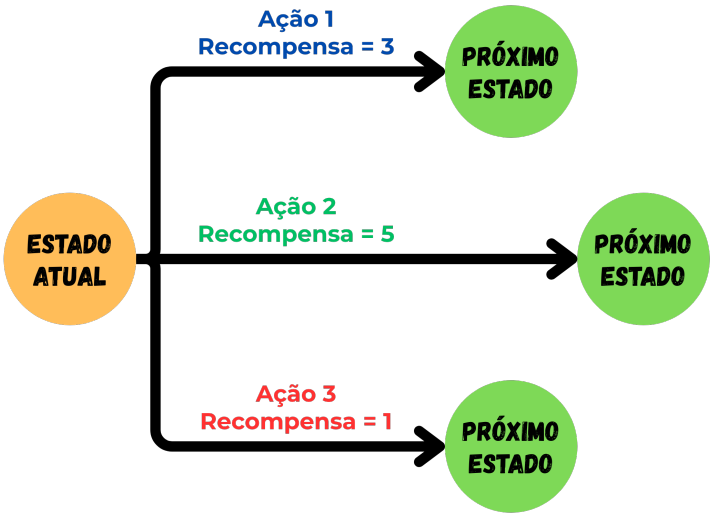
A política (*policy*) descreve como o agente escolhe suas ações para cada estado em que se encontrar. É, em essência, a estratégia ou a "regra de decisão" do agente. Podemos pensar na política como uma função π que, para cada estado s, indica qual ação a deve ser tomada:

$$\pi(s) = a \quad \text{(ação escolhida para o estado s)}$$

Dependendo do método de aprendizagem, a política pode ser determinística (o agente tem sempre uma ação específica para cada estado) ou estocástica (o agente escolhe ações de acordo com certas probabilidades associadas a cada estado). Em qualquer dos casos, o que o agente busca é ajustar π para maximizar a recompensa total esperada, ao longo de várias interações.

Uma maneira de pensar na política é vê-la como um conjunto de "instruções" que o agente consultaria a cada momento para saber qual ação executar. Conforme aprende, essas instruções vão sendo afinadas até que se torne possível, com sorte, alcançar o maior ganho possível de recompensas.

3.1.8 Markov Decision Processes



Em um nível mais formal, costuma-se modelar problemas de aprendizagem por reforço como Processos de Decisão de Markov (*Markov Decision Processes*, ou MDPs). Um MDP é definido por um conjunto de estados S, um conjunto de ações A, uma função de transição de probabilidade P que descreve a probabilidade de ir para um estado s' quando se executa a ação a no estado s e uma função de recompensa R. No entanto, para muitos problemas, a formulação intuitiva "estado $\rightarrow$ ação $\rightarrow$ recompensa + próximo estado" já é suficiente para compreender a essência do aprendizado.

Fique Alerta!

Em MDPs grandes, o grafo fica imenso. Nesses casos, usa-se amostragem ou abstrações (agrupamento de estados) para simplificar a visualização.

3.1.9 Aplicando seus conhecimentos

Ao iniciar sua jornada no campo da aprendizagem por reforço, é interessante trabalhar com cenários simples de tentativa e erro, como labirintos. O código abaixo permite simular o comportamento de um agente que navega por um labirinto, aprendendo a encontrar o objetivo enquanto recebe recompensas durante o processo. Essa abordagem oferece uma excelente oportunidade para entender melhor o papel das políticas na aprendizagem por reforço e como elas influenciam o comportamento do agente.

Copie e Teste!

```
import random
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from IPython.display import clear_output

# Labirinto padrão representado por uma matriz
labirinto_default = [
    ['S', '.', '.', '#', '.'],
    ['.', '#', '.', '.', '.'],
    ['.', '#', 'G', '#', '.'],
    ['.', '.', '.', '.', '.'],
    ['#', '.', '#', '.', '.']
]

# Variáveis globais
labirinto = []
inicio = (0, 0)
objetivo = (2, 2)
ações = ['cima', 'baixo', 'esquerda', 'direita']

# Função para inicializar o labirinto
def inicializar_labirinto():
    global labirinto
    labirinto = [linha[:] for linha in labirinto_default]  # Copia
    o labirinto padrão
```

```
    return labirinto

# Função para desenhar o labirinto
def desenhar_labirinto():
    fig, ax = plt.subplots()
    for i in range(5):
        for j in range(5):
            if labirinto[i][j] == '#':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
   color='black'))
            elif labirinto[i][j] == 'S':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
   color='green'))
            elif labirinto[i][j] == 'G':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
   color='gold'))
            else:
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
   edgecolor='gray', facecolor='white'))

    ax.set_xlim(0, 5)
    ax.set_ylim(0, 5)
    ax.set_aspect('equal')
    ax.axis('off')
    plt.show()

# Função para atualizar o labirinto com a escolha do usuário
def atualizar_labirinto(x, y, tipo):
    global labirinto, inicio, objetivo
    if tipo == 'inicio':
        labirinto[inicio[0]][inicio[1]] = '.'  # Remove o marcador
    do início anterior
        inicio = (x, y)
        labirinto[x][y] = 'S'
    elif tipo == 'objetivo':
        labirinto[objetivo[0]][objetivo[1]] = '.'  # Remove o
   marcador do objetivo anterior
        objetivo = (x, y)
        labirinto[x][y] = 'G'
    elif tipo == 'parede':
        labirinto[x][y] = '#'

    desenhar_labirinto()

# Função para definir se a posição é válida
def pos_valida(pos):
    x, y = pos
    return 0 <= x < 5 and 0 <= y < 5 and labirinto[x][y] != '#'

# Função para mover o agente
```

```
def mover(pos, ação):
    x, y = pos
    if ação == 'cima': x -= 1
    elif ação == 'baixo': x += 1
    elif ação == 'esquerda': y -= 1
    elif ação == 'direita': y += 1
    nova_pos = (x, y)
    return nova_pos if pos_valida(nova_pos) else pos

# Função para calcular a recompensa
def recompensa(pos):
    if pos == objetivo:
        return 10
    elif labirinto[pos[0]][pos[1]] == '#':
        return -5
    else:
        return -1

# Função para visualizar o caminho percorrido
def visualizar_caminho(caminho, recompensa_total, numero_passos):
    fig, ax = plt.subplots(figsize=(6, 6))

    for i in range(5):
        for j in range(5):
            if labirinto[i][j] == '#':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
color='black'))
            elif labirinto[i][j] == 'S':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
color='green'))
            elif labirinto[i][j] == 'G':
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
color='gold'))
            else:
                ax.add_patch(patches.Rectangle((j, 4 - i), 1, 1,
edgecolor='gray', facecolor='white'))

    # Novo: mapa para registrar quantas vezes cada posição foi
visitada
    visitas = {}

    for idx, (x, y) in enumerate(caminho):
        pos = (x, y)
        count = visitas.get(pos, 0)
        deslocamento_x = (count % 3 - 1) * 0.2
        deslocamento_y = ((count // 3) % 3 - 1) * 0.2

        ax.text(y + 0.5 + deslocamento_x, 4 - x + 0.5 +
deslocamento_y, str(idx),
```

```
                ha='center', va='center', fontsize=10, color='blue
    ')

        visitas[pos] = count + 1  # Incrementa a contagem da
    posição

     ax.set_xlim(0, 5)
     ax.set_ylim(0, 5)
     ax.set_aspect('equal')
     ax.axis('off')

     # Colocando a legenda abaixo do labirinto
     ax.legend([f"Recompensa total: {recompensa_total}", f"Passos:
    {numero_passos-1}"],
               loc='upper center', bbox_to_anchor=(0.5, -0.02),
    ncol=2)

     plt.show()

# Função para executar o episódio com a política aleatória
def executar_episodio(politica='aleatoria', max_passos=30, mostrar
    =False):
     pos = inicio
     total_recompensa = 0
     caminho = [pos]

     for _ in range(max_passos):
         if politica == 'aleatoria':
             ação = random.choice(ações)

         nova_pos = mover(pos, ação)
         r = recompensa(nova_pos)
         total_recompensa += r
         pos = nova_pos
         caminho.append(pos)
         if pos == objetivo:
             break

     if mostrar:
         visualizar_caminho(caminho, total_recompensa, len(caminho)
    )

     return total_recompensa, len(caminho), caminho

# Função para rodar as simulações
def rodar_simulacoes(num_episodios=2):
     clear_output(wait=True)
     politica = 'aleatoria'
     print(f"\nPolítica: {politica.upper()}")
     for _ in range(num_episodios):
```

```
        r, passos, caminho = executar_episodio(politica, mostrar=
    True)

# Inicializar o labirinto e rodar simulações
inicializar_labirinto()
rodar_simulacoes(2)  # Executar 2 episódios
```

Resultado Esperado

```
Política: ALEATORIA
```

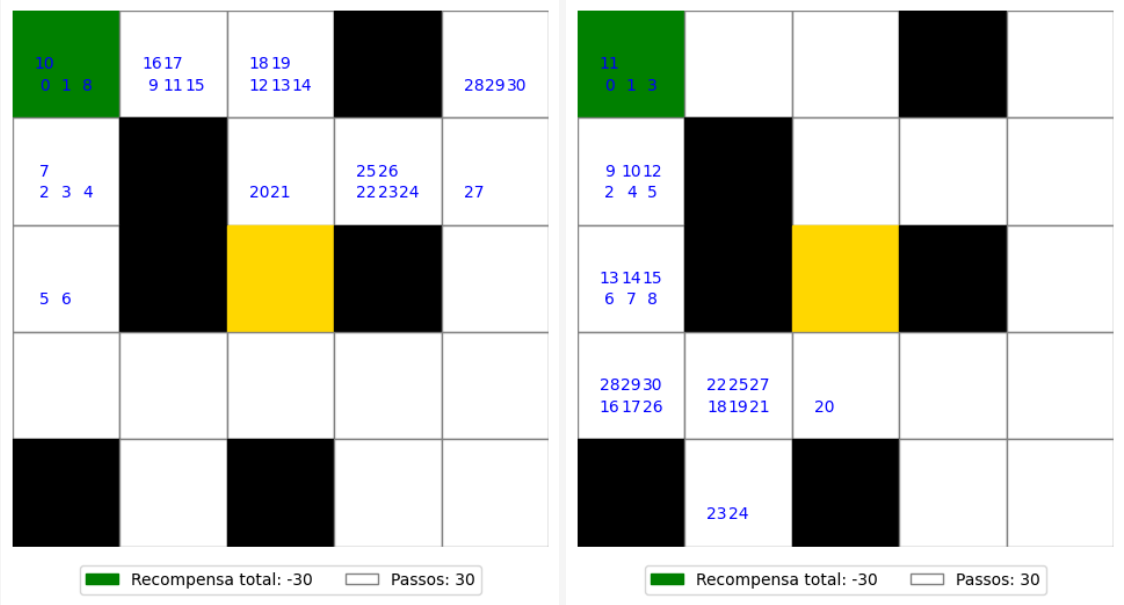


Figura 3.2: Resultados de dois episódios de navegação aleatória no labirinto.

```
Recompensa total: -30, Passos: 30
Recompensa total: -30, Passos: 30
```

No código, o labirinto é representado por uma matriz 5x5. Embora essa política seja simples, ela demonstra como o agente realiza escolhas baseadas nas recompensas recebidas, explorando o ambiente de maneira aleatória. A função de recompensa, que define as recompensas recebidas em diferentes estados, está assim configurada:

```
def recompensa(pos):
    if pos == objetivo:
        return 10
    elif labirinto[pos[0]][pos[1]] == '#':
        return -5
    else:
        return -1
```

Ao ajustar os valores ou a lógica dentro dessa função, por exemplo, alterando a recompensa ao passar por obstáculos ou ao encontrar o objetivo, você pode observar como isso afeta as decisões do agente, potencialmente fazendo com que ele aprenda a explorar o ambiente de maneira mais eficiente. Após a execução do código, o comportamento do agente é guiado pela política definida, que, neste caso, é aleatória.

A política é implementada neste trecho:

```
if politica == 'aleatoria':
    ação = random.choice(ações)
```

Você pode modificar o comportamento do agente escolhendo diferentes políticas ou alterando a lógica de como as ações são selecionadas. Além disso, o código permite definir um número máximo de passos para cada episódio, o que pode ser ajustado no parâmetro *max_passos* da função *executar_episodio*. Para limitar o tempo de exploração do agente, por exemplo, basta reduzir esse número:

```
r, passos, caminho = executar_episodio(politica, max_passos=20,
   mostrar=True)
```

Por fim, você pode modificar o número de episódios executados na simulação ajustando o valor passado para a função *rodar_simulacoes()*. Por exemplo, ao chamar *rodar_simulacoes(2)*, o agente executará 2 episódios. Se você quiser testar com mais ou menos episódios, basta alterar esse número. A quantidade de episódios pode influenciar o comportamento do agente, pois ele terá mais ou menos oportunidades de explorar e aprender no labirinto.

```
rodar_simulacoes(10)
```

Pronto! Agora você pode experimentar e entender melhor como o agente reage em diferentes cenários. Ao ajustar as variáveis, como as recompensas, políticas e número de passos, você verá como isso afeta o comportamento do agente e o processo de aprendizado. Continue explorando e modificando os parâmetros para aprofundar ainda mais sua compreensão sobre os conceitos de exploração, recompensa e aprendizado. Quanto mais você testar e ajustar, mais eficaz será o seu aprendizado sobre como os agentes funcionam em problemas de aprendizagem por reforço!

3.2 Funções de Valor

Iniciando o diálogo...

Imagine que você está treinando um robô para navegar por um labirinto em busca de recompensas. Como podemos medir "quão bom" é cada posição do labirinto, ou "quão boa" é cada ação que o robô pode tomar? É aqui que entram as funções de valor, ferramentas essenciais na aprendizagem por reforço para avaliar estados e ações ao longo do tempo.

3.2.1 Função de Valor de Estado V(s)

A função de valor de estado, $V^{\pi}(s)$, estima o retorno esperado (soma descontada de recompensas) ao começar no estado s e seguir uma política π.

Definição formal:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t+1} \mid s_0 = \mathbf{s}\right]$$

onde:

- r_{t+1} é a recompensa recebida ao transitar do passo t para $t+1$,
- $\gamma \in [0, 1)$ é o fator de desconto, que prioriza recompensas mais próximas,
- a expectativa $\mathbb{E}_{\pi}$ é tomada sobre trajetórias geradas pela política π.

Fique Alerta!

$V^{\pi}(s)$ depende da política π. Se π mudar, seus valores também mudam. O fator γ controla o "horizonte" de planejamento: $\gamma \to 0$ foca em recompensas imediatas; $\gamma \to 1$ valoriza mais o longo prazo.

3.2.2 Função de Valor de Ação

Enquanto $V(s)$ avalia estados, a **função de valor de ação**, $Q^{\pi}(s, a)$, avalia pares estado-ação. Ela responde à pergunta: "Se estou em s e escolho a, qual o retorno esperado ao seguir π a partir daí?"

Definição formal

$$Q^{\pi}(\mathbf{s}, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t+1} \mid s_0 = \mathbf{s}, a_0 = a\right]$$

Caso Prático

Em um labirinto simples 2×2, suponha que o robô possa escolher entre quatro ações: mover-se para o Norte (N), Sul (S), Leste (L) ou Oeste (O). Os estados possíveis em que o robô pode se encontrar, por exemplo, podem ser A, B e assim por diante. A função de valor de ação avalia qual o retorno esperado ao tomar uma determinada ação em um estado específico. Para representar isso, podemos montar uma tabela em que as linhas correspondem aos estados e as colunas às ações. Cada célula da tabela contém o valor de $Q^{\pi}(s, a)$, ou seja, o valor esperado de tomar a ação a no estado s. Por exemplo, uma tabela poderia ser:

	N	S	L	O
A	0.8	0.1	0.5	0.2
B	0.7	0.4	0.3	0.9

Essa tabela está te dizendo o seguinte:

- Se você está no estado A e toma a ação N, o valor do retorno é 0.8
- Se está no estado B e toma a ação O, o valor esperado é 0.9

A função de valor de ação $Q^{\pi}(s, a)$ mede quão boa é uma ação específica em um estado específico. Ela te **ajuda a escolher a melhor ação possível** a partir de um estado.

3.2.3 Equações de Bellman

As Equações de Bellman fornecem relações recursivas que permitem calcular as funções de valor de estado $V^\pi(s)$ e de valor de ação $Q^\pi(s,a)$ de forma iterativa, com base nas recompensas imediatas e nos valores futuros esperados. Elas são fundamentais para a construção e análise de políticas em algoritmos de aprendizado por reforço.

Equação de Bellman de Expectativa para V

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V^\pi(s')]$$

Essa equação diz que o valor de um estado s sob uma política π é a média ponderada dos valores esperados das ações possíveis, considerando:

- a probabilidade de escolher cada ação a definida pela política
- a probabilidade de transição para o próximo estado s'
- a recompensa imediata r
- e o valor descontado do próximo estado $V^\pi(s')$

Equação de Bellman de Expectativa para Q

$$Q^\pi(s,a) = \sum_{s',r} P(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s')Q^\pi(s', a')]$$

Aqui, o valor de $Q^\pi(s,a)$ considera diretamente a ação a escolhida no estado s, e depois avalia todas as ações futuras a' no próximo estado s', seguindo a política π.

Cada valor calculado pelas Equações de Bellman é um tipo de "backup" que representa a soma do retorno imediato, ou seja, a recompensa por tomar uma ação, junto com o valor futuro esperado, o que ainda está por vir, ponderado pelo fator de desconto γ. Essas equações são fundamentais para atualizar estimativas de valor em algoritmos como *Value Iteration*, *Policy Iteration* e *Q-Learning*. O processo leva em conta as probabilidades de transição entre estados e as escolhas feitas pela política.

Equação de Bellman de Otimalidade

Quando buscamos encontrar a política ótima π^*, ou seja, aquela que maximiza o retorno esperado, usamos as versões ótimas das Equações de Bellman, substituindo a expectativa pela melhor escolha possível, o máximo:

$$V^*(s) = \max_a Q^*(s,a), \quad Q^*(s,a) = \sum_{s',r} P(s', r|s, a)[r + \gamma V^*(s')]$$

Essas versões descrevem o valor ótimo de estar em um estado ou de tomar uma ação, assumindo que o agente sempre tomará a melhor decisão em cada passo futuro.

Fique Alerta!

As Equações de Bellman servem como base para propagar *valor* ao longo dos estados. O agente aprende, mesmo sem saber o caminho certo de início, que certas ações levam a estados com maior valor e assim ele melhora sua política com o tempo.

3.2.4 Aplicando seus conhecimentos

No método *Value Iteration*, atualizamos o valor de cada estado V com base nas Equações de Bellman, considerando a melhor ação possível a partir daquele estado. A atualização leva em conta a recompensa imediata e o valor futuro dos estados vizinhos, até que os valores se estabilizem. Abaixo, aplicamos esse processo a um exemplo simples de labirinto 2×2.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Estados nomeados
estados = ['A', 'B', 'C', 'D']
indices_estados = {'A': (0, 0), 'B': (0, 1), 'C': (1, 0), 'D': (1,
    1)}

# Inicialização dos valores dos estados
valores = np.zeros((2, 2))

# Parâmetros
fator_desconto = 0.9
limiar_convergencia = 1e-4
diferenca_maxima = float('inf')

# Recompensas (apenas o estado D tem recompensa)
recompensas = np.zeros((2, 2))
recompensas[1, 1] = 1

# Função para obter os estados vizinhos possíveis
def obter_vizinhos(i, j):
    vizinhos = []
    if i > 0: vizinhos.append((i - 1, j)) # cima
    if i < 1: vizinhos.append((i + 1, j)) # baixo
    if j > 0: vizinhos.append((i, j - 1)) # esquerda
    if j < 1: vizinhos.append((i, j + 1)) # direita
    return vizinhos

# Iteração de Valor (Value Iteration)
historico_iteracoes = []
while diferenca_maxima > limiar_convergencia:
    diferenca_maxima = 0
    novos_valores = np.copy(valores)
```

```
    for i in range(2):
        for j in range(2):
            if (i, j) == (1, 1): # Estado D é terminal
                continue

            vizinhos = obter_vizinhos(i, j)
            valores_possiveis = []
            for vi, vj in vizinhos:
                recompensa = recompensas[vi, vj]
                valor_futuro = valores[vi, vj]
                valores_possiveis.append(recompensa +
   fator_desconto * valor_futuro)

            melhor_valor = max(valores_possiveis)
            novos_valores[i, j] = melhor_valor
            diferenca_maxima = max(diferenca_maxima, abs(
   melhor_valor - valores[i, j]))

    valores = novos_valores
    historico_iteracoes.append(valores.copy())

# Visualização dos valores finais dos estados
plt.figure(figsize=(5, 4))
sns.heatmap(valores, annot=True, fmt=".2f", cmap="YlGnBu", cbar=
   True,
            xticklabels=["B", "D"], yticklabels=["A", "C"])
plt.title("Valores dos Estados após Iteração de Valor")
plt.xlabel("Coluna")
plt.ylabel("Linha")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

# Resultado final
print("Valores finais dos estados:")
print(f"Estado A: {valores[0, 0]:.2f}")
print(f"Estado B: {valores[0, 1]:.2f}")
print(f"Estado C: {valores[1, 0]:.2f}")
print(f"Estado D (terminal): {valores[1, 1]:.2f}")
```

Resultado Esperado

```
Valores finais dos estados:
Estado A: 0.90
Estado B: 1.00
Estado C: 1.00
Estado D (terminal): 0.00
```

Fique Alerta!

Isso mostra que os estados B e C têm o maior valor, pois levam diretamente ao estado D com recompensa. O estado A também tem um bom valor (0.9), pois leva até B ou C.

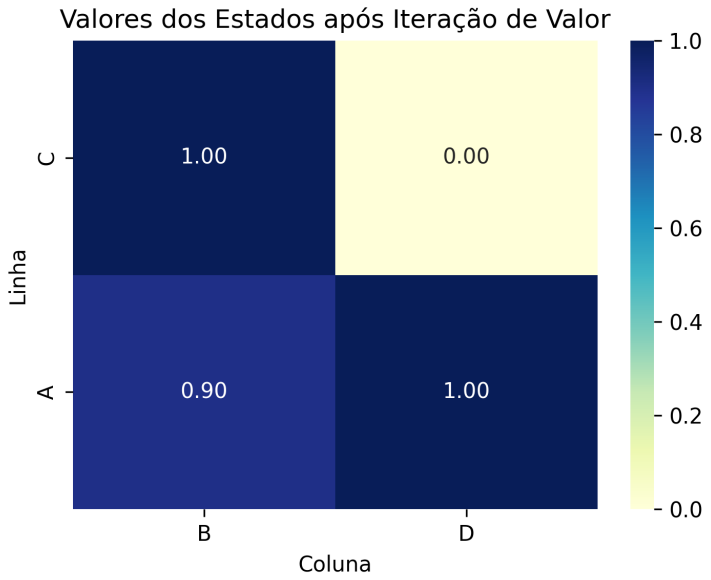


Figura 3.3: Mapa de calor exibindo os valores finais dos estados após a execução do algoritmo de Iteração de Valor.

Conhecendo um pouco mais!

Para aprofundar, explore materiais que detalham métodos como *Policy Iteration* e *Q-Learning*, que usam essas funções de valor em algoritmos de controle.

3.3 Exemplos de Ambiente

Iniciando o diálogo...

Imagine que você é um agente, um robô ou um jogador, navegando em um labirinto ou escolhendo entre várias máquinas caça-níqueis para ganhar moedas. Em Aprendizagem por Reforço, cada "jogo" que seu agente enfrenta é chamado de ambiente. Neste trecho, vamos conhecer dois ambientes clássicos que ilustram de forma clara os desafios de tomar decisões sequenciais, o *GridWorld* e os *Multi-Armed Bandits*.

3.3.1 GridWorld

Um *GridWorld* é uma grade bidimensional onde um agente se move célula a célula, interagindo com o ambiente por meio de ações simples como mover-se para cima, baixo, esquerda ou direita. Em cada passo, o agente pode receber recompensas ao alcançar um objetivo ou penalidades ao colidir com obstáculos, o que torna esse tipo de ambiente ideal para ilustrar

conceitos fundamentais de aprendizado por reforço. Um exemplo típico de *GridWorld* pode ser representado da seguinte forma:

S			G
	X		

Nessa grade, S representa o estado inicial (*Start*), de onde o agente começa sua jornada. A célula G é o objetivo (*Goal*), cuja chegada garante uma recompensa de +1. O obstáculo X é intransponível, representando uma penalidade de -1 caso o agente tente acessá-lo. As demais células são estados neutros, que não concedem recompensa nem penalidade.

Formalmente, cada célula da grade representa um estado. As ações disponíveis ao agente são os movimentos para cima, baixo, esquerda e direita, permitindo movimentação para as quatro direções cardeais. A função de recompensa define que o agente recebe +1 ao alcançar o objetivo, -1 ao colidir com um obstáculo e 0 nas demais transições.

As transições de estado podem ser determinísticas, em que a ação leva diretamente ao estado vizinho, ou estocásticas, caso em que existe uma probabilidade de desvio, por exemplo, 80% de chance de seguir a direção escolhida e 10% para cada uma das direções laterais. Esse tipo de ambiente é extremamente útil para desenvolver e testar algoritmos de tomada de decisão sequencial, como *Value Iteration*, *Policy Iteration*, *Q-learning* e *SARSA*, além de proporcionar visualizações intuitivas para o processo de aprendizado.

Caso Prático

Com base na estrutura mostrada acima, vamos construir uma versão do *GridWorld* em Python, definindo seus estados, ações, recompensas e regras de transição.

Copie e Teste!

```
# Ambiente GridWorld
import gymnasium as gym
from gymnasium import spaces
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output, display
import time

class GridWorldEnv(gym.Env):
    def __init__(self):
        super(GridWorldEnv, self).__init__()
        # Grade do ambiente (3 linhas x 4 colunas)
        self.grid = [
            ['S', '', '', 'G'],
            ['', 'X', '', ''],
            ['', '', '', '']
        ]
        self.linhas = 3
        self.colunas = 4
        self.estado = (0, 0) # posição inicial
```

```
        self.action_space = spaces.Discrete(4)
        self.observation_space = spaces.MultiDiscrete([self.linhas
    , self.colunas])

    def reset(self, seed=None, options=None):
        self.estado = (0, 0)
        return self.estado, {}

    def step(self, acao):
        linha, coluna = self.estado
        if acao == 0 and linha > 0: nova_linha, nova_coluna =
    linha - 1, coluna
        elif acao == 1 and linha < self.linhas - 1: nova_linha,
    nova_coluna = linha + 1, coluna
        elif acao == 2 and coluna > 0: nova_linha, nova_coluna =
    linha, coluna - 1
        elif acao == 3 and coluna < self.colunas - 1: nova_linha,
    nova_coluna = linha, coluna + 1
        else: nova_linha, nova_coluna = linha, coluna

        if self.grid[nova_linha][nova_coluna] == 'X':
            nova_linha, nova_coluna = linha, coluna

        self.estado = (nova_linha, nova_coluna)

        if self.grid[nova_linha][nova_coluna] == 'G':
            return self.estado, 1.0, True, False, {}
        else:
            return self.estado, 0.0, False, False, {}

    def render(self, mode='rgb_array'):
        img = np.ones((300, 400, 3), dtype=np.uint8) * 255
        cores = {'S': (200, 255, 200), 'G': (200, 200, 255), 'X':
    (255, 200, 200), '': (230, 230, 230)}
        for i in range(self.linhas):
            for j in range(self.colunas):
                img[i*100:(i+1)*100, j*100:(j+1)*100] = cores[self
    .grid[i][j]]
        i, j = self.estado
        img[i*100+25:i*100+75, j*100+25:j*100+75] = (0, 0, 0)
        return img

# Testando o ambiente
env = GridWorldEnv()
obs, info = env.reset()
for passo in range(40):
    acao = env.action_space.sample()
    obs, recompensa, finalizado, truncado, info = env.step(acao)
    frame = env.render()
    clear_output(wait=True)
```

```
    plt.imshow(frame)
    plt.axis("off")
    plt.title(f"Passo {passo}")
    display(plt.gcf())
    time.sleep(0.5)
    if finalizado:
        print("Chegou no objetivo!")
        break
env.close()
```

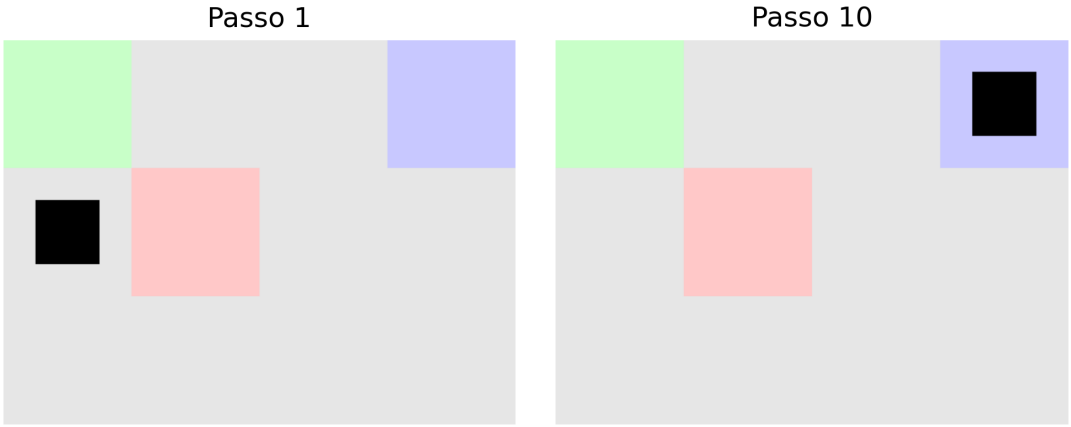


Figura 3.4: Exemplo de execução do GridWorld com agente aleatório.

Aplicando seus conhecimentos

1. Se a recompensa do objetivo fosse +10 em vez de +1, como isso poderia afetar a política aprendida?

2. O que mudaria se adicionássemos um pequeno custo negativo em cada movimento, por exemplo, -0.10, mesmo sem bater em obstáculos ou alcançar o objetivo?

3.3.2 Multi-Armed Bandits

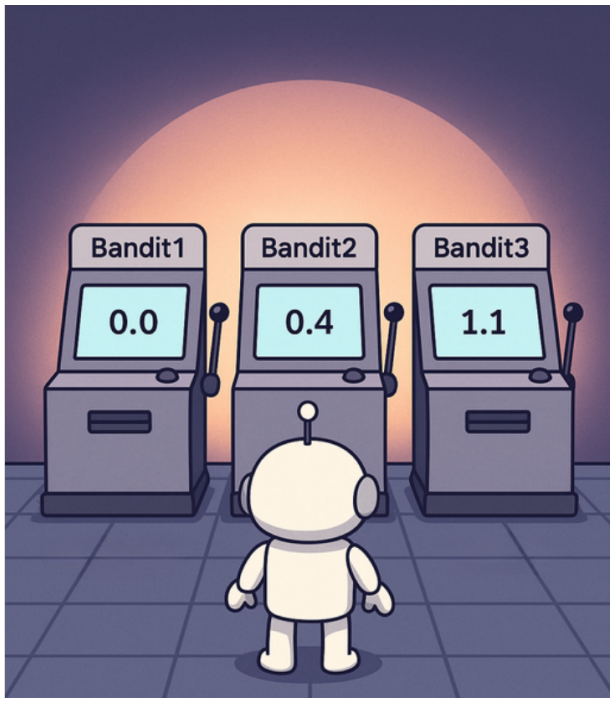


Figura 3.5: Ilustração gerada com o DALL·E, representando um corredor com três máquinas caça-níqueis sobre multi-armed bandits em 16/04/2025.

Imagine-se em um corredor repleto de máquinas caça-níqueis, também chamadas de *bandits*. Cada máquina possui uma distribuição de recompensas desconhecida, ou seja, ao jogar, você recebe um retorno aleatório que depende da máquina escolhida. O desafio do seu agente é decidir, a cada rodada, em qual máquina jogar para maximizar o ganho acumulado ao longo do tempo.

Um dos principais desafios no cenário dos *multi-armed bandits* é o dilema entre exploração e *exploitation*. Explorar significa testar novas máquinas, mesmo que seus resultados anteriores não tenham sido os melhores, com a esperança de descobrir opções mais vantajosas. Por outro lado, explorar no sentido de *exploitation* consiste em continuar jogando na máquina que, até o momento, apresentou o melhor desempenho. Encontrar o equilíbrio entre essas duas estratégias é essencial para maximizar as recompensas ao longo do tempo, evitando tanto o desperdício de oportunidades quanto a estagnação em escolhas subótimas.

Caso Prático

Suponha as três máquinas da imagem com médias de recompensa ainda incertas. Após 10 jogadas em cada uma obtivemos:

- Máquina 1: média ≈ 0.01
- Máquina 2: média ≈ 0.40
- Máquina 3: média ≈ 1.10

Qual delas você escolheria na 21ª jogada? A resposta depende de como você equilibra a exploração de novas oportunidades com a exploração do que já funciona. Abaixo, temos

uma simulação simples de uma estratégia ϵ-greedy, onde o agente escolhe aleatoriamente com probabilidade ϵ e, no restante das vezes, escolhe a máquina com melhor valor estimado, iremos simular com ϵ variando de 0.0 a 0.9.

Copie e Teste!

```
import random
import matplotlib.pyplot as plt
import numpy as np

def simular_bandits(epsilon=0.1, jogadas=200):
    bandits = [
        lambda: random.gauss(0.01, 1),
        lambda: random.gauss(0.40, 1),
        lambda: random.gauss(1.10, 1)
    ]
    contagens = [0] * len(bandits)
    valores = [0.0] * len(bandits)
    for _ in range(jogadas):
        if random.random() < epsilon:
            escolha = random.randint(0, len(bandits) - 1)
        else:
            escolha = valores.index(max(valores))
        recompensa = bandits[escolha]()
        contagens[escolha] += 1
        valores[escolha] += (recompensa - valores[escolha]) /
   contagens[escolha]
    return contagens

epsilons = [0.0, 0.1, 0.3, 0.9]
resultados = [simular_bandits(epsilon=eps, jogadas=300) for eps in
    epsilons]
resultados = np.array(resultados)

largura_barra = 0.25
x = np.arange(len(epsilons))
plt.figure(figsize=(10, 6))
plt.bar(x - largura_barra, resultados[:, 0], width=largura_barra,
   label='Bandit 1 ($\mu \approx 0.01$)', color='skyblue')
plt.bar(x, resultados[:, 1], width=largura_barra, label='Bandit 2
   ($\mu \approx 0.40$)', color='lightgreen')
plt.bar(x + largura_barra, resultados[:, 2], width=largura_barra,
   label='Bandit 3 ($\mu \approx 1.10$)', color='salmon')
plt.xticks(x, [f'$\\epsilon$ = {e}' for e in epsilons])
plt.xlabel("Valor de epsilon ($\\epsilon$)")
plt.ylabel("Número de escolhas")
plt.title("Comparação de escolhas para diferentes valores de $\\
   epsilon$ (3 Bandits)")
plt.legend()
plt.grid(True, axis='y')
```

```
plt.tight_layout()
plt.show()
```

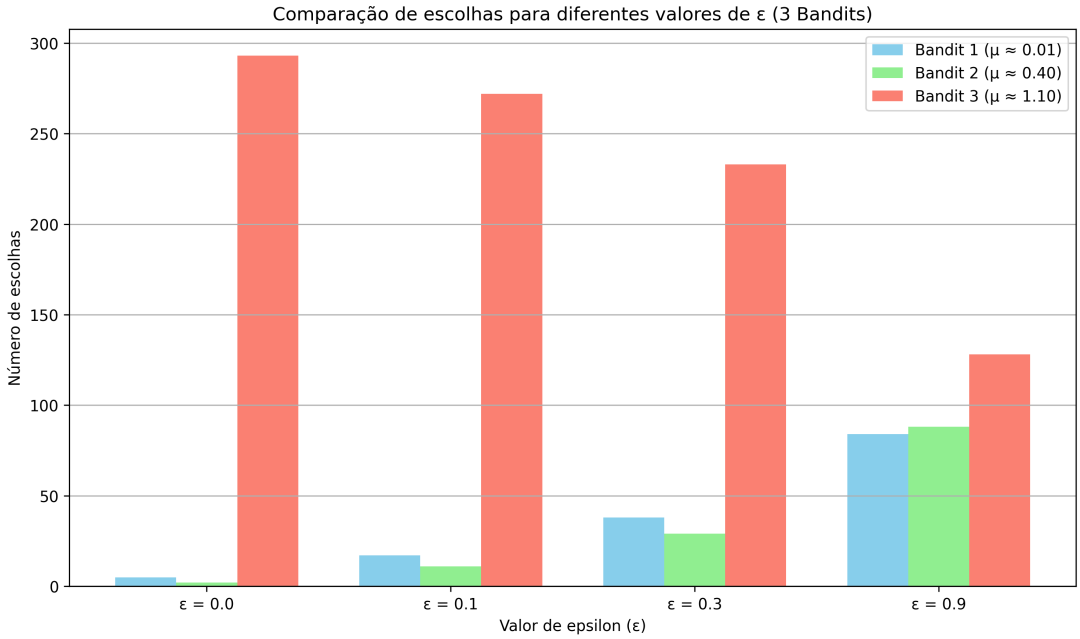


Figura 3.6: Gráfico comparando o número de vezes que cada 'bandit' foi escolhido para diferentes valores de ϵ.

Vamos analisar o que aconteceu

Usando o algoritmo ϵ-greedy, começando em $\epsilon = 0.0$, o agente não explora, sempre escolhe o *bandit* com o maior valor estimado, no caso o Bandit 3, com média 1.10. Assim, a maioria das escolhas deve ser para Bandit 3.

Com $\epsilon = 0.1$, o agente explorará 10% das vezes e, nos outros 90%, escolherá o bandit com o maior valor estimado até o momento. As escolhas ainda devem ser principalmente para Bandit 3, mas com alguma exploração dos outros. Ao chegar com $\epsilon = 0.3$, o agente explorará 30% das vezes, o que leva a uma distribuição mais balanceada das escolhas, especialmente entre Bandit 2 e Bandit 3. Um ponto interessante é $\epsilon = 0.9$, neste aqui o agente explorará 90% das vezes, o que faz com que as escolhas sejam quase aleatórias. Portanto todos os bandits devem ser escolhidos de forma mais equilibrada.

Fique Alerta!

Embora simples, esses exemplos capturam os desafios centrais da Aprendizagem por Reforço: modelar ambientes, lidar com incertezas e equilibrar exploração e *exploitation*.

3.4 Algoritmos clássicos de aprendizagem por reforço

Iniciando o diálogo...

Em ambientes onde decisões precisam ser tomadas em sequência e sob incerteza, como em jogos, robótica ou sistemas de recomendação, os algoritmos clássicos de aprendizagem por reforço fornecem ferramentas poderosas para encontrar políticas ótimas de ação. Entre os mais fundamentais estão o *Value Iteration* e o *Policy Iteration*, dois métodos baseados em programação dinâmica aplicados a Processos de Decisão de Markov (MDPs).

3.4.1 Value Iteration e Policy Iteration

Esses algoritmos assumem que o modelo do ambiente, ou seja, as probabilidades de transição entre estados e recompensas é conhecido, e utilizam esse conhecimento para refinar, a cada iteração, estimativas sobre o valor de estar em determinado estado ou de executar determinada ação. O objetivo final é encontrar uma política ótima, ou seja, uma regra de decisão que maximize a recompensa acumulada ao longo do tempo.

Aqui, vamos explorar como cada algoritmo opera, quais são suas diferenças e semelhanças, e como ocorre o processo de convergência para uma solução ótima. Com isso, construiremos a base conceitual para técnicas mais avançadas e práticas da aprendizagem por reforço.

Value Iteration: Atualização Baseada no Valor

O algoritmo de *Value Iteration* atualiza os valores dos estados diretamente utilizando a equação de Bellman. A ideia central é aproximar, de forma iterativa, o valor ótimo V^* para cada estado, até que as alterações sejam tão pequenas que possamos considerar o processo convergido. A equação de Bellman para um estado s é dada por:

$$V(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]$$

onde:

- $R(s,a)$ é a recompensa imediata ao executar a ação a em s
- γ é o fator de desconto que pondera recompensas futuras ($0 \leq \gamma < 1$)
- $P(s'|s,a)$ é a probabilidade de transição para o estado s' ao executar a em s

Cada iteração do *Value Iteration* utiliza essa equação para atualizar o valor de cada estado, "olhando para frente" e avaliando as recompensas imediatas e futuras. Sua atualização ocorre de forma iterativa:

- **Inicialização:** Começamos atribuindo um valor inicial a todos os estados. Em geral, pode-se utilizar zero ou uma estimativa prévia.
- **Atualização Recursiva:** Em cada iteração, para cada estado s, recalculamos $V(s)$ usando a equação de Bellman. Assim, o algoritmo "propaga" as informações de recompensa ao longo dos estados.

- **Critério de Convergência:** A atualização é repetida até que a mudança em $V(s)$ para todos os estados seja menor que um valor pré-definido ϵ, garantindo que os valores se estabilizaram.

O algoritmo converge porque a operação de atualização é um mapeamento de contração: a diferença entre os valores das iterações diminui a cada passo, levando os valores para um ponto fixo que é o valor ótimo V^*. Esse comportamento é garantido matematicamente pela propriedade do fator de desconto γ, que diminui a influência de recompensas futuras.

Policy Iteration: Avaliação e Melhoria de Políticas

Enquanto o *Value Iteration* foca na atualização direta dos valores dos estados, o *Policy Iteration* alterna entre duas etapas fundamentais:

1. **Avaliação da Política:** Calcula o valor de cada estado para uma política fixa.
2. **Melhoria da Política:** Atualiza a política escolhendo a ação que maximiza a recompensa baseada nos valores calculados.

Essa abordagem "divide e conquista" o problema, tornando a busca pela política ótima mais estruturada. Na etapa de avaliação da política, partindo de uma política inicial, calcula-se o valor $V^{\pi}(s)$ para cada estado s por meio da seguinte equação:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$$

Esse processo pode ser realizado de forma iterativa ou resolvendo um sistema de equações lineares, de modo a determinar o "desempenho" da política atual. Uma vez obtidos os valores $V^{\pi}(s)$, a política é atualizada de forma a escolher a ação que maximize a recompensa esperada:

$$\pi_{\text{novo}}(s) = \arg\max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi}(s') \right]$$

Esse novo conjunto de ações é então utilizado na próxima etapa de avaliação, e o processo se repete até que a política não sofra mais mudanças significativas.

A convergência do *Policy Iteration* ocorre quando a etapa de melhoria da política não resulta em mudanças, ou seja, a política atual é ótima, pois nenhuma ação alternativa oferece uma recompensa melhor. Esse ponto de equilíbrio garante que o algoritmo encontrou uma política que maximiza o valor esperado em todos os estados. Em termos matemáticos, a política ótima π^* satisfaz:

$$\pi^*(s) = \arg\max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right]$$

Fique Alerta!

Embora ambos os algoritmos sejam projetados para encontrar políticas ótimas, suas abordagens diferem.

Critério	Value Iteration	Policy Iteration
Estratégia	Atualiza diretamente os valores dos estados.	Alterna entre avaliação e melhoria da política.
Etapas do Algoritmo	Avaliação e melhoria combinadas em uma única equação iterativa.	Etapas separadas: primeiro avalia, depois melhora.
Facilidade de Implementação	Mais simples de implementar.	Requer estrutura para duas fases distintas.
Velocidade de Convergência	Pode levar mais iterações para convergir.	Converge em menos iterações, geralmente.
Custo Computacional por Iteração	Mais leve, pois só atualiza valores.	Mais pesado, pois cada avaliação pode envolver resolver sistemas lineares.
Escalabilidade	Mais indicado para ambientes muito grandes (com aproximações).	Pode ser menos eficiente em MDPs muito grandes (a menos que aproximado).
Uso Prático	Ideal quando se deseja simplicidade e menos custo por iteração.	Ideal quando se pode investir mais por iteração para alcançar rapidez global.

Tabela 3.1: Comparativo: Value Iteration vs. Policy Iteration.

Conhecendo um pouco mais!

Procure sobre outras estratégias de decisão que também são usadas para lidar com esse dilema, *Softmax* e *Upper Confidence Bound (UCB)*.

3.5 Q-Learning

Iniciando o diálogo...

Imagine que você queira ensinar um robô a atravessar um labirinto sem nunca ter visto o mapa antes. Como fazer com que ele "aprenda" a melhor rota só testando movimentos e recebendo recompensas (por exemplo, +1 ao chegar na saída, −1 ao bater numa parede)? É aqui que entra o *Q-Learning*, um dos algoritmos mais clássicos de aprendizagem por reforço.

Goodfellow, Bengio e Courville (2016) ressaltam que o Q-learning permite a aprendizagem de políticas ótimas sem conhecimento a priori do modelo do ambiente, assim o Q-Learning é um método *off-policy*, significa que o agente aprende o valor de ações ótimas mesmo explorando com outra política. Em vez de estimar diretamente o valor de um estado, estimamos a qualidade (*quality*) de pares estado-ação, chamada de $Q(s, a)$, onde:

- s: descreve a situação do estado atual, por exemplo, posição no labirinto.

- a: descreve o movimento que o agente pode tomar, por exemplo, cima, baixo, esquerda, direita.
- $Q(s, a)$: significa "quão boa" é a ação a no estado s, em termos de recompensa futura esperada.

O agente itera: em cada passo, observa (s, a, r, s'), atualiza $Q(s, a)$ e escolhe novas ações. A cada experiência, ajustamos $Q(s, a)$ assim:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

onde:

- α, usualmente é a taxa de aprendizado ($0 < \alpha \leq 1$), controla o quanto a nova informação substitui a antiga.
- γ é o fator de desconto ($0 \leq \gamma < 1$), pondera a importância de recompensas futuras.
- r é a recompensa imediata obtida ao executar a em s.
- $\max_{a'} Q(s', a')$: estimativa do melhor valor possível a partir do próximo estado s'.

Fique Alerta!

Se α for muito alto, $Q(s, a)$ varia demais e pode não convergir. Se γ for muito baixo, o agente fica "míope", ou seja, valoriza só recompensas imediatas.

3.5.1 Equilibrando exploração e exploitation

Para que um agente aprenda de forma eficiente, é necessário encontrar um equilíbrio entre duas atitudes opostas: explorar novas ações para descobrir possibilidades melhores (*exploration*) e aproveitar o que já sabe sobre as melhores escolhas atuais (*exploitation*). A estratégia ϵ-greedy funciona assim:

- Com **probabilidade** ϵ, o agente escolhe uma ação aleatória, *exploration* = explorando o ambiente.
- Com **probabilidade** $1 - \epsilon$, o agente escolhe a ação que atualmente possui o maior valor estimado, *exploitation* = explorando o que já funciona bem.

Na prática, ϵ costuma diminuir ao longo do tempo, por exemplo, começando em 1.0 e reduzindo gradualmente até 0.1, o que permite muita *exploration* no início da aprendizagem e mais *exploitation* conforme o agente se torna mais confiante sobre as ações.

3.5.2 Resumo

O Q-Learning é poderoso pois, mesmo sem modelo do ambiente, converge sob certas condições à política ótima. A combinação da fórmula de atualização e da política ϵ-greedy garante aprendizado e equilíbrio *exploration/exploitation*.

Conhecendo um pouco mais!

Para aprofundar, procure ler sobre as variações, como *Double Q-Learning* e *DQN*, o Q-Learning com redes neurais.

3.6 SARSA e outras variações

Iniciando o diálogo...

Imagine que você é um(a) explorador(a) em um labirinto desconhecido, onde cada passo revela recompensas e perigos. Como decidir o melhor caminho sem conhecer todo o mapa? No aprendizado por reforço, agentes aprendem a escolher ações a partir de experiências, ajustando seu comportamento para maximizar recompensas futuras. Hoje, vamos conhecer o algoritmo SARSA, um método clássico "on-policy" que combina exploração e atualização em cada passo, e suas variações que ampliam ainda mais seu poder.

3.6.1 O que é SARSA?

SARSA é um acrônimo para *State–Action–Reward–State–Action*, que indica as cinco variáveis usadas em sua regra de atualização:

- S_t: estado atual
- A_t: ação tomada em S_t
- R_{t+1}: recompensa recebida ao transitar para o próximo estado
- S_{t+1}: próximo estado
- A_{t+1}: ação escolhida em S_{t+1}

Diferente do Q-Learning (*off-policy*), que usa a melhor ação futura para atualizar $Q(S_t, A_t)$, o SARSA atualiza com base na ação que o agente realmente irá tomar, seguindo sua política de exploração (por exemplo, ϵ-greedy). A cada transição, ajustamos $Q(S_t, A_t)$ segundo:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

com $\alpha \in (0, 1]$: taxa de aprendizado e $\gamma \in [0, 1]$: fator de desconto de recompensas futuras. Agora imagine uma superfície tridimensional em que o eixo x é o estado, y é a ação, e z é $Q(S_t, A_t)$. A cada passo, você "puxa" o valor $Q(S_t, A_t)$ em direção a um ponto de referência dado por $R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1})$, suavizando a superfície até aproximar-se da função ótima.

Fique Alerta!

SARSA é sensível à política de exploração, se você explora muito (ϵ grande), as atualizações serão mais "ruidosas", mas garantem melhores descobertas de caminhos, mas se explora pouco, pode convergir rápido a políticas subótimas.

O caso prático a seguir ilustra como a função $Q(S_t, A_t)$ evolui ao longo das iterações do algoritmo SARSA, destacando como o modelo se ajusta gradualmente em direção a uma

estimativa mais precisa das melhores ações em cada situação. Vamos explorar esse processo e entender o impacto de diferentes parâmetros no desempenho do aprendizado.

Copie e Teste!

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Criando uma grade de estados e ações
states = np.linspace(0, 10, 30)
actions = np.linspace(0, 5, 30)
S, A = np.meshgrid(states, actions)

# Função Q inicial: com picos e vales
Q_initial = np.sin(S / 2) * np.cos(A) + 0.5 * np.random.randn(*S.
    shape)

# Parâmetros
alpha = 0.1
gamma = 0.9

# Valor alvo artificial (como se fosse r + gamma * Q(s', a'))
Q_target = np.cos(S / 3) * np.sin(A / 2)

# Atualização SARSA: 30 iterações
Q_updated = Q_initial.copy()
for _ in range(30):
    Q_updated += alpha * (Q_target - Q_updated)

# Plotando antes e depois
fig = plt.figure(figsize=(14, 6))

# Superfície original
ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(S, A, Q_initial, cmap='viridis', alpha=0.9)
ax1.set_title('Superfície Q original')
ax1.set_xlabel('Estado (s)')
ax1.set_ylabel('Ação (a)')
ax1.set_zlabel('Q(s, a)')

# Superfície após 30 iterações
ax2 = fig.add_subplot(122, projection='3d')
ax2.plot_surface(S, A, Q_updated, cmap='viridis', alpha=0.9)
ax2.set_title('Superfície Q após 30 iterações SARSA')
ax2.set_xlabel('Estado (s)')
ax2.set_ylabel('Ação (a)')
ax2.set_zlabel('Q(s, a)')
plt.tight_layout()
plt.show()
```

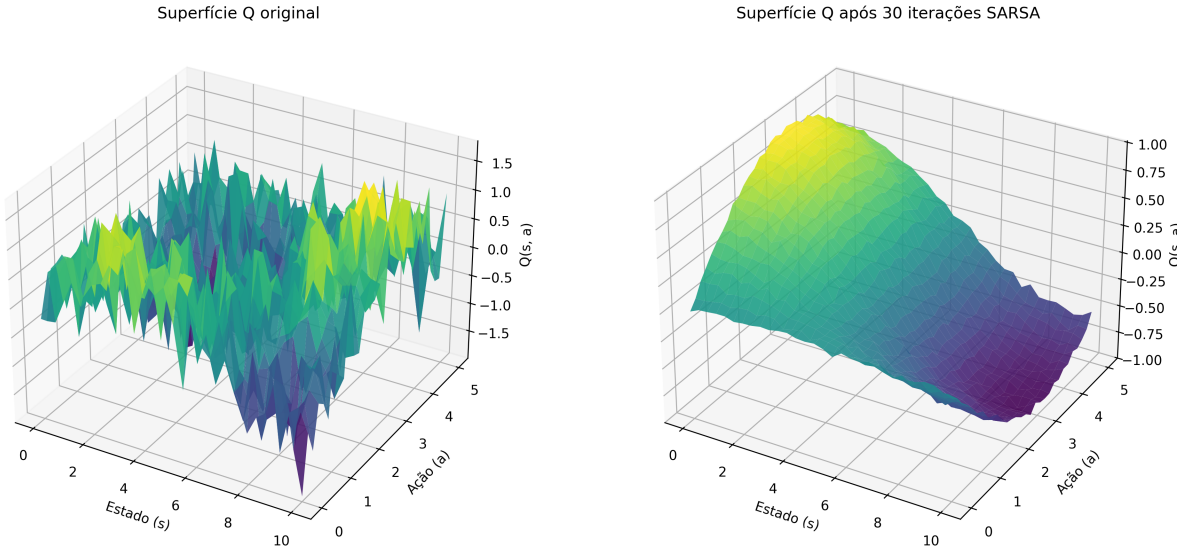


Figura 3.7: Comparação da Superfície da Função Q antes e depois das atualizações SARSA.

A visualização acima ilustra de forma clara a evolução da função Q(s, a) durante o processo de aprendizado. À esquerda, vemos a superfície Q inicial, caracterizada por irregularidades e aleatoriedade. À direita, a superfície foi suavemente ajustada após 30 iterações de atualização SARSA, aproximando-se dos valores-alvo. Esse comportamento demonstra como o algoritmo SARSA ajusta a função de valor-ação ao longo do tempo, tornando-a uma estimativa cada vez mais precisa das melhores ações para cada estado.

Experimente alterar o número de iterações em `for _ in range(30)`, o valor de alpha em `alpha = 0.1` ou até mesmo a equação `Q_target`, e observe como essas modificações influenciam o processo de aprendizado! SARSA aprende de forma gradual, com base na política seguida, e a mudança na superfície ao longo do tempo evidencia essa dinâmica.

3.6.2 Variações de SARSA

Expected SARSA

Em vez de usar $Q(S_{t+1}, A_{t+1})$ para a ação amostrada, utiliza-se o valor esperado sob a política π para calcular a atualização. Isso significa que, em vez de depender de uma única ação amostrada, o algoritmo leva em consideração a média ponderada das recompensas esperadas para todas as ações possíveis no estado seguinte.

Essa abordagem reduz a variância nas atualizações, pois não está sujeita às flutuações que poderiam ocorrer se fosse usada apenas uma amostra de ação. Ao calcular a expectativa, proporciona um aprendizado mais estável e eficiente, especialmente em ambientes com alta variabilidade ou incerteza nas escolhas de ação.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)]$$

SARSA(λ) - eligibility traces

Introduz o conceito de trilhas de elegibilidade para propagar o erro de *Temporal Difference (TD)* não apenas para o último passo, mas também para múltiplos passos anteriores, de acordo com o parâmetro λ. A cada iteração, as trilhas de elegibilidade e os valores $Q(S_t, A_t)$

são atualizados para todos os pares estado-ação que são considerados ”elegíveis”, ou seja, que ainda têm um traço significativo de elegibilidade.

Essa abordagem acelera o aprendizado, pois permite que o agente ajuste seus valores de $Q(s, a)$ de maneira mais eficiente, levando em consideração o impacto de decisões passadas no comportamento presente. Ao combinar as ideias do aprendizado por TD, que considera a diferença entre valores de estados consecutivos, com a visão mais global dos métodos de Monte Carlo, que aprendem após observar episódios completos, o SARSA com trilhas de elegibilidade alcança uma convergência mais rápida e estável.

3.7 Caso Prático: CartPole

Iniciando o diálogo...

Imagine que você está empurrando um carrinho sobre trilhos, com um bastão preso a ele por uma articulação livre. Seu objetivo é manter o bastão sempre na posição vertical, movendo o carrinho para a esquerda ou para a direita. Embora a tarefa pareça simples à primeira vista, ela exige decisões rápidas e precisas a cada fração de segundo, pois qualquer erro pode fazer o bastão cair. No estudo de caso ”CartPole”, vamos explorar como um agente de Aprendizagem por Reforço pode aprender a realizar exatamente essa tarefa: equilibrar o bastão (pole) sobre o carrinho (cart). Para isso, utilizaremos o ambiente `CartPole-v1` da Farama Foundation’s Gymnasium, um fork da Gym da OpenAI, que é uma plataforma poderosa para criar, treinar e avaliar algoritmos de aprendizado de máquina em diversos cenários dinâmicos e desafiadores.

3.7.1 Descrição do Problema

- **Estado**: vetor de quatro valores contínuos $s = [x, \dot{x}, \theta, \dot{\theta}]$
  - x: posição do carrinho
  - $\dot{x}$: velocidade do carrinho
  - θ: ângulo do bastão em relação à vertical
  - $\dot{\theta}$: velocidade angular do bastão
- **Ações:** espaço discreto $\{0, 1\}$
  - 0: empurrar para a esquerda
  - 1: empurrar para a direita
- **Recompensa:** +1 para cada passo de tempo em que $(-12° > \theta < 12°)$ e $(-2.4 > x < 2.4)$.

O **Episódio** termina quando o bastão cai $(-12° \leq \theta \geq 12°)$, o carrinho sai dos limites $(-2.4 \leq x \geq 2.4)$ ou, como no código abaixo, atinge 10 passos.

Fique Alerta!

A recompensa é sempre +1 por passo bem-sucedido. Não há “recompensa negativa” explícita, o sinal de falha está no término abrupto do episódio.

Vamos chamar o ambiente e executar um agente que escolhe ações aleatoriamente, apenas para observar o funcionamento.

Copie e Teste!

```
import gymnasium as gym

# Criação do ambiente CartPole
ambiente = gym.make("CartPole-v1", render_mode=None)

# Reinicia o ambiente e obtém o estado inicial
estado, info = ambiente.reset()

print("Iniciando simulação do CartPole!\n")

# Loop de interação com o ambiente
for passo in range(10):
    # Escolha de ação aleatória: 0 (esquerda) ou 1 (direita)
    acao = ambiente.action_space.sample()

    # Executa a ação e coleta o próximo estado e informações
    proximo_estado, recompensa, terminou, truncado, info =
   ambiente.step(acao)

    # O vetor de estado contém:
    posicao_carrinho = proximo_estado[0]
    velocidade_carrinho = proximo_estado[1]
    angulo_bastao = proximo_estado[2]
    velocidade_angular_bastao = proximo_estado[3]

    angulo_em_graus = angulo_bastao * 180 / 3.1416

    print(f"Passo {passo + 1}")
    print(f"  Ação realizada: {'Esquerda (0)' if acao == 0 else '
   Direita (1)'}")
    print(f"  Posição do carrinho (x): {posicao_carrinho:.3f}")
    print(f"  Velocidade do carrinho ẋ(): {velocidade_carrinho:.3f
   }")
    print(f"  Ângulo do bastão (Theta): {angulo_bastao:.3f} rad ({
   angulo_em_graus:.1f}°)")
    print(f"  Velocidade angular do bastão (Theta): {
   velocidade_angular_bastao:.3f}")
    print(f"  Recompensa recebida: {recompensa}")

    # Dicas sobre o término do episódio
    print(f"  Episódio terminou? {'Sim' if terminou else 'Não'}")
    print(f"  Tempo máximo do episódio atingido? {'Sim' if
   truncado else 'Não'}")
    print()
```

```
    # Condição para reiniciar o episódio
    if terminou or truncado:
        estado, info = ambiente.reset()
        print("Reiniciando o ambiente, pois o bastão caiu ou o
   tempo máximo foi atingido.\n")

# Fecha o ambiente
ambiente.close()
```

Resultado Esperado

```
Iniciando simulação do CartPole!

Passo 1
  Ação realizada: Direita (1)
  Posição do carrinho (x): 0.014
  Velocidade do carrinho (ẋ): 0.215
  Ângulo do bastão (Theta): 0.030 rad (1.7°)
  Velocidade angular do bastão (Theta): -0.275
  Recompensa recebida: 1.0
  Episódio terminou? Não
  Tempo máximo do episódio atingido? Não
[...]
```

Fique Alerta!

Experimente rodar esse código várias vezes e altere a quantidade de episódios no loop de interação com o ambiente em `for passo in range(10)`, basta alterar o número!

Você verá que, em média, o carrinho equilibra o bastão por poucos passos antes de falhar. A simulação do CartPole ilustra como um agente interage com o ambiente para equilibrar o bastão sobre o carrinho. Ao longo dos passos, o agente escolhe ações aleatórias, que influenciam a posição e o ângulo do bastão. Em cada passo, o agente recebe uma recompensa de +1 quando consegue manter o bastão equilibrado e dentro dos limites estabelecidos (como o ângulo de 12° e a posição do carrinho entre -2.4 e 2.4).

Entretanto, o desafio de manter o bastão equilibrado por um período prolongado é grande. Mesmo com ações aleatórias, o agente falha frequentemente, pois não há aprendizado ou estratégia definida para otimizar suas ações. Ao rodar o código várias vezes, o comportamento observado é consistente: o carrinho consegue equilibrar o bastão por alguns passos, mas logo o episódio termina devido à falha do agente em manter o equilíbrio.

Este comportamento é esperado, pois o agente ainda não foi treinado para aprender a política de ações que maximiza sua recompensa ao longo do tempo. Para melhorar o desempenho, seria necessário implementar algoritmos de aprendizado por reforço, como o *Q-learning* ou *Deep Q-Networks (DQN)*, que possibilitam ao agente aprender com a experiência e otimizar suas decisões.

Fique Alerta!

No CartPole, o estado é contínuo, discretizar ou usar Aprendizado por Reforço Profundo é mais eficiente.

Conhecendo um pouco mais!

A documentação completa desse experimento se encontra aqui `https://gymnasium.farama.org/environments/classic_control/cart_pole/`

3.8 Aprendizagem por Reforço Profundo

Iniciando o diálogo...

E se um agente, como um robô ou um personagem de jogo, pudesse aprender sozinho, apenas interagindo com o ambiente, qual é a melhor ação a tomar em cada situação? Na Aprendizagem por Reforço Profundo, isso se torna possível combinando *Reinforcement Learning* clássico com redes neurais profundas. Nesta seção, vamos explorar de forma intuitiva e gráfica como funcionam as *Deep Q-Networks*, um dos métodos mais famosos dessa área. Prepare-se para entender como esses modelos conseguem aprender estratégias sofisticadas apenas com tentativa, erro e muita matemática por trás.

3.8.1 Da Tabela Q à Rede Neural

Em *Q-Learning* clássico, mantemos uma tabela Q que armazena valores $Q(s, a)$ para cada par estado-ação. Quando o número de estados ou ações é muito grande, ou ainda, quando são contínuos, a representação da função $Q(s, a)$ por meio de uma tabela se torna impraticável ou até impossível, devido ao tamanho e à complexidade envolvidos.

A solução proposta pelo *Deep Q-Networks* é substituir essa tabela por uma função aproximadora $Q(s, a; \theta)$, onde θ representa os parâmetros de uma rede neural. Essa rede aprende a estimar os valores Q diretamente a partir das observações, permitindo que o agente generalize seu aprendizado para estados nunca vistos, de forma mais escalável e eficiente.

3.8.2 Arquitetura Básica de uma Deep Q-Network (DQN)

- **Entrada:** O agente recebe como entrada um vetor de características do estado atual s. Esse vetor pode representar, por exemplo, pixels de uma imagem como no ambiente CartPole ou variáveis numéricas descritivas.
- **Rede Neural:** O vetor de entrada passa por uma rede neural profunda, que pode ser composta por:
  - **Camadas totalmente conectadas:** para dados tabulares ou vetores;
  - **Camadas convolucionais:** especialmente úteis para dados visuais, como imagens de jogos.
- **Saída:** A camada final contém um neurônio para cada ação possível. Cada saída representa a estimativa $Q(s, a_i)$: o valor esperado da recompensa ao executar a ação a_i no estado s.

Fique Alerta!

Visualmente, essa rede aproxima a função Q ao longo do tempo de treinamento.

3.8.3 Atualização dos Pesos: Minimizar o Erro de Bellman

Em uma DQN, o processo de aprendizado acontece ajustando os pesos da rede neural para melhorar a estimativa da função $Q(s, a)$. Para isso, usamos como referência o alvo de Bellman, que representa uma estimativa mais atualizada do valor esperado para cada ação:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

Aqui, θ^- são os parâmetros da rede-alvo, que é uma cópia periódica da rede principal com parâmetros θ. O erro de predição (ou erro de Bellman) mede a diferença entre o valor estimado pela rede principal e o alvo calculado com a rede-alvo:

$$\delta = y - Q(s, a; \theta)$$

A função de perda utilizada é o erro quadrático médio (MSE):

$$L(\theta) = \frac{1}{2}[y - Q(s, a; \theta)]^2$$

Para reduzir esse erro e melhorar as estimativas, a rede principal θ é atualizada usando gradiente descendente:

$$\theta \leftarrow \theta + \alpha \delta \nabla_\theta Q(s, a; \theta)$$

3.8.4 Componentes Essenciais de uma DQN

- **Replay Buffer (Memória de Repetição):** Armazena transições (s, a, r, s') ao longo do tempo. Durante o treinamento, são amostrados *mini-batches* aleatórios, o que reduz a correlação temporal entre amostras e estabiliza o aprendizado.
- **Rede-Alvo (Target Network)** θ^-: É uma cópia da rede principal, usada para calcular os alvos y. Ela é atualizada a cada N iterações, copiando os parâmetros da rede principal. Isso evita oscilações e instabilidades no aprendizado.
- **Exploração vs. Explotação (ϵ-greedy):** Com probabilidade ϵ, escolhemos uma ação aleatória (*exploration*); caso contrário, a ação com maior Q (*exploitation*).

3.8.5 Fluxo de Treinamento

Para que uma DQN aprenda a tomar boas decisões, ela passa por um ciclo contínuo de interação com o ambiente, aprendizado com base nas experiências armazenadas e ajustes periódicos nos seus parâmetros. Esse processo permite que o agente melhore sua política gradualmente, mesmo em ambientes complexos e de alta dimensionalidade.

1. **Interagir com o ambiente:**
   - Observar estado s.
   - Escolher ação a (ϵ-greedy).

- Receber recompensa r e próximo estado s'.
- Armazenar (s, a, r, s') no replay buffer.

2. **Aprimorar a rede:**
   - Amostrar mini-batch de transições.
   - Calcular alvos y usando rede-alvo.
   - Computar gradientes da perda e atualizar θ.
3. **Atualizar rede-alvo:**
   - A cada K passos, definir $\theta^- \leftarrow \theta$.
4. **Repetir** o processo até que o agente convirja para uma política eficiente.

Caso Prático

CartPole com DQN
Já trabalhamos com o CartPole utilizando tabelas Q para aprender uma política de controle. Agora, vamos dar um passo adiante: aplicar o algoritmo Deep Q-Network e observar as diferenças. O ambiente é o mesmo: um carrinho deve se mover para a esquerda ou para a direita para manter uma haste em pé. O jogo termina quando a haste se inclina demais ou o carrinho sai dos limites do trilho. A grande mudança está na forma como o agente aprende, pois, em vez de usar uma tabela Q discreta, o DQN utiliza uma rede neural para aproximar a função de valor Q. Isso permite que ele lide com espaços de estado contínuos, como posição, velocidade e ângulo da haste, variáveis que mudam de forma suave e não cabem bem em uma tabela. Com o DQN, o agente aprende a tomar decisões com base em experiências anteriores, armazenadas em um buffer de memória, e melhora sua política ao longo do tempo por meio de treino supervisionado sobre essas experiências. Vamos ver, na prática, como isso funciona e comparar os resultados com o método tabular que já exploramos.

Copie e Teste!

```
import gymnasium as gym
import random
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from collections import deque
import matplotlib.pyplot as plt

# Hiperparâmetros
fator_desconto = 0.99
epsilon = 1.0
epsilon_min = 0.05
decaimento_epsilon = 0.995
taxa_aprendizado = 0.01
tamanho_memoria = 10000
```

```
tamanho_lote = 64
intervalo_atualizacao_alvo = 10
num_episodios = 300

# Ambiente
ambiente = gym.make("CartPole-v1")
dim_estado = ambiente.observation_space.shape[0]
num_acoes = ambiente.action_space.n

# Rede Q
class RedeQ(nn.Module):
    def __init__(self, entradas, saidas):
        super(RedeQ, self).__init__()
        self.camadas = nn.Sequential(
            nn.Linear(entradas, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, saidas)
        )
    def forward(self, estado):
        return self.camadas(estado)

# Inicialização
rede_q = RedeQ(dim_estado, num_acoes)
rede_alvo = RedeQ(dim_estado, num_acoes)
rede_alvo.load_state_dict(rede_q.state_dict())
otimizador = optim.Adam(rede_q.parameters(), lr=taxa_aprendizado)
memoria_replay = deque(maxlen=tamanho_memoria)
criterio_perda = nn.MSELoss()

# Política ε-greedy
def escolher_acao(estado):
    if random.random() < epsilon:
        return ambiente.action_space.sample()
    estado_tensor = torch.FloatTensor(estado).unsqueeze(0)
    with torch.no_grad():
        q_valores = rede_q(estado_tensor)
    return torch.argmax(q_valores).item()

# Loop de treinamento
recompensas_episodios = []
for episodio in range(num_episodios):
    estado, _ = ambiente.reset()
    feito = False
    recompensa_total = 0
    while not feito:
        acao = escolher_acao(estado)
        proximo_estado, recompensa, terminado, truncado, info =
   ambiente.step(acao)
```

```
        feito = terminado or truncado
        memoria_replay.append((estado, acao, recompensa,
    proximo_estado, feito))
        estado = proximo_estado
        recompensa_total += recompensa
        if len(memoria_replay) >= tamanho_lote:
            amostras = random.sample(memoria_replay, tamanho_lote)
            estados, acoes, recompensas, proximos_estados, finais
    = zip(*amostras)

            estados_tensor = torch.FloatTensor(estados)
            acoes_tensor = torch.LongTensor(acoes).unsqueeze(1)
            recompensas_tensor = torch.FloatTensor(recompensas).
    unsqueeze(1)
            proximos_tensor = torch.FloatTensor(proximos_estados)
            finais_tensor = torch.FloatTensor(finais).unsqueeze(1)

            with torch.no_grad():
                q_proximos = rede_alvo(proximos_tensor).max(1,
    keepdim=True)[0]
                alvos = recompensas_tensor + fator_desconto *
    q_proximos * (1 - finais_tensor)

            q_estimados = rede_q(estados_tensor).gather(1,
    acoes_tensor)
            perda = criterio_perda(q_estimados, alvos)

            otimizador.zero_grad()
            perda.backward()
            otimizador.step()

     if episodio % intervalo_atualizacao_alvo == 0:
        rede_alvo.load_state_dict(rede_q.state_dict())

     epsilon = max(epsilon_min, epsilon * decaimento_epsilon)
     recompensas_episodios.append(recompensa_total)
     print(f"Episódio {episodio+1}, Recompensa: {recompensa_total
    :.0f}, Epsilon: {epsilon:.3f}")

ambiente.close()

# Plot das recompensas
plt.figure(figsize=(10, 5))
plt.plot(recompensas_episodios)
plt.xlabel("Episódios")
plt.ylabel("Recompensa Total")
plt.title("Recompensa por Episódio - DQN no CartPole")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Resultado Esperado

```
Episódio 1, Recompensa: 13, Epsilon: 0.995
Episódio 2, Recompensa: 36, Epsilon: 0.990
...
Episódio 299, Recompensa: 24, Epsilon: 0.223
Episódio 300, Recompensa: 68, Epsilon: 0.222
```

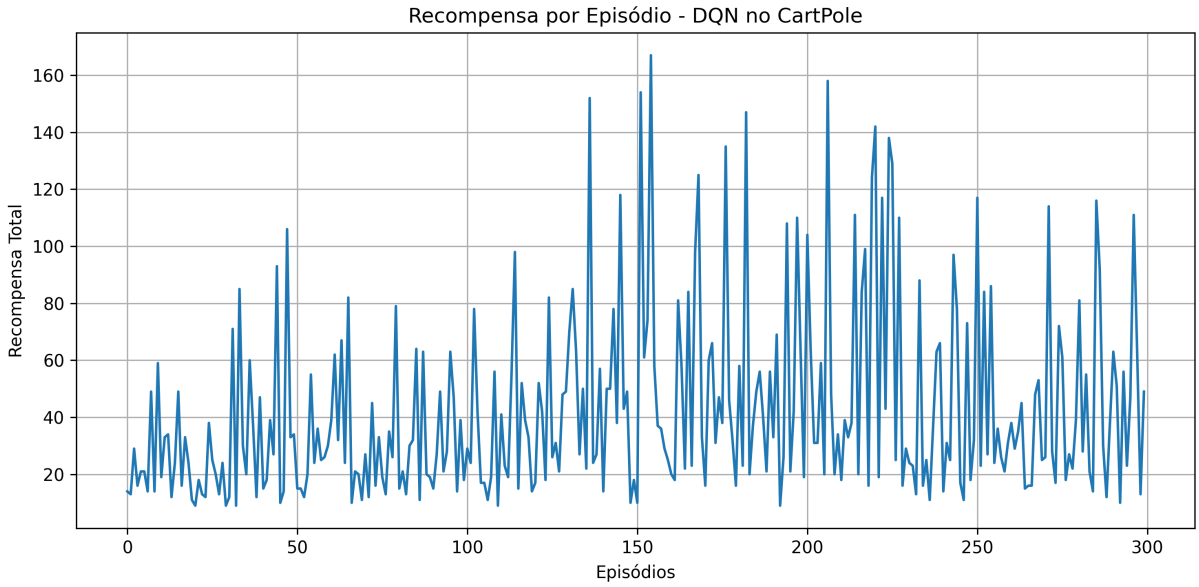


Figura 3.8: Gráfico da recompensa total por episódio durante o treinamento do agente DQN no ambiente CartPole.

Ao final de todos os episódios, o código gera um gráfico de recompensas por episódio. O eixo X representa os episódios (de 1 a 300), e o eixo Y mostra a recompensa total acumulada ao longo de cada episódio. Inicialmente, a recompensa pode ser baixa ou até mesmo 0, pois o agente está explorando o ambiente aleatoriamente (alta exploração com épsilon próximo de 1.0). Ao longo do tempo, espera-se que a recompensa aumente, indicando que o agente está começando a aprender a maximizar a recompensa, ou seja, ele aprende a manter o pêndulo equilibrado por mais tempo.

Fique Alerta!

Sem *replay buffer* e rede-alvo, o DQN tende a divergir, gerando instabilidade. Tamanho do buffer muito pequeno gera amostras correlacionadas; muito grande e demora para aprender de dados novos.

Conhecendo um pouco mais!

Vídeo “Introdução a Deep Q-Learning”
Vídeo “Deep Learning Explicado”

Como melhorar o treinamento?

- **Reduzir épsilon progressivamente:** O valor de épsilon diminui a cada episódio, o que incentiva o agente a explorar menos e a explorar mais suas ações com base na experiência acumulada. Se o agente não estiver aprendendo, você pode ajustar o decaimento do épsilon.
- **Ajustar a taxa de aprendizado:** Se o agente estiver aprendendo muito lentamente, você pode aumentar a taxa de aprendizado, ou se o treinamento estiver instável, pode diminuir.

Observe o impacto no número de passos até balancear o pêndulo. Ao final, você verá seu agente aprendendo a equilibrar o CartPole usando DQN!

Capítulo 4

Experimento Prático

Iniciando o diálogo...

Seja bem-vindo ao nosso primeiro projeto de Aprendizado de Máquina com aplicação prática na agricultura amazônica! Neste experimento, vamos trabalhar com dados reais sobre a produtividade do guaraná, relacionando fatores climáticos, sazonais e fenômenos como o El Niño (ENSO). Nosso objetivo é prever a produtividade das safras e classificá-las como baixa, média ou alta, utilizando técnicas de Aprendizado Supervisionado e, ao final, um toque de Aprendizado por Reforço.

4.1 Etapas iniciais

Antes de começar qualquer análise ou modelagem, precisamos preparar o ambiente de trabalho. Nesta etapa inicial, vamos:

- Importar as bibliotecas essenciais para análise de dados e visualização;
- Verificar o dataset que será utilizado.

Copie e Teste!

```
# Etapa 1 — Importação de bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

4.1.1 O dataset

Abaixo está o dataset que utilizaremos neste experimento. Lembre-se de que a primeira linha contém os identificadores (nomes) das colunas.

```
ano;chuva_durante_floração_mm;chuva_durante_colheita_mm;
chuva_total_anual_mm;...
1984;423,20;516,70;2459,80;...
...
```

4.1.2 Entendendo o problema

Agora que configuramos o ambiente, é hora de mergulhar no problema que queremos resolver. Neste projeto, vamos trabalhar com um conjunto de dados reais sobre a produção agrícola de guaraná no Amazonas, contendo informações sobre:

- Fatores climáticos anuais (chuva, temperatura, umidade);
- Fases do cultivo (floração e colheita);
- Classificação das safras (baixa, média ou alta produtividade);
- Eventos sazonais como o El Niño (ENSO).

O que vamos investigar?

Nesta investigação, nosso objetivo é utilizar o conjunto de dados para desenvolver modelos preditivos com duas finalidades centrais: primeiramente, prever o valor exato da produtividade da safra, o que configura uma tarefa de regressão; e, em segundo lugar, classificar o rendimento da safra em categorias como baixa, média ou alta, uma tarefa de classificação. Contudo, antes da etapa de modelagem, é fundamental realizar uma análise exploratória aprofundada para compreender as variáveis disponíveis, as relações entre elas e a motivação que contextualiza a necessidade dessas previsões.

A importância estratégica da previsão de produtividade

A aplicação de modelos de regressão para prever a produtividade agrícola transcende o exercício acadêmico; ela se estabelece como uma ferramenta de inteligência estratégica para toda a cadeia produtiva. Conforme descrito por Géron (2023), a capacidade de antecipar o volume de uma safra permite um planejamento muito mais eficiente da produção, da logística e da comercialização de culturas de alto valor, como o guaraná, especialmente em regiões com desafios logísticos únicos como o Amazonas.

Essa previsibilidade gera impactos positivos em múltiplos níveis. Para os agricultores, compreender como fatores climáticos e de manejo afetam a produtividade futura permite otimizar o uso de insumos, como irrigação e adubação, e planejar o momento ideal da colheita para maximizar a qualidade e o rendimento. Para gestores públicos e cooperativas, previsões acuradas são vitais para antecipar flutuações na oferta, permitindo a criação de políticas de estabilização de preços, o planejamento de estoques de segurança e a organização da logística de escoamento da produção. Por fim, para pesquisadores e técnicos, esses modelos são fundamentais para validar hipóteses e identificar as complexas relações não-lineares entre clima, solo e práticas agrícolas, fomentando a inovação e o desenvolvimento de culturas mais resilientes.

O caso do guaraná: ciência de dados aplicada à bioeconomia amazônica

O guaraná (Paullinia cupana) não é apenas uma cultura agrícola; é um pilar da bioeconomia e da identidade cultural da região Norte do Brasil, sendo a principal fonte de renda para milhares de agricultores familiares. Aplicar ciência de dados para modelar sua produtividade com base em dados históricos é, portanto, uma forma poderosa de conectar a tecnologia à realidade local, gerando valor e sustentabilidade.

A produção do guaraná é notoriamente sensível às condições climáticas, tornando-a um objeto de estudo ideal para este projeto. A vulnerabilidade da cultura se manifesta de várias formas:

Regime de chuvas: A precipitação em períodos críticos, como a floração e a frutificação, é determinante. Chuvas excessivas podem prejudicar a polinização, enquanto a falta de água pode estressar a planta e reduzir drasticamente o número de frutos.

Variações de temperatura: As temperaturas médias ao longo do ciclo produtivo influenciam o desenvolvimento da planta. Ondas de calor ou frio fora de época podem dessincronizar seus estágios fenológicos e impactar negativamente o resultado da safra.

Eventos climáticos extremos: Fenômenos macroclimáticos como o El Niño (ENSO) representam uma ameaça direta, pois alteram drasticamente os padrões de chuva e temperatura na Amazônia, frequentemente resultando em secas severas que podem levar a perdas massivas na produção.

O objetivo é desenvolver e validar modelos de machine learning que não apenas aprendem os padrões do passado, mas que também se tornem ferramentas práticas para prever e classificar a produtividade de safras futuras, oferecendo um suporte mais robusto à tomada de decisão para todos os envolvidos na cadeia produtiva.

4.1.3 Variáveis do dataset e suas relações

Vamos agora carregar o conjunto de dados com as informações climáticas e de produtividade do guaraná. Esses dados serão a base para todas as etapas de análise e modelagem ao longo do projeto.

Copie e Teste!

```
df = pd.read_csv("dataset_produtividade_guarana.csv", sep=';',
   decimal=',')
```

Renomeando as colunas

Algumas colunas do dataset original possuem nomes longos, por isso vamos renomear para facilitar a leitura e o uso nos gráficos e modelos.

Copie e Teste!

```
df.rename(columns={
    'chuva_durante_floração_mm': 'chuva_flor',
    'chuva_durante_colheita_mm': 'chuva_colheita',
    'chuva_total_anual_mm': 'chuva_total',
    'anomalia_chuva_floração_mm': 'anomalia_flor',
    'temperatura_média_floração_C': 'temp_flor',
    'umidade_relativa_média_floração_%': 'umid_flor',
    'evento_ENSO': 'ENSO',
    'produtividade_kg_por_ha': 'produtividade',
    'produtividade_safra': 'safra'
}, inplace=True)
```

Ajustes adicionais

A variável de umidade estava expressa em porcentagem. Vamos convertê-la para escala fracionária (ex: 80% → 0.80). Também definimos o ano como índice da tabela, pois ele será útil para análises temporais.

Copie e Teste!

```
df['umid_flor'] = df['umid_flor'] / 100
df.set_index('ano', inplace=True)
df.head()
```

Resultado Esperado

```
      chuva_flor  chuva_colheita  chuva_total  anomalia_flor
ano
1984       423.2           516.7       2459.8           93.4
1985       363.0           540.0       2773.6           33.2
1986       320.6           666.9       2694.5           -9.2
1987       360.0           180.4       2165.2           30.2
1988       419.4           693.8       2932.8           89.6

   temp_flor   umid_flor     ENSO   produtividade   safra
    26.16       0.8425   Neutro        110     media
    25.78       0.8409   Neutro        107     media
    25.46       0.8424   El Niño       109     media
    26.91       0.8656   El Niño        74     baixa
    27.80       0.8559   La Niña       102     media
```

Tarefas de aprendizado supervisionado e objetivos do projeto

Com o dataset carregado e compreendido, podemos agora definir as duas tarefas principais que resolveremos ao longo deste projeto.

- **Regressão: onde vamos prever a produtividade.** Nesta tarefa, o objetivo será construir um modelo que receba como entrada variáveis climáticas e sazonais e gere como saída a produtividade estimada em kg/ha. Isso nos permitirá prever a produtividade com base em cenários climáticos futuros, auxiliando o planejamento agrícola.
- **Classificação, onde vamos categorizar a safra.** Nesta tarefa, vamos transformar a produtividade em categorias discretas: Baixa, Média e Alta. O modelo de classificação tentará prever o rótulo da safra, o que é útil para: diagnósticos rápidos, alertas sazonais e análises qualitativas.

Objetivo final

Ao longo do projeto, vamos construir modelos supervisionados para essas duas tarefas, avaliar seus desempenhos e visualizar seus comportamentos. Além disso, encerraremos com um desafio opcional de Aprendizado por Reforço, que explora decisões sequenciais aplicadas à irrigação. Agora que o problema está compreendido, vamos explorar os dados!

4.2 Análise Exploratória dos Dados (EDA)

Antes de treinar qualquer modelo, é fundamental entender o comportamento dos dados, pois a Análise Exploratória de Dados nos ajuda a:

- Visualizar a distribuição das variáveis;
- Detectar padrões, tendências e possíveis *outliers*;
- Investigar relações entre clima e produtividade;
- Identificar quais fatores parecem mais influentes.

Nesta etapa, vamos construir diversos gráficos, como dispersões entre variáveis climáticas e produtividade, *boxplots* comparando categorias (ex.: ENSO), histogramas para entender a distribuição dos dados, correlações entre variáveis e componentes principais (PCA) para redução de dimensionalidade. Esses gráficos serão essenciais para formular hipóteses e escolher abordagens adequadas de modelagem.

4.2.1 Diagnóstico inicial do dataset

Vamos começar explorando a estrutura geral do conjunto de dados. Nesta etapa inicial, buscamos responder perguntas como:

- Quantas observações existem?
- Quais são os tipos de variáveis?
- Há dados ausentes?
- Como se comportam os valores mínimos, máximos e médios?

Para isso, utilizaremos os métodos `.info()`, `.isnull()` e `describe()`, que nos fornecem um panorama estatístico e estrutural do dataframe.

Copie e Teste!

```
# Ver informações gerais do dataframe
df.info()

# Verificar valores ausentes
print("\nValores ausentes por coluna:")
print(df.isnull().sum())

# Resumo estatístico
df.describe().T
```

Resultado Esperado

```
<class 'pandas.core.frame.DataFrame'>
Index: 40 entries, 1984 to 2023
Data columns (total 9 columns):
```

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   chuva_flor       40 non-null     float64
 1   chuva_colheita   40 non-null     float64
 2   chuva_total      40 non-null     float64
 3   anomalia_flor    40 non-null     float64
 4   temp_flor        40 non-null     float64
 5   umid_flor        40 non-null     float64
 6   ENSO             40 non-null     object
 7   produtividade    40 non-null     int64
 8   safra            40 non-null     object
dtypes: float64(6), int64(1), object(2)
memory usage: 3.1+ KB

Valores ausentes por coluna:
chuva_flor         0
chuva_colheita     0
chuva_total        0
anomalia_flor      0
temp_flor          0
umid_flor          0
ENSO               0
produtividade      0
safra              0
dtype: int64
```

	count	mean	std	min	25%	50%	75%	max
chuva_flor	40.0	350.110000	103.369868	132.6000	282.900000	360.9000	420.35000	657.5000
chuva_colheita	40.0	467.175000	177.934808	39.7000	363.650000	489.3500	580.25000	756.9000
chuva_total	40.0	2532.797500	382.051048	1889.4000	2243.575000	2469.4500	2826.85000	3432.9000
anomalia_flor	40.0	20.310000	103.369868	-197.2000	-46.900000	31.1000	90.55000	327.7000
temp_flor	40.0	27.313000	0.949186	24.8300	26.895000	27.5050	27.91000	28.6300
umid_flor	40.0	0.820178	0.037551	0.7468	0.787375	0.8289	0.85285	0.8787
produtividade	40.0	137.300000	70.966298	50.0000	98.750000	123.0000	157.75000	399.0000

Figura 4.1: Resumo estatístico das variáveis numéricas do conjunto de dados.

4.2.2 Gráficos exploratórios

Agora que conhecemos a estrutura geral dos dados, vamos utilizar visualizações para entender melhor o comportamento das variáveis. É bom sempre lembrar que a análise gráfica permite detectar padrões e tendências, comparar produtividades em diferentes condições climáticas, verificar a distribuição dos dados e explorar relações entre variáveis contínuas e categóricas.

Boxplot: ENSO × Produtividade

O fenômeno climático El Niño (ENSO) influencia diretamente os padrões de chuva e temperatura, impactando a produtividade agrícola. No gráfico a seguir, comparamos os níveis de produtividade em anos classificados como El Niño, La Niña ou Neutro. O *boxplot* é especialmente útil para visualizar tendências associadas a cada evento, a variação interna de produtividade em cada grupo e a presença de possíveis outliers.

Copie e Teste!

```
sns.set(style="whitegrid", palette="colorblind")
sns.boxplot(
    data=df,
    x='ENSO',
    y='produtividade',
    order=['La Niña', 'Neutro', 'El Niño']
)
plt.title('Produtividade vs. Evento ENSO', fontsize=14)
plt.xlabel('Evento ENSO', fontsize=12)
plt.ylabel('Produtividade (kg/ha)', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```

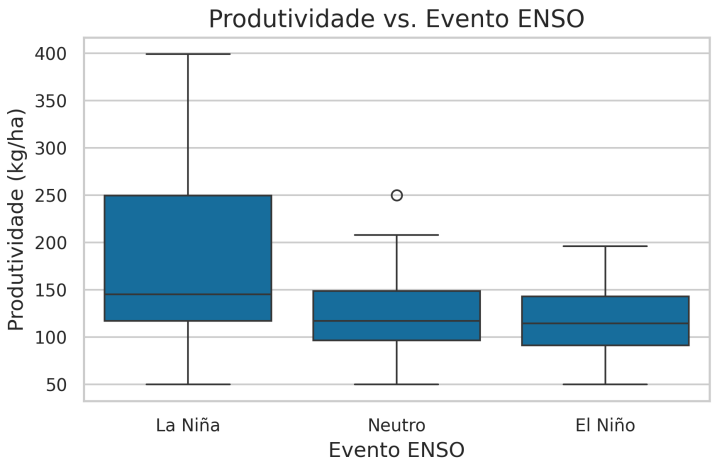


Figura 4.2: Boxplot comparando a distribuição da produtividade agrícola sob diferentes fases do fenômeno ENSO (El Niño-Southern Oscillation). Cada caixa representa a dispersão dos dados, incluindo a mediana (linha central), os quartis (bordas da caixa) e os valores atípicos (pontos), permitindo uma análise visual do impacto de cada evento na safra.

Scatter: Temperatura × Produtividade

Agora, vamos explorar a relação entre a temperatura média durante a floração e a produtividade da safra. No gráfico de dispersão que será gerado com o código abaixo, verá que cada ponto representa um ano, e a cor de cada ponto indica o tipo de evento ENSO. Observe

atentamente os padrões visuais para identificar:

- As faixas de temperatura que parecem mais adequadas para otimizar a produtividade;
- Como os diferentes eventos ENSO (El Niño, La Niña, Neutro) afetam a produtividade da safra.

A partir dessa análise, você poderá tirar conclusões sobre as condições climáticas ideais para o cultivo e os impactos dos eventos climáticos no rendimento agrícola.

Copie e Teste!

```
sns.scatterplot(data=df, x='temp_flor', y='produtividade',
                hue='ENSO', s=80, alpha=0.8)
plt.title('Temperatura durante floração vs. Produtividade',
   fontsize=14)
plt.xlabel('Temperatura média durante floração (°C)', fontsize=12)
plt.ylabel('Produtividade (kg/ha)', fontsize=12)
plt.legend(title='Evento ENSO')
```

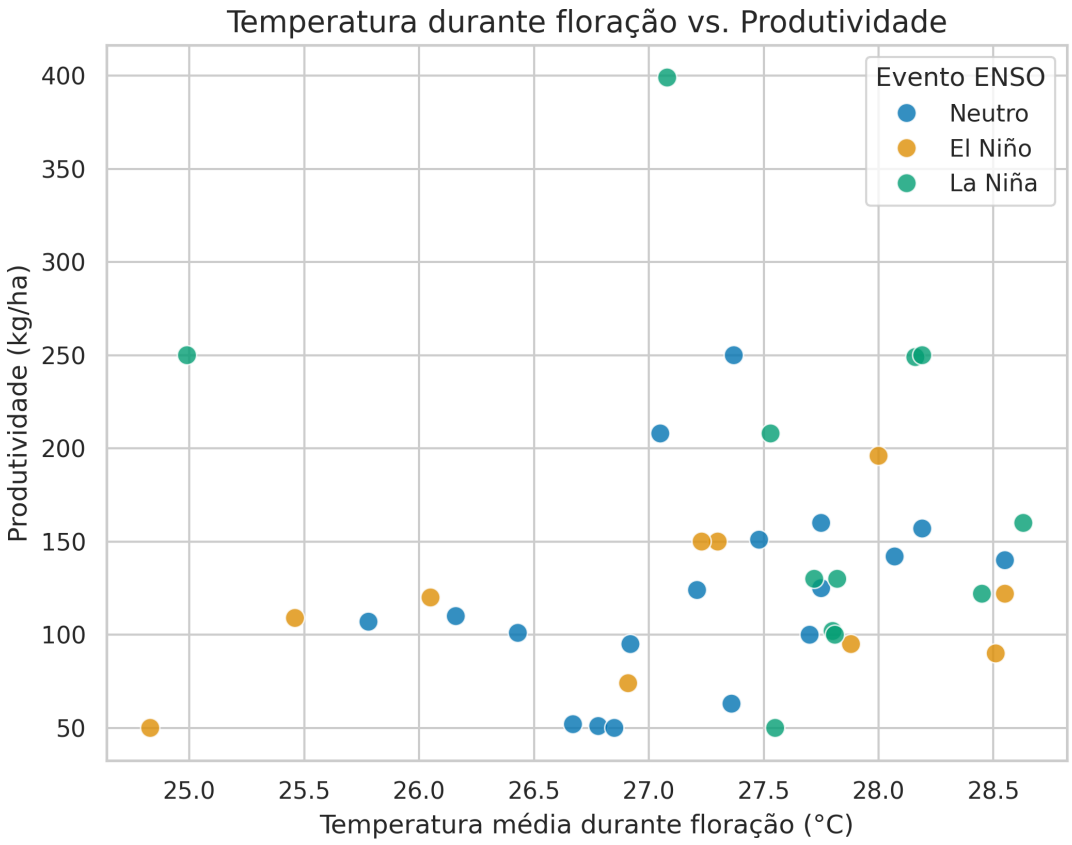


Figura 4.3: Relação entre a temperatura na floração e a produtividade, segmentado por eventos ENSO.

Histogramas de variáveis numéricas

Agora, vamos analisar os histogramas que mostram a distribuição das variáveis numéricas do dataset. Observe atentamente os gráficos e busque identificar:

- Assimetrias ou concentrações nas distribuições das variáveis;
- Valores extremos (outliers) que possam indicar dados atípicos;
- As faixas mais frequentes para variáveis como temperatura, umidade, chuva e produtividade.

Essas observações são fundamentais para entender a distribuição dos dados e podem ajudar a identificar tendências ou problemas no dataset.

Copie e Teste!

```
df.select_dtypes(include='number').hist(bins=15, figsize=(12,8))
plt.suptitle("Distribuições das variáveis numéricas")
plt.show()
```

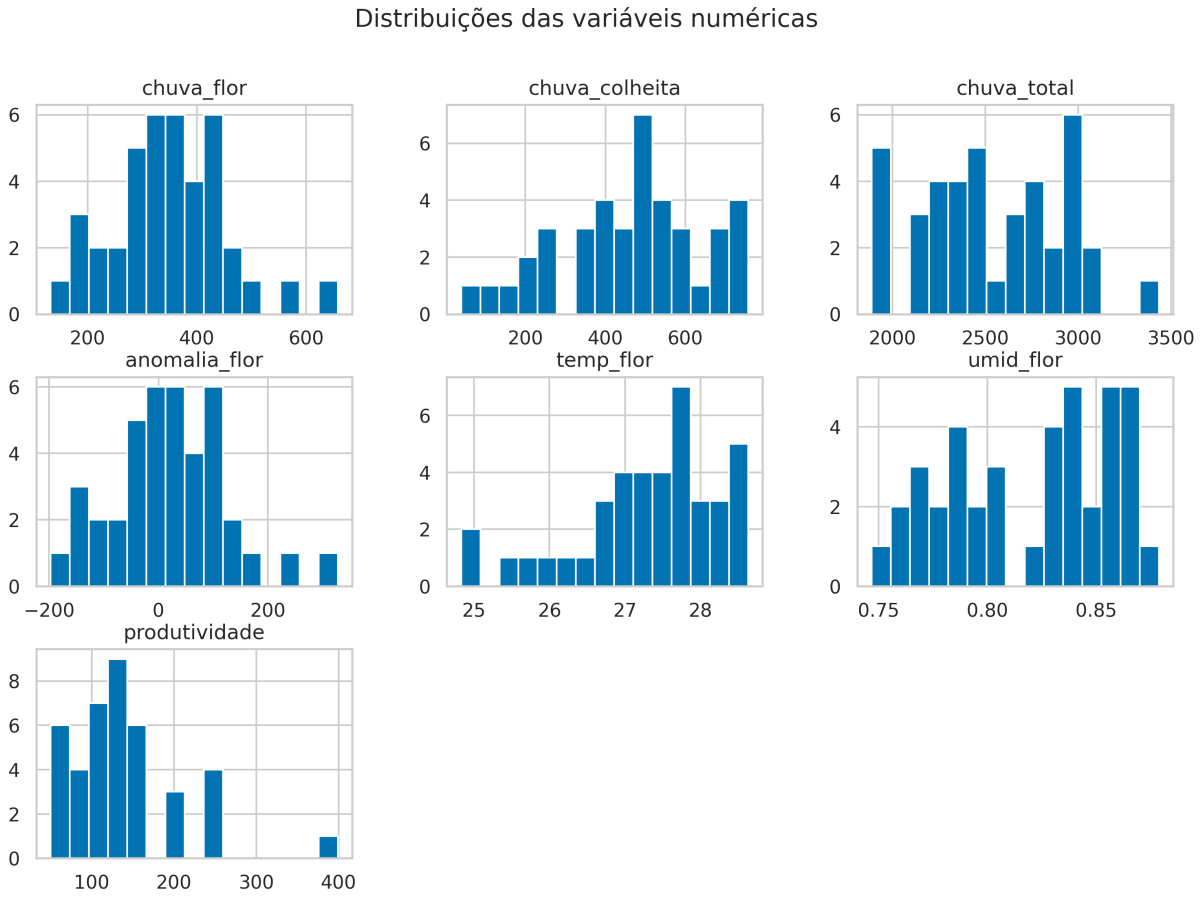


Figura 4.4: Histogramas de frequência para as variáveis numéricas do conjunto de dados, utilizados na análise exploratória inicial.

Heatmap de Correlação

Agora vamos analisar a correlação entre as variáveis numéricas do dataset utilizando um mapa de calor (*heatmap*). Ao observar o heatmap, identifique:

- Relações fortes, tanto positivas quanto negativas, entre as variáveis;
- Variáveis redundantes, ou seja, aquelas com alta correlação entre si;
- Fatores que possam influenciar diretamente a produtividade.

Essa análise é essencial para selecionar as variáveis mais relevantes para a modelagem e evitar problemas como a multicolinearidade, que pode afetar a precisão dos modelos.

Copie e Teste!

```
# Seleciona só as colunas numéricas relevantes
variaveis_numericas = df.select_dtypes(include='number')

# Calcula a matriz de correlação
correlacao = variaveis_numericas.corr()

# Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(
    correlacao,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    linewidths=0.5,
    square=True,
    cbar_kws={"shrink": .8},
    vmin=-1, vmax=1
)
plt.title('Matriz de Correlação entre Variáveis Numéricas')
plt.tight_layout()
plt.show()
```

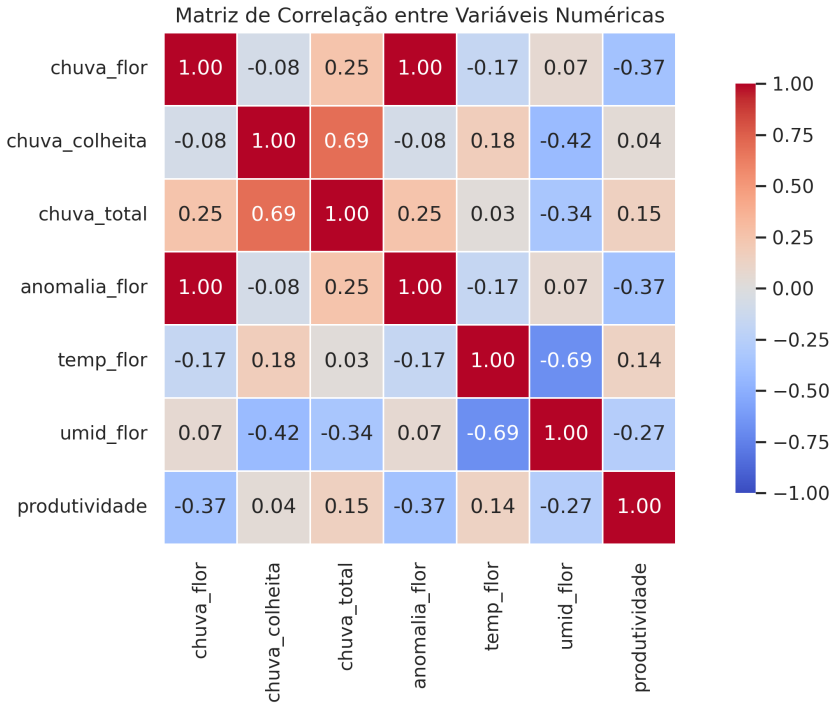


Figura 4.5: Heatmap da matriz de correlação de Pearson entre as variáveis numéricas. A intensidade e a cor de cada célula representam a força e a direção (positiva ou negativa) da relação linear entre um par de variáveis.

Pairplot

No gráfico a seguir, combinaremos múltiplas dispersões e histogramas em uma única visualização, com o objetivo de visualizar pares de variáveis simultaneamente, facilitando a análise de suas relações, observar como a produtividade se distribui em relação a cada uma das variáveis e identificar agrupamentos visuais de acordo com o tipo de safra (baixa, média, alta). É útil como uma etapa preliminar para avaliar a relação entre as variáveis antes de aplicar técnicas mais avançadas, como PCA (Análise de Componentes Principais) ou regressão.

Copie e Teste!

```
# Seleciona as variáveis numéricas (sem o ano)
cols_plot = ['chuva_flor', 'chuva_colheita', 'chuva_total',
             'anomalia_flor', 'temp_flor', 'umid_flor', '
   produtividade']

sns.pairplot(
    df[cols_plot],
    corner=True,      # evita duplicação acima/abaixo da diagonal
    diag_kind='hist', # ou 'kde'
    plot_kws={'alpha': 0.7, 's': 40, 'edgecolor': 'k'}
)
plt.suptitle("Matriz de Dispersão entre Variáveis", fontsize=14, y
   =1.02)
plt.show()
```

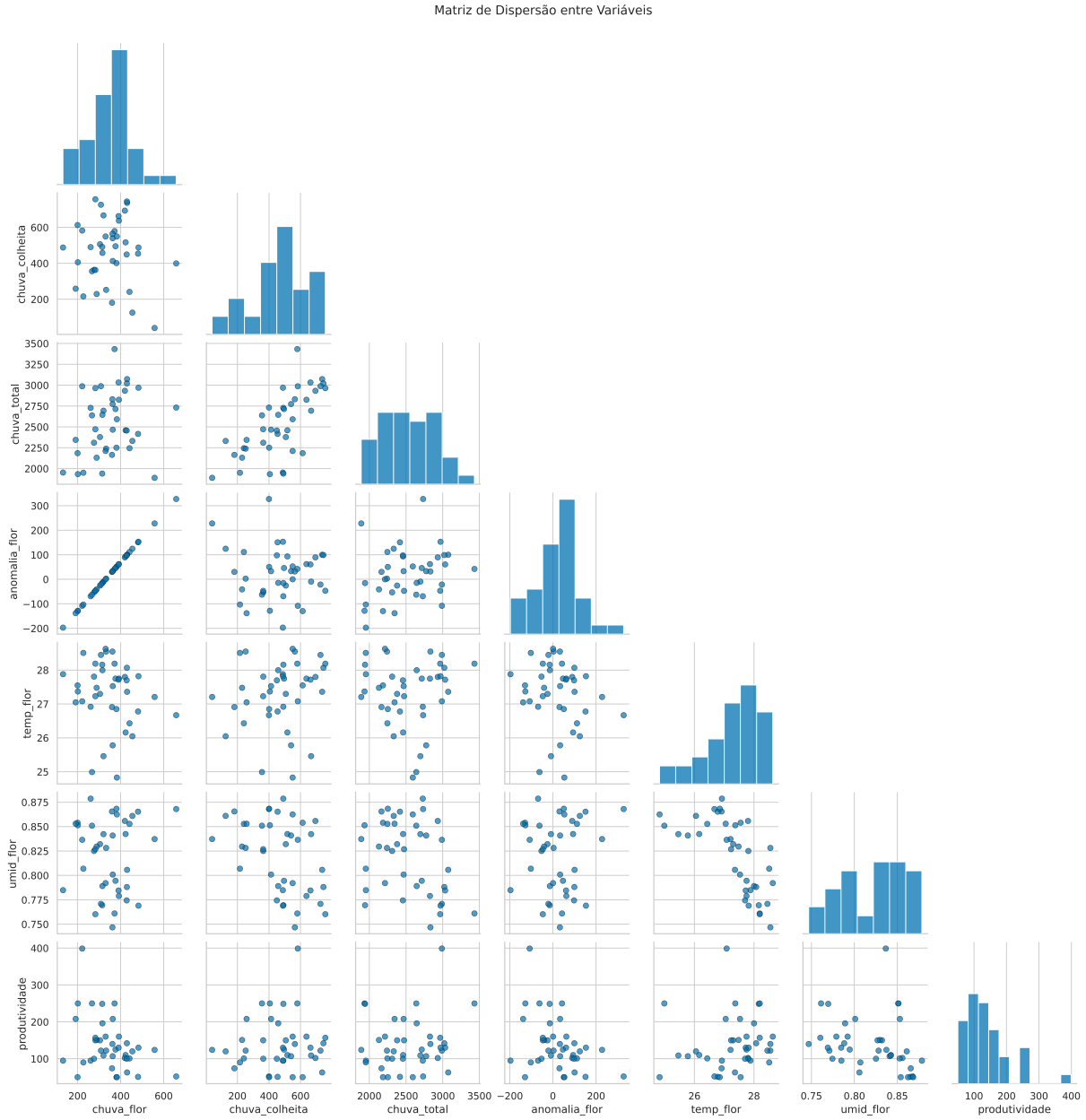


Figura 4.6: Matriz de gráficos de dispersão (pairplot) para análise das relações bivariadas e distribuições univariadas. Os gráficos de dispersão (fora da diagonal) mostram a correlação entre pares de variáveis, enquanto os histogramas (na diagonal) ilustram a distribuição de cada variável individualmente.

4.3 Engenharia de Atributos

Após explorar visualmente os dados, é hora de preparar as variáveis para os modelos de Machine Learning. Esse processo é conhecido como engenharia de atributos e envolve diversas etapas, como transformações de variáveis, criação de novas variáveis, codificação de variáveis categóricas e representações numéricas para variáveis categóricas. O objetivo dessa preparação é fornecer aos modelos entradas mais informativas e organizadas, mantendo a interpretação original das variáveis. Com isso, conseguimos melhorar a qualidade dos dados, sem perder o contexto. Aqui, vamos abordar três pontos principais: a criação de uma variável

de chuva relativa, a codificação do evento ENSO em variáveis binárias e a transformação da variável-alvo da classificação (safra) em valores numéricos.

4.3.1 Criação de variáveis sazonais e climáticas

A variável `chuva_relativa` representa a fração da chuva total anual que ocorreu durante a floração. Essa abordagem pode ser mais informativa do que simplesmente usar os valores absolutos de chuva, pois permite uma comparação mais direta da influência da chuva na fase crítica para a produtividade da safra.

Copie e Teste!

```
# 1. Chuva relativa durante floração
df['chuva_relativa'] = df['chuva_flor'] / df['chuva_total']

# 2. Binário: anomalia positiva ou não
df['anomalia_bin'] = (df['anomalia_flor'] > 0).astype(int)
```

4.3.2 Codificação de variáveis categóricas (ENSO)

A variável ENSO representa o tipo de evento climático do ano, podendo ser classificado como Neutro, El Niño ou La Niña. Como os modelos de machine learning exigem que todas as variáveis sejam numéricas, precisamos converter essas variáveis categóricas em números.

Para fazer essa conversão, utilizamos a técnica de codificação *one-hot (dummies)*, que cria colunas binárias para cada categoria. No caso da variável ENSO, geraremos duas novas colunas: `ENSO_La Niña` e `ENSO_Neutro`. Essas colunas terão valores 0 ou 1, representando a presença ou ausência de cada evento climático em um determinado ano.

Copie e Teste!

```
# 3. Codificar ENSO como variáveis dummies
df = pd.get_dummies(df, columns=['ENSO'], drop_first=True) # cria
   ENSO_El Niño e ENSO_La Niña

df.head(10)
df.filter(like='ENSO').tail(10)
```

ano	chuva_flor	chuva_colheita	chuva_total	anomalia_flor	temp_flor	umid_flor	produtividade	safra	chuva_relativa	anomalia_bin	ENSO_La Niña	ENSO_Neutro
1984	423.2	516.7	2459.8	93.4	26.16	0.8425	110	media	0.172047	1	False	True
1985	363.0	540.0	2773.6	33.2	25.78	0.8409	107	media	0.130877	1	False	True
1986	320.6	666.9	2694.5	-9.2	25.46	0.8424	109	media	0.118983	0	False	False
1987	360.0	180.4	2165.2	30.2	26.91	0.8656	74	baixa	0.166266	1	False	False
1988	419.4	693.8	2932.8	89.6	27.80	0.8559	102	media	0.143003	1	True	False
1989	480.9	454.1	2416.7	151.1	26.78	0.8653	51	baixa	0.198990	1	False	True
1990	657.5	399.6	2732.2	327.7	26.67	0.8680	52	baixa	0.240649	1	False	True
1991	454.5	125.3	2332.2	124.7	26.05	0.8610	120	media	0.194880	1	False	False
1992	557.5	39.7	1889.4	227.7	27.21	0.8373	124	alta	0.295067	1	False	True
1993	380.3	401.2	2250.9	50.5	26.85	0.8684	50	baixa	0.168955	1	False	True

Figura 4.7: Exibição das 10 primeiras linhas do DataFrame após a engenharia de atributos.

ano	ENSO_La Niña	ENSO_Neutro
2014	False	True
2015	False	False
2016	True	False
2017	False	True
2018	False	True
2019	False	True
2020	True	False
2021	True	False
2022	True	False
2023	False	False

Figura 4.8: Exibição das 10 últimas linhas das colunas relacionadas ao ENSO após a codificação.

4.4 Preparação para o modelo

Com os atributos devidamente construídos, o próximo passo é preparar os dados para o treinamento dos modelos de Machine Learning. Essa etapa é fundamental para garantir que os modelos recebam as informações no formato mais adequado e que os resultados obtidos sejam confiáveis e justos. A preparação dos dados envolve algumas ações importantes: selecionar as variáveis preditoras e a variável-alvo, separar variáveis numéricas e binárias, padronizar os dados numéricos e dividir o conjunto em dados de treino e teste de forma coerente.

4.4.1 Seleção de variáveis

Neste momento, selecionamos a variável-alvo da tarefa de regressão, que é a produtividade. Para compor o conjunto de variáveis preditoras (X), removemos as colunas safra (que será utilizada na tarefa de classificação) e produtividade (pois é o valor que queremos prever). Essa seleção será utilizada na construção dos modelos de regressão linear, tanto com os dados originais quanto com a redução de dimensionalidade via PCA.

Copie e Teste!

```
# 1. Definindo X e y
X = df.drop(columns=['produtividade', 'safra']) # safra é para
    classificação
y = df['produtividade']
```

4.4.2 Identificação dos tipos de variáveis

Para preparar os dados de forma adequada, precisamos separar as variáveis numéricas das variáveis binárias, já que cada tipo exige um tratamento específico durante o pré-processamento. As variáveis numéricas são contínuas e devem ser padronizadas antes de serem utilizadas em modelos que são sensíveis à escala dos dados. No nosso caso, as seguintes colunas foram classificadas como numéricas:

Copie e Teste!

```
# Lista de colunas numéricas
colunas_numericas = ['chuva_flor', 'chuva_colheita', 'chuva_total'
    , 'anomalia_flor', 'temp_flor', 'umid_flor', 'chuva_relativa']
```

Já as variáveis binárias representam categorias codificadas com valores 0 ou 1, e podem ser usadas diretamente nos modelos sem necessidade de padronização. Aqui estão as colunas identificadas como binárias:

Copie e Teste!

```
# Lista de colunas binárias
colunas_binarias = ['anomalia_bin', 'ENSO_La Niña', 'ENSO_Neutro']
```

Essa separação nos permite aplicar transformações específicas a cada grupo, como escalonamento apenas para as variáveis numéricas e uso direto das variáveis binárias nos algoritmos de aprendizado.

4.4.3 Padronização com ColumnTransformer

Para garantir que todas as variáveis numéricas estejam na mesma escala, utilizamos o `StandardScaler`, que transforma os dados para que tenham média zero e desvio padrão igual a um. Essa padronização é fundamental, especialmente em algoritmos sensíveis à escala, como regressão linear, SVMs e PCA.

Entretanto, nem todas as variáveis precisam ser transformadas. As variáveis binárias já estão em formato adequado (0 ou 1) e devem ser mantidas como estão. Para resolver isso de forma eficiente, utilizamos o `ColumnTransformer`, uma ferramenta do scikit-learn que permite aplicar diferentes transformações a diferentes grupos de colunas dentro de um mesmo pipeline. Dessa forma, conseguimos padronizar apenas as variáveis numéricas enquanto preservamos as variáveis binárias, garantindo um pré-processamento coerente e automatizado.

Copie e Teste!

```
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer

# 3. Criando o transformador
preprocessador = ColumnTransformer(transformers=[('num',
    StandardScaler(), colunas_numericas),('bin', 'passthrough',
    colunas_binarias)])
```

4.4.4 Divisão do conjunto de treino/teste

A divisão entre dados de treino e teste é uma etapa essencial para avaliar a capacidade de generalização dos modelos. Neste projeto, optamos por separar 70% dos dados para treino e 30% para teste, respeitando a ordem cronológica dos registros.

Para isso, a divisão foi feita com o parâmetro `shuffle=False`, o que significa que os dados não foram embaralhados. Essa escolha é intencional: ao preservar a ordem temporal dos anos, simulamos um cenário mais realista, em que o modelo aprende com dados do passado e é testado em anos futuros. Essa estratégia é especialmente importante em contextos sazonais ou climáticos, onde a dependência temporal pode influenciar os resultados.

Copie e Teste!

```
from sklearn.model_selection import train_test_split

# 4. Separando treino e teste sem embaralhar (respeitando ordem
    temporal)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, shuffle=False)
```

4.5 Redução de dimensionalidade com PCA

Antes de treinarmos os modelos, é interessante avaliar se conseguimos reduzir a dimensionalidade do dataset sem comprometer significativamente a informação contida nos dados. Para isso, utilizamos a técnica de Análise de Componentes Principais (PCA), que transforma o conjunto original de variáveis em novas direções, chamadas componentes, capazes de explicar a maior parte da variância presente nos dados.

A aplicação do PCA pode trazer diversos benefícios. Além de facilitar a visualização dos dados em um espaço bidimensional, ele também pode reduzir o ruído, ajudar a evitar o sobreajuste (*overfitting*) e possibilitar representações mais compactas dos dados, com menos variáveis. No nosso caso, aplicaremos o PCA após a padronização, extrairemos os dois primeiros componentes principais e analisaremos a quantidade de variância explicada por cada um, bem como a distribuição dos dados no plano formado por PC1 × PC2.

4.5.1 Avaliação da variância explicada (Scree Plot)

Antes de decidirmos quantos componentes manter, é importante avaliar quanto da variância total dos dados cada componente principal consegue explicar. Para isso, utilizamos o gráfico de variância explicada, conhecido como *scree plot*.

Esse gráfico nos auxilia a identificar um ponto de corte adequado: ou seja, quantos componentes conseguem reter uma boa proporção da informação original, justificando a redução de dimensionalidade. A partir dessa análise, podemos aplicar o PCA de forma mais informada e eficiente.

Copie e Teste!

```
from sklearn.decomposition import PCA

# Aplica o ColumnTransformer (padronização)
X_padronizado = preprocessador.fit_transform(X)

# Aplica PCA com todos os componentes (não limita n_components
   ainda)
pca_full = PCA()
pca_full.fit(X_padronizado)

# Scree Plot
plt.plot(range(1, len(pca_full.explained_variance_ratio_)+1),
   pca_full.explained_variance_ratio_, marker='o')
plt.title('Scree Plot - Variância Explicada por Componente')
plt.xlabel('Componente Principal')
plt.ylabel('Proporção da Variância')
plt.grid(True)
plt.tight_layout()
plt.show()

# Mostrar numericamente
for i, v in enumerate(pca_full.explained_variance_ratio_):
    print(f"PC{i+1}: {v:.2%}")
```

Resultado Esperado

```
PC1: 41.34%
PC2: 28.80%
PC3: 16.50%
PC4: 4.29%
PC5: 3.51%
PC6: 3.18%
PC7: 1.22%
PC8: 0.98%
PC9: 0.18%
PC10: 0.00%
```

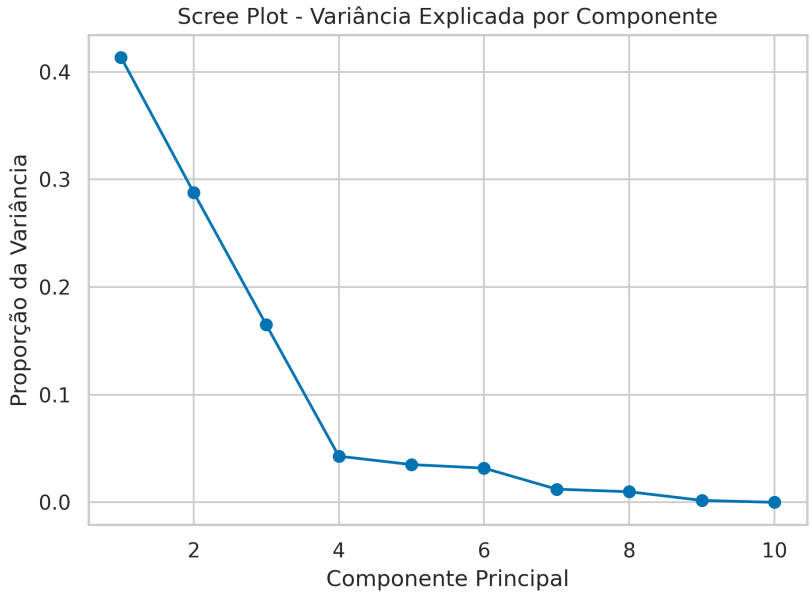


Figura 4.9: Scree plot para determinar o número ótimo de componentes principais a serem retidos, mostrando o ponto de inflexão da curva de variância.

4.5.2 Aplicação do PCA (2 componentes)

Com base na análise do scree plot, decidimos aplicar o PCA utilizando dois componentes principais. Esses componentes, chamados de PC1 e PC2, representam as novas dimensões que concentram a maior parte da variabilidade dos dados, reduzindo a complexidade do conjunto original de variáveis. Essas duas novas variáveis passam a resumir a estrutura dos dados de forma mais compacta e serão utilizadas nas próximas etapas do projeto, servindo como entradas para os modelos de regressão e classificação que utilizam a abordagem com PCA.

Copie e Teste!

```
# Aplica PCA com 2 componentes
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_padronizado)

# Cria df_PCA com componentes e variáveis-alvo
df_PCA = pd.DataFrame(X_pca, columns=['PC1', 'PC2'], index=X.index
    )
df_PCA['produtividade'] = df['produtividade']
df_PCA['safra'] = df['safra']
```

4.5.3 Visualização 2D dos componentes principais

Nesta etapa, projetamos os dados no plano bidimensional formado pelos dois primeiros componentes principais (PC1 e PC2). Cada ponto no gráfico representa um ano do conjunto de dados, e as cores indicam a classe da safra correspondente: baixa, média ou alta. Essa visualização é útil para analisar possíveis agrupamentos visuais conforme o tipo de safra, além de avaliar a separabilidade entre as classes no novo espaço dimensional. Se os grupos estiverem

relativamente bem distribuídos ou separados, isso pode indicar que o PCA está capturando padrões relevantes, o que, por sua vez, pode ajudar os modelos a generalizarem melhor durante o processo de aprendizado.

Copie e Teste!

```
sns.scatterplot(data=df_PCA, x='PC1', y='PC2', hue='safra', s=80,
   alpha=0.8)
plt.title('PCA - Componentes Principais coloridos por Safra')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.legend(title='Safra')
plt.tight_layout()
plt.show()
```

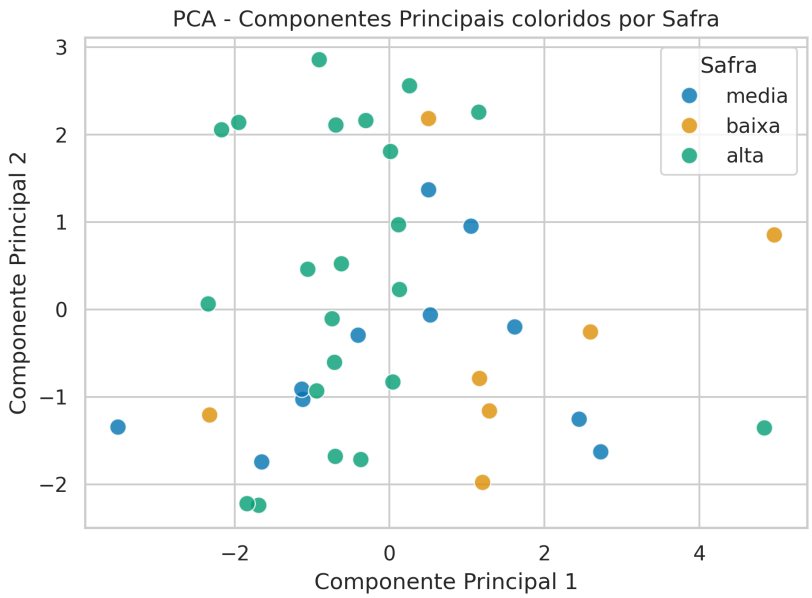


Figura 4.10: Projeção dos dados nos dois primeiros componentes principais (PC1 e PC2). A coloração por tipo de safra permite avaliar a separabilidade das classes no espaço de características reduzido, indicando a formação de agrupamentos visuais.

4.6 Previsão da Produtividade: Regressão Linear

Nesta etapa, construiremos modelos de regressão linear com o objetivo de prever a produtividade da safra a partir de variáveis climáticas e sazonais. Vamos considerar dois cenários distintos: o primeiro utilizando as variáveis originais do dataset, sem qualquer tipo de redução de dimensionalidade, e o segundo empregando os componentes principais gerados via PCA.

Em ambos os cenários, aplicaremos dois tipos de modelo: a Regressão Linear Simples, que busca encontrar a melhor combinação linear das variáveis para prever a produtividade, e a Regressão com Regularização L^2 (Ridge), que introduz uma penalidade para reduzir o risco de sobreajuste e melhorar a capacidade de generalização do modelo. Ao final dessa etapa, com-

pararemos os modelos construídos por meio de métricas quantitativas, visualizações gráficas e interpretação da função de custo, buscando entender qual abordagem se adapta melhor aos dados.

4.6.1 Modelos com variáveis originais (sem PCA)

Neste primeiro cenário, utilizaremos diretamente as variáveis originais do conjunto de dados como entrada para os modelos de regressão. A ideia aqui é avaliar o desempenho dos modelos sem qualquer transformação ou compactação dos dados.

Serão aplicados dois modelos: o primeiro será uma Regressão Linear tradicional, que ajusta os coeficientes com base na minimização do erro quadrático médio; o segundo será uma Regressão Ridge, que adiciona um termo de penalização L^2 à função de custo. Essa regularização busca reduzir a complexidade do modelo, especialmente quando há muitas variáveis correlacionadas, ajudando a evitar o sobreajuste.

Copie e Teste!

```
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
```

Regressão Linear Simples

Neste passo, treinamos um modelo de regressão linear simples utilizando o pipeline previamente definido, o qual já incorpora a padronização das variáveis numéricas. O objetivo do modelo é ajustar uma função linear que relacione diretamente as variáveis climáticas e sazonais com a produtividade da safra. Essa abordagem fornece uma base interpretável para compreender como cada fator influencia a variável-alvo e serve como referência inicial para comparação com modelos mais sofisticados ou regularizados.

Copie e Teste!

```
# Pipeline: pré-processador + modelo
pipeline_original = make_pipeline(preprocessador, LinearRegression
    ())

# Treinamento
pipeline_original.fit(X_train, y_train)

# Previsão
y_pred_orig = pipeline_original.predict(X_test)

# Avaliação
mse_orig = mean_squared_error(y_test, y_pred_orig)
rmse_orig = mse_orig ** 0.5
r2_orig = r2_score(y_test, y_pred_orig)
print(f"[Regressão linear] RMSE: {rmse_orig:.2f} | R²: {r2_orig
    :.2%}")
```

Resultado Esperado

```
[Regressão linear] RMSE: 71.57 | R²: -857.97%
```

Regressão Linear com Regularização (Ridge)

Agora, aplicamos a regularização L^2 utilizando o modelo Ridge. Esse tipo de regularização adiciona um termo de penalização à função de custo, o que ajuda a reduzir o sobreajuste (*overfitting*), especialmente em datasets que apresentam alta variância ou multicolinearidade entre as variáveis. A intensidade da penalização é controlada por um hiperparâmetro, denominado `alpha`, que ajusta o quanto a regularização influencia o modelo. A escolha adequada é essencial para encontrar um equilíbrio entre o ajuste do modelo aos dados e a sua capacidade de generalização.

Copie e Teste!

```
# Pipeline com regularização L2 (Ridge)
lambda_regressao = 1 # testar vários valores para lambda
pipeline_ridge = make_pipeline(preprocessador, Ridge(alpha=
    lambda_regressao))

# Treinamento
pipeline_ridge.fit(X_train, y_train)

# Previsão
y_pred_ridge = pipeline_ridge.predict(X_test)

# Avaliação
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
rmse_ridge = mse_ridge ** 0.5
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"[Regularização Ridge (L²) | λ = {lambda_regressao}] RMSE:
    {rmse_ridge:.2f} | R²: {r2_ridge:.2%}")
```

Resultado Esperado

```
[Regularização Ridge (L²) | λ = 1] RMSE: 68.30 | R²: -772.31%
```

4.6.2 Modelos com componentes principais (PCA)

Neste segundo cenário, utilizamos os dois primeiros componentes principais (PC1 e PC2) obtidos por meio do PCA para treinar os modelos. O objetivo dessa abordagem é avaliar se a projeção dos dados em um espaço de menor dimensionalidade pode melhorar a capacidade de generalização do modelo, ao reduzir o risco de sobreajuste e captar as principais variáveis que explicam a variância dos dados.

Regressão Linear Simples sobre PCA

Neste caso, treinamos o modelo de regressão linear simples diretamente sobre os dois componentes principais extraídos, PC1 e PC2. A ideia é comparar o desempenho desse modelo em um espaço de menor dimensão com o modelo de regressão linear treinado nas variáveis originais, para verificar se a redução da dimensionalidade impacta positivamente a capacidade de previsão da produtividade.

Copie e Teste!

```
# Definindo X e y com base no df_PCA
X_pca = df_PCA[['PC1', 'PC2']]
y_pca = df_PCA['produtividade']

# Divisão temporal (como fizemos antes)
X_pca_train, X_pca_test, y_pca_train, y_pca_test =
    train_test_split(X_pca, y_pca, test_size=0.2, shuffle=False)

# Modelo linear simples com PCA
modelo_pca = LinearRegression()
modelo_pca.fit(X_pca_train, y_pca_train)

# Previsão
y_pred_pca = modelo_pca.predict(X_pca_test)

# Avaliação
rmse_pca = mean_squared_error(y_pca_test, y_pred_pca) ** 0.5
r2_pca = r2_score(y_pca_test, y_pred_pca)

print(f"[PCA + Regressão linear] RMSE: {rmse_pca:.2f} | R²: {
    r2_pca:.2%}")
```

Resultado Esperado

```
[PCA + Regressão linear] RMSE: 43.28 | R²: -250.26%
```

Regressão Linear com Regularização (Ridge) sobre PCA

A combinação da Análise de Componentes Principais (PCA) com a Regressão Ridge é uma estratégia robusta para construir modelos de regressão linear, especialmente em cenários com alta dimensionalidade e multicolinearidade (quando as variáveis preditoras são correlacionadas entre si). O processo funciona em duas etapas principais, onde cada uma aborda um desafio específico:

1. **Transformação com PCA:** Primeiramente, o PCA é aplicado ao conjunto de variáveis preditoras. Ele transforma as variáveis originais, potencialmente correlacionadas, em um novo conjunto de variáveis linearmente não correlacionadas, chamadas de **componentes principais**. Esses componentes são ordenados pela quantidade de variância dos dados originais que eles capturam. Ao selecionar apenas os primeiros componentes,

conseguimos uma representação de dimensionalidade reduzida que ainda retém a maior parte da informação relevante, simplificando a estrutura dos dados.

2. **Regularização com Ridge:** Em seguida, o modelo de Regressão Ridge é treinado não sobre os dados originais, mas sobre os componentes principais selecionados. A regularização Ridge adiciona uma **penalidade L2** (proporcional à soma dos quadrados dos coeficientes) à função de custo do modelo. Essa penalidade desencoraja coeficientes com magnitudes muito grandes, o que é a causa principal do sobreajuste (*overfitting*).

A **sinergia** entre as duas técnicas é o ponto-chave:

- O PCA resolve o problema da **multicolinearidade** em sua origem, fornecendo ao modelo de Ridge um conjunto de preditores ortogonais (não correlacionados).
- Isso torna a tarefa de regularização do Ridge mais estável e eficaz, pois o modelo não precisa mais "decidir" como distribuir os pesos entre variáveis redundantes.
- O resultado é um modelo duplamente otimizado: simplificado em sua estrutura pelo PCA e protegido contra o sobreajuste pela regularização Ridge.

Copie e Teste!

```
# Modelo com Ridge sobre PCA
lambda_regressao = 1 # testar vários valores para lambda
modelo_pca_ridge = Ridge(alpha=lambda_regressao)
modelo_pca_ridge.fit(X_pca_train, y_pca_train)

# Previsão
y_pred_pca_ridge = modelo_pca_ridge.predict(X_pca_test)

# Avaliação
rmse_pca_ridge = mean_squared_error(y_pca_test, y_pred_pca_ridge)
   ** 0.5
r2_pca_ridge = r2_score(y_pca_test, y_pred_pca_ridge)

print(f"[PCA + Regularização Ridge (L²) | λ = {lambda_regressao}]
   RMSE: {rmse_pca_ridge:.2f} | R²: {r2_pca_ridge:.2%}")
```

Resultado Esperado

```
[PCA + Regularização Ridge (L²) | λ = 1]
      RMSE: 42.98 | R²: -245.38%
```

4.6.3 Comparação dos Modelos

Após treinar os modelos, o próximo passo é compará-los para avaliar seu desempenho sob diferentes perspectivas. Esta comparação envolve:

- Tabela com métricas de desempenho (RMSE e R^2)
- Curvas de previsão vs. variáveis explicativas

- Gráficos da função de custo (1D e 2D)
- Análise gráfica dos resíduos

Tabela Comparativa: Regressão Regularizada com e sem PCA

Aqui, comparamos os resultados obtidos pelos quatro modelos testados:

- Regressão Linear Simples
- Regressão Ridge
- Regressão com PCA
- Regressão com PCA + Ridge

As principais métricas de avaliação utilizadas para a comparação são:

- **RMSE (Root Mean Squared Error):** Mede a diferença entre os valores previstos e os reais, sendo que valores menores indicam um modelo mais preciso.
- **R^2 (coeficiente de determinação):** Indica a proporção da variância da variável-alvo que é explicada pelo modelo. Quanto mais próximo de 1, melhor é a explicação da variância.

Abaixo, temos a tabela comparativa com o desempenho dos modelos para diferentes valores de λ (hiperparâmetro de regularização do Ridge). Com isso, podemos avaliar os modelos não só com base em métricas quantitativas como RMSE e R^2, mas também observar o impacto da redução de dimensionalidade e regularização no desempenho geral.

λ (Ridge)	RMSE sem PCA	R^2 sem PCA	RMSE com PCA	R^2 com PCA
0	71.57	−857.97%	43.28	−250.26%
0.1	71.19	−847.87%	43.25	−249.76%
1	68.30	−772.31%	42.98	−245.38%
10	54.97	−465.14%	40.61	−208.48%
100	34.91	−127.95%	31.20	−82.07%
1000	26.38	−30.09%	25.97	−26.16%
10000	25.61	−22.67%	25.57	−22.30%
30000	25.56	−22.21%	25.55	−22.09%
100000	25.55	−22.06%	25.54	−22.02%
300000	25.54	−22.01%	25.54	−22.00%
1000000	25.54	−22.00%	25.54	−21.99%

Tabela 4.1: Comparação de RMSE e R^2 para Regressão Ridge com e sem PCA.

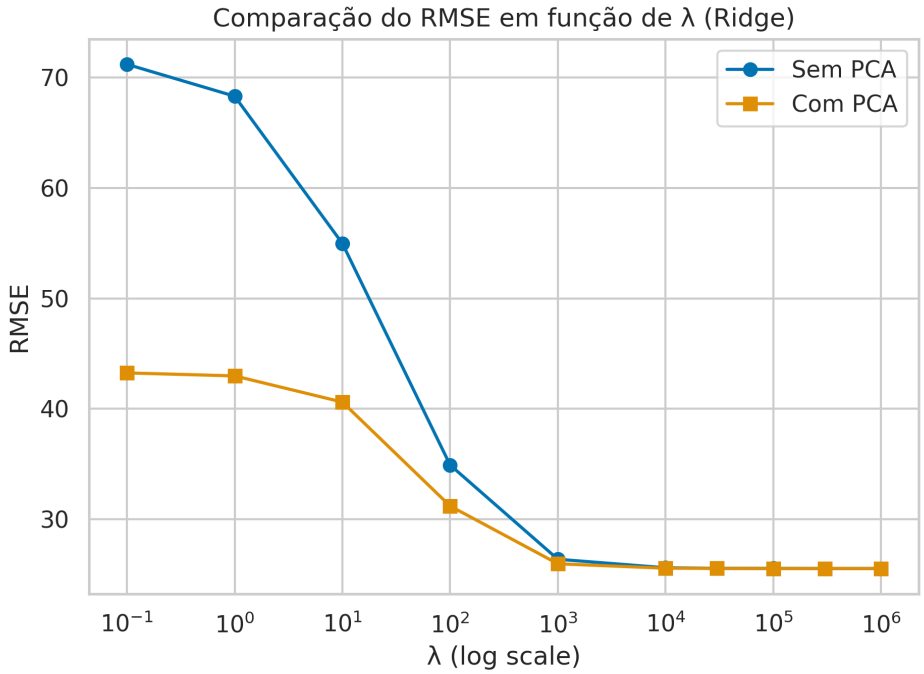


Figura 4.11: Evolução do Erro Quadrático Médio da Raiz (RMSE) em função do hiperparâmetro de regularização λ. As curvas comparam o desempenho do modelo com e sem a aplicação prévia de PCA, mostrando a convergência do erro à medida que a regularização aumenta.

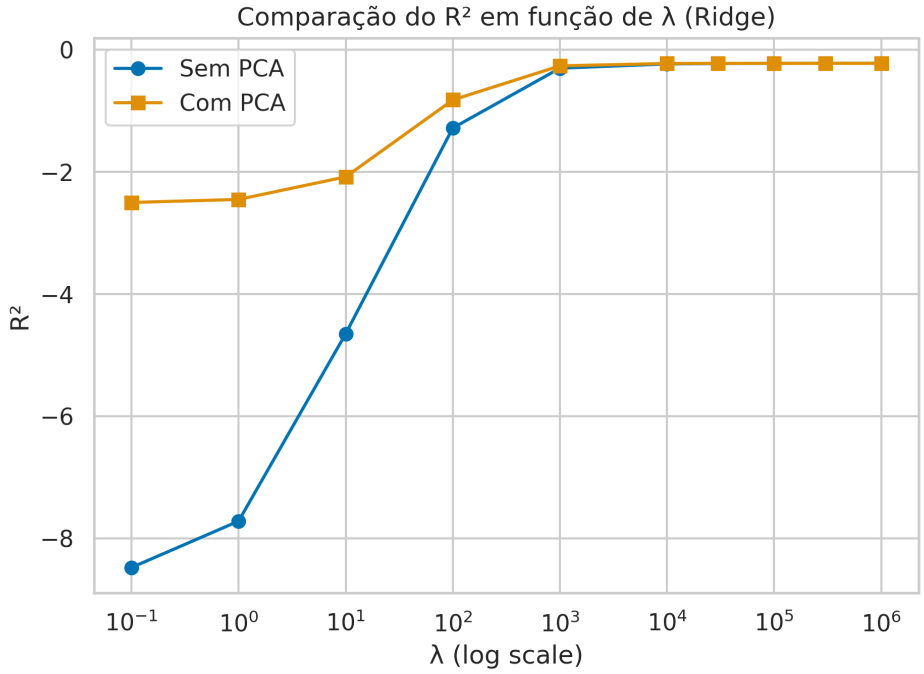


Figura 4.12: Evolução do Coeficiente de Determinação (R^2) em função do hiperparâmetro de regularização λ. A visualização demonstra como a regularização e o PCA melhoram o ajuste do modelo, elevando o valor de R^2 de patamares negativos para mais próximos de zero.

Copie e Teste!

```
# Dados para plotagem
lambdas = [0.1, 1, 10, 100, 1000, 10000, 30000, 100000, 300000,
   1000000]
rmse_sem_pca = [71.19, 68.30, 54.97, 34.91, 26.38, 25.61, 25.56,
   25.55, 25.54, 25.54]
rmse_com_pca = [43.25, 42.98, 40.61, 31.20, 25.97, 25.57, 25.55,
   25.54, 25.54, 25.54]

plt.plot(lambdas, rmse_sem_pca, marker='o', label='Sem PCA')
plt.plot(lambdas, rmse_com_pca, marker='s', label='Com PCA')
plt.xscale('log')
plt.xlabel("λ (log scale)")
plt.ylabel("RMSE")
plt.title("Comparação do RMSE em função de λ (Ridge)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# R² para cada lambda (sem e com PCA)
r2_sem_pca = [-8.48, -7.72, -4.65, -1.28, -0.30, -0.23, -0.2221,
   -0.2206, -0.2201, -0.2200]
r2_com_pca = [-2.50, -2.45, -2.08, -0.82, -0.2616, -0.2230,
   -0.2209, -0.2202, -0.2200, -0.2199]

plt.plot(lambdas, r2_sem_pca, marker='o', label='Sem PCA')
plt.plot(lambdas, r2_com_pca, marker='s', label='Com PCA')
plt.xscale('log')
plt.xlabel("λ (log scale)")
plt.ylabel("R²")
plt.title("Comparação do R² em função de λ (Ridge)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Relação entre Variáveis e Produtividade

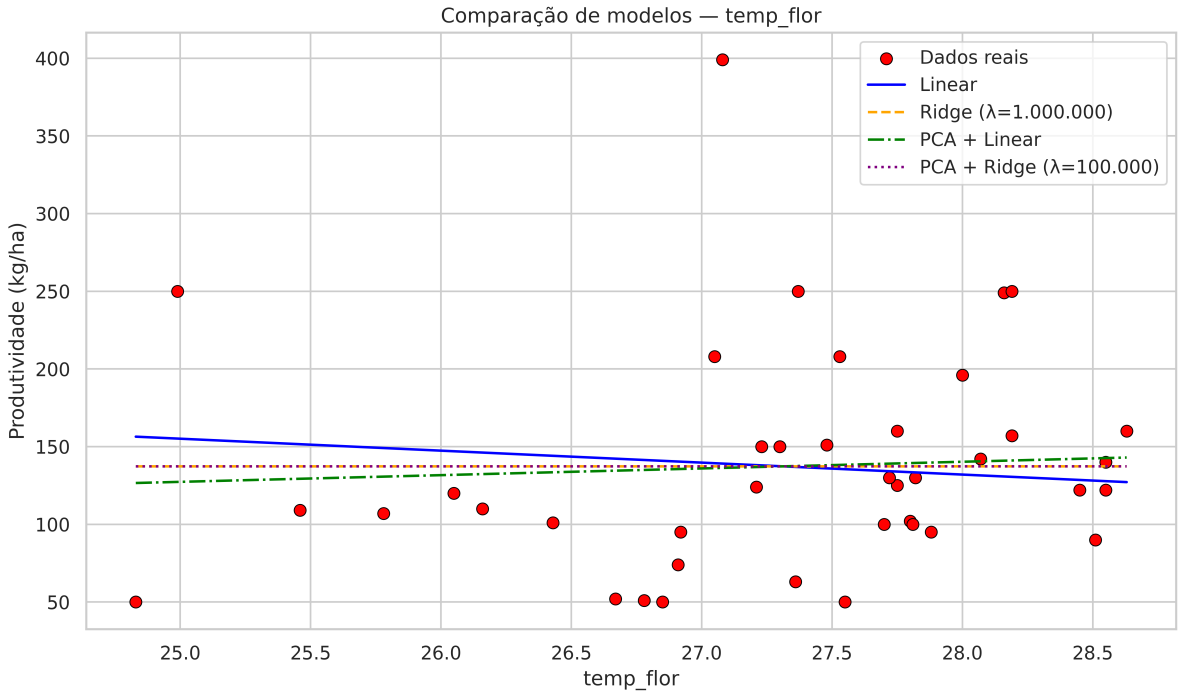


Figura 4.13: Comparativo visual do desempenho de diferentes modelos de regressão para prever a produtividade com base na temperatura durante a floração. As linhas representam as predições de cada modelo, permitindo uma análise da sua capacidade de ajuste aos dados observados.

Aqui, comparamos os valores reais de produtividade com as previsões feitas por cada modelo, levando em consideração cada variável explicativa. Para cada variável como temperatura, umidade e chuva, os gráficos apresentam:

- Pontos reais (representados em vermelho).
- A curva prevista por cada modelo, incluindo:
  - Regressão Linear sem PCA;
  - Regressão Ridge sem PCA;
  - Regressão Linear com PCA;
  - Regressão Ridge com PCA.

Além disso, incluímos gráficos para os dois primeiros componentes principais (PC1 e PC2), que são combinações lineares das variáveis originais. Essa visualização tem como objetivo identificar o tipo de relação entre as variáveis explicativas e a produtividade, avaliar o comportamento qualitativo dos modelos em relação aos dados reais e verificar se a redução de dimensionalidade com PCA resultou em representações mais alinhadas com as previsões de produtividade.

Copie e Teste!

```
def plot_modelos_para_variavel(x_var, X, y, scaler, pca_model,
   modelo_linear, modelo_ridge, modelo_pca, modelo_pca_ridge):
    x_index = X.columns.get_loc(x_var)
    x_vals = np.linspace(X[x_var].min(), X[x_var].max(), 100)
    X_mean = X.mean().to_numpy()

    X_input = np.tile(X_mean, (100, 1))
    X_input[:, x_index] = x_vals
    X_input_df = pd.DataFrame(X_input, columns=X.columns)
    X_input_scaled = scaler.transform(X_input_df)
    X_input_pca = pca_model.transform(X_input_scaled)

    y_linear = modelo_linear.predict(X_input_scaled)
    y_ridge = modelo_ridge.predict(X_input_scaled)
    y_pca = modelo_pca.predict(X_input_pca)
    y_pca_ridge = modelo_pca_ridge.predict(X_input_pca)

    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=X[x_var], y=y, color='red', label='Dados
   reais', s=50, edgecolor='black')
    plt.plot(x_vals, y_linear, label='Linear', linestyle='-',
   color='blue')
    plt.plot(x_vals, y_ridge, label='Ridge λ(=1.000.000)',
   linestyle='--', color='orange')
    plt.plot(x_vals, y_pca, label='PCA + Linear', linestyle='-.',
   color='green')
    plt.plot(x_vals, y_pca_ridge, label='PCA + Ridge λ(=100.000)',
    linestyle=':', color='purple')
    plt.xlabel(x_var)
    plt.ylabel('Produtividade (kg/ha)')
    plt.title(f'Comparação de modelos — {x_var}')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# 1. Reconstrução de X e y
X = df[['chuva_flor', 'chuva_colheita', 'chuva_total', '
   anomalia_flor', 'temp_flor', 'umid_flor', 'chuva_relativa']]
y = df['produtividade']

# 2. Padronização
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. PCA
pca_model = PCA(n_components=2)
X_pca = pca_model.fit_transform(X_scaled)
```

```
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'], index=df.
    index)
df[['PC1', 'PC2']] = df_pca

# 4. Modelos treinados separadamente
modelo_linear = LinearRegression().fit(X_scaled, y)
modelo_ridge = Ridge(alpha=1e6).fit(X_scaled, y)
modelo_pca = LinearRegression().fit(X_pca, y)
modelo_pca_ridge = Ridge(alpha=1e5).fit(X_pca, y)

plot_modelos_para_variavel('temp_flor', X, y, scaler, pca_model,
    modelo_linear, modelo_ridge, modelo_pca, modelo_pca_ridge)
```

Função custo

A seguir, exploramos visualmente como a função custo varia em diferentes cenários. Primeiro, analisamos a variação do erro para um único coeficiente (curva 1D), o que ajuda a entender como ajustes sutis impactam o desempenho do modelo. Em seguida, criamos superfícies bidimensionais que mostram a forma da função de custo para pares de variáveis, revelando o relevo que o modelo "navega" na busca por mínimos. Por fim, plotamos a função de custo no espaço dos componentes principais (PC1 × PC2), observando como a redução de dimensionalidade influencia esse processo.

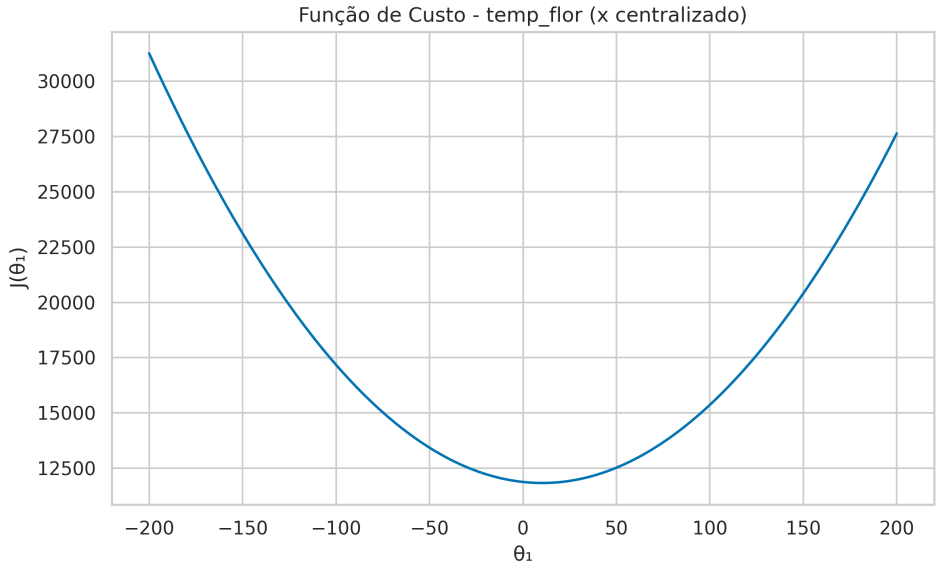


Figura 4.14: Visualização unidimensional da função de custo. O gráfico mostra como o erro quadrático médio (MSE) varia ao se alterar apenas o coeficiente associado à variável 'temperatura durante a floração'.

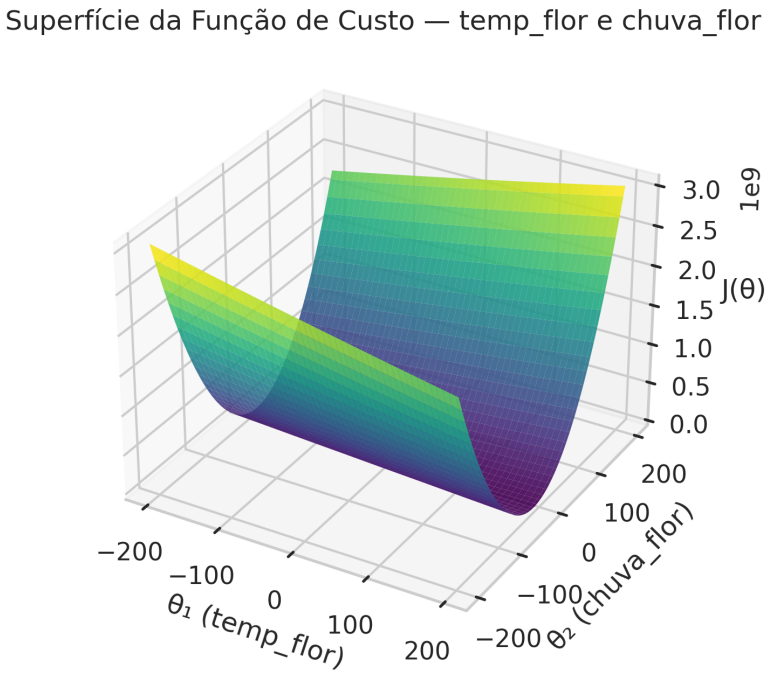


Figura 4.15: Superfície de custo bidimensional em função dos coeficientes das variáveis 'temperatura durante a floração' e 'chuva durante a floração'. As elipses inclinadas indicam que as variáveis são correlacionadas, criando um caminho mais complexo para o algoritmo de otimização encontrar o ponto de mínimo global (o fundo do vale).

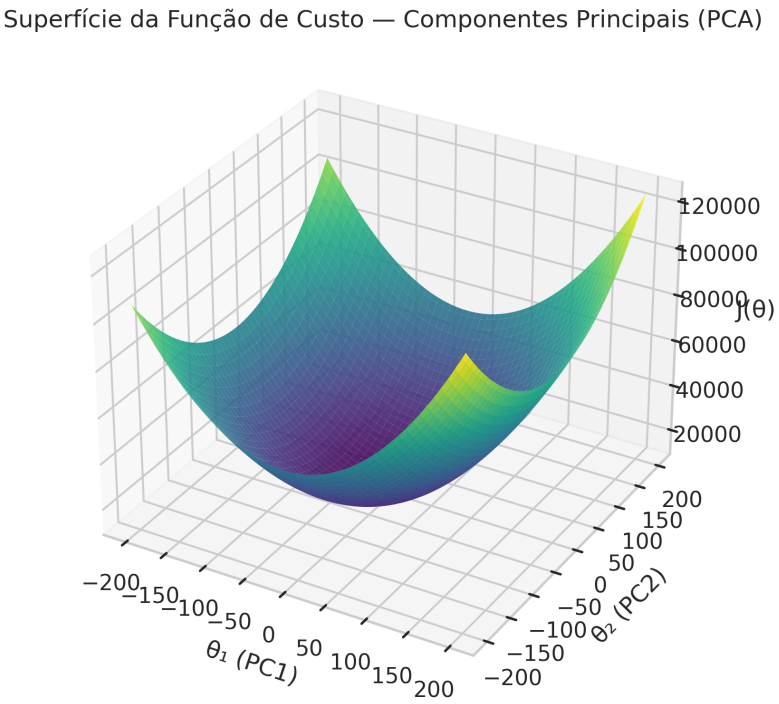


Figura 4.16: Superfície de custo no espaço dos dois primeiros Componentes Principais (PC1 e PC2). Note como a transformação PCA realinhou os eixos: os contornos agora são elipses regulares e alinhadas. Isso ocorre porque os componentes principais não são correlacionados, o que simplifica a topografia da função e facilita a convergência do otimizador.

Fique Alerta!

Repare que a superfície da função de custo no espaço PC1 × PC2 é quase simétrica, isso acontece porque o PCA gera variáveis ortogonais entre si, o que elimina a multicolinearidade e como resultado, o gradiente desce de forma mais eficiente, sem precisar ”zigueza-guear”por vales inclinados.

Copie e Teste!

```
# Curva 1D da função custo
def plot_funcao_custo_1D(x_var, X, y, intervalo=(-200, 200),
   pontos=200):
    """
    Plota a função de custo J(Theta 1) para uma regressão
   univariada com a variável x_var.
    """
    x = X[x_var].values
    y = y.values
    m = len(y)

    # Centraliza x para eliminar o intercepto implicitamente
    x_centralizado = x - x.mean()

    theta1_vals = np.linspace(intervalo[0], intervalo[1], pontos)
    custos = [(1 / (2 * m)) * np.sum((theta1 * x_centralizado - y)
    ** 2) for theta1 in theta1_vals]

    plt.figure(figsize=(8, 5))
    plt.plot(theta1_vals, custos)
    plt.xlabel("Theta 1")
    plt.ylabel("J(Theta 1)")
    plt.title(f"Função de Custo - {x_var} (x centralizado)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_funcao_custo_1D('temp_flor', X, y)
```

Copie e Teste!

```
# Superfície 2D da função custo
from itertools import combinations
from mpl_toolkits.mplot3d import Axes3D

def plot_funcao_custo_2D(x_vars, X, y, range_theta=(-200, 200),
   pontos=100):
    """
    Plota a superfície da função de custo J(Theta 1, Theta 2) para
    duas variáveis.
```

```
    """
    x1 = X[x_vars[0]].values
    x2 = X[x_vars[1]].values
    y = y.values
    m = len(y)

    # Matriz de entrada com intercepto
    X_mat = np.vstack([np.ones(m), x1, x2]).T

    # Geração de grid de Theta 1 e Theta 2 (intercepto Theta □
   fixado em 0 para simplificação)
    theta1_vals = np.linspace(range_theta[0], range_theta[1],
   pontos)
    theta2_vals = np.linspace(range_theta[0], range_theta[1],
   pontos)
    J_vals = np.zeros((pontos, pontos))

    for i in range(pontos):
        for j in range(pontos):
            theta = np.array([0, theta1_vals[i], theta2_vals[j]])
            h = X_mat @ theta
            J_vals[j, i] = (1 / (2 * m)) * np.sum((h - y) ** 2)

    # Superfície
    T1, T2 = np.meshgrid(theta1_vals, theta2_vals)
    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(T1, T2, J_vals, cmap='viridis', edgecolor='
   none', alpha=0.9)

    ax.set_xlabel(f"Theta 1 ({x_vars[0]})")
    ax.set_ylabel(f"Theta 2 ({x_vars[1]})")
    ax.set_zlabel("J(Theta)")
    ax.set_title(f"Superfície da Função de Custo — {x_vars[0]} e {
   x_vars[1]}")
    fig.subplots_adjust(right=0.5)
    plt.show()

# Para um gráfico específico
plot_funcao_custo_2D(['temp_flor', 'chuva_flor'], X, y)
```

Copie e Teste!

```
# Superfície 2D da função custo com PCA
def plot_funcao_custo_2D_PCA(X_pca, y, range_theta=(-200, 200),
   pontos=100):
    """
    Plota a superfície da função de custo J(Theta 1, Theta 2)
   usando os componentes principais PC1 e PC2.
```

```
    """
    pc1 = X_pca[:, 0]
    pc2 = X_pca[:, 1]
    m = len(y)

    X_mat = np.vstack([np.ones(m), pc1, pc2]).T
    theta1_vals = np.linspace(range_theta[0], range_theta[1],
   pontos)
    theta2_vals = np.linspace(range_theta[0], range_theta[1],
   pontos)
    J_vals = np.zeros((pontos, pontos))

    for i in range(pontos):
        for j in range(pontos):
            theta = np.array([0, theta1_vals[i], theta2_vals[j]])
            h = X_mat @ theta
            J_vals[j, i] = (1 / (2 * m)) * np.sum((h - y) ** 2)

    T1, T2 = np.meshgrid(theta1_vals, theta2_vals)
    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(T1, T2, J_vals, cmap='viridis', edgecolor='
   none', alpha=0.9)

    ax.set_xlabel("Theta 1 (PC1)")
    ax.set_ylabel("Theta 2 (PC2)")
    ax.set_zlabel("J(Theta)")
    ax.set_title("Superfície da Função de Custo — Componentes
   Principais (PCA)")
    fig.subplots_adjust(right=0.85)
    plt.show()

plot_funcao_custo_2D_PCA(X_pca, y)
```

Análise gráfica dos resíduos

Os resíduos representam a diferença entre os valores reais de produtividade e os valores previstos pelos modelos. Analisar esses resíduos é essencial para entender se os modelos estão cometendo erros sistemáticos ou aleatórios. Por exemplo, é possível identificar se os erros estão concentrados em safras de baixa, média ou alta produtividade, ou ainda se há viés nas previsões. Idealmente, esperamos encontrar uma distribuição simétrica dos resíduos em torno de zero, sem tendências evidentes de crescimento ou decaimento, o que indicaria que o modelo não está cometendo erros sistemáticos. Quanto mais próximos de zero forem os resíduos, melhor o desempenho do modelo. Nesta etapa, comparamos graficamente os resíduos de cada abordagem para identificar padrões e avaliar a qualidade das previsões.

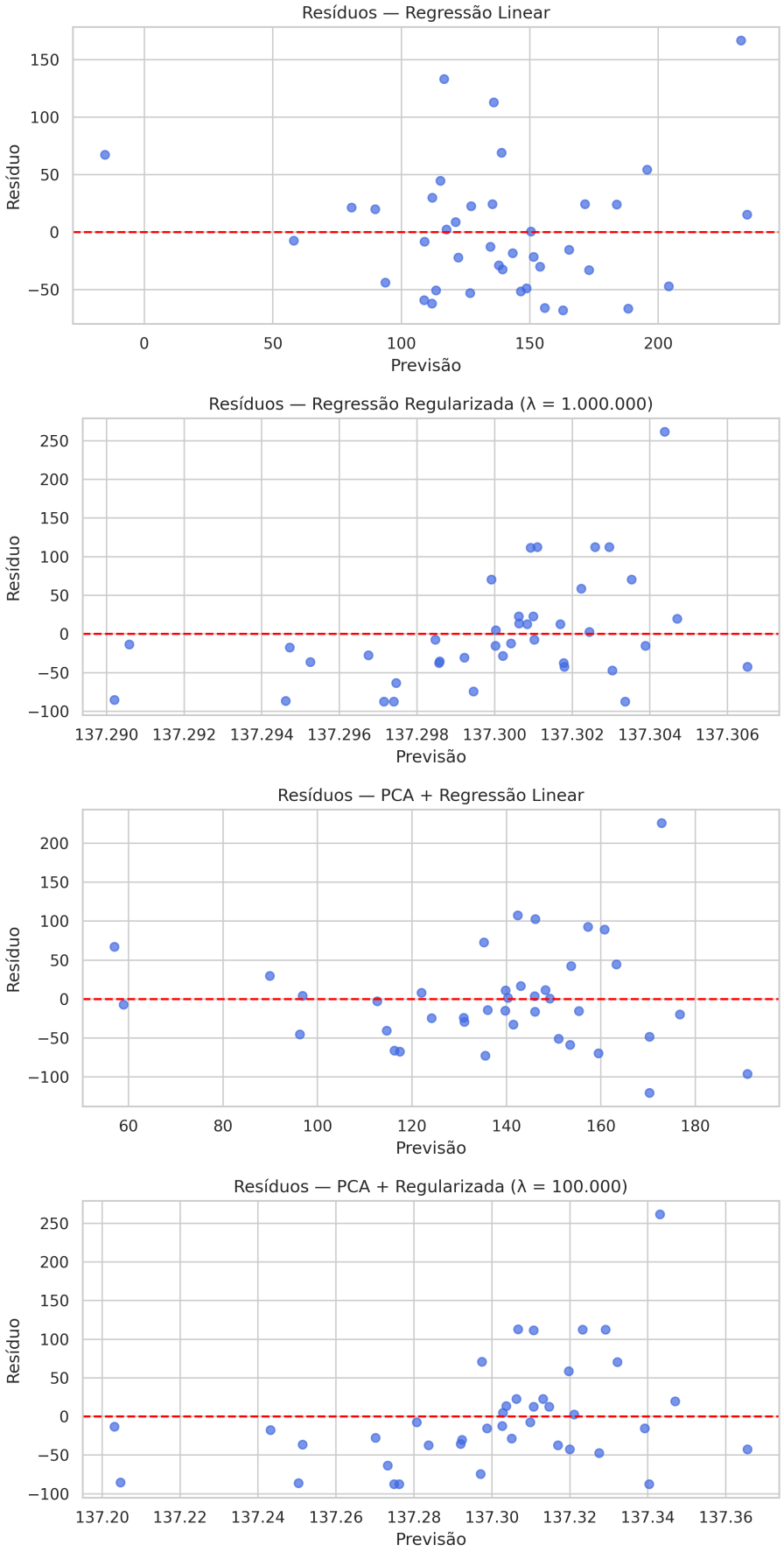


Figura 4.17: Análise de Resíduos para os diferentes modelos de regressão.

Copie e Teste!

```
def plot_residuos(y_true, y_pred, titulo):
    """
    Plota os resíduos de um modelo específico.
    """
    residuos = y_true - y_pred
    plt.figure(figsize=(8, 4))
    plt.scatter(y_pred, residuos, color='royalblue', alpha=0.7)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Previsão")
    plt.ylabel("Resíduo")
    plt.title(f"Resíduos — {titulo}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    print("\n")

# Previsões
y_pred_linear = modelo_linear.predict(X_scaled)
y_pred_ridge = modelo_ridge.predict(X_scaled)
y_pred_pca = modelo_pca.predict(X_pca)
y_pred_pca_ridge = modelo_pca_ridge.predict(X_pca)

# Gráficos de resíduos
plot_residuos(y, y_pred_linear, "Regressão Linear")
plot_residuos(y, y_pred_ridge, "Regressão Regularizada λ( =
   1.000.000)")
plot_residuos(y, y_pred_pca, "PCA + Regressão Linear")
plot_residuos(y, y_pred_pca_ridge, "PCA + Regularizada λ( =
   100.000)")
```

Discussão: Trade-offs na Regressão Linear Regularizada

Ao longo desta seção, exploramos como diferentes configurações impactam a qualidade da previsão da produtividade do guaraná. A comparação entre os cenários com e sem PCA e com diferentes níveis de regularização L2 (Ridge) nos permite tirar algumas conclusões importantes:

- Modelos sem regularização apresentaram desempenho desastroso, especialmente sem PCA, com RMSE elevado e R^2 extremamente negativo. Isso evidencia um caso claro de sobreajuste severo, mesmo com poucos dados, resultado da colinearidade entre variáveis e da instabilidade nos coeficientes da regressão.
- A introdução da regularização L2 ajuda a controlar esse comportamento. À medida que aumentamos λ, observamos uma redução significativa do erro até um ponto de estabilização. Contudo, esse ganho traz um trade-off natural: maior penalização implica menor capacidade do modelo de capturar variações mais complexas nos dados, refletido pela persistência de valores negativos de R^2.

- O uso do PCA antes da regressão melhora discretamente os resultados, principalmente por eliminar redundâncias e simplificar o espaço de variáveis. No entanto, mesmo com essa redução de dimensionalidade, o desempenho ainda é limitado pela natureza linear da regressão e pelas características dos dados.
- Na prática, a melhor performance foi alcançada com PCA + Ridge com $\lambda \approx 100000$, embora o R^2 continue próximo de zero. Isso indica que o modelo consegue estimar valores medianos de produtividade com certa estabilidade, mas ainda não é capaz de capturar variações extremas ou relações não lineares mais complexas.

4.7 Classificação da Safra

Além da previsão contínua da produtividade por meio de regressão, é possível também abordar o problema como uma tarefa de classificação multiclasse, atribuindo diretamente uma das categorias: baixa, média ou alta produtividade. Essa abordagem é particularmente útil para gerar alertas simples, identificar padrões em safras anteriores e apoiar decisões operacionais em cenários onde a precisão numérica exata da produtividade não é essencial, bastando saber se a safra tende a ser fraca, mediana ou excelente.

O objetivo desta seção é construir dois modelos de classificação, utilizando os mesmos dados climáticos e sazonais explorados anteriormente. O primeiro modelo será treinado com as variáveis originais do dataset, enquanto o segundo utilizará apenas os dois primeiros componentes principais extraídos via PCA. Isso nos permitirá avaliar se a redução de dimensionalidade também beneficia a tarefa de classificação, além da regressão.

Para avaliar o desempenho desses modelos, utilizaremos métricas clássicas de classificação, como acurácia (proporção de acertos totais), F1-score (média harmônica entre precisão e revocação, útil em classes desbalanceadas) e a matriz de confusão, que detalha os acertos e erros por categoria. Também incluiremos visualizações mais avançadas, como as curvas ROC e as fronteiras de decisão em 2D, que ajudam a entender como o modelo separa as diferentes classes no espaço de atributos.

Fique Alerta!

É importante notar que, apesar da similaridade com a tarefa de regressão, a natureza da saída muda: aqui, o modelo retorna uma classe discreta, em vez de um valor contínuo como a produtividade (kg/ha). Isso exige adaptações no pré-processamento e na forma de interpretar os resultados.

4.7.1 Preparação dos Dados

Para preparar os dados para essa nova tarefa, começamos convertendo a variável-alvo `safra` em valores numéricos: baixa será representada por 0, média por 1 e alta por 2. Em seguida, selecionamos as mesmas variáveis preditoras utilizadas na regressão e aplicamos uma padronização com `StandardScaler`, a fim de garantir que todas as variáveis tenham média zero e variância unitária. Também realizamos uma versão paralela do dataset com apenas os dois componentes principais (PC1 e PC2), o que permitirá uma comparação direta entre os modelos com e sem PCA.

Por fim, dividimos o conjunto de dados em treino e teste, utilizando uma proporção de 70% para treino e 30% para teste. Essa divisão é feita com estratificação por classe, garantindo

que todas as categorias de safra estejam proporcionalmente representadas em ambas as partes. Com os dados preparados, podemos então treinar os modelos e comparar os resultados.

Copie e Teste!

```
# 1. Mapeia a variável alvo (safra)
mapa_safra = {'baixa': 0, 'media': 1, 'alta': 2}
df['safra_num'] = df['safra'].map(mapa_safra)

# 2. Define variáveis preditoras (mesmas da regressão)
X_class = df[['chuva_flor', 'chuva_colheita', 'chuva_total', '
   anomalia_flor', 'temp_flor', 'umid_flor', 'chuva_relativa']]
y_class = df['safra_num']

# 3. Padronização
from sklearn.preprocessing import StandardScaler
scaler_class = StandardScaler()
X_class_scaled = scaler_class.fit_transform(X_class)

# 4. PCA (opcional — será usado para um dos modelos)
from sklearn.decomposition import PCA
pca_class = PCA(n_components=2)
X_class_pca = pca_class.fit_transform(X_class_scaled)

# 5. Divisão treino/teste
from sklearn.model_selection import train_test_split
X_train_class, X_test_class, y_train_class, y_test_class =
   train_test_split(
    X_class_scaled, y_class, test_size=0.3, random_state=42,
   stratify=y_class
)
X_train_pca, X_test_pca, _, _ = train_test_split(
    X_class_pca, y_class, test_size=0.3, random_state=42, stratify
   =y_class
)
```

4.7.2 Treinamento dos Modelos

Para a tarefa de classificação multiclasse da safra, utilizaremos como modelo base a Regressão Logística. Apesar do nome, esse modelo é amplamente reconhecido por seu desempenho em tarefas de classificação, inclusive quando envolvem mais de duas categorias, como é o caso aqui de baixa, média ou alta produtividade.

A Regra básica da regressão logística é interpretar a saída do modelo como probabilidades associadas a cada classe. Após o cálculo dessas probabilidades, o modelo prediz como classe final aquela com o maior valor entre elas. Essa abordagem fornece não apenas uma decisão, mas também uma medida de confiança associada a cada predição.

Neste trabalho, empregaremos a estratégia conhecida como ”one-vs-rest”, na qual o modelo aprende a distinguir cada classe em relação a todas as outras. Isso permite adaptar facilmente a regressão logística para cenários multiclasse, sem perder a simplicidade e a efici-

ência do algoritmo. Treinaremos dois modelos distintos, com estruturas paralelas para fins de comparação. O primeiro modelo será ajustado com as variáveis climáticas e sazonais originais, devidamente padronizadas. Já o segundo utilizará apenas os dois primeiros componentes principais (PC1 e PC2), obtidos previamente via PCA.

Copie e Teste!

```
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier

# 1. Modelo com variáveis originais
modelo_classico = OneVsRestClassifier(LogisticRegression(max_iter
   =1000))
modelo_classico.fit(X_train_class, y_train_class)

# 2. Modelo com PCA
modelo_pca_class = OneVsRestClassifier(LogisticRegression(max_iter
   =1000))
modelo_pca_class.fit(X_train_pca, y_train_class)
```

4.7.3 Avaliação dos Modelos

Após o treinamento dos modelos de classificação, é essencial avaliar seu desempenho de maneira rigorosa e interpretável. Para isso, comparamos os resultados dos modelos com e sem PCA utilizando um conjunto diversificado de métricas e visualizações. As métricas adotadas para essa análise incluem a acurácia, que mede a proporção de classificações corretas em relação ao total de exemplos; o F1-score médio macro, que calcula o equilíbrio entre precisão e recall para cada classe, considerando todas com igual importância; e a matriz de confusão, que revela com clareza onde o modelo está acertando e onde ocorrem os principais erros de classificação.

Além disso, utilizamos a curva ROC multiclasse, que permite avaliar a capacidade do modelo de distinguir entre as diferentes classes, mesmo em cenários com mais de duas categorias. Para facilitar a análise e manter a consistência, desenvolvemos uma função de avaliação que automatiza a geração de todas essas métricas e gráficos.

Copie e Teste!

```
from sklearn.metrics import confusion_matrix, accuracy_score,
   f1_score, classification_report, roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import seaborn as sns

def plotar_curva_roc_multiclasse(y_true, y_score, classes, titulo=
   "Modelo"):
    y_bin = label_binarize(y_true, classes=classes)
    n_classes = y_bin.shape[1]
    fpr, tpr, roc_auc = {}, {}, {}
```

```
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    fpr["micro"], tpr["micro"], _ = roc_curve(y_bin.ravel(),
   y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure(figsize=(8, 6))
    colors = cycle(['#1f77b4', '#ff7f0e', '#2ca02c'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Classe
    {i} (AUC = {roc_auc[i]:.2f})')

    plt.plot(fpr["micro"], tpr["micro"], color='black', linestyle=
   '--', label=f"Média micro (AUC = {roc_auc['micro']:.2f})")
    plt.plot([0, 1], [0, 1], 'k--', lw=1)
    plt.xlabel("Taxa de Falsos Positivos (FPR)")
    plt.ylabel("Taxa de Verdadeiros Positivos (TPR)")
    plt.title(f"Curva ROC Multiclasse — {titulo}")
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def avaliar_modelo_classificacao(nome, y_true, y_pred, y_prob=None
   , classes=[0, 1, 2]):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels
   =['Baixa', 'Média', 'Alta'], yticklabels=['Baixa', 'Média', '
   Alta'])
    plt.xlabel("Predito")
    plt.ylabel("Real")
    plt.title(f"Matriz de Confusão — {nome}", pad=12)
    plt.tight_layout()
    plt.show()

    print(f"\nAvaliação — {nome}")
    print("Acurácia:", accuracy_score(y_true, y_pred))
    print("F1-score (macro):", f1_score(y_true, y_pred, average='
   macro'))
    print("\nRelatório de Classificação:")
    print(classification_report(y_true, y_pred, target_names=['
   Baixa', 'Média', 'Alta'], zero_division=0))

    if y_prob is not None:
        plotar_curva_roc_multiclasse(y_true, y_prob, classes,
   titulo=nome)
```

```
# Previsões e probabilidades - Modelo Sem PCA
y_pred_classico = modelo_classico.predict(X_test_class)
y_prob_classico = modelo_classico.predict_proba(X_test_class)
avaliar_modelo_classificacao("Modelo Sem PCA", y_test_class,
   y_pred_classico, y_prob_classico)
```

Resultado Esperado

```
Avaliação — Modelo Sem PCA
Acurácia: 0.5833333333333334
F1-score (macro): 0.3833333333333333

Relatório de Classificação:
              precision    recall  f1-score   support
       Baixa       0.33      0.50      0.40         2
       Média       0.00      0.00      0.00         3
        Alta       0.67      0.86      0.75         7
    accuracy                           0.58        12
   macro avg       0.33      0.45      0.38        12
weighted avg       0.44      0.58      0.50        12
```

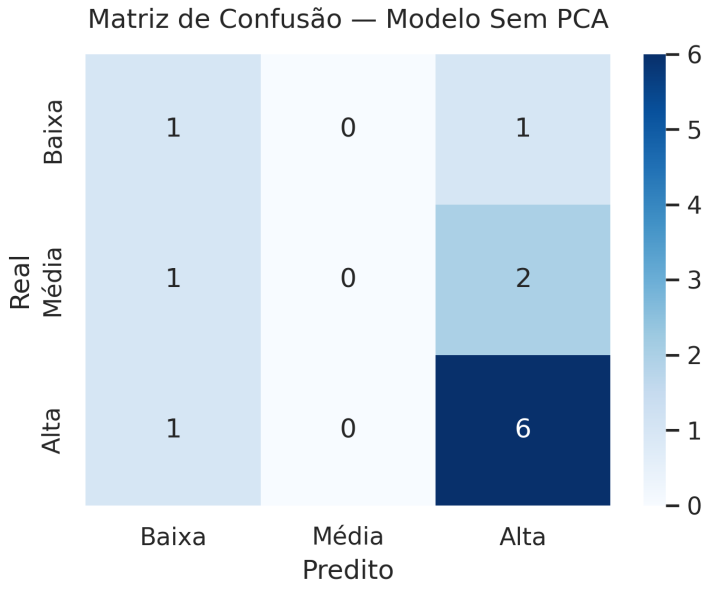


Figura 4.18: Matriz de confusão para o modelo de classificação treinado nos dados originais (sem PCA). A matriz revela a dificuldade do modelo em identificar a classe 'Média', que não teve nenhuma instância classificada corretamente (linha central).

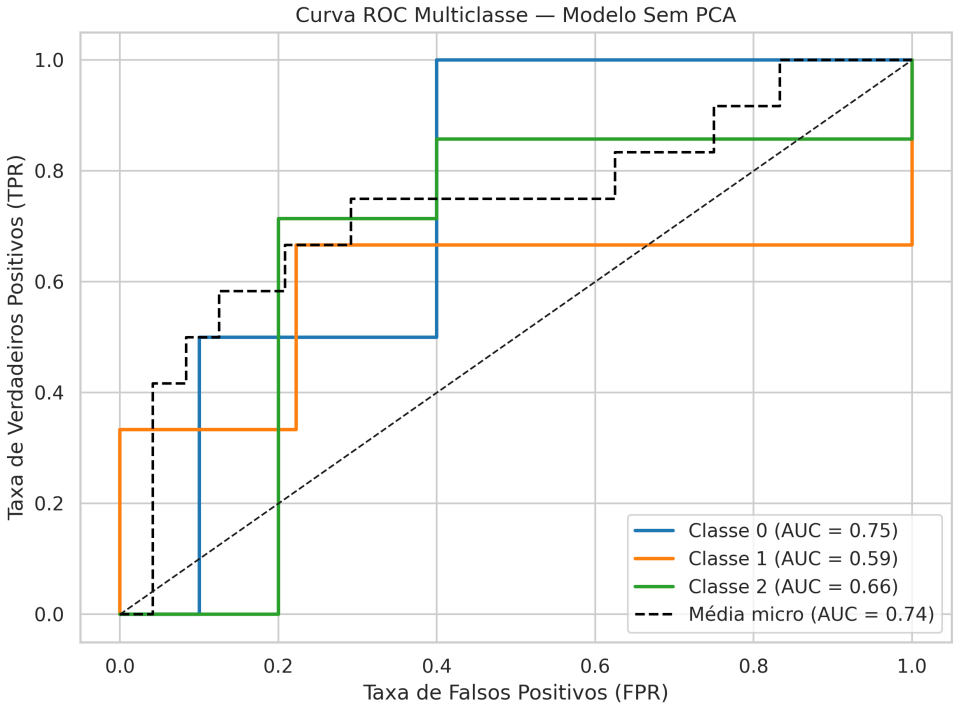


Figura 4.19: Curva ROC multiclasse para o modelo sem PCA. O valor da AUC para cada classe indica a capacidade do modelo de distinguir aquela classe das demais. A linha tracejada preta representa a média micro, fornecendo uma medida de desempenho agregado através de todas as classes.

Copie e Teste!

```
# Previsões e probabilidades com PCA
y_pred_pca = modelo_pca_class.predict(X_test_pca)
y_prob_pca = modelo_pca_class.predict_proba(X_test_pca)
avaliar_modelo_classificacao("Modelo com PCA", y_test_class,
   y_pred_pca, y_prob_pca)
```

Resultado Esperado

```
Avaliação — Modelo com PCA
Acurácia: 0.5833333333333334
F1-score (macro): 0.3833333333333333

Relatório de Classificação:
              precision    recall  f1-score   support
       Baixa       0.33      0.50      0.40         2
       Média       0.00      0.00      0.00         3
        Alta       0.67      0.86      0.75         7
    accuracy                           0.58        12
   macro avg       0.33      0.45      0.38        12
weighted avg       0.44      0.58      0.50        12
```

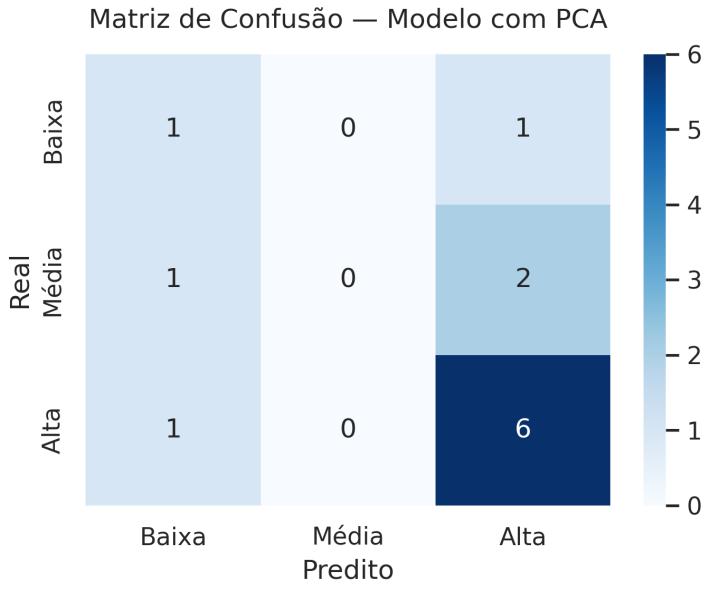


Figura 4.20: Matriz de confusão para o modelo de classificação treinado após a aplicação do PCA. Comparando com o modelo sem PCA (Figura 4.18), esta matriz permite analisar o impacto da redução de dimensionalidade nos padrões de acertos e erros de classificação em relação às classes 'Baixa', 'Média' e 'Alta'.

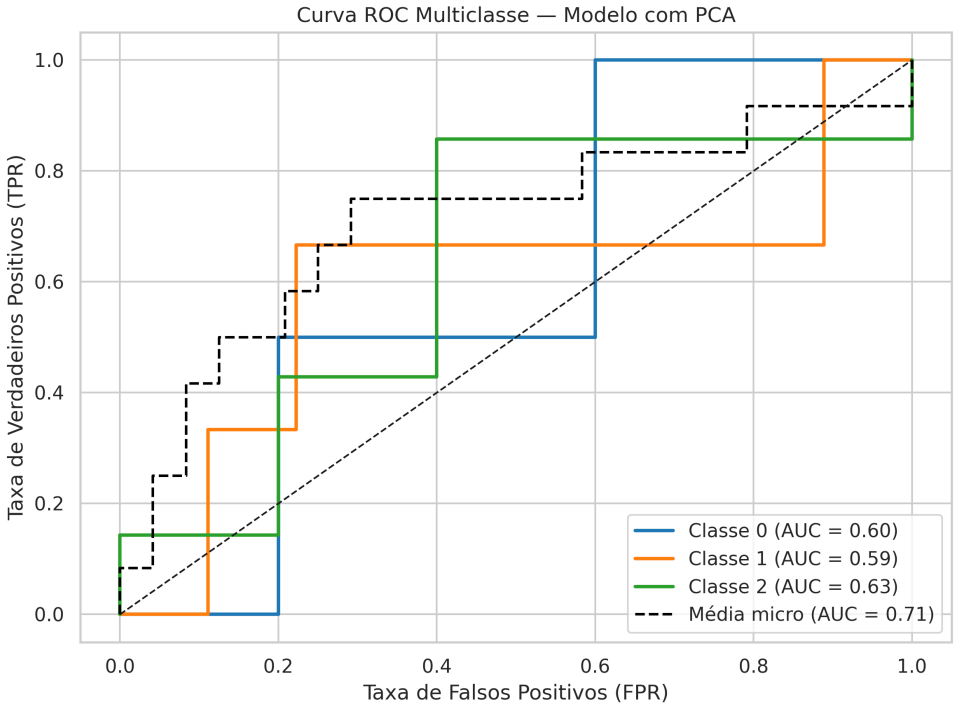


Figura 4.21: Curva ROC multiclasse para o modelo com PCA. Esta visualização complementa a análise da matriz de confusão, mostrando a capacidade discriminatória do modelo para cada classe no novo espaço de características. A comparação com a curva ROC sem PCA (Figura 4.19) pode indicar ganhos ou perdas na separabilidade das classes devido à redução de dimensionalidade.

4.7.4 Visualização das Fronteiras de Decisão

Uma das vantagens de utilizar apenas dois componentes principais (PC1 e PC2) é a possibilidade de representar visualmente as fronteiras de decisão do modelo em um plano bidimensional. Essa visualização permite compreender de forma intuitiva como o modelo está segmentando o espaço dos dados em regiões associadas a cada classe de safra, baixa, média ou alta.

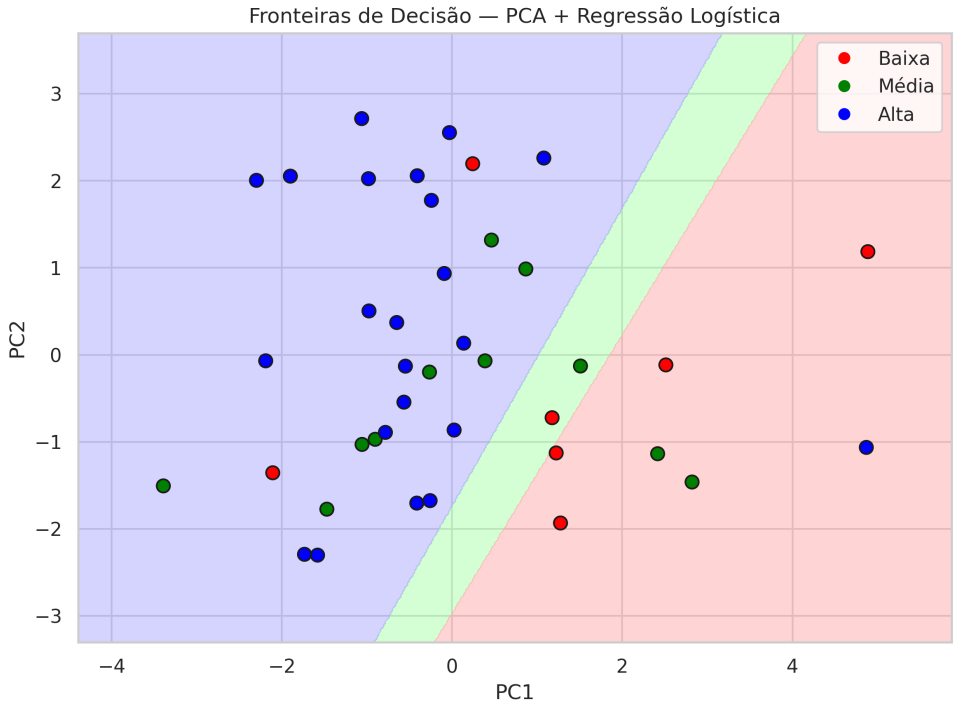


Figura 4.22: Fronteiras de decisão do modelo de Regressão Logística no espaço bidimensional dos componentes principais (PC1 e PC2). As linhas que separam as regiões coloridas mostram como o modelo segmenta o espaço de características para classificar as safras.

Copie e Teste!

```
from matplotlib.colors import ListedColormap

def plot_fronteira_decisao_2D(X_pca, y_true, modelo, titulo="
   Fronteiras de Decisão (PCA)"):
    h = 0.02 # Passo da malha
    x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
    y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(
   y_min, y_max, h))

    Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ['red', 'green', 'blue']
```

```
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.5)
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_true, cmap
   =ListedColormap(cmap_bold), edgecolor='k', s=60)
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.title(titulo)
    plt.legend(handles=scatter.legend_elements()[0], labels=['
   Baixa', 'Média', 'Alta'])
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# Aplicar para os dados do PCA e o modelo treinado com PCA
plot_fronteira_decisao_2D(X_class_pca, y_class, modelo_pca_class,
   titulo="Fronteiras de Decisão — PCA + Regressão Logística")
```

4.7.5 Discussão dos Resultados

Nesta etapa final, avaliamos o desempenho dos modelos de classificação multiclasse em dois cenários distintos: O modelo treinado com os dados padronizados, mas sem PCA; e o modelo treinado com os dois primeiros componentes principais, ou seja, com PCA. Ambos os modelos foram avaliados por meio de diferentes métricas, incluindo matriz de confusão, acurácia, F1-score macro e curvas ROC por classe com média micro.

Os resultados mostraram que os dois modelos tiveram desempenhos praticamente idênticos, com valores muito semelhantes de acurácia, F1-score e área sob a curva ROC (AUC). Isso indica que a aplicação do PCA não comprometeu o desempenho do modelo, mas também não trouxe ganhos significativos em termos de classificação.

Apesar da equivalência nos resultados, o uso do PCA teve um papel fundamental do ponto de vista didático. Ele permitiu a visualização clara das fronteiras de decisão do modelo em um plano bidimensional, facilitando a compreensão de como o modelo separa as diferentes classes. Além disso, possibilitou representar geometricamente as curvas ROC multiclasse, reforçando conceitos de classificação probabilística e avaliação de desempenho. Por fim, serviu como ponto de partida para discutir o equilíbrio entre complexidade do modelo e capacidade de generalização, um tema central em aprendizado de máquina.

4.8 Aprendizagem por Reforço com Irrigação Inteligente

Nos tópicos anteriores, trabalhamos com aprendizado supervisionado, no qual fornecemos exemplos prontos como dados climáticos e produtividade, e o modelo aprende a prever o resultado correto. Mas, e se o modelo não souber a resposta certa de antemão? E se ele precisar interagir com um ambiente, tomar decisões, receber feedback e aprender com as consequências dessas decisões? É exatamente esse o papel da Aprendizagem por Reforço (*Reinforcement Learning*).

Para aplicar a aprendizagem por reforço de forma prática e intuitiva, vamos modelar um ambiente agrícola simplificado, no qual o agente deve decidir como irrigar o solo. Esse ambiente pode apresentar três estados distintos do solo: seco, quando há necessidade urgente

de água; ideal, quando a umidade está equilibrada; e encharcado, quando há excesso de água, o que pode ser prejudicial à planta. Cada um desses estados será representado por um código numérico: 0 para ”seco”, 1 para ”ideal”e 2 para ”encharcado”.

Copie e Teste!

```
import random
import pandas as pd
from IPython.display import display

# Parâmetros
alpha = 0.5      # taxa de aprendizado
gamma = 0.9      # fator de desconto
epsilon = 0.2    # chance de explorar

# Inicializa a Tabela Q
q_table = {
    'seco': {'regar': 0.0, 'pouca_agua': 0.0, 'nao_regar': 0.0},
    'ideal': {'regar': 0.0, 'pouca_agua': 0.0, 'nao_regar': 0.0},
    'encharcado': {'regar': 0.0, 'pouca_agua': 0.0, 'nao_regar':
   0.0},
}

# Função de transição
def transicao(estado, acao):
    if estado == 'seco':
        if acao == 'regar': return 'ideal'
        else: return 'seco'
    elif estado == 'ideal':
        if acao == 'regar': return 'encharcado'
        elif acao == 'pouca_agua': return 'ideal'
        else: return 'seco'
    elif estado == 'encharcado':
        if acao == 'nao_regar': return 'ideal'
        else: return 'ideal'

# Função de recompensa
def recompensa(estado, acao):
    if estado == 'seco':
        if acao == 'regar': return 5
        elif acao == 'pouca_agua': return 2
        else: return -1
    elif estado == 'ideal':
        if acao == 'nao_regar': return 5
        elif acao == 'pouca_agua': return 2
        else: return -3
    elif estado == 'encharcado':
        if acao == 'nao_regar': return 2
        elif acao == 'pouca_agua': return -1
        else: return -5
```

```
# Treinamento
historico = []
for episodio in range(1, 51):
    estado = random.choice(['seco', 'ideal', 'encharcado'])
    for passo in range(1): # Simplificado
        if random.random() < epsilon:
            acao = random.choice(['regar', 'pouca_agua', '
   nao_regar'])
        else:
            acao = max(q_table[estado], key=q_table[estado].get)

        prox_estado = transicao(estado, acao)
        r = recompensa(estado, acao)
        max_q_prox = max(q_table[prox_estado].values())
        q_atual = q_table[estado][acao]
        q_novo = q_atual + alpha * (r + gamma * max_q_prox -
   q_atual)
        q_table[estado][acao] = q_novo

        historico.append({
            'Episódio': episodio, 'Estado': estado, 'Ação': acao,
            'Recompensa': r, 'Próximo estado': prox_estado, 'Q(s,a
   )': round(q_novo, 2)
        })
        estado = prox_estado

# Resultados
q_df = pd.DataFrame(q_table).T
display(q_df.style.background_gradient(cmap="YlGn"))
historico_df = pd.DataFrame(historico)
display(historico_df.tail(10))
```

	regar	pouca_agua	nao_regar
seco	20.705319	3.462569	10.406105
ideal	-0.147300	5.298162	21.718561
encharcado	-2.299931	1.510972	20.897934

Figura 4.23: Tabela Q final após o treinamento do agente de irrigação. Cada célula representa o valor esperado (qualidade) de se tomar uma ação (coluna) a partir de um determinado estado (linha). Os valores mais altos indicam a política ótima que o agente aprendeu para maximizar sua recompensa ao longo do tempo.

	Episódio	Estado	Ação	Recompensa	Próximo estado	Q(s,a)
40	41	encharcado	nao_regar	2	ideal	14.59
41	42	encharcado	nao_regar	2	ideal	16.34
42	43	ideal	nao_regar	5	seco	19.80
43	44	encharcado	nao_regar	2	ideal	18.08
44	45	encharcado	nao_regar	2	ideal	18.95
45	46	seco	regar	5	ideal	20.71
46	47	seco	nao_regar	-1	seco	10.41
47	48	ideal	nao_regar	5	seco	21.72
48	49	encharcado	nao_regar	2	ideal	20.25
49	50	encharcado	nao_regar	2	ideal	20.90

Figura 4.24: Histórico das últimas 10 decisões tomadas pelo agente de irrigação, mostrando o estado observado e a ação escolhida em cada passo de tempo. Esta visualização serve para uma análise qualitativa da política aprendida, permitindo observar o comportamento do agente em resposta a diferentes condições do ambiente.

4.8.1 Como analisar?

Após os 50 episódios de simulação, o agente construiu uma Tabela Q, onde cada valor representa a qualidade esperada de uma ação em determinado estado. É com base nessa tabela que ele decide o que fazer no futuro. Para cada estado do ambiente, a ação com maior valor na Tabela Q é considerada a mais vantajosa, ou seja, a decisão preferida pelo agente. Se o aprendizado ocorreu de forma adequada, esperamos ver algo assim:

- No estado **seco**, a melhor ação deve ser **pouca_agua** ou **regar**
- No estado **ideal**, as melhores escolhas devem ser **nao_regar** ou **pouca_agua**
- No estado **encharcado**, a ação mais segura tende a ser **nao_regar**

4.8.2 Perguntas para reflexão

1. Qual ação o agente aprendeu para cada estado? Observe a Tabela Q: qual ação tem o maior valor em cada linha? Isso está de acordo com as recompensas que você definiu?
2. O agente aprendeu a evitar ações prejudiciais? Ações com penalidades (exemplo: regar no estado encharcado) ficaram com valores baixos?
3. Se o agente pudesse treinar por mais tempo, por exemplo 500 episódios, o que você imagina que aconteceria? Os valores da Tabela Q tenderiam a se estabilizar? As decisões do agente seriam ainda mais confiáveis?

4. E se mudarmos as recompensas? Altere os valores da matriz de recompensas e execute a simulação novamente. Quais ações passaram a ser preferidas? Algum comportamento do agente mudou?

Fique Alerta!

Diferentemente do aprendizado supervisionado, o agente não precisa conhecer a resposta certa. Ele aprende com as consequências de suas escolhas, ajustando sua estratégia ao longo do tempo. Esse é o grande poder da Aprendizagem por Reforço: descobrir, por tentativa e erro, como tomar boas decisões em ambientes dinâmicos.

4.8.3 Desafio Final: Experimente e teste o agente

Agora que você viu como o agente aprende a partir de recompensas e transições de estado, é hora de assumir o controle e explorar diferentes cenários por conta própria. Façamos o seguinte:

- **Altere as recompensas:** Penalize mais o desperdício de água. Recompense melhor ações que mantêm o solo em estado ideal.
- **Aumente o número de episódios:** Altere `range(1, 51)` para `range(1, 201)` e observe. A Tabela Q se estabiliza melhor? O agente passa a evitar ações ruins com mais consistência?
- **Experimente diferentes valores de épsilon:** Teste com 0.0, 0.1, 0.5, 1.0. O agente aprende melhor com mais exploração? O que acontece se ele nunca explorar?
- **Crie um novo estado:** Por exemplo: `muito_seco`. Adicione regras e recompensas novas. Como o agente reage?
- **Altere as transições de estado:** E se regar muito sempre levasse a encharcamento, mesmo quando o solo estava ideal?

Fique Alerta!

Fique sempre de olho na Tabela Q final: ela é o reflexo do que o agente ”aprendeu”. A melhor maneira de dominar Aprendizagem por Reforço é experimentar, errar, testar novamente e interpretar os resultados.

4.8.4 O que aprendemos de verdade?

Neste curso, exploramos como o aprendizado de máquina pode ser usado para enfrentar desafios reais, como prever e classificar a produtividade do guaraná no município de Maués, no coração do Amazonas. Trabalhamos com dados reais, com toda a sua riqueza e imperfeição para entender os limites e o potencial da inteligência artificial diante de perguntas complexas do mundo real, onde:

- Aplicamos regressão linear para prever a produtividade (kg/ha);
- Utilizamos classificação multiclasse para categorizar a safra em “baixa”, “média” ou “alta”;
- Testamos técnicas de pré-processamento, como padronização, PCA e regularização;

- Visualizamos os erros, resíduos, curvas de decisão e até as superfícies de custo;
- Ainda fizemos um desafio de aprendizagem por reforço, simulando decisões de irrigação.

Entretanto mesmo com um pipeline bem estruturado, os modelos apresentaram desempenho limitado. **O motivo?** As variáveis climáticas e sazonais não explicaram bem a produtividade. Nem mesmo com PCA ou regularização houve melhora significativa. E isso não é um erro do aluno, é uma lição essencial: Que em projetos reais, nem sempre os dados disponíveis são suficientes para resolver o problema.

Devemos entender que aprender machine learning não é só treinar modelos, mas entender o problema e os dados; que as limitações dos dados não são falhas, mas sim oportunidades para pensar criticamente; a ciência de dados na agricultura é complexa, mas cheia de potencial; E a realidade amazônica exige criatividade, adaptação e respeito ao contexto.

AGORA É COM VOCÊ: EXPLORE, MODELE, EXPERIMENTE, E COLHA OS FRUTOS DO SEU APRENDIZADO!

Referências Bibliográficas

[1] GÉRON, Aurélien. **Aprenda Machine Learning com Scikit-Learn, Keras e TensorFlow**. 3. ed. Sebastopol: O'Reilly Media, 2023.

[2] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. Cambridge: MIT Press, 2016. Disponível em: `https://www.deeplearningbook.org`.

[3] MARQUES, Bruno Torres; MARQUES, Leonardo Torres. **Aprendizado de máquina: uma abordagem para descoberta de conhecimento**. [S.l.]: Novas Edições Acadêmicas, 2022.

[4] MITCHELL, Tom M. **Machine Learning**. New York: WCB/McGraw-Hill, 1997.

[5] MUELLER, John Paul. **Aprendizado de máquina para leigos**. Rio de Janeiro: Alta Books, 2019.

[6] NG, Andrew. **Machine Learning Specialization**. [Curso online]. [S.l.]: DeepLearning.AI; Stanford Online, 2022. Disponível em: `https://www.coursera.org/specializations/machine-learning-introduction`.

[7] NG, Andrew. **Machine Learning Yearning: technical strategy for AI engineers, in the era of deep learning**. [S.l.]: deeplearning.ai project, 2018. Disponível em: `https://info.deeplearning.ai/machine-learning-yearning-book/`.

[8] RASCHKA, Sebastian. **Python Machine Learning: machine learning e deep learning com Python, Scikit-learn e TensorFlow 2**. São Paulo: Novatec, 2021.

[9] RUSSELL, Stuart; NORVIG, Peter. **Artificial Intelligence: a modern approach**. 3. ed. Upper Saddle River: Prentice Hall, 2010.

FOTO DE AMAR PRECIADO

ISBN: 978-65-01-74472-8

CITHA

Capacitação e Interiorização em
Tecnologias Habilitadoras na Amazônia