

Aprendizado de máquinas

Thiago Rodrigo Ramos

17 de junho de 2025

Sumário

1	Introdução	7
1.1	Um breve histórico do aprendizado estatístico	7
1.1.1	Inferência vs Predição	8
1.1.2	As duas culturas de Breiman	9
1.2	Algumas tarefas clássicas de aprendizado	10
1.3	Exemplos	10
1.3.1	Salários	10
1.3.2	Mercado de ações	11
1.4	Um conselho: a importância de ser ruim antes de ser bom	12
2	Fundamentos do aprendizado supervisionado	13
2.1	Formulação do problema	13
2.2	Decomposição de viés-variância em regressão	15
2.3	Data Splitting e Validação Cruzada	18
3	Introdução à regressão via mínimos quadrados	21
3.1	Mínimos quadrados	21
3.2	Resolução Numérica	25
3.3	Estimativas de erros para regressão linear	25
3.4	Um pouco de inferência	27
4	Introdução à classificação via regressão logística	29
4.1	Classificador de Bayes	29
4.1.1	O Modelo Logístico	31
4.1.2	Estimando os Coeficientes da Regressão	32
4.2	Um pouco de otimização - O método de Newton	33
4.3	Outras métricas de avaliação	35
4.4	Ajuste do limiar de decisão	36
4.4.1	Escolhendo o limiar de decisão	37
5	KNN	41
5.1	KNN para classificação	41
5.2	KNN para regressão	44
5.3	O que é treinado no KNN?	44

5.4 Regressão Linear vs. KNN	45
6 Modelos baseados em árvores	49
6.1 Árvores de decisão	49
6.1.1 Divisão Binária Recursiva	50
6.1.2 Poda de Árvores (Tree Pruning)	52
6.1.3 Árvores de Classificação	53
6.1.4 Árvores versus modelos lineares	56
6.1.5 Vantagens e Desvantagens das Árvores	58
6.2 Bagging	58
6.2.1 Estimativa do Erro Out-of-Bag (OOB)	59
6.2.2 Importância de Variáveis em Modelos Bagged	61
6.3 Random Forests	61
6.3.1 Impacto dos Parâmetros B e m	62
6.3.2 Uso de Random Forests em Altas Dimensões	63
7 Seleção de modelos lineares e regularização	65
7.1 Seleção do melhor subconjunto (Best Subset Selection)	66
7.1.1 Seleção Foward e Backward	67
7.2 Ridge e Lasso	72
7.2.1 Regressão Ridge (Ridge Regression)	72
7.2.2 Regressão Lasso	73
7.2.3 Formulações Alternativas de Ridge e Lasso	73
7.2.4 Interpretação via encolhimento	75
7.2.5 Interpretação Bayesiana de Ridge e Lasso	76
8 Boosting	79
8.1 AdaBoost	79
8.1.1 Cálculo do erro empírico	82
8.1.2 Um pouco de teoria	84
8.2 Gradient Boosting	86
9 SVM	89
9.1 Caso separável	90
9.1.1 Problema primal	90
9.1.2 Um pouco de otimização convexa	92
9.1.3 Vetores de suporte	92
9.1.4 Um breve comentário sobre dualidade	93
9.1.5 Problema dual	94
9.2 Caso não separável	95
9.3 O truque do kernel	96
9.3.1 Formulações primal e dual com kernel	97

SUMÁRIO	5
10 Redução de dimensão	99
10.1 Análise de componentes principais	99
10.1.1 Variância explicada	101
10.1.2 Decomposição em valores singulares	102
10.1.3 Aplicação: redução de dimensão	106
10.1.4 Aplicação: compressão de dados	107
10.2 Projeções aleatórias	108
10.2.1 Aplicação	109
10.3 t-SNE	110
10.3.1 Aplicação 1	111
11 K-Médias	113
A Revisão	117
A.1 Álgebra linear	117
A.1.1 Multiplicações	119
A.1.2 Mudança de base	120
A.1.3 Produtos internos	121
A.1.4 Ortogonalidade	124
A.2 Probabilidade	129
A.2.1 Variáveis aleatórias	129
A.2.2 Probabilidade condicional e independência	130
A.2.3 Algumas fórmulas importantes	130
A.2.4 Esperança e desigualdade de Markov	131
A.2.5 Variância e a desigualdade de Chebyshev	131
A.2.6 Covariância	132
A.2.7 Teoremas assintóticos	133
A.2.8 Função geradora de momentos	134
B Guia de desigualdades	135
C Ferramentas computacionais	137
C.1 Git	137
C.2 Python	138
C.3 Poetry	138

Material do curso

Todo o material utilizado neste curso, incluindo códigos e notebooks, pode ser acessado no repositório do GitHub: https://github.com/thiagorr162/curso_aprendizado.

Referências principais

O conteúdo deste curso é baseado em referências que cobrem tópicos fundamentais de aprendizado de máquina e estatística. [Izbicki and dos Santos \(2020\)](#) introduz o aprendizado de máquina com ênfase em uma abordagem estatística, voltada ao público brasileiro. [James et al. \(2013\)](#) apresentam métodos estatísticos aplicados à aprendizagem supervisionada e não supervisionada, com exemplos em *R* e *Python*. [Hastie et al. \(2001\)](#) abordam técnicas avançadas e a teoria estatística por trás de algoritmos de aprendizado de máquina. [Shalev-Shwartz and Ben-David \(2014\)](#) desenvolvem a teoria do aprendizado e a análise de algoritmos, com foco na compreensão matemática das técnicas. [Mohri et al. \(2018\)](#) tratam de conceitos fundamentais de generalização e estabilidade, além de fornecer uma base teórica para diversos algoritmos modernos.

Todas esses livros podem ser baixadas online de forma legal nos sites dos autores.

Capítulo 1

Introdução

Aprendizado de máquina é um termo utilizado para descrever sistemas capazes de identificar automaticamente padrões e regularidades em dados ([Shalev-Shwartz and Ben-David, 2014](#)). Nos últimos anos, essa área consolidou-se como uma ferramenta indispensável para atividades que envolvem a análise e interpretação de grandes volumes de informação. Hoje em dia, essa tecnologia está presente em nosso cotidiano: motores de busca ajustam seus resultados para atender melhor às nossas consultas (ao mesmo tempo em que exibem anúncios), filtros de *spam* são aperfeiçoados para proteger nossas caixas de e-mail, e sistemas de detecção de fraudes asseguram a integridade de transações financeiras realizadas com cartões de crédito. Além disso, câmeras digitais reconhecem rostos, assistentes virtuais em *smartphones* interpretam comandos de voz e veículos utilizam algoritmos inteligentes para prevenir acidentes. O aprendizado de máquina também desempenha papel crucial em diversas áreas da ciência, como a bioinformática, a medicina e a astronomia.

1.1 Um breve histórico do aprendizado estatístico

Como descrito em [James et al. \(2013\)](#), embora o termo *aprendizado estatístico* seja relativamente recente, muitos dos conceitos fundamentais da área foram estabelecidos há bastante tempo. No início do século XIX, surgiu o método dos mínimos quadrados, que representa uma das primeiras formas do que hoje conhecemos como regressão linear. Essa técnica foi aplicada com sucesso, inicialmente, em problemas de astronomia. A regressão linear é amplamente utilizada para prever variáveis quantitativas, como o salário de um indivíduo, por exemplo.

Com o objetivo de prever variáveis qualitativas — como determinar se um paciente sobreviverá ou não, ou se o mercado financeiro terá alta ou queda —, foi proposta em 1936 a análise discriminante linear. Já na década de 1940, autores sugeriram uma abordagem alternativa: a regressão logística. No início dos anos 1970, o conceito de *modelos lineares generalizados* foi introduzido, englobando tanto a regressão linear quanto a logística como casos particulares dentro de uma estrutura mais ampla.

Até o final da década de 1970, diversas técnicas para aprendizado a partir de dados já estavam disponíveis, embora fossem predominantemente lineares, devido às limitações computacionais da época para modelagem de relações não lineares. A partir dos anos 1980, com o avanço da

tecnologia, métodos não lineares passaram a ser mais acessíveis. Nesse período surgiram as árvores de decisão para classificação e regressão, seguidas pelos modelos aditivos generalizados. Ainda nos anos 1980, as redes neurais ganharam destaque, e nos anos 1990, as máquinas de vetor de suporte (*support vector machines*) foram introduzidas.

Desde então, o aprendizado estatístico consolidou-se como um subcampo da estatística dedicado à modelagem e predição em cenários supervisionados e não supervisionados. Nos últimos anos, o progresso na área foi impulsionado pela crescente disponibilidade de softwares poderosos e acessíveis, como a linguagem de programação Python, que é gratuito e de código aberto. Esse avanço vem contribuindo para ampliar o alcance das técnicas de aprendizado estatístico, tornando-as uma ferramenta essencial não apenas para estatísticos e cientistas da computação, mas também para profissionais de diversas outras áreas.

1.1.1 Inferência vs Predição

Como descrito em [Izbicki and dos Santos \(2020\)](#), em problemas supervisionados, é importante distinguir entre dois objetivos fundamentais: a inferência e a predição. Essas duas abordagens guiam a forma como modelos são construídos e avaliados.

- **Objetivo inferencial** diz respeito à compreensão da relação entre as covariáveis x e a variável resposta Y . Nesse caso, queremos responder perguntas como: quais covariáveis são mais relevantes para explicar Y ? Qual a direção e a magnitude do efeito de cada preditor? Esse tipo de análise é útil quando o interesse está em interpretar o modelo, entender a estrutura dos dados ou formular hipóteses científicas.
- **Objetivo preditivo**, por outro lado, está focado em construir uma função $g : \mathbb{R}^d \rightarrow \mathbb{R}$ que tenha boa capacidade de prever Y para novas observações não vistas durante o treinamento. O sucesso neste contexto é medido pela capacidade do modelo em generalizar para dados futuros, mesmo que isso ocorra às custas de uma menor interpretabilidade do modelo.

Para ilustrar essas distinções, vejamos dois exemplos práticos:

- No **Exemplo 1.3** (*Isomap face data*), cada observação consiste em uma imagem de um rosto humano, e o objetivo é prever a direção para a qual a pessoa está olhando (variável y) com base nos pixels da imagem (variáveis x). Esse é um exemplo puramente preditivo, pois a principal meta é estimar corretamente a direção do olhar em novas imagens. O modelo não busca explicar quais regiões da imagem são mais relevantes ou como cada pixel individual influencia a resposta, mas sim gerar boas previsões.
- Já no **Exemplo 1.4** (*Million Song Dataset*), o banco de dados contém informações sobre diversas características de músicas (como timbre, energia, dançabilidade etc.) e o ano de lançamento de cada uma delas. Nesse caso, o problema tem um caráter misto. Por um lado, queremos prever o ano de lançamento a partir das covariáveis disponíveis (objetivo preditivo). Por outro, pode haver interesse em entender como cada característica da música se relaciona com o ano de lançamento, como por exemplo investigar se músicas dos anos 70 são de fato mais "dançantes" do que as atuais (objetivo inferencial).

Portanto, enquanto alguns problemas são essencialmente preditivos ou inferenciais, outros envolvem uma combinação dos dois. Essa distinção é relevante, pois impacta tanto a escolha do modelo quanto a forma de interpretá-lo e validá-lo.

1.1.2 As duas culturas de Breiman

Leo Breiman foi um estatístico renomado, conhecido por suas contribuições fundamentais à estatística e ao aprendizado de máquina, incluindo o desenvolvimento de métodos como random forests [Breiman \(2001a\)](#). Em seu influente artigo "*Statistical Modeling: The Two Cultures*" ([Breiman, 2001b](#)), Breiman discute duas abordagens distintas para modelagem estatística. Ele inicia seu artigo com o seguinte resumo:

"There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown. The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools."

O artigo teve grande impacto na comunidade estatística e no campo do aprendizado de máquina. Nele, Breiman argumenta que existem duas culturas distintas na modelagem de dados: a **cultura dos modelos estocásticos**, que assume que os dados são gerados por um modelo probabilístico especificado (como regressão linear, modelos lineares generalizados etc.), e a **cultura algorítmica**, que foca na construção de algoritmos preditivos eficazes sem necessariamente se preocupar com a interpretação ou com a modelagem explícita da distribuição dos dados (como árvores de decisão, random forests, redes neurais, entre outros).

Breiman defende que a estatística tradicional estava excessivamente focada em modelos estocásticos, o que limitava seu impacto em problemas práticos, enquanto métodos algorítmicos — amplamente utilizados fora da estatística, especialmente na ciência da computação — estavam mais bem adaptados para resolver problemas com grandes volumes de dados e alta complexidade.

Hoje em dia, o artigo de Breiman é amplamente citado e considerado um marco que antecipou a ascensão de métodos de aprendizado de máquina dentro da estatística e da ciência de dados. No entanto, sua visão também recebeu críticas. Alguns argumentam que as duas culturas não são mutuamente excludentes e que há um valor significativo na modelagem estatística clássica, especialmente quando a interpretação dos parâmetros e a inferência causal são importantes. Além disso, com o avanço dos métodos de aprendizado estatístico e da estatística bayesiana, muitos pesquisadores propõem abordagens híbridas que combinam modelagem interpretável com o poder preditivo dos algoritmos.

Atualmente, o artigo de Breiman é visto como uma provocação importante que incentivou a comunidade a repensar o papel da estatística em problemas do mundo real, mas também é reconhecido que tanto a modelagem estocástica quanto a preditiva têm seu espaço e relevância dependendo do contexto e dos objetivos da análise.

1.2 Algumas tarefas clássicas de aprendizado

A seguir, apresentamos algumas tarefas clássicas de aprendizado de máquina que têm sido amplamente estudadas ([Mohri et al., 2018](#)):

- **Classificação:** consiste em atribuir uma categoria a cada item. Por exemplo, na classificação de documentos, o objetivo é rotular cada texto com categorias como política, negócios, esportes ou clima. Já na classificação de imagens, cada imagem pode ser categorizada como carro, trem ou avião. Em geral, o número de categorias é limitado a algumas centenas, mas pode ser consideravelmente maior em tarefas complexas, como reconhecimento óptico de caracteres (OCR), classificação de textos ou reconhecimento de fala.
- **Regressão:** envolve a predição de um valor numérico contínuo para cada item. Exemplos comuns incluem a previsão de preços de ações ou de indicadores econômicos. Diferentemente da classificação, em regressão o erro de uma predição depende da distância entre o valor real e o valor estimado, enquanto na classificação normalmente não há uma medida de proximidade entre as categorias.
- **Rankeamento:** trata-se de aprender a ordenar itens de acordo com algum critério. Um exemplo típico é o rankeamento de páginas em um motor de busca, onde o sistema precisa retornar os resultados mais relevantes para uma consulta. Outras aplicações de rankeamento aparecem em sistemas de extração de informações e em processamento de linguagem natural.
- **Agrupamento (Clustering):** busca organizar um conjunto de itens em subconjuntos homogêneos. Algoritmos de agrupamento são especialmente úteis na análise de grandes volumes de dados. Na análise de redes sociais, por exemplo, técnicas de clustering são usadas para identificar comunidades ou grupos com características similares dentro de uma rede.
- **Redução de dimensionalidade ou aprendizado de variedades:** refere-se ao processo de transformar uma representação original de dados em uma representação de menor dimensão, preservando certas propriedades estruturais importantes. Um exemplo comum ocorre no pré-processamento de imagens digitais em tarefas de visão computacional.

1.3 Exemplos

1.3.1 Salários

Nesta análise, utilizamos um conjunto de dados que contém informações sobre salários de trabalhadores da região do Atlântico dos Estados Unidos (Fig. 1.3.1). O foco é explorar como fatores

como idade, nível de escolaridade e o ano em que o salário foi registrado influenciam os valores salariais.

	year	age	marital	race	education	region	jobclass	health	health_ins	logwage	wage
0	2006	18	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	75.043154
1	2004	24	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	70.476020
2	2003	45	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	130.982177
3	2003	43	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	154.685293
4	2005	50	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	75.043154

Figura 1.1: Exemplo de registros do conjunto de dados de salários.

Exercício 1. Utilizando o código nesse [link](#). Faça uma análise do comportamento entre as variáveis de idade e salário. Faça o mesmo para nível de escolaridade e salário.

1.3.2 Mercado de ações

Enquanto o conjunto de dados de salários aborda a previsão de uma variável numérica contínua, neste exemplo o objetivo é prever um resultado qualitativo. Trata-se de um problema clássico de classificação, em que desejamos prever categorias ao invés de valores numéricos. Um exemplo

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.1913	0.959	Up
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	Up
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	Down
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.2760	0.614	Up
4	2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213	Up

Figura 1.2: Exemplo de registros do conjunto de dados de ações.

interessante envolve dados do mercado financeiro (Fig. 1.2), que incluem as variações diárias do índice S&P 500 ao longo de um período de cinco anos, entre 2001 e 2005. Esse conjunto de dados, que chamaremos de *Smarket*, busca prever a direção do mercado em um determinado dia (se irá subir ou cair), utilizando como variáveis explicativas as mudanças percentuais dos cinco dias anteriores.

Diferente da tarefa de regressão, aqui o desafio consiste em classificar o movimento do mercado como sendo uma alta (*Up*) ou uma baixa (*Down*). Embora o comportamento passado do índice possa não fornecer uma regra clara para prever o movimento do dia seguinte, pequenas tendências ou padrões podem ser identificados com métodos de aprendizado estatístico.

Exercício 2. Explorar os dados do mercado de ações utilizando esse [código](#).

1.4 Um conselho: a importância de ser ruim antes de ser bom

É natural que, quando começamos a fazer algo, a gente faça essa coisa muito malfeita ou cheia de defeitos. Isso é comum em qualquer processo de aprendizagem, e sempre foi assim, desde o início dos tempos.

Quando comecei a programar em Python, muita coisa sobre a linguagem eu aprendi por conta própria, apesar de já ter feito alguns cursos básicos em C. Programei de forma amadora em Python por muitos anos, até que, no doutorado, precisei aprender a programar de forma mais organizada e profissional. Lembro que, nessa época, um amigo da pós-graduação me apresentou ao "submundo da programação". Foi aí que aprendi muito do que sei hoje sobre terminal do Linux, Git, e foi também quando comecei a usar o Vim.

Uma das coisas que esse amigo me mostrou foi o Pylint, que nada mais é do que um verificador de bugs e qualidade de código para Python. O Pylint é bem rigoroso na análise, e ainda te dá, ao final, uma nota que vai até 10. Nessa fase, apesar de já ter evoluído bastante, meus códigos ainda recebiam notas por volta de 6 ou 7. Resolvi então rodar o Pylint nos meus códigos antigos pra ter uma noção de quão ruins eles eram — e a nota final foi -900. Pois é, existe um limite superior para o quão bem você consegue fazer algo, mas aparentemente o fundo do poço é infinito.

O que eu queria mostrar com essa história é que faz parte do processo de aprendizado ser ruim no começo e melhorar com o tempo. Falo isso porque, hoje em dia, com o crescimento dos LLMs, a gente fica tentado a pular essa etapa de errar muito até acertar, e ir direto pra fase em que escrevemos códigos limpos, bem comentados, identados e organizados. Mas não se enganem: apesar da aparência profissional, depender de LLMs pra escrever tudo atrapalha justamente essa parte essencial de aprender errando.

Nessas notas, vários exercícios envolvem escrever códigos em Python. Meu conselho é: não tenham vergonha de errar, de escrever soluções ruins ou confusas. Isso é absolutamente normal. Vocês estão aqui para evoluir — e errar faz parte do processo.

Capítulo 2

Fundamentos do aprendizado supervisionado

O aprendizado supervisionado é uma das principais áreas do aprendizado de máquina e da estatística, tendo como objetivo construir modelos capazes de prever uma variável de interesse Y a partir de um conjunto de variáveis explicativas X . Este paradigma baseia-se em dados rotulados, ou seja, em observações para as quais tanto as covariáveis quanto a variável de resposta são conhecidas.

2.1 Formulação do problema

Ao longo deste capítulo, assumiremos que dispomos de uma amostra de dados $(X_i, Y_i)_{i=1}^n$, em que cada par (X_i, Y_i) é uma realização de um mesmo par de variáveis aleatórias (X, Y) . Além disso, adotaremos a hipótese de que estas observações são *i.i.d.* (independentes e identicamente distribuídas). Esta suposição simplifica a análise teórica, permitindo o uso de ferramentas como leis dos grandes números e teoremas de concentração. Na prática, embora a hipótese de *i.i.d.* nem sempre seja completamente satisfeita, ela é uma aproximação útil e bastante comum em aplicações reais.

Nosso objetivo será utilizar esse conjunto de dados (ou uma parte dele) para aprender um modelo preditivo, que denotaremos por \hat{g} , de forma que $\hat{g}(X) \approx Y$. O significado da relação de aproximação " \approx " será discutido a seguir.

Ao construir um modelo preditivo $\hat{g}(X)$ para estimar uma variável de interesse Y , é essencial definir uma métrica que quantifique o erro cometido pelas previsões. Essa métrica é chamada de **função de perda**, e mede a discrepância entre o valor verdadeiro Y e a predição $\hat{g}(X)$. Duas escolhas comuns para problemas com resposta contínua (regressão) são:

$$L(Y, \hat{g}(X)) = \begin{cases} (Y - \hat{g}(X))^2 & (\text{erro quadrático}) \\ |Y - \hat{g}(X)| & (\text{erro absoluto}). \end{cases}$$

No caso de problemas com resposta discreta (classificação), um exemplo comum de função de perda é a chamada *perda 0-1*, definida por:

$$L(Y, \hat{g}(X)) = \mathbb{I}\{g(X) \neq Y\},$$

a qual simplesmente contabiliza a quantidade de erros cometidos pelo classificador \hat{g} , isto é, o número de vezes em que a predição difere do valor verdadeiro.

A escolha da função de perda impacta diretamente as propriedades do modelo e como ele responde a diferentes tipos de dados ou outliers.

Erro de teste, também chamado de *erro de generalização*, é o erro de predição em uma amostra de teste independente:

$$R_{\mathcal{D}}(\hat{g}) = \mathbb{E}[L(Y, \hat{g}(X)) \mid \mathcal{D}]$$

onde X e Y são sorteados de sua distribuição conjunta (população). Note que \hat{g} **depende de \mathcal{D}** ! Aqui, o conjunto de treinamento \mathcal{D} é fixo, e o erro de teste se refere ao erro para esse conjunto de treinamento específico. Uma quantidade relacionada é o *erro esperado de predição* (ou erro esperado de teste):

$$R(\hat{g}) = \mathbb{E}[L(Y, \hat{g}(X))] = \mathbb{E}[\text{Err}_{\mathcal{D}}].$$

Note que a esperança acima leva em conta toda a aleatoriedade envolvida, incluindo a aleatoriedade do conjunto de treinamento que gerou \hat{g} .

Nosso objetivo será a estimativa de $\text{Err}_{\mathcal{D}}$, embora veremos que Err é mais acessível do ponto de vista estatístico, e a maioria dos métodos busca estimar efetivamente esse erro esperado. Estimar $\text{Err}_{\mathcal{D}}$ de maneira condicional não é muito viável na prática.

Erro de treinamento é a perda média sobre a amostra de treinamento:

$$\hat{R}(\hat{g}) = \frac{1}{N} \sum_{i=1}^N L(y_i, \hat{g}(x_i)).$$

Nosso interesse está em conhecer o erro de teste esperado do modelo \hat{g} . À medida que o modelo se torna mais complexo, ele se ajusta melhor aos dados de treinamento e passa a capturar estruturas subjacentes mais complicadas. Com isso, ocorre uma redução no viés, mas um aumento na variância. Existe, portanto, um nível intermediário de complexidade do modelo que minimiza o erro esperado de teste.

Infelizmente, o erro de treinamento não é uma boa estimativa do erro de teste. O erro de treinamento tipicamente diminui à medida que a complexidade do modelo aumenta, podendo até atingir zero quando essa complexidade é suficientemente alta. Entretanto, um modelo com erro de treinamento igual a zero está *superajustado* (overfit) aos dados de treinamento e, geralmente, apresentará baixa capacidade de generalização.

Ao longo deste capítulo, descreveremos diversos métodos para estimar o erro de teste esperado de um modelo. Tipicamente, o modelo dependerá de um parâmetro ou de um conjunto de parâmetros de ajuste α , de modo que podemos escrever a predição como $\hat{g}_\alpha(x)$. O parâmetro α controla a complexidade do modelo, e nosso objetivo é encontrar o valor de α que minimize o erro esperado de teste, ou seja, que produza o menor erro médio. Para simplificar a notação, omitiremos frequentemente a dependência explícita de $\hat{g}(x)$ em α .

É importante destacar que temos, na verdade, dois objetivos distintos ao avaliar modelos preditivos:

- **Seleção de modelo:** consiste em comparar diferentes modelos ou configurações a fim de escolher o que apresenta o melhor desempenho.

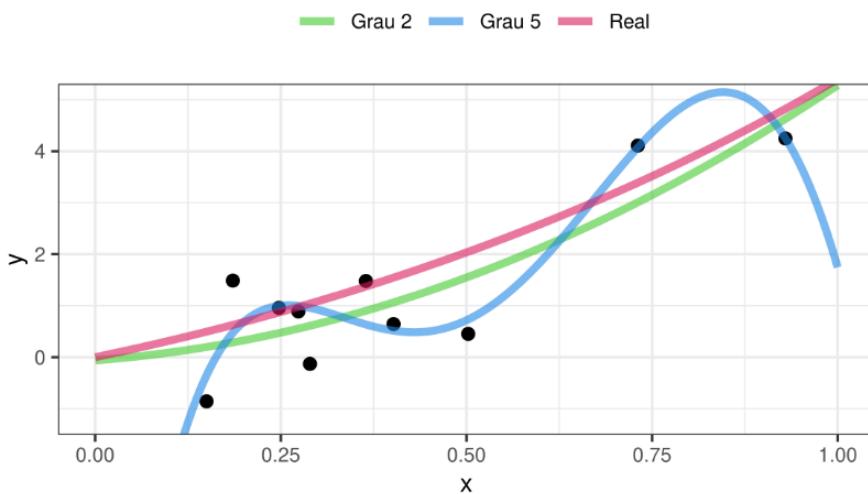


Figura 2.1: Exemplo de sobreajuste. Imagem retirada de ([Izbicki and dos Santos, 2020](#)).

- **Avaliação de modelo:** uma vez escolhido o modelo final, o objetivo passa a ser estimar o seu erro de predição (ou erro de generalização) em novos dados.

Para entender como o erro de predição se comporta em função da complexidade do modelo, começamos analisando o caso da **regressão**, onde é possível decompor o erro de teste esperado em termos de *viés* e *variância*. Essa decomposição fornece intuições valiosas sobre o compromisso entre subajuste e sobreajuste, e servirá de base conceitual para as discussões posteriores sobre classificação e seleção de modelos.

2.2 Decomposição de viés-variância em regressão

As origens dos métodos de regressão remontam a mais de 200 anos, com as contribuições de Legendre (1805) e Gauss (1809), que introduziram o método dos mínimos quadrados para modelar o movimento dos planetas ao redor do Sol. Atualmente, a estimativa de funções de regressão é um dos pilares fundamentais da estatística.

Embora as primeiras soluções para este problema sejam antigas, apenas nas últimas décadas, com o avanço das tecnologias de computação e armazenamento, novas abordagens puderam ser exploradas. Em especial, o crescimento exponencial da quantidade de dados disponíveis tem impulsionado o desenvolvimento de métodos que fazem menos suposições sobre o comportamento real dos fenômenos estudados.

Esse cenário trouxe novos desafios: por exemplo, métodos clássicos muitas vezes não conseguem lidar adequadamente com bancos de dados em que o número de variáveis excede o número de observações, uma situação comum nos contextos atuais. Além disso, aplicações envolvendo dados complexos — como imagens ou textos — têm se tornado frequentes e demandam técnicas mais sofisticadas.

De modo geral, o objetivo de um modelo de regressão é capturar a relação entre uma variável aleatória de interesse $Y \in \mathbb{R}$ e um vetor de covariáveis $\mathbf{x} = (x_1, \dots, x_p) \in \mathbb{R}^p$. O foco está em

estimar a chamada função de regressão, definida por

$$r(\mathbf{x}) := \mathbb{E}[Y | X = \mathbf{x}].$$

A motivação para estudar essa função está relacionada ao problema de minimizar o erro quadrático. Para ilustrar, considere uma variável aleatória Z e a função objetivo

$$\phi(t) = \mathbb{E}[(Z - t)^2],$$

em que buscamos o valor $t \in \mathbb{R}$ que minimiza $\phi(t)$. Derivando em relação a t e igualando a zero, obtemos:

$$\phi'(t) = \mathbb{E}[2(Z - t)] = 0 \Leftrightarrow t = \mathbb{E}[Z].$$

Portanto, o valor ótimo de t que minimiza o erro quadrático é justamente a esperança de Z .

Esse raciocínio se estende naturalmente ao contexto de regressão. Nossa objetivo passa a ser encontrar uma função $g(\mathbf{x})$ que minimize

$$\phi(g) = \mathbb{E}[(Y - g(X))^2] = \mathbb{E}[\mathbb{E}[(Y - g(\mathbf{x}))^2 | X = \mathbf{x}]].$$

Fixando $X = \mathbf{x}$, $g(\mathbf{x})$ se comporta como um número, e, pelo argumento anterior, a minimização local de $\mathbb{E}[(Y - g(\mathbf{x}))^2 | X = \mathbf{x}]$ ocorre quando $g^*(\mathbf{x}) = \mathbb{E}[Y | X = \mathbf{x}]$.

Assim, a função de regressão $r(\mathbf{x})$ é, sob a métrica de erro quadrático, a melhor escolha para aproximar Y em função de \mathbf{x} .

Exercício 3. Seja

$$\varphi(t) = \mathbb{E}[|Z - t|].$$

Encontre o t que minimiza a expressão acima.

Agora, suponha que o modelo verdadeiro seja dado por $Y = g^*(X) + \varepsilon$, onde ε é um ruído com $\mathbb{E}[\varepsilon] = 0$, independente das covariáveis X . Nossa objetivo é estudar o erro quadrático esperado de um modelo preditivo $\hat{g}(x)$ que tenta estimar y a partir de x .

O erro de predição é medido pela perda quadrática $(y - \hat{g}(x))^2$, e buscamos decompor o valor esperado desse erro em três componentes: variância, viés ao quadrado e ruído irredutível, considerando x fixado.

A função ótima g^* é definida como:

$$g^*(x) = \arg \min_g \mathbb{E}[(y - g(x))^2].$$

Para a decomposição, começamos adicionando e subtraindo $g^*(x)$ no termo de erro esperado:

$$\mathbb{E}[(y - \hat{g}(x))^2] = \mathbb{E}[(y - g^*(x) + g^*(x) - \hat{g}(x))^2].$$

Expandindo o quadrado, obtemos:

$$\mathbb{E}[(y - g^*(x))^2] + \mathbb{E}[(g^*(x) - \hat{g}(x))^2] + 2\mathbb{E}[(y - g^*(x))(g^*(x) - \hat{g}(x))].$$

O primeiro termo, $\mathbb{E}[(y - g^*(x))^2]$, representa a variância do ruído ε e, portanto, é irredutível. O segundo termo, $\mathbb{E}[(g^*(x) - \hat{g}(x))^2]$, captura o erro introduzido pelo modelo $\hat{g}(x)$. O terceiro termo é nulo, pois $y - g^*(x) = \varepsilon$ tem média zero e é independente de $\hat{g}(x)$.

Portanto, temos a seguinte decomposição:

$$\mathbb{E}[(y - \hat{g}(x))^2] = \underbrace{\mathbb{E}[(y - g^*(x))^2]}_{\text{ruído irredutível}} + \mathbb{E}[(g^*(x) - \hat{g}(x))^2].$$

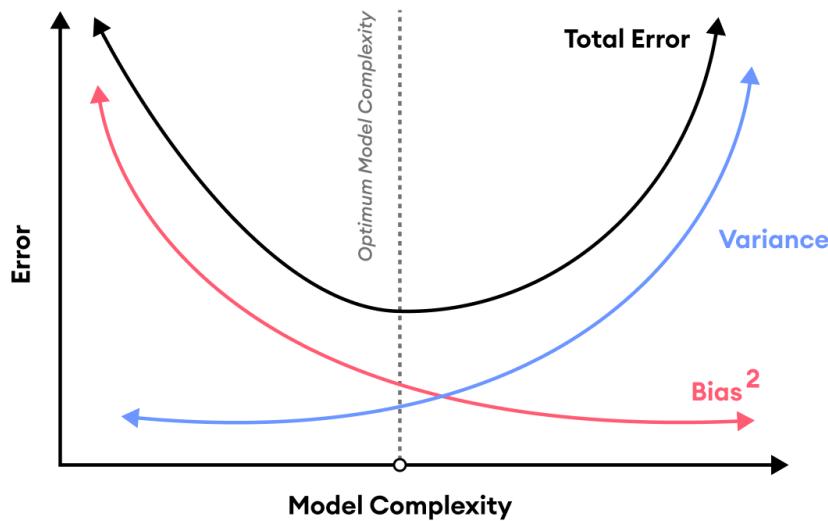
Agora, o termo $\mathbb{E}[(g^*(x) - \hat{g}(x))^2]$ pode ser decomposto em viés e variância:

$$\begin{aligned}\mathbb{E}[(g^*(x) - \hat{g}(x))^2] &= \mathbb{E}[(g^*(x) - \mathbb{E}[\hat{g}(x)]) + \mathbb{E}[\hat{g}(x)] - \hat{g}(x))^2] \\ &= (g^*(x) - \mathbb{E}[\hat{g}(x)])^2 + \text{Var}(\hat{g}(x)),\end{aligned}$$

onde o termo cruzado é novamente nulo por independência.

Finalmente, temos a seguinte decomposição clássica:

$$\mathbb{E}[(y - \hat{g}(x))^2] = \underbrace{\mathbb{E}[(y - g^*(x))^2]}_{\text{ruído irredutível}} + \underbrace{(g^*(x) - \mathbb{E}[\hat{g}(x)])^2}_{\text{viés}^2} + \underbrace{\text{Var}(\hat{g}(x))}_{\text{variância}}.$$



Essa decomposição nos permite entender um dos conceitos fundamentais em aprendizado de máquina e estatística: o **trade-off viés-variância**. Em termos simples, existe uma relação inversa entre o viés e a variância de um modelo. Modelos mais simples tendem a ter baixa variância, pois suas previsões mudam pouco ao variar a amostra de treinamento, mas frequentemente apresentam alto viés por não capturarem toda a complexidade da função $g^*(x)$. Por outro lado, modelos mais complexos conseguem ajustar melhor os dados e reduzir o viés, mas tendem a ter alta variância, pois são mais sensíveis a flutuações nas amostras de treino. O desafio central em modelagem preditiva é encontrar um equilíbrio entre essas duas quantidades, de modo a minimizar o erro total. Esse equilíbrio é essencial para garantir que o modelo generalize bem para novos dados, sem ser excessivamente simples (subajuste) ou excessivamente complexo (sobreajuste).

Resumindo: no caso da perda quadrática, o erro total pode ser decomposto como:

$$\text{Erro Total} = \text{Erro Irredutível} + \text{Viés}^2 + \text{Variância}.$$

Exercício 4. Acesse o notebook [aqui](#) e faça experimentos com o código mudando o tamanho da amostra, *random state*, grau dos polinômios, etc. Além disso, aplique o método de validação cruzada *k-fold*.

2.3 Data Splitting e Validação Cruzada

Em situações onde há abundância de dados, uma abordagem comum é dividir aleatoriamente o conjunto em três partes: um conjunto de treinamento, um conjunto de validação e um conjunto de teste. O conjunto de treinamento é utilizado para ajustar os modelos; o conjunto de validação serve para estimar o erro de predição e realizar a seleção do modelo; e o conjunto de teste é reservado para avaliar o erro de generalização final do modelo escolhido.

IMPORTANTE! Idealmente, o conjunto de teste deve ser mantido isolado — como se estivesse em um "cofre"— e só ser acessado ao final da análise de dados. Caso utilizemos o conjunto de teste de forma repetida durante a seleção do modelo, escolhendo aquele com menor erro no teste, acabaremos subestimando o verdadeiro erro de generalização, às vezes de maneira significativa.

Não há uma regra única para definir a quantidade de observações em cada uma das três divisões, pois isso depende da razão sinal-ruído dos dados e do tamanho da amostra disponível. Uma divisão típica é utilizar cerca de 50% dos dados para treinamento e 25% para validação e teste, respectivamente.



Figura 2.2: Divisão em treino, validação e teste. Tirado de ([Hastie et al., 2001](#)).

No Python, podemos realizar essa divisão utilizando a biblioteca `scikit-learn`. O procedimento padrão é, primeiramente, dividir os dados em duas partes: treino e teste. Em seguida, subdividir a parte de treino em treino e validação.

Listing 2.1: Divisão em treino, validação e teste

```

1 from sklearn.model_selection import train_test_split
2
3 # Suponha que temos dados X e respostas Y
4 # X: matriz de covariaveis (n_samples x n_features)
5 # Y: vetor de respostas (n_samples,)
6
7
8 # Primeira divisao: treino e restante (validacao + teste)
9 X_train, X_rest, Y_train, Y_rest = train_test_split(
10     X, Y, test_size=0.5, random_state=42)
11

```

```

12 # Segunda divisao: validacao e teste a partir do restante
13 X_val, X_test, Y_val, Y_test = train_test_split(
14     X_rest, Y_rest, test_size=0.5, random_state=42)
15
16 print("Treino:", X_train.shape)
17 print("Validacao:", X_val.shape)
18 print("Teste:", X_test.shape)

```

No exemplo acima, separamos 25% dos dados para o conjunto de teste e, dos 75% restantes, cerca de 33% foi alocado para validação. Assim, o resultado final aproximado seria: 50% dos dados para treino, 25% para validação e 25% para teste, como discutido anteriormente.

ATENÇÃO: É importante definir o argumento `random_state` para garantir reproduzibilidade da divisão dos dados.

Embora a divisão treino/validação/teste seja bastante comum, em situações onde a quantidade de dados é limitada, desperdiçar uma parte significativa da amostra apenas para validação pode ser custoso. Nesse contexto, uma estratégia amplamente utilizada é a **validação cruzada**, em especial o *k-fold cross-validation*.

A ideia do *k-fold* é dividir o conjunto de dados em k subconjuntos (ou *folds*) de tamanhos aproximadamente iguais. Em cada uma das k iterações, utilizamos $k - 1$ desses subconjuntos para treinar o modelo e o subconjunto restante para validá-lo. No final, o erro de validação é calculado como a média dos erros obtidos em cada uma das iterações.

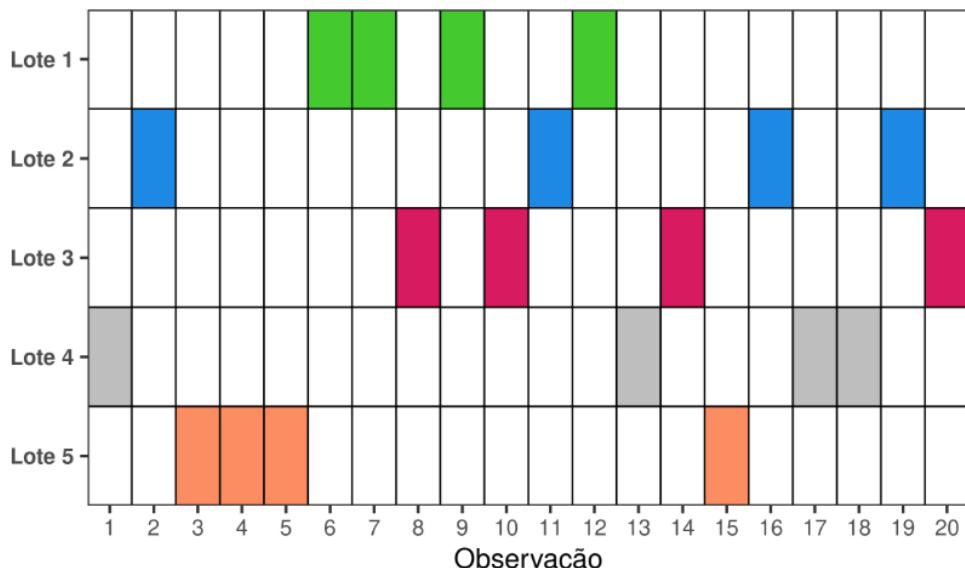


Figura 2.3: Esquema com amostra de tamanho 20 e 5 folds. Retirado de [Izbicki and dos Santos \(2020\)](#).

Essa técnica tem como vantagem utilizar o máximo de dados possível para treinamento em

cada repetição, reduzindo a variância da estimativa do erro de generalização. Além disso, o *k-fold* ajuda a mitigar a dependência da divisão aleatória dos dados, já que cada observação é utilizada tanto para treino quanto para validação ao longo do processo.

No Python, a implementação do *k-fold cross-validation* pode ser feita utilizando a biblioteca `scikit-learn` da seguinte maneira:

Listing 2.2: Validação cruzada k-fold

```

1 from sklearn.model_selection import KFold
2
3 # Suponha que temos dados X e respostas Y
4 k = 5
5 kf = KFold(n_splits=k, shuffle=True, random_state=42)
6
7 for fold, (train_index, val_index) in enumerate(kf.split(X)):
8     X_train, X_val = X[train_index], X[val_index]
9     Y_train, Y_val = Y[train_index], Y[val_index]
10    print(f"Fold {fold+1}:")
11    print("Treino:", X_train.shape, "Validacao:", X_val.shape)

```

No exemplo acima, o conjunto de dados é dividido em $k = 5$ subconjuntos. O argumento `shuffle=True` garante que as observações sejam embaralhadas antes da divisão em folds, e o `random_state` garante a reprodutibilidade dos resultados.

Exercício 5. Acesse o notebook [aqui](#) e faça experimentos com o código mudando o tamanho da amostra, `random state`, etc.

Capítulo 3

Introdução à regressão via mínimos quadrados

3.1 Mínimos quadrados

O método dos mínimos quadrados tem suas origens no início do século XIX e está intimamente ligado à história da astronomia e da estatística. Ele foi introduzido formalmente por Carl Friedrich Gauss, que o utilizava desde 1795 em seus trabalhos com órbitas planetárias, embora o primeiro a publicar sobre o método tenha sido Adrien-Marie Legendre, em 1805.

Legendre apresentou o método em seu trabalho sobre o cálculo de órbitas cometárias, propondo uma técnica para ajustar curvas a dados experimentais minimizando a soma dos quadrados dos erros. Poucos anos depois, em 1809, Gauss publicou seu famoso livro *Theoria Motus Corporum Coelestium*, onde apresentou uma justificativa probabilística do método com base na distribuição normal dos erros.

Desde então, os mínimos quadrados tornaram-se uma das ferramentas fundamentais em estatística, ciência de dados e análise numérica, sendo aplicados em regressão linear, ajuste de modelos, filtragem de sinais e muitos outros contextos científicos e tecnológicos.

Motivados por esses contextos, consideremos agora o seguinte problema: queremos encontrar um vetor β tal que

$$y = X\beta.$$

Essa expressão possui solução se, e somente se, $y \in \text{Ran } X$. Mas o que podemos fazer quando queremos resolver uma equação que não tem solução? Por exemplo, se $y = X\beta + \varepsilon$ onde ε é um erro aleatório de medição?

À primeira vista, essa pode parecer uma pergunta boba, pois se não há solução, então não há solução. No entanto, situações em que desejamos resolver uma equação sem solução podem surgir naturalmente — por exemplo, quando a equação vem de dados experimentais. Se não houver nenhum erro, o vetor do lado direito y pertence à imagem de X , e a equação é consistente.

Porém, na prática, é impossível eliminar completamente os erros de medição. Assim, pode acontecer de uma equação que teoricamente deveria ser consistente não possuir solução.

O que fazer, então, nessa situação?

A ideia mais simples é escrever o erro na forma

$$\|X\beta - y\|$$

e tentar encontrar o vetor β que minimiza essa quantidade. Se conseguirmos encontrar um β tal que o erro seja zero, então o sistema é consistente e temos uma solução exata. Caso contrário, obtemos a chamada *solução de mínimos quadrados*.

O nome *mínimos quadrados* vem do fato de que minimizar $\|X\beta - y\|$ é equivalente a minimizar

$$\|X\beta - y\|^2 = \sum_{k=1}^m |(X\beta)_k - y_k|^2 = \sum_{k=1}^m \left| \sum_{j=1}^n X_{k,j}\beta_j - y_k \right|^2,$$

isto é, estamos minimizando a soma dos quadrados de funções lineares.

Existem diversas maneiras de encontrar a solução de mínimos quadrados. Se estivermos em \mathbb{R}^n , e tudo for real, podemos ignorar os valores absolutos. Nesse caso, basta calcular as derivadas parciais em relação a cada β_j e encontrar onde todas elas se anulam — o que nos dará o ponto de mínimo.

Exercício 6. Encontre a solução do problema de mínimos quadrados derivando e igualando à zero.

Existe uma forma mais simples de encontrar o mínimo. De fato, ao considerarmos todos os vetores β , o vetor $X\beta$ percorre todo o espaço imagem de X , ou seja, $\text{Ran } X$. Portanto, minimizar $\|X\beta - y\|$ equivale a calcular a menor distância de y até $\text{Ran } X$.

Assim, $\|X\beta - y\|^2$ é mínima se, e somente se,

$$X\beta = P_{\text{Ran } X}y,$$

onde $P_{\text{Ran } X}$ denota a projeção ortogonal de y sobre o subespaço imagem de X .

Se conhecemos uma base ortogonal $\mathbf{v}_1, \dots, \mathbf{v}_n$ de $\text{Ran } X$, podemos calcular $P_{\text{Ran } X}y$ pela fórmula:

$$P_{\text{Ran } X}y = \sum_{k=1}^n \frac{\langle y, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2} \mathbf{v}_k.$$

Caso só tenhamos uma base qualquer de $\text{Ran } X$, é necessário utilizar o processo de Gram-Schmidt para ortogonalizá-la antes de aplicar a fórmula.

Existe, no entanto, uma alternativa mais direta. A condição de que $X\beta = P_{\text{Ran } X}y$ é equivalente a exigir que o vetor $y - X\beta$ seja ortogonal a $\text{Ran } X$, ou seja,

$$y - X\beta \perp \text{coluna de } X.$$

Seja $X = [a_1, a_2, \dots, a_n]$, onde cada a_k é uma coluna. A condição acima é equivalente a:

$$\langle y - X\beta, a_k \rangle = 0, \quad \text{para } k = 1, 2, \dots, n.$$

Ou, de forma matricial:

$$X^*(y - X\beta) = 0,$$

o que é equivalente à chamada *equação normal*:

$$X^*X\beta = X^*y.$$

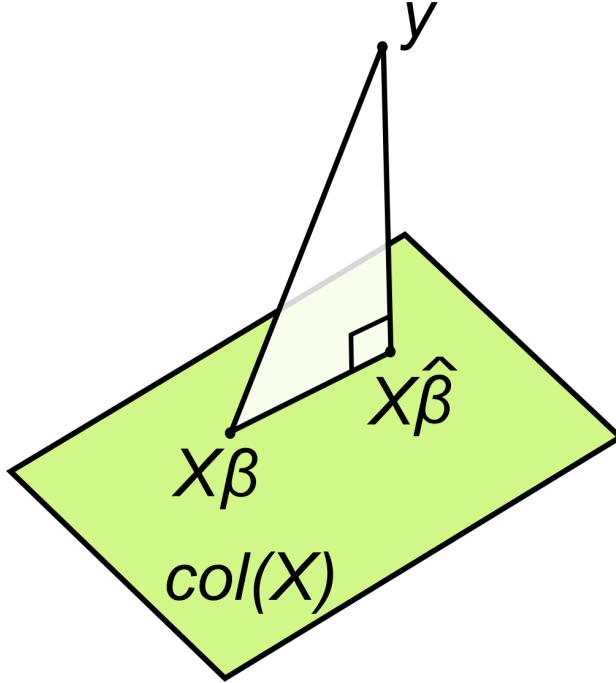


Figura 3.1: Projeção ortogonal de y no espaço coluna de X .

A solução dessa equação nos fornece a solução de mínimos quadrados da equação $X\beta = y$. Note que a solução é única se, e somente se, X^*X for inversível.

Agora, se β é uma solução da *equação normal* $X^*X\beta = X^*y$ (ou seja, uma solução de mínimos quadrados da equação $X\beta = y$), então $X\beta = P_{\text{Ran } X}y$. Assim, para encontrar a projeção ortogonal de y sobre o espaço coluna $\text{Ran } X$, basta resolver a equação normal $X^*X\beta = X^*y$ e multiplicar a solução por X .

Se o operador X^*X for inversível, então a solução da equação normal é dada por:

$$\beta = (X^*X)^{-1}X^*y,$$

e, portanto, a projeção ortogonal $P_{\text{Ran } X}y$ pode ser escrita como:

$$P_{\text{Ran } X}y = X(X^*X)^{-1}X^*y.$$

Como isso vale para todo y , obtemos a expressão matricial da projeção ortogonal sobre o espaço coluna de X :

$$P_{\text{Ran } X} = X(X^*X)^{-1}X^*.$$

Observação 1. Note que a expressão

$$P_{\text{Ran } X} = X(X^*X)^{-1}X^*.$$

é uma generalização matricial para a projeção ortogonal em vetores da forma

$$\frac{vv^T}{v^Tv} = v(v^Tv)^{-1}v^T,$$

já que para um vetor x qualquer,

$$\frac{\langle v, x \rangle}{\|v\|^2}v = \frac{v^Tx}{\|v\|^2}v = \frac{vv^Tx}{\|v\|^2} = \frac{vv^T}{\|v\|^2}x.$$

Exemplo 1. Suponha que sabemos que a relação entre x e y é dada por uma parábola da forma

$$y = a + bx + cx^2,$$

e queremos ajustar essa parábola aos dados observados. Os coeficientes desconhecidos a, b, c devem satisfazer o sistema:

$$a + bx_k + cx_k^2 = y_k, \quad k = 1, 2, \dots, n.$$

Em forma matricial, esse sistema pode ser escrito como:

$$\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Por exemplo, para os dados do exemplo anterior, devemos resolver a equação de mínimos quadrados:

$$\begin{pmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Então calculamos:

$$X^T X = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 3 \\ 4 & 1 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 3 & 9 \end{pmatrix} = \begin{pmatrix} 5 & 2 & 18 \\ 2 & 18 & 26 \\ 18 & 26 & 114 \end{pmatrix}.$$

E também:

$$X^T y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 3 \\ 4 & 1 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 4 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ -5 \\ 31 \end{pmatrix}.$$

Portanto, a equação normal $X^T X \beta = X^T y$ é:

$$\begin{pmatrix} 5 & 2 & 18 \\ 2 & 18 & 26 \\ 18 & 26 & 114 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 9 \\ -5 \\ 31 \end{pmatrix},$$

cuja solução única é:

$$a = \frac{86}{77}, \quad b = -\frac{62}{77}, \quad c = \frac{43}{154}.$$

Portanto, a parábola que melhor se ajusta aos dados é:

$$y = \frac{86}{77} - \frac{62}{77}x + \frac{43}{154}x^2.$$

3.2 Resolução Numérica

A forma fechada da solução dos mínimos quadrados, dada por

$$(X^T X) \hat{\beta} = X^T y \Rightarrow \hat{\beta} = (X^T X)^{-1} X^T y,$$

embora útil em termos analíticos, pode ser instável numericamente e custosa para grandes dimensões. Assim, é comum empregar abordagens numéricas mais robustas.

Fatoração QR. Um método bastante utilizado é a decomposição QR, na qual a matriz $X \in \mathbb{R}^{n \times p}$ é escrita como $X = QR$, onde Q possui colunas ortonormais (ou seja, $Q^T Q = I$) e $R \in \mathbb{R}^{p \times p}$ é triangular superior. Essa fatoração evita a inversão direta da matriz $X^T X$ e proporciona maior estabilidade numérica.

Como $X^T X = R^T Q^T QR = R^T R$, temos:

$$(X^T X) \hat{\beta} = X^T y \iff R^T R \hat{\beta} = R^T Q^T y \iff R \hat{\beta} = Q^T y.$$

Portanto, basta resolver um sistema linear triangular com matriz R , o que é computacionalmente eficiente. O custo total permanece na ordem de $\mathcal{O}(d^3)$. Em situações onde d é grande, métodos iterativos como o gradiente conjugado também podem ser considerados (ver Golub e Loan, 1996).

3.3 Estimativas de erros para regressão linear

Como vimos no primeiro capítulo, um conceito central na análise de métodos de aprendizado de máquina é compreender o comportamento de viés e variância do modelo.

Vamos assumir que os dados seguem o modelo

$$y = \langle x, \beta \rangle + \varepsilon,$$

com $\mathbb{E}[\varepsilon] = 0$ e $\text{Var}(\varepsilon) = \sigma^2$ e os pontos x estão fixos, ou seja, a aleatoriedade é apenas sobre o erro ε . Seja $\hat{\beta} = (X^T X)^{-1} X^T y$ a solução obtida por mínimos quadrados. Então:

$$\mathbb{E}[\hat{\beta}] = (X^\top X)^{-1} X^\top \mathbb{E}[y] = (X^\top X)^{-1} X^\top X \beta = \beta,$$

isto é, o estimador é não-viesado.

Agora vamos estimar o risco dado que estamos utilizando o estimador $\hat{\beta}$, ou seja:

$$R(\hat{\beta}) = \frac{1}{n} \mathbb{E} [\|y - X\hat{\beta}\|^2],$$

Precisamos ter atenção redobrada na aleatoriedade da expressão acima, isso porque o erro que existe no nosso dataset no momento da estimação de $\hat{\beta}$ é potencialmente diferente do erro utilizado no cálculo do risco. Podemos representar isso da seguinte forma

$$R(\hat{\beta}) = \frac{1}{n} \mathbb{E} [\|y' - X(X^\top X)^{-1} X^\top y\|^2].$$

Novamente, temos um y' que será utilizado no cálculo do risco, e um y que é utilizado no cálculo de $\hat{\beta}$. Suponha então que $y' = X\beta + \varepsilon'$ e que $y = X\beta + \varepsilon$. Logo, a expressão acima fica:

$$\begin{aligned} R(\hat{\beta}) &= \frac{1}{n} \mathbb{E} [\|y' - X(X^\top X)^{-1} X^\top y\|^2] \\ &= \frac{1}{n} \mathbb{E} [\|X\beta + \varepsilon' - X(X^\top X)^{-1} X^\top (X\beta + \varepsilon)\|^2] \\ &= \frac{1}{n} \mathbb{E} [\|\varepsilon' - X(X^\top X)^{-1} X^\top \varepsilon\|^2] \\ &= \frac{1}{n} \mathbb{E} [\|\varepsilon'\|^2] - \frac{2}{n} \mathbb{E} [\langle \varepsilon', X(X^\top X)^{-1} X^\top \varepsilon \rangle] + \frac{1}{n} \mathbb{E} [\varepsilon^T X(X^\top X)^{-1} X^\top X(X^\top X)^{-1} X^\top \varepsilon] \\ &= \sigma^2 - \frac{2}{n} \mathbb{E} [\varepsilon'^T X(X^\top X)^{-1} X^\top \varepsilon] + \frac{1}{n} \mathbb{E} [\varepsilon^T X(X^\top X)^{-1} X^\top \varepsilon] \end{aligned}$$

Agora note que temos dois casos possíveis:

- Se $\varepsilon' = \varepsilon$, isto é, $y = y'$, temos que a expressão acima fica:

$$\begin{aligned} R(\hat{\beta}) &= \sigma^2 - \frac{1}{n} \mathbb{E} [\varepsilon^T X(X^\top X)^{-1} X^\top \varepsilon] \\ &= \sigma^2 - \frac{1}{n} \mathbb{E} [\text{tr}(\varepsilon^T X(X^\top X)^{-1} X^\top \varepsilon)] \\ &= \sigma^2 - \frac{1}{n} \mathbb{E} [\text{tr}(\varepsilon \varepsilon^T X(X^\top X)^{-1} X^\top)] \\ &= \sigma^2 - \frac{\sigma^2 p}{n} \end{aligned}$$

e como estamos usando o mesmo y para estimar e avaliar o modelos, essa quantidade se refere ao **erro de treinamento!**

- Se ε' e ε são independentes, isto é, y e y' são independentes, então

$$\mathbb{E} [\varepsilon'^T X(X^\top X)^{-1} X^\top \varepsilon] = \mathbb{E} [\varepsilon'^T] \mathbb{E} [X(X^\top X)^{-1} X^\top \varepsilon] = 0$$

e portanto a expressão fica:

$$\begin{aligned} R(\hat{\beta}) &= \sigma^2 \frac{1}{n} \mathbb{E} [\varepsilon^T X(X^\top X)^{-1} X^\top \varepsilon] \\ &= \sigma^2 + \frac{\sigma^2 p}{n}. \end{aligned}$$

Nesse caso, como o y utilizado para treinamento e o y' utilizado para estimação do risco são diferentes, a quantidade acima se refere ao **erro de teste!**

- O erro de treinamento esperado é $(1 - \frac{p}{n})\sigma^2$, enquanto o erro de teste esperado é $(1 + \frac{p}{n})\sigma^2$. Isso mostra que o erro de treinamento subestima o erro de teste em $\frac{2\sigma^2 p}{n}$, caracterizando o *overfitting*. Essa diferença pode ser usada para seleção de modelos.
- Para que o risco excessivo seja pequeno comparado a σ^2 , é necessário que $\frac{p}{n}$ seja pequeno. Isso dificulta a aplicação direta de mínimos quadrados em situações de alta dimensionalidade, onde $p \approx n$ ou $p \gg n$. Nesses casos, técnicas de regularização, como regressão ridge ou penalizações com norma- ℓ_1 , são necessárias.

3.4 Um pouco de inferência

Perceba que a teoria acima não depende de informações sobre os dados — a única coisa que fizemos foi encontrar a melhor aproximação de y no espaço gerado pelas colunas de X . Ou seja, tratamos X e y como vetores fixos, e a projeção ortogonal $P_{\text{Ran } X}y$ é puramente uma construção geométrica.

Suponha agora que temos um modelo probabilístico associado aos dados:

$$Y = \langle x, \beta \rangle + \varepsilon, \quad \varepsilon \sim N(0, 1),$$

onde $x \in \mathbb{R}^p$ é um vetor fixo (não aleatório), $\beta \in \mathbb{R}^p$ é o vetor de parâmetros desconhecido, e ε é um erro aleatório com distribuição normal padrão.

Se coletamos n observações (x_i, Y_i) , $i = 1, \dots, n$, podemos escrever o modelo vetorialmente como:

$$y = X\beta + \varepsilon,$$

onde:

- $y \in \mathbb{R}^n$ é o vetor de respostas;
- $X \in \mathbb{R}^{n \times p}$ é a matriz cujas linhas são os vetores x_i^T ;
- $\varepsilon \sim N(0, I_n)$ é o vetor de erros independentes com variância 1.

Neste caso, a estimativa de mínimos quadrados:

$$\hat{\beta} = (X^* X)^{-1} X^* y$$

é também o *estimador de máxima verossimilhança* de β sob o modelo gaussiano. Além disso, por ser combinação linear de variáveis gaussianas, $\hat{\beta}$ é também uma variável aleatória com distribuição normal.

Teorema 1. Se $y = X\beta + \varepsilon$, com $\varepsilon \sim N(0, I_n)$, então:

$$\hat{\beta} = (X^* X)^{-1} X^* y \sim N(\beta, (X^* X)^{-1}).$$

Demonstração. Note que:

$$\hat{\beta} = (X^* X)^{-1} X^* y = (X^* X)^{-1} X^* (X\beta + \varepsilon) = \beta + (X^* X)^{-1} X^* \varepsilon.$$

Como $\varepsilon \sim N(0, I_n)$, e $(X^* X)^{-1} X^* \varepsilon$ é uma combinação linear de gaussianas, então:

$$\hat{\beta} \sim N\left(\beta, (X^* X)^{-1} X^* X (X^* X)^{-1}\right) = N\left(\beta, (X^* X)^{-1}\right).$$

□

Essa propriedade permite que façamos *inferência estatística* sobre os coeficientes β . Por exemplo, para testar a hipótese nula:

$$H_0 : \beta_j = 0,$$

podemos usar o fato de que:

$$\hat{\beta}_j \sim N(\beta_j, \sigma_j^2), \quad \text{com } \sigma_j^2 = [(X^* X)^{-1}]_{jj}.$$

Ou seja, o valor padronizado

$$Z_j = \frac{\hat{\beta}_j}{\sqrt{\sigma_j^2}} \sim N(0, 1) \quad \text{sob } H_0.$$

Isso nos permite construir intervalos de confiança e calcular valores- p .

Exemplo 2. Suponha que queremos um intervalo de confiança para β_j com nível de confiança $1 - \alpha$. Como $\hat{\beta}_j \sim N(\beta_j, \sigma_j^2)$, temos:

$$\mathbb{P}\left(\hat{\beta}_j - z_{\alpha/2} \sqrt{\sigma_j^2} \leq \beta_j \leq \hat{\beta}_j + z_{\alpha/2} \sqrt{\sigma_j^2}\right) = 1 - \alpha,$$

onde $z_{\alpha/2}$ é o quantil superior de ordem $1 - \alpha/2$ da normal padrão.

Para mais detalhes sobre inferência de parâmetros, ver ([James et al., 2013](#), Capítulo 3).

Bibliografia

[Bach \(2024\)](#)

Capítulo 4

Introdução à classificação via regressão logística

4.1 Classificador de Bayes

Sejam $(X_1, Y_1), \dots, (X_n, Y_n)$ amostras independentes e identicamente distribuídas, onde $X_i \in \mathbb{R}^p$ representa um vetor de características e $Y_i \in \{0, 1\}$ é o rótulo associado. O objetivo é construir um classificador $\hat{g} : \mathbb{R}^p \rightarrow \{0, 1\}$ que prediga Y a partir de X .

Uma forma comum de avaliar o desempenho de um classificador g é por meio da **função de perda 0–1**, definida por

$$L(g(x), y) = \begin{cases} 0, & \text{se } g(x) = y, \\ 1, & \text{caso contrário.} \end{cases}$$

O **risco verdadeiro** associado a g é dado por

$$R(g) = \mathbb{E}[L(g(X), Y)],$$

onde a esperança é tomada em relação à distribuição conjunta de (X, Y) .

Como vimos anteriormente, no contexto de regressão com perda quadrática, a função ótima é dada por

$$g^*(x) = \mathbb{E}[Y | X = x],$$

isto é, a média condicional de y dado X .

No caso de classificação binária com perda 0–1, a função ótima assume uma forma diferente. Vamos agora investigar qual é a forma da função g^* que minimiza o risco verdadeiro.

Teorema 2 (Classificador de Bayes e risco 0–1). *Seja $\eta(x) = \mathbb{P}(Y = 1 | X = x)$. O classificador de Bayes g^* é definido por*

$$g^*(x) = \begin{cases} 1, & \text{se } \eta(x) \geq 0,5, \\ 0, & \text{caso contrário.} \end{cases}$$

Denotamos por R^ o menor risco verdadeiro possível, isto é,*

$$R^* = \inf_g \mathbb{E}[L(g(X), Y)].$$

Então, valem os seguintes resultados:

(a) $R(h^*) = R^*$, ou seja, h^* minimiza o risco verdadeiro e é um classificador de Bayes.

(b) Para qualquer classificador h , o excesso de risco satisfaz

$$R(h) - R^* = 2 \mathbb{E}_X [|\eta(X) - 0,5| \cdot \mathbb{I}\{h(X) \neq h^*(X)\}].$$

Demonstração. **Item (a).** Note que

$$\begin{aligned} \mathbb{E}[L(g(x), Y)] &= \mathbb{E}[\mathbb{I}\{g(x) \neq Y\}] \\ &= \mathbb{E}_X [\mathbb{E}[\mathbb{I}\{g(x) \neq Y\} | X = x]]. \end{aligned}$$

Vamos analisar a expressão dentro da esperança condicional em X . Temos:

$$\begin{aligned} \mathbb{E}[\mathbb{I}\{g(x) \neq Y\} | X = x] &= \mathbb{P}(g(x) \neq Y | X = x) \\ &= \mathbb{P}(0 \neq Y | X = x) \mathbb{I}\{g(x) = 0\} + \mathbb{P}(1 \neq Y | X = x) \mathbb{I}\{g(x) = 1\} \\ &= \mathbb{P}(Y = 1 | X = x) \mathbb{I}\{g(x) = 0\} + \mathbb{P}(Y = 0 | X = x) \mathbb{I}\{g(x) = 1\} \\ &= \eta(x) \mathbb{I}\{g(x) = 0\} + (1 - \eta(x)) \mathbb{I}\{g(x) = 1\}. \end{aligned}$$

Nosso objetivo é encontrar a função g que **minimiza** essa expressão. Para isso, basta decidir, para cada x , se devemos escolher $g(x) = 0$ ou $g(x) = 1$. Analisamos caso a caso:

- Se $\eta(x) \geq 0,5$, então $1 - \eta(x) \leq \eta(x)$. Nesse caso, a expressão é minimizada escolhendo $g(x) = 1$, pois isso anula o termo com $\eta(x)$, restando apenas o menor entre os dois.
- Se $\eta(x) < 0,5$, então $\eta(x) < 1 - \eta(x)$, e a expressão é minimizada com $g(x) = 0$.

Portanto, a função g^* definida no teorema de fato minimiza o risco verdadeiro, o que conclui a demonstração do item (a).

Item (b). Queremos mostrar que, para qualquer classificador g ,

$$R(h) - R^* = 2 \mathbb{E}_X [|\eta(X) - 0,5| \cdot \mathbb{I}\{h(X) \neq h^*(X)\}].$$

Começamos com a definição do excesso de risco:

$$\begin{aligned} R(h) - R(g^*) &= \mathbb{E}_X [\eta(X) \mathbb{I}\{g(X) = 0\} + (1 - \eta(X)) \mathbb{I}\{g(X) = 1\}] \\ &\quad - \mathbb{E}_X [\eta(X) \mathbb{I}\{g^*(X) = 0\} + (1 - \eta(X)) \mathbb{I}\{g^*(X) = 1\}]. \end{aligned}$$

Note que a expressão dentro da esperança é nula quando $g(x) = g^*(x)$, logo só precisamos analisar o que acontece quando $g(x) \neq g^*(x)$.

- Se $g^*(x) = 1$, só precisamos analisar o caso de $g(x) = 0$ e daí temos que

$$\begin{aligned} R(h) - R(g^*) &= \mathbb{E}_X [\eta(X) \mathbb{I}\{g(X) = 0\} + (1 - \eta(X)) \mathbb{I}\{g(X) = 1\}] \\ &\quad - \mathbb{E}_X [\eta(X) \mathbb{I}\{g^*(X) = 0\} + (1 - \eta(X)) \mathbb{I}\{g^*(X) = 1\}] \\ &= -(1 - \eta(X)) + \eta(X) \\ &= 2\eta(x) - 1. \end{aligned}$$

Note que como assumimos que $g^*(x) = 1$, então $\eta(x) \geq 0,5$ e portanto $2\eta(x) - 1 \geq 0$.

- Se $g^*(x) = 0$, concluímos de forma análoga que

$$R(h) - R(g^*) = 1 - 2\eta(x).$$

Nesse caso, como $g^*(x) = 0$, temos que $\eta(x) < .5$ e dessa forma $1 - 2\eta(x) \geq 0$. De fato, podemos expressar a solução de forma geral na seguinte forma:

$$R(h) - R^* = 2 \mathbb{E}_X [|\eta(X) - 0,5| \cdot \mathbb{I}\{h(X) \neq h^*(X)\}].$$

Isso prova o item (b). □

A expressão do item (b) nos dá uma intuição importante: o excesso de risco é mais sensível a erros de classificação quando $\eta(X)$ está longe de 0,5, ou seja, quando a incerteza sobre a classe é baixa. Nesses pontos, errar significa contrariar uma alta confiança, o que resulta em um aumento maior no risco. Por outro lado, se $\eta(X)$ está próximo de 0,5, mesmo que $g(X) \neq g^*(X)$, o impacto no risco é pequeno. Assim, o termo $|\eta(X) - 0,5|$ atua como um peso que penaliza mais fortemente os erros onde a decisão ótima é mais evidente.

4.1.1 O Modelo Logístico

Uma vez entendido o papel da função $\eta(x) = \mathbb{P}(Y = 1 | X = x)$ como componente central da regra de Bayes, torna-se natural buscar maneiras de aproximar essa função a partir dos dados via um estimador g . Um dos modelos mais clássicos para esse fim é a **regressão logística**.

A ideia é modelar diretamente a probabilidade condicional $\mathbb{P}(Y = 1 | X = x)$ como uma função dos preditores. Um primeiro impulso seria utilizar um modelo linear do tipo $g(x) = \langle \beta, x \rangle$, mas essa abordagem apresenta um problema fundamental: a função linear pode assumir valores fora do intervalo $[0, 1]$, o que é inaceitável para probabilidades.

Para contornar essa limitação, adotamos uma função que mapeia qualquer valor real para o intervalo $(0, 1)$. Na regressão logística, usamos a função logística:

$$g(x) = \frac{e^{\langle \beta, x \rangle}}{1 + e^{\langle \beta, x \rangle}}.$$

É fácil ver que

$$1 - g(x) = \frac{1}{1 + e^{\langle \beta, x \rangle}}.$$

Essa formulação garante que $g(x) \in (0, 1)$ para todo $x \in \mathbb{R}^p$. Podemos reescrever essa expressão como

$$\frac{g(x)}{1 - g(x)} = e^{\langle \beta, x \rangle}.$$

A razão $g(x)/[1 - g(x)]$ é chamada de *odds* e pode assumir qualquer valor entre 0 e ∞ . Valores de odds próximos de 0 indicam probabilidades muito baixas, enquanto valores muito altos indicam alta probabilidade da classe positiva. Por exemplo, $g(x) = 0,2$ implica uma odds

de $\frac{0,2}{1-0,2} = 1/4$, o que corresponde a 1 em cada 5 indivíduos da classe positiva. Já $g(x) = 0,9$ implica uma odds de $\frac{0,9}{1-0,9} = 9$, ou seja, 9 em cada 10.

Odds são tradicionalmente usadas em vez de probabilidades em contextos como apostas, pois se relacionam mais diretamente com estratégias de decisão.

Tomando o logaritmo de ambos os lados da expressão anterior, obtemos:

$$\log \left(\frac{g(x)}{1-g(x)} \right) = \langle \beta, x \rangle.$$

A expressão à esquerda é chamada de *log odds* ou *logito*. Observamos, assim, que o modelo de regressão logística possui um logito linear em x , o que permite tanto uma interpretação estatística clara quanto facilidade de ajuste computacional.

4.1.2 Estimando os Coeficientes da Regressão

Os coeficientes β no modelo logístico são desconhecidos e devem ser estimados a partir dos dados de treinamento disponíveis. Embora uma abordagem possível seja o uso de mínimos quadrados não lineares, o método mais comum e preferido é o de *máxima verossimilhança*, por apresentar melhores propriedades estatísticas.

A intuição básica por trás do uso da máxima verossimilhança para ajustar um modelo de regressão logística é a seguinte: buscamos encontrar um vetor β tal que a probabilidade predita $\hat{g}(x_i)$ para cada observação se aproxime o máximo possível dos valores reais observados $Y_i \in \{0, 1\}$.

Em outras palavras, queremos que $\hat{g}(x_i) \approx 1$ para os indivíduos com $Y_i = 1$, e $\hat{g}(x_i) \approx 0$ para os indivíduos com $Y_i = 0$. Essa ideia pode ser formalizada por meio de uma função chamada *função de verossimilhança*, dada por:

$$\ell(\beta) = \prod_{i:Y_i=1} g(x_i) \prod_{i:Y_i=0} (1 - g(x_i)).$$

Mais especificamente, podemos reescrever a função de verossimilhança como:

$$\prod_{k=1}^n \left(\frac{e^{\beta_0 + \sum_{i=1}^p \beta_i x_{k,i}}}{1 + e^{\beta_0 + \sum_{i=1}^p \beta_i x_{k,i}}} \right)^{y_k} \left(\frac{1}{1 + e^{\beta_0 + \sum_{i=1}^p \beta_i x_{k,i}}} \right)^{1-y_k}.$$

Essa forma incorpora tanto os casos em que $Y_k = 1$ quanto $Y_k = 0$ em uma única expressão compacta.

Tomando o logaritmo da função de verossimilhança, obtemos a **log-verossimilhança**:

$$\log \ell(\beta) = \sum_{k=1}^n \left[y_k \log \left(\frac{e^{\beta_0 + \sum_{i=1}^d \beta_i x_{k,i}}}{1 + e^{\beta_0 + \sum_{i=1}^d \beta_i x_{k,i}}} \right) + (1 - y_k) \log \left(\frac{1}{1 + e^{\beta_0 + \sum_{i=1}^d \beta_i x_{k,i}}} \right) \right].$$

Simplificando os logaritmos, essa expressão pode ser reescrita como:

$$\log \ell(\beta) = \sum_{k=1}^n \left[y_k \langle \beta, x_k \rangle - \log \left(1 + e^{\langle \beta, x_k \rangle} \right) \right],$$

onde $x_k \in \mathbb{R}^d$ inclui o termo de intercepto (ou seja, supomos $x_k = (1, x_{k,1}, \dots, x_{k,d})$) e $\beta \in \mathbb{R}^{d+1}$.

4.2 Um pouco de otimização - O método de Newton

Antes de maximizar a log-verossimilhança, introduzimos o **Método de Newton**. O Método de Newton é um algoritmo iterativo usado para encontrar raízes de uma função. No caso univariado, sua implementação segue os seguintes passos:

1. Encontre a reta tangente à função $f(x)$ no ponto atual (x_n, y_n) , com:

$$y = f'(x_n)(x - x_n) + f(x_n).$$

2. Calcule a interseção da reta tangente com o eixo x , isto é, determine x_{n+1} tal que $f(x_{n+1}) = 0$. Para isso, impomos:

$$0 = f'(x_n)(x_{n+1} - x_n) + f(x_n),$$

o que resulta em:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

3. Avalie a função no novo ponto:

$$y_{n+1} = f(x_{n+1}).$$

4. Verifique o critério de parada: se $y_{n+1} - y_n \approx 0$, então:

`return y_{n+1} (convergência atingida).`

5. Caso contrário, atualize o ponto:

$$x_n \leftarrow x_{n+1}, \quad y_n \leftarrow y_{n+1},$$

e volte ao passo 1.

Para uma ilustração, ver o seguinte [gif](#).

Em resumo, o Método de Newton para uma variável consiste em atualizar iterativamente o ponto atual segundo a fórmula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

até que a diferença entre x_n e x_{n+1} seja suficientemente pequena, isto é, até que $|x_n - x_{n+1}| \approx 0$. Esse procedimento corresponde a encontrar o ponto onde a função f se anula, aproximando-se da raiz por meio das tangentes locais.

Se f é uma função fortemente convexa com hessiana Lipschitz contínua, então, desde que o ponto inicial x_0 esteja suficientemente próximo de $x_* = \arg \min f(x)$, a sequência x_0, x_1, x_2, \dots gerada pelo Método de Newton converge para o minimizador (necessariamente único) x_* de f com taxa de convergência quadrática.

Embora o Método de Newton seja frequentemente apresentado como um algoritmo para encontrar raízes de funções (isto é, soluções de $f(x) = 0$), ele também pode ser usado para encontrar pontos críticos de uma função $f: \mathbb{R}^d \rightarrow \mathbb{R}$, ou seja, pontos onde o gradiente $\nabla f(x)$ se anula.

Para maximizar uma função f , podemos aplicar o Método de Newton à equação $\nabla f(x) = 0$. Nesse caso, a atualização iterativa assume a forma:

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k),$$

onde $\nabla f(x_k)$ é o gradiente de f no ponto x_k , e $\nabla^2 f(x_k)$ é a hessiana (a matriz de derivadas segundas).

Se a matriz hessiana for negativa definida, então x_k está em uma vizinhança de um ponto de máximo local de f , e a iteração de Newton caminha nessa direção. O método é particularmente eficiente quando f é suave (duas vezes diferenciável) e fortemente côncava, pois a convergência é rápida e quadrática nas proximidades do máximo.

Em problemas de otimização como a regressão logística, onde queremos maximizar a log-verossimilhança, esse procedimento é especialmente útil: basta aplicar o Método de Newton à função log-verossimilhança $\log \ell(\beta)$ para obter uma sequência de estimativas para os coeficientes β .

Para maximizar a log-verossimilhança de uma regressão logística, usamos o Método de Newton aplicando-o à função

$$\ell(\beta) = \sum_{i=1}^n \left[Y_i \langle \beta, x_i \rangle - \log \left(1 + e^{\langle \beta, x_i \rangle} \right) \right].$$

- **Gradiente:** o vetor gradiente da log-verossimilhança em relação a $\beta \in \mathbb{R}^d$ é dado por:

$$\nabla \ell(\beta) = \sum_{i=1}^n (Y_i - g(x_i)) x_i,$$

onde $g(x_i) = \frac{e^{\langle \beta, x_i \rangle}}{1 + e^{\langle \beta, x_i \rangle}}$ é a predição do modelo logístico no ponto x_i . Essa expressão pode ser escrita matricialmente como:

$$\nabla \ell(\beta) = X^\top (Y - \hat{g}),$$

em que $X \in \mathbb{R}^{n \times d}$ é a matriz de design, $Y \in \mathbb{R}^n$ o vetor de respostas e $\hat{g} \in \mathbb{R}^n$ o vetor das probabilidades preditas.

- **Hessiana:** a matriz hessiana da log-verossimilhança é dada por:

$$\nabla^2 \ell(\beta) = -X^\top W X,$$

onde $W \in \mathbb{R}^{n \times n}$ é uma matriz diagonal com elementos

$$W_{ii} = g(x_i)(1 - g(x_i)).$$

Exercício 7. Prove a expressão do gradiente e hessiana em dimensão 1 e se convença que vale para dimensões maiores.

Note que a hessiana é negativa definida sempre que $0 < g(x_i) < 1$, o que garante que estamos maximizando uma função côncava. O método de Newton atualiza os parâmetros pela regra:

$$\beta_{k+1} = \beta_k - [\nabla^2 \ell(\beta_k)]^{-1} \nabla \ell(\beta_k).$$

No notebook que você encontra [aqui](#), apresentamos uma implementação do método de Newton para resolver o problema de regressão logística via maximização da verossimilhança.

Exercício 8. Entenda o código acima.

4.3 Outras métricas de avaliação

Nem sempre a função de risco $R(g) = \mathbb{E}[\mathbb{I}(Y \neq g(X))] = \mathbb{P}(Y \neq g(X))$ fornece uma visão completa da qualidade de um classificador g . Em certos contextos, como detecção de doenças raras, o risco pode parecer pequeno mesmo quando o desempenho do modelo está longe do ideal.

Considere, por exemplo, um cenário em que $Y = 1$ representa um paciente doente, e $Y = 0$, um paciente saudável. Suponha que, numa amostra de 1.000 indivíduos, apenas 10 estejam doentes. Um classificador trivial que sempre prediz $g(x) = 0$ (isto é, que todos são saudáveis) terá risco muito baixo, pois $\mathbb{P}(Y = 1)$ é pequena. No entanto, tal modelo falha completamente em identificar os casos realmente relevantes — os pacientes doentes.

Para melhor avaliar o desempenho de classificadores, especialmente em situações de desbalanceamento de classes, é comum recorrer ao uso de *matrizes de confusão*, que distinguem corretamente os diferentes tipos de acerto e erro. A seguir, apresentamos um exemplo típico:

Predição	Valor real	
	$Y = 0$	$Y = 1$
$g(x) = 0$	VN (negativo verdadeiro)	FN (falso negativo)
$g(x) = 1$	FP (falso positivo)	VP (positivo verdadeiro)

Tabela 4.1: Matriz de confusão.

A partir dessa tabela, podemos definir diversas métricas de desempenho. Uma delas é:

- **Sensibilidade (ou recall):**

$$\text{Sensibilidade} = \frac{\text{VP}}{\text{VP} + \text{FN}}$$

que representa a proporção de indivíduos doentes corretamente identificados pelo classificador.

- **Especificidade:**

$$\text{Especificidade} = \frac{\text{VN}}{\text{VN} + \text{FP}}$$

Mede a proporção de indivíduos saudáveis corretamente identificados como tais.

- **Valor predictivo positivo (ou precisão):**

$$\text{VPP} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

Indica, entre os exemplos classificados como positivos, quantos de fato são positivos.

- **Valor predictivo negativo:**

$$\text{VPN} = \frac{\text{VN}}{\text{VN} + \text{FN}}$$

Representa a proporção de negativos corretos entre os exemplos classificados como negativos.

- **Medida F1:**

$$F1 = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

que é a média harmônica entre sensibilidade (S) e precisão (VPP), sendo útil para balancear ambas as medidas.

Considere novamente o classificador constante $g(x) \equiv 0$. Neste caso, ele nunca prevê a classe positiva. Isso resulta em sensibilidade igual a zero (pois nenhum positivo é detectado) e especificidade igual a um (todos os negativos são corretamente identificados). Apesar de apresentar um bom desempenho segundo a especificidade, esse modelo é incapaz de detectar os casos relevantes (os positivos), tornando seu uso questionável.

Por isso, é fundamental considerar múltiplas métricas, principalmente em contextos de desequilíbrio entre classes — como em problemas médicos, onde a classe positiva pode representar uma condição rara.

As estatísticas derivadas da matriz de confusão devem ser interpretadas como estimativas amostrais de probabilidades condicionais. Por exemplo, a sensibilidade estima $\mathbb{P}(g(X) = 1 | Y = 1)$, enquanto a especificidade estima $\mathbb{P}(g(X) = 0 | Y = 0)$. Para garantir validade e evitar vieses, especialmente sobreajuste, é importante que essas métricas sejam calculadas em uma amostra separada de teste ou validação.

4.4 Ajuste do limiar de decisão

Classificadores probabilísticos, como a regressão logística, produzem uma estimativa da probabilidade condicional $g(x) = \mathbb{P}(Y = 1 | X = x)$. Para converter essa predição contínua em uma decisão binária (classe 0 ou 1), é necessário escolher um *limiar de corte* $t \in [0, 1]$. O valor padrão mais comum é $t = 0,5$, ou seja, prediz-se a classe positiva sempre que $g(x) \geq 0,5$.

No entanto, esse limiar pode ser ajustado de forma estratégica. Em particular, quando as classes estão desbalanceadas — por exemplo, se a classe positiva é rara — o uso de um limiar padrão pode resultar em desempenho insatisfatório. Um classificador pode, por exemplo, raramente atribuir probabilidades maiores que 0,5 à classe positiva, mesmo quando está relativamente confiante, simplesmente porque a classe é rara nos dados.

Ao reduzir o limiar t , tornamo-nos mais propensos a prever a classe positiva, o que tende a aumentar a **sensibilidade (recall)** — isto é, aumentamos a chance de detectar os verdadeiros positivos. Por outro lado, essa mudança normalmente reduz a **precisão (VPP)**, já que mais exemplos negativos podem ser erroneamente classificados como positivos.

De modo análogo, aumentar o limiar tende a aumentar a precisão, mas à custa da sensibilidade.

Limiar t	Recall	Precisão
baixo ($\downarrow t$)	alto	baixo
alto ($\uparrow t$)	baixo	alto

Esse comportamento reflete um trade-off clássico. Em aplicações médicas, por exemplo, pode-se preferir alta sensibilidade para evitar deixar passar casos positivos, mesmo que isso implique

em mais falsos positivos. Já em sistemas de recomendação, pode ser mais importante manter alta precisão, mesmo que alguns casos positivos sejam perdidos.

Na prática, o limiar ideal pode ser escolhido com base em uma métrica combinada (como F1-score) ou por análise da curva ROC ou curva precisão-recall, conforme os objetivos do problema.

4.4.1 Escolhendo o limiar de decisão

A escolha do limiar t é, em muitos casos, uma decisão crítica e depende diretamente do contexto e dos custos associados aos diferentes tipos de erro. Não existe um valor universalmente ótimo: o melhor limiar depende dos objetivos do problema, das consequências práticas dos erros e da distribuição das classes. Abaixo exibimos algumas possibilidades para encontrar o corte ótimo.

Maximizar F1

Uma forma prática de selecionar t é por meio da otimização de alguma métrica de desempenho, como a **F1-score**, que busca um equilíbrio entre precisão e recall. O limiar ótimo nesse caso seria:

$$t^* = \arg \max_t \text{F1-score}(t).$$

Curva ROC

A **curva ROC** (Receiver Operating Characteristic) é uma ferramenta gráfica amplamente utilizada para avaliar o desempenho de classificadores binários. Ela é construída variando o limiar de decisão t e observando como isso afeta duas métricas fundamentais:

- **Taxa de verdadeiros positivos (TVP) ou sensibilidade:**

$$\text{TVP} = \frac{\text{VP}}{\text{VP} + \text{FN}},$$

que mede a proporção de positivos corretamente identificados.

- **Taxa de falsos positivos (TFP):**

$$\text{TFP} = \frac{\text{FP}}{\text{FP} + \text{VN}},$$

que mede a proporção de negativos incorretamente classificados como positivos.

A curva ROC é um gráfico onde a TFP é colocada no eixo horizontal e a TVP no eixo vertical. Cada ponto da curva corresponde ao desempenho do classificador para um determinado limiar $t \in [0, 1]$.

O comportamento da curva é o seguinte:

- Para limiares muito baixos (por exemplo, $t \approx 0$), o classificador tende a prever quase tudo como positivo. Isso resulta em uma TVP alta, mas também em uma TFP alta.
- Para limiares muito altos (por exemplo, $t \approx 1$), o classificador quase nunca prevê a classe positiva. Assim, tanto a TVP quanto a TFP são baixas.
- Conforme o limiar varia, a curva se forma conectando esses pares (TFP, TVP).

Um classificador perfeito atingiria o ponto $(0, 1)$, ou seja, sem falsos positivos e com todos os verdadeiros positivos detectados. Na prática, quanto mais a curva ROC se aproxima do canto superior esquerdo, melhor é o desempenho do classificador.

Uma métrica associada à curva ROC é a **área sob a curva** (AUC — Area Under the Curve). O valor da AUC varia de 0 a 1:

- AUC = 1 indica um classificador perfeito;
- AUC = 0,5 corresponde a um classificador aleatório (linha diagonal);
- AUC < 0,5 sugere que o classificador está pior que o acaso (e poderia ser melhorado invertendo as previsões).

A curva ROC é especialmente útil quando queremos analisar o comportamento do classificador sob diferentes limiares, ou quando o custo dos erros de diferentes tipos não é simétrico. No entanto, em contextos com classes desbalanceadas, a curva ROC pode ser menos informativa — nesses casos, a curva precisão-recall costuma oferecer uma análise mais sensível.

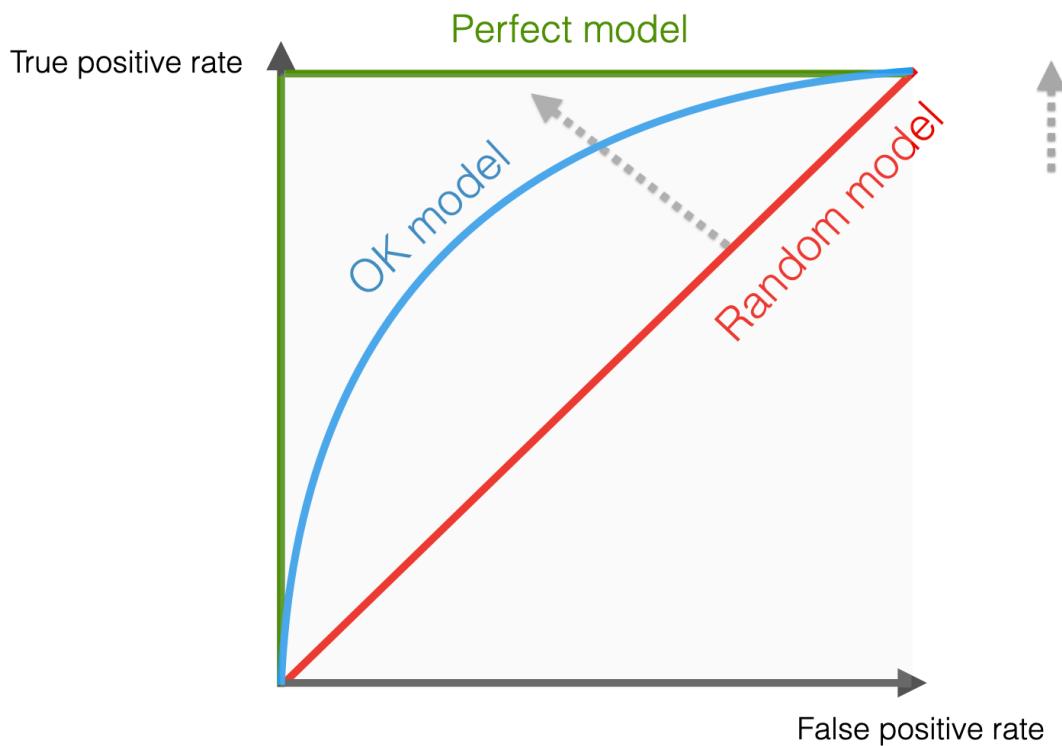


Figura 4.1: Curva ROC

Exercício 9. Acesse [esse site aqui](#) e brinque com a simulação de escolha de corte para classificação.

Curva Precisão-Recall

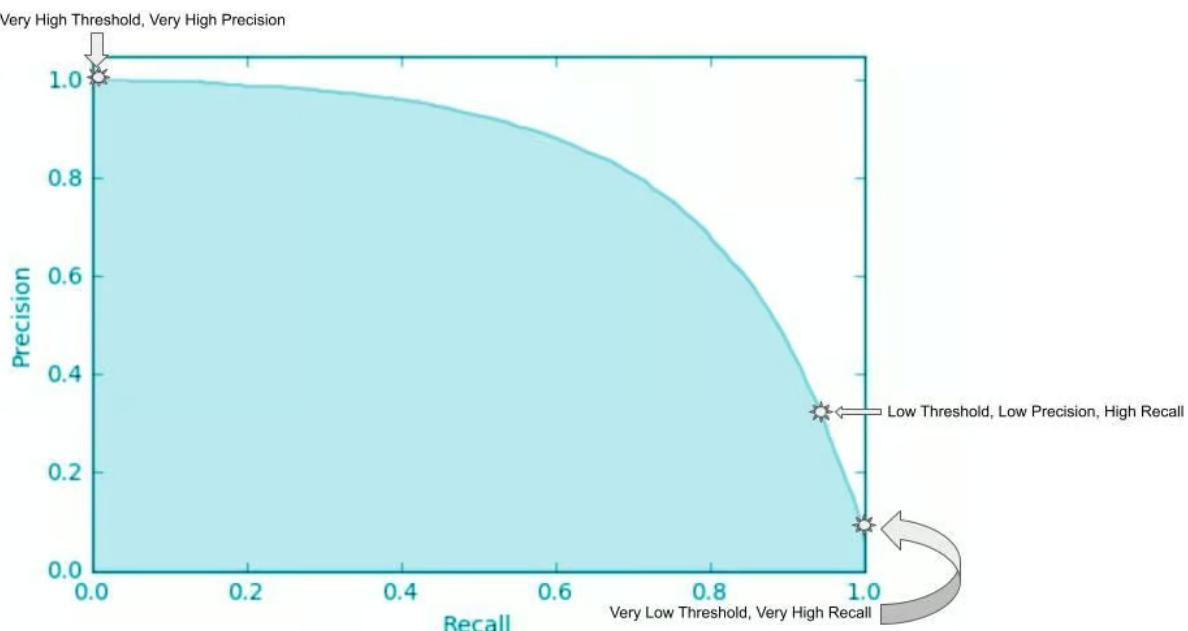
A **curva precisão-recall** é outra ferramenta gráfica fundamental para avaliar classificadores binários, especialmente em cenários onde as classes estão desbalanceadas — ou seja, quando a classe positiva é muito menos frequente que a negativa.

Enquanto a curva ROC mostra o trade-off entre sensibilidade (ou taxa de verdadeiros positivos) e a taxa de falsos positivos, a curva precisão-recall foca diretamente em duas métricas de interesse: Precision e Recall.

Para construir a curva, variamos o limiar de decisão $t \in [0, 1]$ usado para converter probabilidades em classificações, e calculamos os pares (Recall, Precisão) correspondentes a cada valor de t . Ao plotar esses pontos no plano, obtemos a curva precisão-recall.

Quanto mais a curva se aproxima do canto superior direito (alta precisão e alto recall), melhor é o desempenho do classificador. A área sob a curva (PR AUC — Precision-Recall Area Under Curve) pode ser usada como uma medida global de desempenho, assim como a AUC da curva ROC.

Em cenários onde a classe positiva é rara, a curva ROC pode dar uma falsa sensação de bom desempenho, pois o classificador pode ter uma taxa de falsos positivos muito baixa simplesmente porque há poucos exemplos positivos para errar. Já a curva precisão-recall é mais sensível a esse tipo de situação: ela penaliza mais severamente classificadores que não conseguem manter boa precisão ao tentar capturar mais positivos.



Exercício 10. Rode o seguinte [código](#) e faça alterações para se familiarizar com a parte computacional.

Capítulo 5

KNN

5.1 KNN para classificação

Idealmente, gostaríamos sempre de prever respostas qualitativas usando o classificador de Bayes, que é ótimo em termos teóricos. No entanto, na prática, não conhecemos a distribuição condicional de Y dado X , o que torna impossível a construção direta do classificador de Bayes. Por isso, ele é tratado como um padrão-ouro inatingível, servindo apenas como referência para avaliar outros métodos.

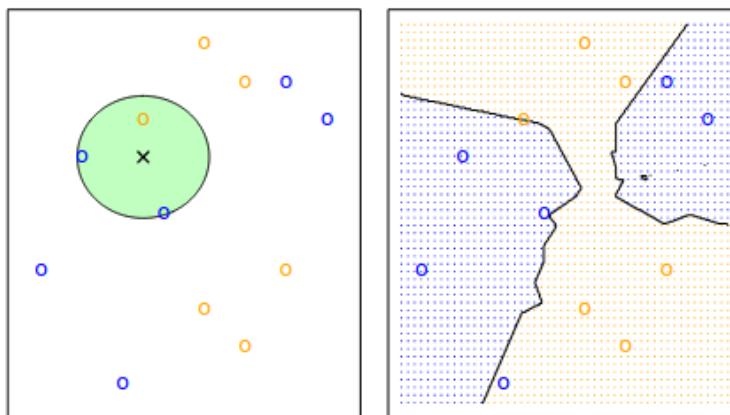


Figura 5.1: KNN com $k = 3$.

Diversas abordagens tentam estimar a distribuição condicional de Y dado X , e classificam uma nova observação atribuindo-a à classe com maior *probabilidade estimada*. Um método simples e amplamente utilizado é o dos *K-vizinhos mais próximos* (KNN). Dado um valor de K e uma nova observação x_0 , o KNN identifica os K pontos mais próximos de x_0 no conjunto de treinamento. Denotando esse subconjunto por \mathcal{N}_0 , a probabilidade condicional da classe j é estimada como:

$$\mathbb{P}(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathbb{I}(y_i = j).$$

Por fim, o KNN atribui à observação x_0 a classe com a maior probabilidade estimada.

Por exemplo, suponha que escolhemos $K = 3$, e que os três vizinhos mais próximos de x_0 sejam dois da classe azul e um da laranja. A probabilidade estimada para a classe azul será $2/3$, e para a classe laranja, $1/3$. Assim, o KNN classificará x_0 como pertencente à classe azul. Ao variar K , as fronteiras de decisão do classificador também mudam.

Apesar de ser uma abordagem bastante simples, o KNN pode gerar classificadores que se aproximam surpreendentemente bem do classificador de Bayes. Por exemplo, em um conjunto de dados simulado, com $K = 10$, o erro de teste do KNN foi de aproximadamente 0.1363, muito próximo do erro de Bayes, que era 0.1304.

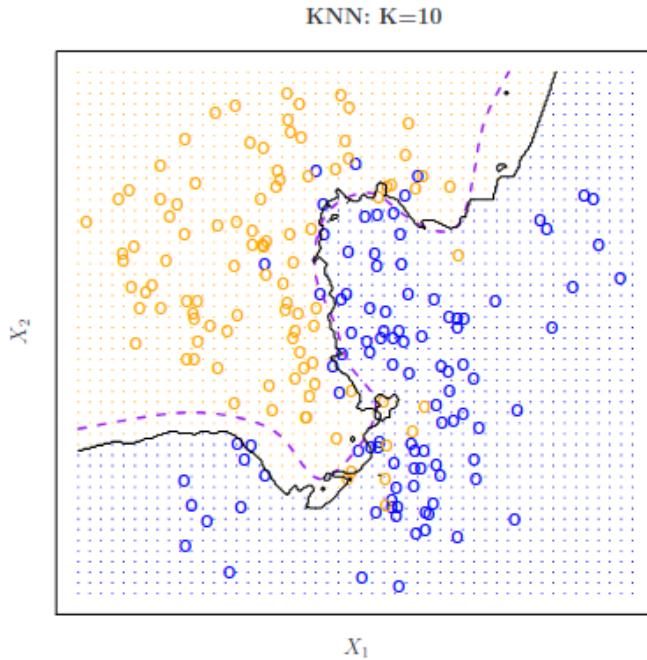


Figura 5.2: A curva preta representa a fronteira de decisão do classificador KNN, utilizando $K = 10$. A fronteira de decisão de Bayes é indicada pela linha tracejada roxa. As duas fronteiras são bastante semelhantes.

O valor de K influencia fortemente o desempenho do KNN. Quando $K = 1$, o modelo se torna extremamente flexível, adaptando-se até aos ruídos dos dados – o que resulta em baixo viés, mas alta variância. Já para valores muito grandes de K , o classificador se torna excessivamente rígido, levando a uma maior taxa de erro devido ao viés elevado. Nesse cenário, os dados são "suavizados" demais, ignorando padrões mais sutis.

Essa relação entre viés e variância se manifesta de forma clássica no gráfico de erro de teste versus $1/K$: à medida que K aumenta, a variância diminui e o viés cresce, formando uma curva em formato de U. O ponto ideal de K é geralmente aquele que minimiza o erro de teste – frequentemente em torno de $K = 10$.

Escolher corretamente o nível de flexibilidade (ou complexidade) do modelo é essencial tanto em tarefas de regressão quanto de classificação.

Note que dado um ponto de teste $x \in \mathbb{R}^p$, o algoritmo ingênuo compara esse ponto com todos os pontos do conjunto de treinamento para calcular a resposta prevista. Isso resulta em uma complexidade de $\mathcal{O}(np)$ por ponto de teste em \mathbb{R}^p . Quando p é grande, esse custo se torna

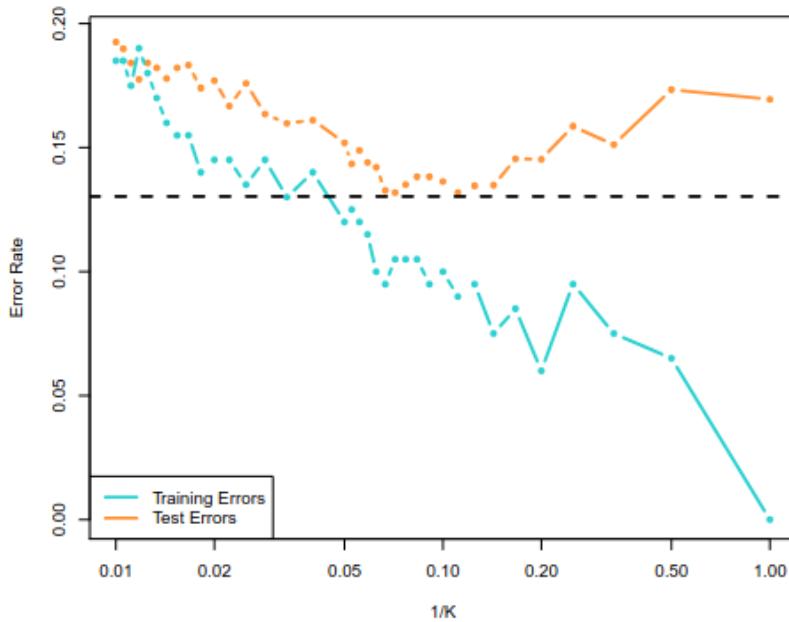


Figura 5.3: A taxa de erro de treinamento do KNN (em azul, 200 observações) e a taxa de erro de teste (em laranja, 5.000 observações) nos dados da Figura 2.13, à medida que o nível de flexibilidade (avaliado por $1/K$ em escala log

elevado tanto em tempo quanto em memória. Existem técnicas de indexação para busca de vizinhos mais próximos (possivelmente aproximadas), como as chamadas *k-d trees*, que apresentam complexidade logarítmica em n (embora com tempo adicional de compilação) e uso de memória que pode crescer exponencialmente com a dimensão.

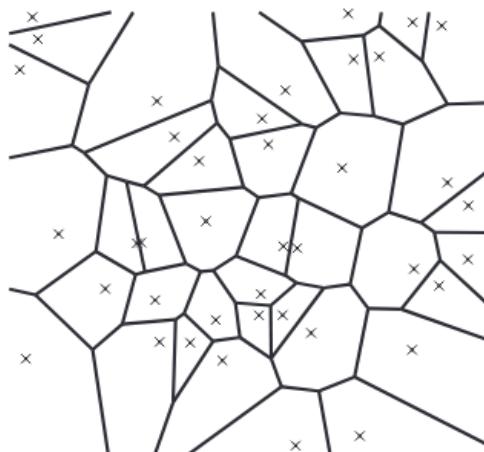


Figura 5.4: Diagrama de Voronoi associado ao algoritmo KNN. Cada região representa o conjunto de pontos que seriam classificados da mesma forma com base no vizinho mais próximo.

5.2 KNN para regressão

O método de regressão KNN é bastante semelhante ao classificador KNN. Dado um valor de K e um ponto de predição x_0 , a regressão KNN identifica as K observações de treinamento mais próximas de x_0 , representadas por \mathcal{N}_0 . Em seguida, estima $f(x_0)$ utilizando a média das respostas correspondentes. Em outras palavras,

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in \mathcal{N}_0} y_i.$$

A figura abaixo ilustra dois ajustes do KNN sobre um conjunto de dados com $p = 2$ predtores. O ajuste com $K = 1$ aparece no painel à esquerda, enquanto o painel da direita mostra o ajuste com $K = 9$. Observa-se que, para $K = 1$, o KNN interpola perfeitamente as observações de treinamento, resultando em um ajuste com descontinuidades abruptas. Para $K = 9$, o ajuste continua sendo uma função por partes, mas o uso da média sobre nove vizinhos suaviza as regiões constantes, produzindo uma predição mais suave.

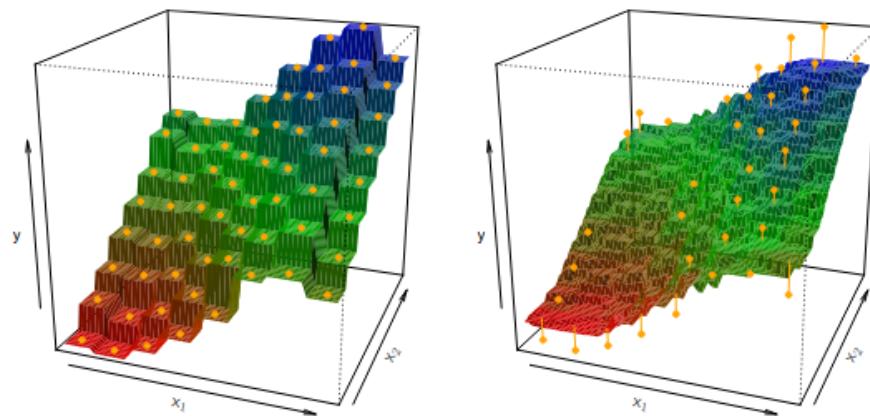


Figura 5.5: Knn para conjunto de 64 pontos. Do lado esquerdo $K = 1$ e do lado direito $K = 9$.

De modo geral, o valor ideal de K está ligado ao compromisso entre viés e variância. Um valor pequeno de K resulta em um modelo altamente flexível, com baixo viés mas alta variância, pois a predição em uma dada região depende fortemente de uma única observação. Já valores maiores de K reduzem a variância, mas podem aumentar o viés.

5.3 O que é treinado no KNN?

Diferentemente de muitos algoritmos de aprendizado supervisionado, o *K-Nearest Neighbors* (KNN) não realiza um processo explícito de treinamento no sentido tradicional de ajustar parâmetros internos a partir dos dados. No KNN, o chamado *treinamento* consiste simplesmente em armazenar o conjunto de dados de treino, ou seja, memorizar os pares (X_i, Y_i) .

Durante a fase de predição, dado um novo ponto X , o algoritmo identifica os K pontos mais próximos entre os dados armazenados, com base em uma medida de distância (como a distância

Euclidiana). Em seguida, para tarefas de regressão, a predição é obtida calculando a média (ou outro agregador) dos valores Y_i correspondentes a esses vizinhos. Para tarefas de classificação, a predição é feita atribuindo a classe mais frequente entre os vizinhos.

Assim, o KNN é considerado um método *lazy learner*, pois posterga todo o trabalho de generalização até o momento da predição, ao contrário de métodos como regressão linear ou redes neurais, que constroem um modelo explícito durante o treinamento.

5.4 Regressão Linear vs. KNN

Em que situações a regressão linear — uma abordagem paramétrica — supera métodos não paramétricos como o KNN? A resposta é simples: *uma abordagem paramétrica terá melhor desempenho se sua forma funcional estiver próxima da verdadeira relação entre as variáveis*.

A figura abaixo mostra um exemplo com dados gerados a partir de um modelo linear unidimensional. As linhas pretas representam a função verdadeira $f(X)$, enquanto as curvas azuis indicam os ajustes feitos pelo KNN com $K = 1$ e $K = 9$. O ajuste com $K = 1$ é muito irregular, enquanto o ajuste com $K = 9$ é bem mais suave e se aproxima melhor da função real.

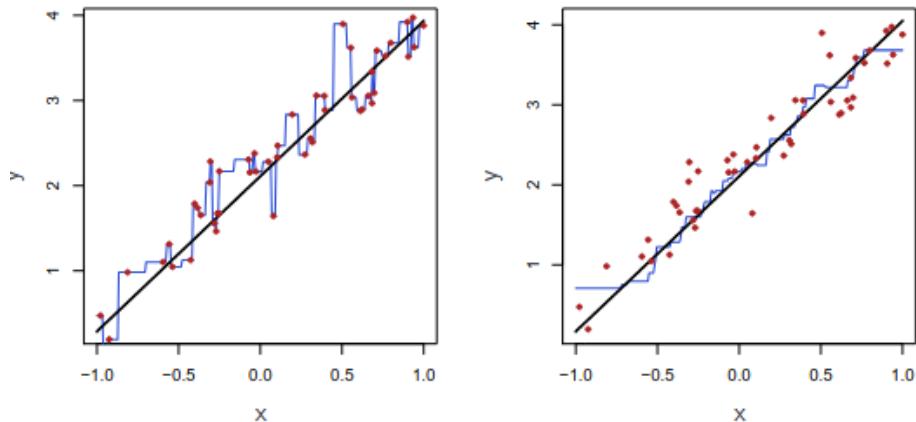


Figura 5.6: Ajustes de $\hat{f}(X)$ usando regressão KNN em um conjunto de dados unidimensional com 50 observações. A relação verdadeira é representada pela linha preta contínua. **Esquerda:** A curva azul corresponde a $K = 1$ e interpola os dados de treinamento, passando exatamente pelos pontos. **Direita:** A curva azul corresponde a $K = 9$ e fornece um ajuste mais suave.

Como a relação subjacente é linear, é difícil para um método não paramétrico competir. A flexibilidade extra do KNN aumenta a variância da predição, sem uma compensação no viés. A linha azul tracejada na figura abaixo representa o ajuste feito por regressão linear — quase perfeito nesse caso. O painel da direita da mesma figura mostra que a regressão linear supera o KNN em termos de erro médio quadrático (MSE) de teste. A linha verde na figura representa esse erro do KNN conforme o valor de $1/K$. Os erros do KNN são consistentemente maiores que os da regressão linear, exceto quando K é grande, situação em que o desempenho do KNN se aproxima do da regressão linear. No entanto, com K pequeno, o KNN tem desempenho significativamente pior.

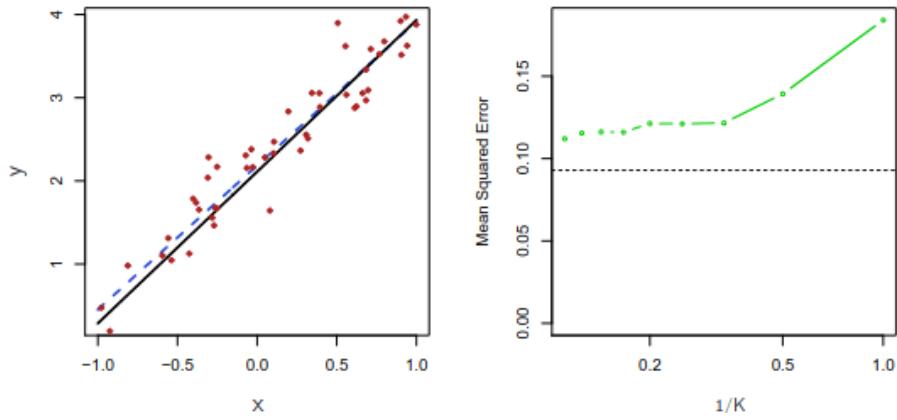


Figura 5.7: **Esquerda:** A linha tracejada azul representa o ajuste por mínimos quadrados. Como $f(X)$ é de fato linear (representada pela linha preta), a regressão linear fornece uma excelente estimativa de $f(X)$. **Direita:** A linha horizontal tracejada indica o erro quadrático médio (MSE) de teste da regressão linear, enquanto a linha verde contínua mostra o MSE do KNN como função de $1/K$ (em escala logarítmica). A regressão linear alcança um erro de teste menor que o KNN, pois a relação é linear. No caso da regressão KNN, os melhores resultados ocorrem para valores altos de K , o que corresponde a valores pequenos de $1/K$.

Na prática, a relação entre X e Y raramente é exatamente linear. A figura abaixo explora o desempenho da regressão linear e do KNN em situações com diferentes níveis de não-linearidade. Quando a relação verdadeira é quase linear (linha superior), a regressão linear mantém o menor erro de teste. Conforme a não-linearidade aumenta (linha inferior), o KNN supera substancialmente a regressão linear em todos os valores de K . Note que o MSE do KNN permanece estável, enquanto o da regressão linear cresce significativamente. Esse comportamento ressalta que, em contextos não lineares, métodos não paramétricos podem ser preferíveis.

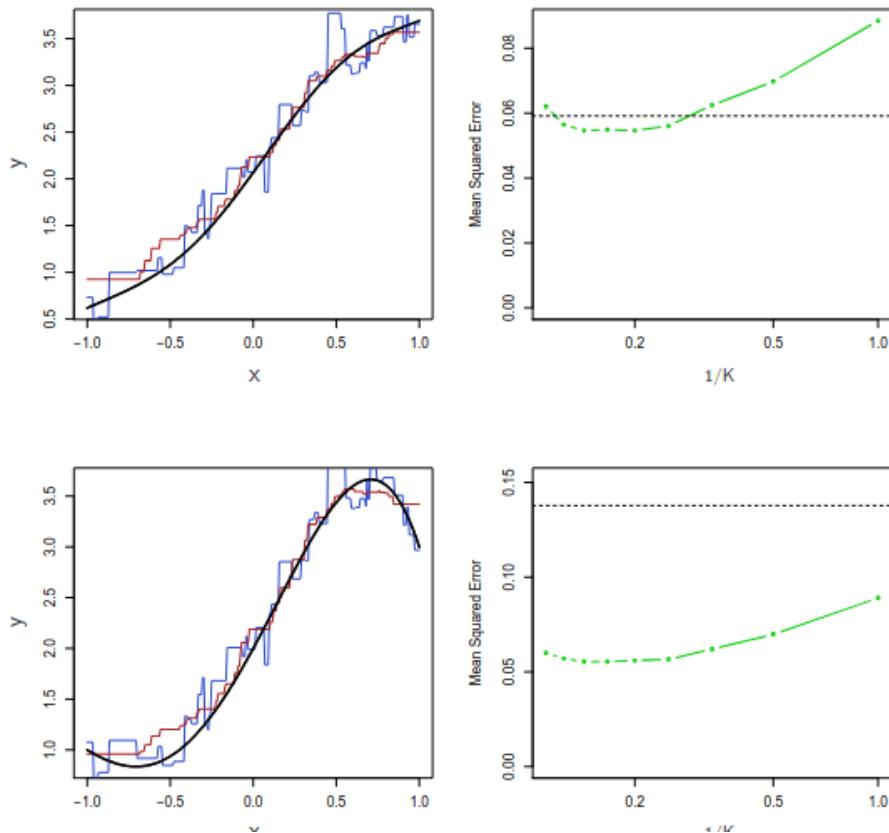


Figura 5.8: **Acima à esquerda:** Em um cenário com relação levemente não linear entre X e Y (linha preta contínua), são exibidos os ajustes do KNN com $K = 1$ (azul) e $K = 9$ (vermelho). **Acima à direita:** Para esse caso levemente não linear, mostra-se o erro quadrático médio (MSE) de teste para a regressão linear (linha preta horizontal) e para o KNN com diferentes valores de $1/K$ (linha verde). **Abaixo à esquerda e à direita:** Como no painel superior, mas considerando agora uma relação fortemente não linear entre X e Y .

O exemplos anteriores mostram que o KNN tende a ter desempenho ligeiramente inferior à regressão linear em relações lineares, mas muito superior quando a relação é não-linear. Em situações reais, como geralmente não conhecemos a forma da relação verdadeira, o KNN pode ser uma escolha segura: no pior caso, será apenas ligeiramente inferior à regressão linear, mas pode superar substancialmente em casos não-lineares.

Exercício 11. Entender o código [aqui](#).

Vale destacar que ambas as figuras consideram apenas o caso com $p = 1$ preditor. Em dimensões mais altas, o KNN tende a ter desempenho pior que a regressão linear, devido à **maldição da dimensionalidade**.

Capítulo 6

Modelos baseados em árvores

Neste capítulo, descrevemos os métodos baseados em árvores para tarefas de regressão e classificação. Essas abordagens consistem em *estratificar* ou *segmentar* o espaço dos preditores em um número reduzido de regiões simples. Para fazer uma predição em uma nova observação, utilizamos geralmente a média (em regressão) ou a moda (em classificação) das respostas das observações de treinamento que pertencem à mesma região. Como as regras de divisão utilizadas podem ser representadas na forma de uma árvore, essas abordagens são conhecidas como métodos de *árvore de decisão*.

6.1 Árvores de decisão

Os métodos baseados em árvores são simples e de fácil interpretação. No entanto, muitas vezes não alcançam o mesmo desempenho preditivo que os melhores métodos supervisionados. Por isso, além das árvores de decisão, introduzimos também técnicas como *bagging*, *florestas aleatórias*, *boosting* e *Bayesian additive regression trees*.

Essas técnicas envolvem a construção de múltiplas árvores que, em seguida, são combinadas para gerar uma predição por consenso. Veremos que, embora a combinação de muitas árvores possa resultar em ganhos expressivos de acurácia preditiva, isso costuma vir acompanhado de uma redução na interpretabilidade do modelo.

O processo de construção de uma árvore de regressão pode ser dividido, grosseiramente, em duas etapas:

1. Dividimos o espaço dos preditores — isto é, o conjunto de valores possíveis para X_1, X_2, \dots, X_p — em J regiões distintas e não sobrepostas, R_1, R_2, \dots, R_J .
2. Para cada observação que cai em uma região R_j , fazemos a mesma predição, que consiste simplesmente na média das respostas Y_i associadas às observações de treino que pertencem àquela região.

Por exemplo, suponha que, na Etapa 1, obtemos duas regiões, R_1 e R_2 . Se a média das respostas nas observações de treino de R_1 é 10, e em R_2 é 20, então para uma nova observação x :

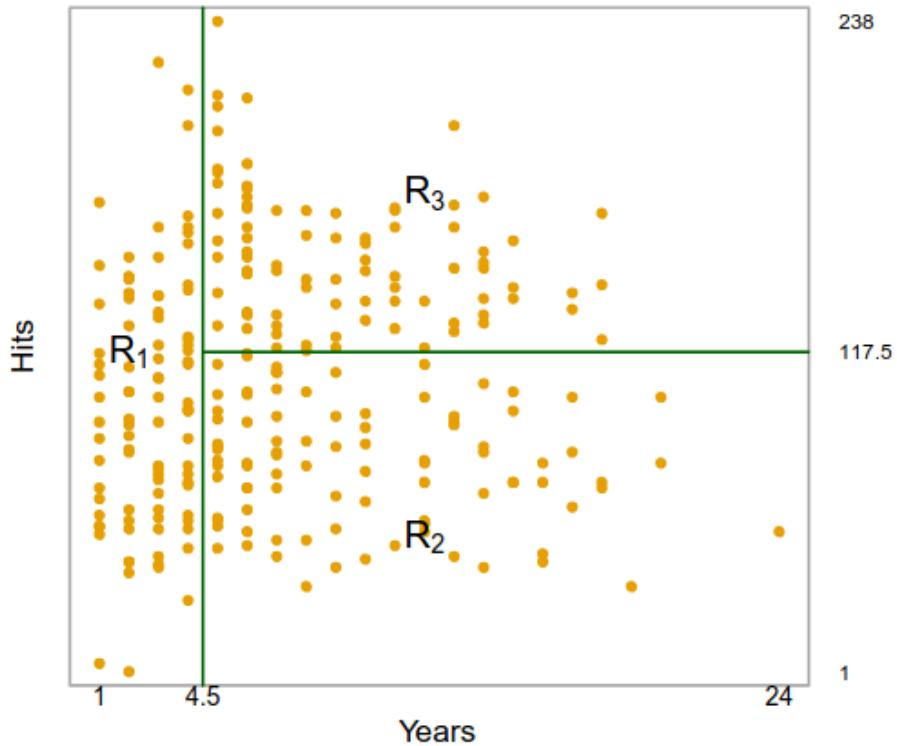


Figura 6.1: Uma possível partição usando árvores

$$\hat{Y} = \begin{cases} 10, & \text{se } x \in R_1, \\ 20, & \text{se } x \in R_2. \end{cases}$$

Na prática, as regiões R_1, \dots, R_J são escolhidas como retângulos de alta dimensão (chamados *boxes*) por simplicidade e facilidade de interpretação. O objetivo é minimizar a soma dos erros quadráticos (RSS):

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \quad (6.1)$$

onde \hat{y}_{R_j} é a média das respostas das observações de treino em R_j .

6.1.1 Divisão Binária Recursiva

Como é computacionalmente inviável testar todas as possíveis partições, adotamos uma abordagem gulosa (*greedy*) e de cima para baixo (*top-down*), conhecida como **divisão binária recursiva**. A cada passo, escolhemos a melhor divisão possível naquele momento, sem necessariamente garantir que seja a melhor árvore globalmente.

Para realizar a divisão binária recursiva, escolhemos um preditor X_j e um ponto de corte s tal que a divisão do espaço dos preditores em duas regiões,

$$R_1(j, s) = \{X \mid X_j < s\} \quad \text{e} \quad R_2(j, s) = \{X \mid X_j \geq s\},$$

gera a maior redução possível no erro quadrático (RSS). O objetivo é encontrar os valores de j e s que minimizem:

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

onde \hat{y}_{R_1} e \hat{y}_{R_2} são as médias das respostas nas regiões correspondentes.

Esse processo é feito de forma gulosa: a cada passo, procuramos a melhor divisão possível localmente, sem considerar futuras divisões. Começamos com toda a base de dados como uma única região e, a cada iteração, escolhemos a divisão que mais reduz o RSS. Ao invés de dividir sempre o espaço completo, passamos a dividir uma das regiões existentes.

O processo continua até que algum critério de parada seja atingido, como por exemplo: nenhuma região contendo mais que cinco observações.

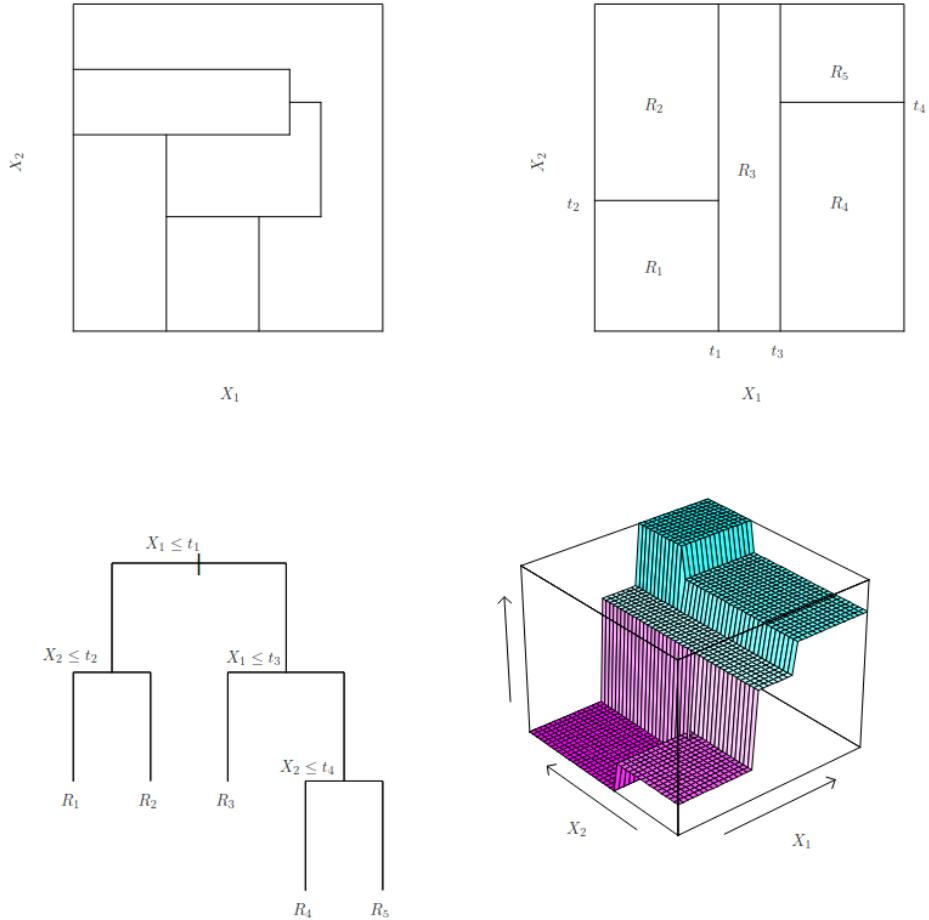


Figura 6.2: **Cima à esquerda:** Uma partição do espaço bidimensional dos preditores que **não poderia** ser obtida por divisão binária recursiva. **Cima à direita:** Resultado de uma divisão binária recursiva em duas dimensões. **Embaixo à esquerda:** Árvore de decisão correspondente à partição mostrada acima à direita. **Embaixo à direita:** Superfície de predição correspondente à árvore, destacando os degraus definidos pelas regiões.

Uma vez criadas as regiões R_1, \dots, R_J , a predição para uma nova observação é dada pela

média das respostas das observações de treinamento que pertencem à mesma região.

6.1.2 Poda de Árvores (Tree Pruning)

O processo de divisão recursiva pode produzir boas previsões no conjunto de treinamento, mas tende a sofrer com *overfitting*, levando a um mau desempenho no conjunto de teste. Isso ocorre porque a árvore resultante pode ser complexa demais.

Uma árvore menor, com menos divisões (isto é, com menos regiões R_1, \dots, R_J), pode ter menor variância e interpretação mais simples, ao custo de um leve aumento de viés. Uma alternativa ao crescimento total da árvore é interromper o processo quando a redução no RSS (soma dos erros quadráticos) gerada por uma divisão não ultrapassar um certo limiar.

Uma abordagem mais eficaz é crescer uma árvore grande T_0 , e depois podá-la para obter uma *subárvore*. A meta é selecionar a subárvore que leva ao menor erro de teste. Podemos estimar esse erro usando validação cruzada ou um conjunto de validação.

No entanto, como o número de subárvores possíveis é muito grande, utilizamos o método de **poda por complexidade de custo** (ou *cost-complexity pruning*, também chamado *weakest link pruning*).

Definimos uma sequência de árvores indexadas por um parâmetro de ajuste $\alpha \geq 0$. Para cada valor de α , escolhemos a subárvore $T \subseteq T_0$ que minimiza:

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|, \quad (8.4)$$

onde:

- $|T|$ é o número de nós terminais (ou regiões) da subárvore T ,
- R_m é a região associada ao m -ésimo nó terminal,
- \hat{y}_{R_m} é a média das respostas em R_m ,
- α penaliza árvores mais complexas.

Quando $\alpha = 0$, essa equação reduz-se ao erro de treinamento. À medida que α aumenta, a penalização por complexidade aumenta, favorecendo árvores menores.

O aumento de α causa a poda sucessiva dos galhos da árvore T_0 , de forma aninhada e prevável. Podemos então selecionar o melhor valor de α usando validação cruzada.

Uma vez escolhido α , a subárvore correspondente é então treinada no conjunto completo.

Algorithm 1 Construção de uma Árvore de Regressão com Poda

-
- 1: Use divisão binária recursiva para crescer uma árvore grande no conjunto de treino, parando apenas quando cada nó terminal tiver menos que um número mínimo de observações.
 - 2: Aplique a poda por complexidade de custo (*cost complexity pruning*) à árvore grande, gerando uma sequência de subárvore ótimas em função de α .
 - 3: Use validação cruzada com K blocos para escolher α :
 - 4: **for** cada bloco $k = 1, \dots, K$ **do**
 - 5: (a) Repita os Passos 1 e 2 usando todos os dados exceto o k -ésimo bloco.
 - 6: (b) Avalie o erro quadrático médio de predição no k -ésimo bloco, como função de α .
 - 7: **end for**
 - 8: Para cada valor de α , calcule o erro médio nos K blocos e selecione o α que minimiza esse erro médio.
 - 9: Retorne a subárvore do Passo 2 correspondente ao α escolhido.
-

Exercício 12. Alterar o código [aqui](#) para usar k -fold.

6.1.3 Árvores de Classificação

Uma árvore de classificação é muito parecida com uma árvore de regressão, com a diferença de que ela é usada para prever uma resposta qualitativa em vez de uma quantitativa. Em uma árvore de regressão, a resposta prevista para uma observação é a média das respostas Y das observações de treinamento que pertencem ao mesmo nó terminal. Já no caso de uma árvore de classificação, a previsão corresponde à classe mais frequente entre as observações de treinamento do nó terminal. Além de determinar a classe predita para cada região terminal, também é comum analisar as proporções das classes entre as observações de treinamento que caem em cada região.

O processo de construção de uma árvore de classificação é bastante similar ao utilizado em árvores de regressão, baseando-se em divisões binárias recursivas. No entanto, como estamos lidando com variáveis qualitativas, não podemos usar a soma de quadrados residual (RSS) como critério para realizar as divisões. Uma alternativa natural ao RSS é a *taxa de erro de classificação*, que mede a proporção de observações em uma região que não pertencem à classe mais frequente. Formalmente, ela é dada por:

$$E = 1 - \max_k \hat{p}_{mk}, \quad (6.2)$$

onde \hat{p}_{mk} representa a proporção de observações de treinamento na região m que pertencem à classe k .

Apesar de sua simplicidade, a taxa de erro de classificação não é suficientemente sensível para orientar o crescimento da árvore de maneira eficiente. Por essa razão, outras duas métricas são geralmente preferidas. A primeira é o *índice de Gini*, definido como:

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}). \quad (6.3)$$

O índice de Gini mede a variabilidade total entre as K classes. Ele assume valores baixos quando as proporções \hat{p}_{mk} estão próximas de zero ou um, indicando que o nó é puro.

Outra medida bastante utilizada é a *entropia*, que é dada por:

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}. \quad (6.4)$$

Assim como o índice de Gini, a entropia também assume valores pequenos quando o nó é puro. De fato, os valores numéricos do índice de Gini e da entropia costumam ser bastante próximos.

Na prática, ao construir uma árvore de classificação, costuma-se utilizar o índice de Gini ou a entropia para avaliar a qualidade de uma divisão, já que essas métricas são mais sensíveis à pureza dos nós do que a taxa de erro de classificação. Qualquer uma dessas três métricas pode ser empregada no processo de *poda* da árvore. No entanto, quando o objetivo final é maximizar a acurácia de predição da árvore final, a taxa de erro de classificação geralmente é preferida para guiar a poda.

Cálculo prático de Gini e Entropia

Suponha que, em um nó m , existam 10 observações distribuídas entre duas classes: Classe 0 com 4 observações e Classe 1 com 6 observações. Assim, as proporções de cada classe são:

$$\hat{p}_{m,0} = \frac{4}{10} = 0,4, \quad \hat{p}_{m,1} = \frac{6}{10} = 0,6.$$

O índice de Gini é definido por:

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

Para este exemplo, temos:

$$G = (0,4)(1 - 0,4) + (0,6)(1 - 0,6) = 0,4 \times 0,6 + 0,6 \times 0,4 = 0,24 + 0,24 = 0,48.$$

A entropia é dada por:

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log_2 \hat{p}_{mk}.$$

Logo:

$$D = - (0,4 \log_2(0,4) + 0,6 \log_2(0,6)).$$

Calculando os logaritmos:

$$\log_2(0,4) \approx -1,3219, \quad \log_2(0,6) \approx -0,73697.$$

Portanto:

$$D = - (0,4 \times (-1,3219) + 0,6 \times (-0,73697)) = -(-0,52876 - 0,44218) = 0,97094.$$

Como exemplo extremo, considere um nó puro com apenas observações da Classe 1. Neste caso, temos:

$$\hat{p}_{m,0} = 0, \quad \hat{p}_{m,1} = 1.$$

Assim, o índice de Gini será:

$$G = 0 \times (1 - 0) + 1 \times (1 - 1) = 0,$$

e a entropia será:

$$D = -(0 \times \log(0) + 1 \times \log(1)) = 0.$$

Na notação \hat{p}_{mk} , o índice m representa o nó terminal considerado e k indica a classe. Por exemplo, $\hat{p}_{3,0}$ é a proporção de observações da classe 0 no nó 3.

Vamos fazer um exemplo completo. Considere o seguinte conjunto de dados:

X	Classe
1	Azul
2	Vermelho
3	Azul
4	Vermelho
5	Azul

Suponha que realizamos um corte em $X = 3,5$.

Os dados ficam divididos da seguinte forma:

- **Grupo à esquerda** ($X \leq 3,5$): observações $\{1, 2, 3\}$, contendo 2 Azuis e 1 Vermelho. A proporção de Azuis é $p = \frac{2}{3}$, resultando em um índice de Gini:

$$G_{\text{esq}} = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9} \approx 0,444.$$

- **Grupo à direita** ($X > 3,5$): observações $\{4, 5\}$, com 1 Azul e 1 Vermelho. A proporção de Azuis é $p = \frac{1}{2}$, e o índice de Gini é:

$$G_{\text{dir}} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0,5.$$

O índice de Gini ponderado da divisão é:

$$G = \frac{3}{5} \cdot 0,444 + \frac{2}{5} \cdot 0,5 = 0,466.$$

Durante o crescimento da árvore, nosso objetivo é realizar divisões que minimizem o Gini ou a entropia, já que valores baixos dessas métricas indicam maior pureza dos nós. Portanto, entre todas as divisões possíveis, escolhe-se aquela que leva ao menor valor de Gini ou entropia nos nós resultantes. Isso equivale a buscar o maior ganho de pureza após o corte.

Intuitivamente, o índice de Gini e a entropia são medidas da *impureza* de um nó. Ambas indicam o quanto misturadas estão as classes dentro do nó. Quando todas as observações no nó pertencem à mesma classe, dizemos que o nó é puro, e tanto o Gini quanto a entropia valem zero. À medida que a mistura entre as classes aumenta, os valores de Gini e entropia aumentam. O índice de Gini pode ser interpretado como a probabilidade de uma classificação errada se

atribuirmos aleatoriamente uma classe a uma observação do nó, com base nas proporções das classes. Já a entropia mede a quantidade de *incerteza* ou *informação* necessária para descrever a classe de uma observação do nó. Nós com alta entropia têm alta incerteza, enquanto nós com baixa entropia contêm observações predominantemente de uma única classe.

Considere uma variável aleatória Y que assume valores em um conjunto finito $\mathcal{Y} = \{y_1, y_2, \dots, y_K\}$, com probabilidades associadas p_1, p_2, \dots, p_K , onde $p_k = \mathbb{P}(Y = y_k)$ e $\sum_{k=1}^K p_k = 1$. A entropia da distribuição de Y é dada por

$$H(Y) = - \sum_{k=1}^K p_k \log_2(p_k).$$

A entropia mede o grau de incerteza ou imprevisibilidade da variável Y . Ela é pequena quando uma classe domina, com probabilidade próxima de um, e é grande quando as probabilidades são semelhantes entre as classes.

Agora mostramos que a entropia é maximizada quando todas as classes têm a mesma probabilidade. Usamos a desigualdade de Jensen, lembrando que a função $\log_2(x)$ é côncava. Escrevendo a entropia como

$$H(Y) = \mathbb{E} \left[\log_2 \frac{1}{p(Y)} \right],$$

aplicamos Jensen com a variável aleatória $Z = 1/p(Y)$, obtendo

$$H(Y) \leq \log_2 \mathbb{E} \left[\frac{1}{p(Y)} \right].$$

A esperança $\mathbb{E}[1/p(Y)]$ pode ser escrita como

$$\mathbb{E} \left[\frac{1}{p(Y)} \right] = \sum_{k=1}^K p_k \cdot \frac{1}{p_k} = \sum_{k=1}^K 1 = K.$$

Portanto,

$$H(Y) \leq \log_2 K.$$

A igualdade ocorre quando p_k é constante para todos os k , ou seja, quando $p_k = 1/K$. Isso prova que a distribuição uniforme é a que maximiza a entropia.

6.1.4 Árvores versus modelos lineares

Árvores de regressão e classificação apresentam uma natureza bastante diferente em relação aos métodos clássicos de regressão e classificação discutidos nos Capítulos 3 e 4. Por exemplo, a regressão linear assume um modelo do tipo

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j,$$

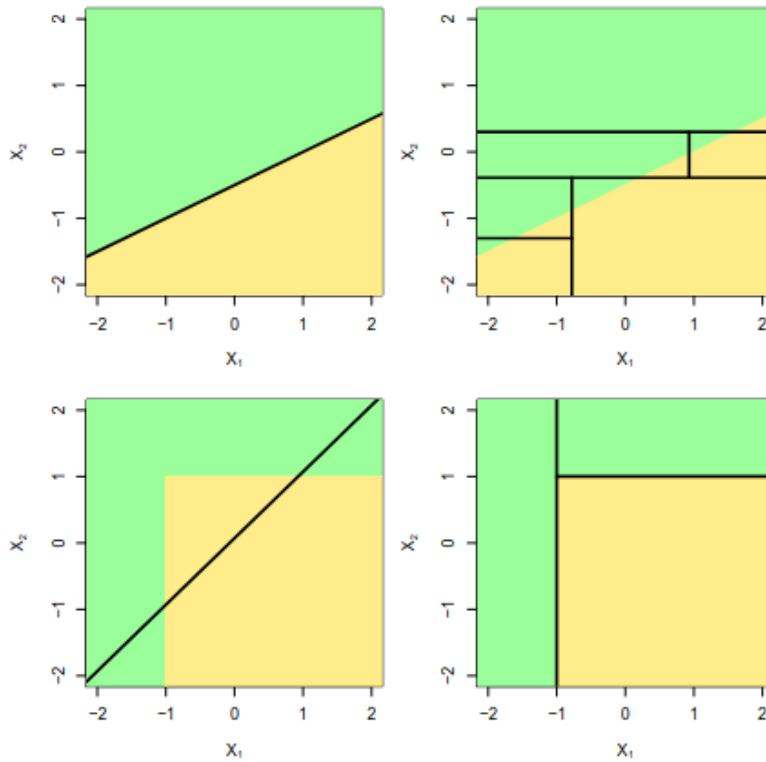
enquanto uma árvore de regressão assume um modelo da forma

$$f(X) = \sum_{m=1}^M c_m \cdot 1_{(X \in R_m)},$$

onde R_1, \dots, R_M representam uma partição do espaço de preditores.

Qual abordagem é melhor depende do problema. Se a relação entre as variáveis preditoras e a resposta pode ser bem aproximada por um modelo linear, como na equação acima, métodos como a regressão linear tendem a ter bom desempenho e podem superar uma árvore de regressão, que não explora essa estrutura linear. No entanto, se a relação entre os preditores e a resposta é altamente não-linear e complexa, como indicado pelo segundo modelo, as árvores de decisão podem apresentar desempenho superior aos métodos lineares.

Um exemplo ilustrativo é apresentado na Figura abaixo. No primeiro caso, a fronteira de decisão real é linear. O modelo linear (painel superior esquerdo) supera a árvore de decisão (painel superior direito), que realiza cortes paralelos aos eixos. No segundo caso, a fronteira de decisão verdadeira é não-linear. O modelo linear falha em capturar a fronteira real (painel inferior esquerdo), enquanto a árvore de decisão consegue uma boa separação (painel inferior direito).



É importante observar que outros fatores além do erro de teste também podem influenciar a escolha de um método de aprendizado estatístico. Em algumas situações, a interpretabilidade e a facilidade de visualização podem tornar as árvores uma escolha preferida, mesmo que o erro de previsão não seja o menor possível.

6.1.5 Vantagens e Desvantagens das Árvores

Árvores de decisão, tanto para regressão quanto para classificação, possuem diversas vantagens em relação aos métodos clássicos discutidos anteriormente.

- Árvores de decisão são, em geral, muito fáceis de explicar e interpretar. Muitas vezes, elas são ainda mais intuitivas do que modelos lineares. Alguns autores sugerem que as árvores refletem melhor o modo como as pessoas tomam decisões em situações práticas.
- Além disso, podem ser representadas graficamente, o que torna sua interpretação acessível mesmo para não especialistas, especialmente quando as árvores são pequenas.
- Árvores também lidam de forma natural com preditores qualitativos, sem exigir a criação de variáveis indicadoras (dummies).
- No entanto, as árvores frequentemente não atingem o mesmo nível de acurácia preditiva que outros métodos de regressão e classificação mais sofisticados.
- Outra limitação importante é a instabilidade: pequenas alterações nos dados podem levar a mudanças significativas na estrutura da árvore estimada.

Uma forma de superar essas limitações é combinar muitas árvores de decisão usando métodos como *Bagging* e *Florestas aleatórias*. Esses métodos serão apresentados na próxima seção e podem melhorar substancialmente o desempenho preditivo das árvores.

6.2 Bagging

As árvores de decisão discutidas anteriormente sofrem de alta variância. Isso significa que, se dividirmos o conjunto de treinamento em duas partes aleatórias e ajustarmos uma árvore de decisão em cada parte, os resultados obtidos podem ser bastante diferentes. Por outro lado, procedimentos com baixa variância produzem resultados semelhantes quando aplicados repetidamente a diferentes conjuntos de dados. A regressão linear, por exemplo, tende a apresentar baixa variância, especialmente quando a razão entre n e p é moderadamente grande.

Bootstrap aggregation, ou *bagging*, é um procedimento geral para reduzir a variância de um método de aprendizado estatístico. Essa técnica é particularmente útil no contexto de árvores de decisão.

Lembre que, para um conjunto de n observações independentes Z_1, \dots, Z_n , cada uma com variância σ^2 , a variância da média \bar{Z} das observações é dada por σ^2/n . Ou seja, a média de várias observações reduz a variância.

Inspirado por esse princípio, uma maneira natural de reduzir a variância e aumentar a acurácia de teste é gerar muitos subconjuntos de treinamento, ajustar um modelo preditivo em cada um e então calcular a média das previsões. Se denotarmos as previsões como $\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$, a previsão agregada é:

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x).$$

Na prática, não temos acesso a múltiplos conjuntos de treinamento independentes. Em vez disso, usamos o *bootstrap*, gerando B conjuntos de treinamento *bootstrap* amostrados com reposição do conjunto original. Ajustamos o modelo em cada conjunto para obter $\hat{f}^{*b}(x)$ e, finalmente, calculamos a média:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x).$$

Esse procedimento é chamado de *bagging*.

Ao aplicar o bagging em árvores de regressão, construímos B árvores em conjuntos de treinamento bootstrap e combinamos as previsões. Essas árvores são crescidas profundamente e não são podadas. Portanto, cada árvore individual tem **alta variância e baixo viés**. Ao combinar as árvores, a variância é reduzida substancialmente, enquanto o viés permanece baixo. Assim:

O bagging reduz a variância sem aumentar significativamente o viés.

Para problemas de classificação, a extensão é direta. Para cada observação de teste, registramos a classe predita por cada uma das B árvores e usamos o voto majoritário para determinar a classe final.

Além da intuição básica, podemos obter uma fórmula explícita para a variância da predição agregada. Suponha que cada árvore $\hat{f}^b(x)$ tenha variância σ^2 , e que a correlação entre as previsões de duas árvores quaisquer seja ρ . Então, a variância da média das árvores é:

$$\text{Var}(\hat{f}_{\text{bag}}(x)) = \rho\sigma^2 + \frac{(1 - \rho)\sigma^2}{B}.$$

Para valores grandes de B , o segundo termo se aproxima de zero e a variância total se reduz a $\rho\sigma^2$.

Isso mostra que o bagging reduz a variância, mas a redução depende da correlação ρ entre as árvores. Quanto menor a correlação, maior será a redução de variância ao aplicar o bagging.

Essa ideia não é restrita a árvores de decisão. Em princípio, qualquer modelo com alta variância pode se beneficiar do bagging. No entanto, árvores de decisão são especialmente adequadas porque têm *baixo viés e alta variância*, características que tornam possível reduzir a variância sem sacrificar a acurácia. Além disso, no contexto de florestas aleatórias e bagging de árvores, costuma-se crescer as árvores até a pureza máxima dos nós. Isso aumenta ainda mais a variância individual das árvores e, consequentemente, o potencial de redução de variância ao aplicar o bagging.

6.2.1 Estimativa do Erro Out-of-Bag (OOB)

Existe uma maneira bastante simples de estimar o erro de teste de um modelo bagged, sem precisar realizar validação cruzada ou utilizar um conjunto de validação separado. O ponto chave do bagging é que cada árvore é ajustada usando subconjuntos bootstrap das observações originais. Em média, cada árvore usa aproximadamente dois terços das observações para treinamento. O restante, cerca de um terço, não é utilizado naquela árvore específica e essas observações são chamadas de *out-of-bag* (OOB).

Essa proporção de aproximadamente um terço pode ser calculada da seguinte forma. Cada árvore do bagging é ajustada com um conjunto bootstrap, onde n observações são amostradas com reposição a partir das n observações originais. Para uma observação específica i , a probabilidade de ela não ser escolhida em uma única amostragem é

$$1 - \frac{1}{n}.$$

Como n amostras são feitas para formar o conjunto bootstrap, a probabilidade de que i nunca seja selecionada é

$$\left(1 - \frac{1}{n}\right)^n.$$

Quando n é grande, podemos usar a aproximação

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} \approx 0,368.$$

Portanto, cerca de 36,8% das observações não são usadas no ajuste de uma árvore específica e ficam como OOB para essa árvore.

Para estimar o erro OOB, consideramos a predição para a i -ésima observação utilizando apenas as árvores nas quais essa observação foi OOB, ou seja, não foi usada no treinamento daquela árvore. Em média, cada observação será OOB para aproximadamente $0,368 \times B$ árvores, onde B é o número total de árvores do ensemble. Para cada observação, podemos então calcular a predição média (no caso de regressão) ou aplicar uma votação majoritária (no caso de classificação), usando apenas essas árvores.

Esse processo leva a uma predição OOB única para cada observação. Repetindo para todas as n observações do conjunto de dados, podemos calcular o erro quadrático médio OOB (para regressão) ou a taxa de erro OOB (para classificação).

O erro OOB fornece uma estimativa válida do erro de teste do modelo bagged, pois as previsões são feitas apenas com árvores que não usaram a observação correspondente em seu treinamento. Quando B é suficientemente grande, o erro OOB torna-se praticamente equivalente ao erro estimado por validação cruzada *leave-one-out*. Isso torna o OOB uma alternativa conveniente e computacionalmente eficiente para estimar o erro de teste, especialmente em grandes conjuntos de dados onde a validação cruzada seria onerosa.

Algorithm 2 Cálculo do erro Out-of-Bag (OOB)

```

1: Dado um conjunto de treinamento com  $n$  observações e um número de árvores  $B$ 
2: for  $b = 1$  até  $B$  do
3:   Sorteie um conjunto bootstrap com  $n$  observações (com reposição)
4:   Ajuste uma árvore  $\hat{f}^{*b}$  usando o conjunto bootstrap
5:   Registre quais observações não foram usadas na árvore  $b$  (estas serão OOB para a árvore
    $b$ )
6: end for
7: for cada observação  $i = 1$  até  $n$  do
8:   Identifique todas as árvores onde  $i$  foi OOB
9:   if há pelo menos uma árvore OOB para  $i$  then
10:    Preveja  $\hat{y}_i^{OOB}$  usando as árvores onde  $i$  foi OOB
11:   end if
12: end for
13: Compare  $\hat{y}_i^{OOB}$  com  $y_i$  verdadeiro e calcule o erro médio

```

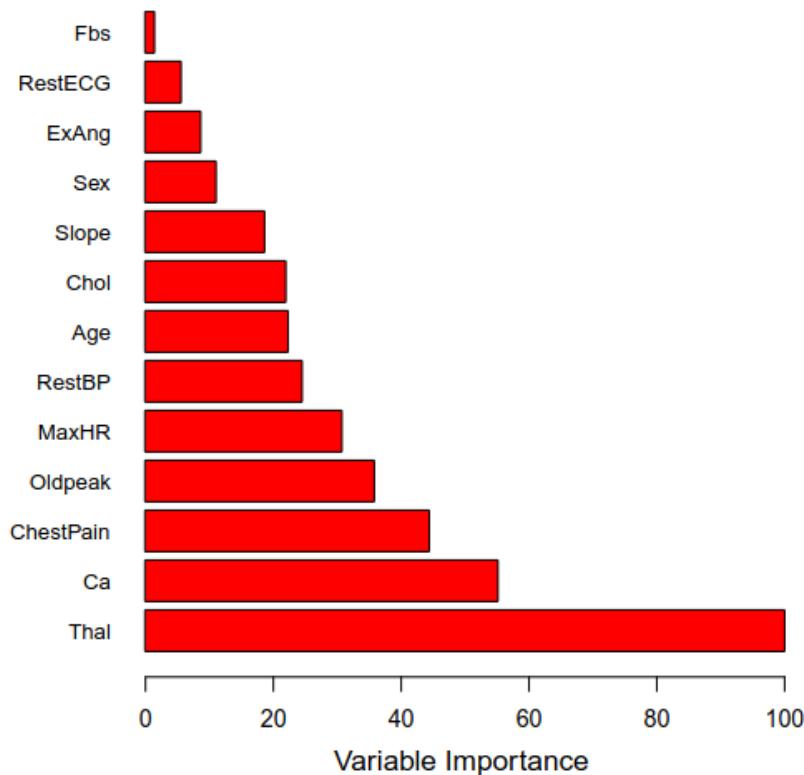
6.2.2 Importância de Variáveis em Modelos Bagged

Como discutido anteriormente, o uso do bagging normalmente resulta em melhoria na acurácia de predição quando comparado ao uso de uma única árvore. No entanto, essa melhoria geralmente ocorre às custas da interpretabilidade do modelo. Uma das vantagens fundamentais das árvores de decisão é sua fácil visualização e a possibilidade de identificar diretamente quais variáveis são mais importantes. Quando combinamos muitas árvores — como no bagging — torna-se impraticável representar visualmente o modelo final ou identificar facilmente as variáveis mais relevantes.

Apesar dessa limitação, é possível obter uma medida global da importância das variáveis em ensembles. Para árvores de regressão, calcula-se a redução total no RSS (soma dos quadrados dos resíduos) atribuída a divisões feitas por cada variável, somando essa quantidade sobre todas as B árvores do ensemble. No caso de árvores de classificação, soma-se a redução no índice de Gini associada a cortes que usam cada variável. Variáveis que resultam em maiores reduções acumuladas são consideradas mais importantes no processo de decisão.

6.3 Random Forests

As *random forests* representam uma extensão do bagging que busca reduzir a correlação entre as árvores, um problema que ocorre naturalmente quando todas as árvores podem escolher livremente entre todas as variáveis disponíveis. No bagging puro, as árvores tendem a usar repetidamente as variáveis mais fortes, levando a árvores altamente correlacionadas e, consequentemente, limitando a redução de variância do ensemble. Para mitigar esse problema, as random forests introduzem uma modificação simples, porém eficaz: a cada divisão de uma árvore, um subconjunto aleatório de m variáveis é selecionado entre as p variáveis disponíveis, e apenas essas podem ser usadas no corte.



Essa estratégia força diferentes árvores a considerar diferentes variáveis em suas divisões iniciais, promovendo diversidade entre as árvores e reduzindo a correlação entre elas. Normalmente, o número de variáveis candidatas m é escolhido como $m \approx \sqrt{p}$, embora outros valores possam ser usados dependendo do problema.

A principal diferença entre bagging e random forests reside justamente nessa escolha do tamanho do subconjunto de variáveis em cada divisão. Quando $m = p$, a random forest se reduz ao bagging. Para valores menores de m , as árvores são forçadas a explorar combinações diferentes de variáveis, o que aumenta a diversidade do ensemble e reduz a variância da predição final. Contudo, se m for muito pequeno, isso pode levar a um aumento do viés, já que as árvores podem deixar de considerar variáveis relevantes em divisões cruciais.

6.3.1 Impacto dos Parâmetros B e m

O número de árvores B no ensemble controla a quantidade de modelos combinados. Aumentar B tende a reduzir a variância da predição final, sem aumentar o viés. Já o parâmetro m , que determina o número de variáveis candidatas em cada divisão, regula a decorrelação entre as árvores. Valores menores de m incentivam uma maior diversidade entre as árvores, promovendo redução da variância, especialmente em conjuntos de dados com variáveis altamente correlacionadas. No entanto, uma escolha de m muito pequena pode elevar o viés do modelo.

6.3.2 Uso de Random Forests em Altas Dimensões

O uso de random forests é especialmente benéfico em situações de alta dimensionalidade, isto é, quando o número de preditores p é grande em relação ao número de observações n . Nesses cenários, é comum que muitas variáveis sejam irrelevantes ou altamente correlacionadas. A estratégia de seleção aleatória de variáveis em cada divisão ajuda a evitar que todas as árvores concentrem suas divisões nas mesmas variáveis dominantes, forçando o modelo a considerar uma gama mais ampla de preditores. Isso aumenta a diversidade entre as árvores e melhora a capacidade de generalização do ensemble.

Além disso, random forests lidam bem com problemas onde $p > n$, como em dados genômicos, imagens ou texto, e conseguem tolerar a presença de muitas variáveis ruidosas ou não informativas. Outra vantagem prática é que o método geralmente não requer uma etapa prévia de seleção de variáveis, pois o processo de amostragem aleatória e a própria métrica de importância de variáveis ajudam a mitigar o impacto de preditores irrelevantes.

Exercício 13. Entender os códigos [aqui](#).

Capítulo 7

Seleção de modelos lineares e regularização

No contexto de regressão, o modelo linear padrão é dado por:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon,$$

onde Y é a variável resposta, X_1, \dots, X_p são as variáveis preditoras, e ϵ representa o erro aleatório.

Esse modelo é normalmente ajustado via mínimos quadrados. Apesar de sua simplicidade, o modelo linear apresenta vantagens importantes, especialmente em termos de inferência e interpretabilidade. Em muitos problemas reais, ele é surpreendentemente competitivo mesmo quando comparado a modelos não-lineares mais complexos.

Existem dois principais motivos para substituir o ajuste por mínimos quadrados por outros procedimentos:

- **Performance Preditiva.** Quando a relação entre resposta e preditores é aproximadamente linear, o ajuste por mínimos quadrados resulta em estimadores com viés baixo. Se o número de observações n for muito maior do que o número de variáveis p , ou seja, $n \gg p$, os estimadores também apresentam baixa variância, o que tende a resultar em boa performance em dados de teste.

Entretanto, se n não for muito maior do que p , a variância dos coeficientes aumenta significativamente, levando a overfitting e má generalização. No caso em que $p > n$, não existe solução única para os coeficientes que minimizam os erros quadráticos: há infinitas soluções possíveis.

Cada uma dessas soluções resulta em erro zero nos dados de treino, mas em geral apresentam alto erro nos dados de teste devido à alta variância. Ao *restringir* ou *encolher* os coeficientes estimados, conseguimos reduzir substancialmente a variância, com aumento de viés geralmente desprezível — o que melhora a performance preditiva em dados não vistos.

- **Interpretabilidade do Modelo.** É comum que, em uma regressão com múltiplos preditores, algumas variáveis não estejam realmente associadas à resposta. Incluir essas variáveis *irrelevantes* aumenta a complexidade do modelo e dificulta sua interpretação.

Ao remover essas variáveis — por exemplo, forçando os coeficientes correspondentes a serem zero — obtemos um modelo mais simples e fácil de interpretar. Note que o

Existem muitas alternativas, clássicas e modernas, ao uso de mínimos quadrados para ajustar o modelo linear. Neste capítulo, são discutidas duas alternativas:

- **Seleção de Subconjuntos (Subset Selection):** envolve identificar um subconjunto das p variáveis que acreditamos estarem associadas à resposta. Ajusta-se o modelo linear por mínimos quadrados usando apenas essas variáveis selecionadas.
- **Encolhimento (Shrinkage):** ajusta-se um modelo utilizando todas as p variáveis, mas os coeficientes são *encolhidos* em direção a zero. Esse encolhimento, também chamado de *regularização*, reduz a variância do modelo. Em alguns casos, os coeficientes podem até ser forçados a zero, o que permite realizar seleção de variáveis implicitamente.

7.1 Seleção do melhor subconjunto (Best Subset Selection)

A seleção do melhor subconjunto consiste em ajustar modelos de regressão para todas as combinações possíveis de subconjuntos das p variáveis preditoras. Por exemplo, ajustamos todos os modelos com uma variável, depois com duas, e assim por diante até o modelo com todas as p variáveis. Para cada tamanho de subconjunto k , comparamos todos os modelos de tamanho k e selecionamos aquele com melhor desempenho nos dados de treinamento, geralmente o que possui menor erro residual quadrático (RSS). Em seguida, entre os modelos de tamanhos diferentes, escolhemos aquele que apresenta melhor desempenho preditivo, avaliando-o em dados de validação ou usando critérios como C_p , AIC, BIC ou R^2 ajustado. O procedimento pode ser descrito da seguinte forma:

Algorithm 3 Best Subset Selection

- 1: Defina \mathcal{M}_0 como o modelo nulo, sem preditores, que prediz a média da amostra.
 - 2: **for** $k = 1, 2, \dots, p$ **do**
 - 3: Ajuste todos os $\binom{p}{k}$ modelos com exatamente k preditores.
 - 4: Escolha o melhor modelo \mathcal{M}_k dentre esses, com base no menor Erro Quadrático ou maior R^2 .
 - 5: **end for**
 - 6: Escolha o melhor modelo dentre $\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_p$ com base em:
 - erro em um conjunto de validação;
 - ou validação cruzada.
-

Uma dificuldade importante desse método está no fato de que RSS e R^2 tendem a melhorar conforme adicionamos mais variáveis, o que favorece modelos mais complexos e com maior risco de sobreajuste. Por isso, é essencial usar critérios que penalizam a complexidade ou avaliar diretamente o erro de teste, via validação cruzada ou conjunto de validação separado.

Apesar de sua simplicidade conceitual, a seleção do melhor subconjunto enfrenta sérias limitações computacionais. O número de modelos possíveis cresce exponencialmente com p , já

que existem 2^p subconjuntos. Por exemplo, com $p = 10$, há cerca de mil modelos possíveis; com $p = 20$, mais de um milhão; e com $p = 40$, torna-se praticamente inviável testar todas as possibilidades, mesmo com computadores modernos.

Vale destacar que o procedimento pode ser estendido para modelos não paramétricos. A seleção dos modelos pode ser guiada por medidas como o erro de validação ou critérios baseados em perda preditiva empírica, calculados em conjuntos de validação ou via validação cruzada. O desempenho de cada subconjunto de variáveis pode ser avaliado com base na capacidade preditiva do modelo ajustado de forma não paramétrica, como estimadores por k -vizinhos mais próximos, árvores de decisão, ou métodos kernel.

7.1.1 Seleção Foward e Backward

Por razões computacionais, a seleção do melhor subconjunto torna-se inviável para valores grandes de p . Além disso, um espaço de busca muito amplo pode levar à seleção de modelos com ótimo desempenho nos dados de treino, mas que não generalizam bem. Isso ocorre porque, quanto mais modelos avaliamos, maior a chance de encontrar um que se ajusta bem ao acaso. Assim, métodos que restringem o espaço de busca, como a *seleção stepwise*, oferecem uma alternativa prática e eficiente.

A **seleção stepwise** explora um número bem menor de modelos do que a busca exaustiva. Um exemplo comum é a **seleção stepwise para frente (forward stepwise selection)**. Nesse procedimento, começamos com o modelo nulo, que não contém nenhuma variável preditora. Em cada etapa, adicionamos à equação a variável que proporciona a maior melhora no ajuste, medida por critérios como R^2 ou RSS. Esse processo se repete até que todas as variáveis estejam no modelo, ou até um critério de parada ser atingido.

A seguir, descrevemos formalmente o procedimento:

Algorithm 4 Forward Stepwise Selection

- 1: Comece com o modelo nulo \mathcal{M}_0
 - 2: **for** $k = 0, 1, \dots, p - 1$ **do**
 - 3: Considere os $p - k$ modelos que adicionam uma nova variável ao modelo \mathcal{M}_k
 - 4: Escolha o modelo \mathcal{M}_{k+1} que oferece a maior melhoria no ajuste (menor RSS ou maior R^2)
 - 5: **end for**
 - 6: Ao final, escolha o melhor modelo entre $\mathcal{M}_0, \dots, \mathcal{M}_p$ com base em erro de validação, AIC, BIC, R^2 ajustado ou validação cruzada
-

Comparado à seleção exaustiva, que requer ajustar 2^p modelos, a seleção forward requer bem menos ajustes. Especificamente, no passo k , consideramos $p - k$ modelos, resultando em um total de:

$$1 + \sum_{k=0}^{p-1} (p - k) = 1 + \frac{p(p + 1)}{2}$$

modelos avaliados. Por exemplo, com $p = 20$, a seleção exaustiva requer mais de um milhão de ajustes, enquanto a stepwise requer apenas 211.

Apesar da grande vantagem computacional, a seleção stepwise não garante encontrar o melhor modelo possível. Isso ocorre porque decisões tomadas em etapas anteriores afetam as opções disponíveis nas próximas. Por exemplo, suponha que o melhor modelo com uma variável inclua X_1 , mas o melhor modelo com duas variáveis inclua X_2 e X_3 . Como o método já adicionou X_1 na primeira etapa, o modelo com duas variáveis será forçado a incluir X_1 , podendo assim descartar a melhor combinação possível de duas variáveis. Portanto, o método pode falhar em encontrar o melhor subconjunto global de variáveis.

Ainda assim, na prática, a seleção stepwise costuma apresentar bom desempenho e é uma alternativa útil quando o número de variáveis é grande demais para permitir uma busca exaustiva.

Assim como a seleção stepwise para frente, a *seleção stepwise para trás* (*backward stepwise selection*) é uma alternativa eficiente à seleção do melhor subconjunto. No entanto, ao contrário do método forward, que começa com nenhum preditor e adiciona variáveis uma a uma, o método backward começa com o modelo completo (contendo todas as p variáveis) e vai removendo iterativamente as variáveis menos úteis, uma por vez.

A cada etapa, entre todos os modelos que removem uma variável do modelo atual, escolhe-se aquele com melhor ajuste — tipicamente o de menor erro residual quadrático (RSS) ou maior R^2 . O processo continua até restar apenas o modelo nulo, que não inclui nenhum preditor. O algoritmo é descrito a seguir:

Algorithm 5 Backward Stepwise Selection

- 1: Defina \mathcal{M}_p como o modelo completo, contendo todos os p preditores
 - 2: **for** $k = p, p - 1, \dots, 1$ **do**
 - 3: Considere os k modelos que removem uma das variáveis de \mathcal{M}_k , resultando em modelos com $k - 1$ variáveis
 - 4: Escolha o melhor entre esses modelos e denote-o por \mathcal{M}_{k-1}
 - 5: **end for**
 - 6: Escolha o melhor modelo final entre $\mathcal{M}_0, \dots, \mathcal{M}_p$, com base em erro de validação, AIC, BIC, R^2 ajustado ou validação cruzada
-

Assim como o método forward, a seleção backward avalia um número total de modelos igual a $1 + \frac{p(p+1)}{2}$, o que é muito menor que os 2^p modelos da seleção exaustiva. Isso permite aplicar o método backward mesmo quando p é grande demais para a busca completa.

Uma limitação importante, porém, é que a seleção backward só pode ser usada quando o número de observações n é maior do que o número de variáveis p , pois o modelo inicial completo precisa ser ajustável. Já o método forward pode ser aplicado mesmo quando $p > n$, sendo útil em contextos de alta dimensionalidade.

É importante lembrar que, assim como o forward, o backward também não garante encontrar o melhor subconjunto global de variáveis. O caminho de remoções escolhidas influencia o resultado final, e o método pode ignorar subconjuntos que oferecem ajuste superior por não estarem em sua trajetória de busca.

Exemplo: Seleção de Variáveis com Modelos Não Paramétricos no Python

A seguir, mostramos como aplicar a seleção sequencial de variáveis usando um classificador não paramétrico, o *k*-Nearest Neighbors (*k*NN), com o dataset clássico *iris*. Usamos a classe `SequentialFeatureSelector` da biblioteca `scikit-learn`:

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
```

Primeiro, carregamos os dados:

```
X, y = load_iris(return_X_y=True)
```

Aqui, *X* representa as variáveis preditoras e *y* representa a variável de resposta (as espécies das flores). Em seguida, definimos o estimador *k*-NN com 3 vizinhos:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

Criamos o seletor sequencial, especificando que queremos selecionar 3 variáveis:

```
sfs = SequentialFeatureSelector(knn, n_features_to_select=3)
```

Ajustamos o seletor aos dados:

```
sfs.fit(X, y)
```

Após o ajuste, podemos verificar quais variáveis foram selecionadas com:

```
sfs.get_support()
# Saída: array([ True, False,  True,  True])
```

A saída indica que as colunas 0, 2 e 3 foram selecionadas (valores `True`), enquanto a coluna 1 foi descartada. Podemos transformar o conjunto original *X* para conter apenas as colunas selecionadas:

```
sfs.transform(X).shape
# Saída: (150, 3)
```

O novo conjunto de dados contém 150 amostras e apenas 3 variáveis, conforme especificado. Este exemplo mostra como técnicas de seleção de variáveis podem ser usadas com modelos não paramétricos de forma prática e eficiente, utilizando validação interna para guiar a escolha das melhores combinações de preditores.

Comparação de Modelos com Seleção de Variáveis em Pipeline

A seguir, descrevemos um procedimento completo para comparar três modelos de classificação — Regressão Logística, k-Nearest Neighbors (kNN) e Floresta Aleatória — utilizando uma etapa de seleção de variáveis incorporada ao pipeline. A comparação é feita com base no desempenho de validação, e o melhor modelo é então avaliado no conjunto de teste.

1. Importação das bibliotecas e carregamento dos dados.

Utilizamos o conjunto de dados `iris`, que já está disponível no `scikit-learn`, e dividimos em três partes: treino, validação e teste.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import numpy as np

X, y = load_iris(return_X_y=True)

# 60% treino, 20% validação, 20% teste
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=0)
```

2. Definição dos modelos a serem comparados.

Criamos um dicionário com os três classificadores que desejamos comparar. Eles serão usados dentro dos pipelines com seleção de variáveis.

```
models = {
    "Regressão Logística": LogisticRegression(max_iter=1000),
    "kNN": KNeighborsClassifier(n_neighbors=3),
    "Floresta Aleatória": RandomForestClassifier(random_state=0)
}
```

3. Criação dos pipelines com seleção sequencial de variáveis.

Para cada modelo, criamos um Pipeline que inclui primeiro a seleção de 3 variáveis com `SequentialFeatureSelector`, e depois o classificador.

```

pipelines = [
    name: Pipeline([
        ('select', SequentialFeatureSelector(model, n_features_to_select=3)),
        ('clf', model)
    ])
    for name, model in models.items()
]

```

4. Treinamento e avaliação de cada pipeline no conjunto de validação.

Cada pipeline é ajustado usando apenas os dados de treino. Em seguida, fazemos previsões no conjunto de validação e calculamos a acurácia.

```

for name, pipe in pipelines.items():
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_val)
    acc = accuracy_score(y_val, y_pred)
    print(f"{name}: Acurácia na validação = {acc:.3f}")

```

5. Escolha do melhor modelo e avaliação no conjunto de teste.

Após identificar o modelo com melhor desempenho na validação, reagrupamos os dados de treino e validação para refazer o ajuste final. Em seguida, avaliamos o desempenho final no conjunto de teste.

```

# Exemplo: suponha que a Regressão Logística teve melhor desempenho
best_pipe = pipelines["Regressão Logística"]

# Avaliamos no teste
y_test_pred = best_pipe.predict(X_test)
print("Acurácia final no teste:", accuracy_score(y_test, y_test_pred))

```

Esse processo garante uma comparação justa entre modelos, pois a seleção de variáveis é feita internamente em cada pipeline, evitando o uso indevido de dados de validação. Além disso, ao separar os conjuntos de dados, conseguimos avaliar corretamente a capacidade de generalização dos modelos.

7.2 Ridge e Lasso

7.2.1 Regressão Ridge (Ridge Regression)

A regressão ridge é uma técnica de regularização que busca ajustar todos os p preditores, mas penalizando coeficientes grandes, o que ajuda a reduzir a variância do modelo.

Lembre que, na regressão linear comum, os coeficientes $\beta = (\beta_1, \dots, \beta_p)$ são obtidos minimizando a soma dos erros quadráticos (RSS):

$$\text{RSS} = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

Na regressão ridge, em vez disso, minimizamos a seguinte função de custo:

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2,$$

onde $\lambda \geq 0$ é o parâmetro de regularização, que controla a força da penalização. O segundo termo, chamado *penalidade de encolhimento (shrinkage penalty)*, impõe uma penalização a coeficientes grandes e força os valores de β_j a se aproximarem de zero.

Em notação matricial, assumindo que os dados foram centrados (isto é, sem intercepto β_0), a função a ser minimizada pode ser escrita como:

$$\hat{\beta}^{\text{Ridge}} = \arg \min_{\beta} \{ \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \},$$

onde:

- $X \in \mathbb{R}^{n \times p}$ é a matriz de design;
- $Y \in \mathbb{R}^n$ é o vetor resposta;
- $\beta \in \mathbb{R}^p$ é o vetor de coeficientes;
- $\|\cdot\|_2$ denota a norma Euclidiana (ou ℓ_2 -norma).

Essa formulação mostra claramente que o objetivo é encontrar um vetor de coeficientes que, além de se ajustar bem aos dados (minimizando o erro), tenha norma pequena, evitando coeficientes muito grandes.

O parâmetro λ precisa ser ajustado separadamente, geralmente via validação cruzada. Quando $\lambda = 0$, obtemos o modelo de mínimos quadrados ordinários (OLS). À medida que $\lambda \rightarrow \infty$, todos os coeficientes tendem a zero.

A solução do problema de Ridge é dada minimizando:

$$J(\beta) = \|Y - X\beta\|^2 + \lambda \|\beta\|^2$$

Derivando e igualando a zero:

$$\nabla J = -2X^\top Y + 2(X^\top X + \lambda I)\beta = 0 \Rightarrow \hat{\beta} = (X^\top X + \lambda I)^{-1} X^\top Y$$

A matriz $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$ é sempre invertível para $\lambda > 0$, mesmo que $\mathbf{X}^\top \mathbf{X}$ seja singular. De fato, seja $\mathbf{A} = \mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$, com $\lambda > 0$. Queremos mostrar que \mathbf{A} é invertível e para isso, vamos mostrar que \mathbf{A} é definida positiva.

Para todo $\mathbf{v} \neq 0$:

$$\mathbf{v}^\top \mathbf{A} \mathbf{v} = \mathbf{v}^\top \mathbf{X}^\top \mathbf{X} \mathbf{v} + \lambda \mathbf{v}^\top \mathbf{v} = \|\mathbf{X} \mathbf{v}\|^2 + \lambda \|\mathbf{v}\|^2$$

Como $\lambda > 0$ e $\|\mathbf{v}\|^2 > 0$, então $\mathbf{v}^\top \mathbf{A} \mathbf{v} > 0$. Logo, \mathbf{A} é definida positiva e, portanto, invertível.

7.2.2 Regressão Lasso

A regressão Lasso (*Least Absolute Shrinkage and Selection Operator*) é uma alternativa à regressão ridge que realiza regularização com norma ℓ_1 . Ela busca minimizar:

$$\hat{\boldsymbol{\beta}}^{\text{Lasso}} = \arg \min_{\boldsymbol{\beta}} \{ \|\mathbf{Y} - \mathbf{X} \boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 \}, \quad \text{onde } \|\boldsymbol{\beta}\|_1 = \sum_{j=1}^p |\beta_j|$$

Comparando com ridge, a diferença é que a penalização com norma ℓ_2 (quadrado dos coeficientes) foi substituída pela norma ℓ_1 . Essa troca faz com que o Lasso produza soluções esparsas — ou seja, com muitos coeficientes exatamente iguais a zero — promovendo assim **seleção de variáveis** automaticamente.

Ao contrário do ridge, o Lasso não possui solução fechada. Isso ocorre porque a função de custo não é diferenciável em pontos onde $\beta_j = 0$ (devido ao valor absoluto). Portanto, a solução é obtida por métodos numéricos como:

- **Coordinate descent (descida por coordenadas)**: método iterativo que otimiza um coeficiente por vez, mantendo os outros fixos.
- **Least Angle Regression (LARS)**: técnica eficiente que gera o caminho completo de soluções à medida que λ varia.

Enquanto ridge encolhe todos os coeficientes, mas não os zera, o Lasso força alguns coeficientes a serem exatamente zero quando λ é suficientemente grande. Isso facilita a interpretação dos modelos, além de funcionar como um mecanismo de seleção de variáveis.

7.2.3 Formulações Alternativas de Ridge e Lasso

As formulações de Ridge e Lasso com penalização λ também podem ser escritas como problemas com restrições equivalentes. No caso do Lasso, podemos reescrevê-lo como:

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X} \boldsymbol{\beta}\|^2 \quad \text{sujeito a } \sum_{j=1}^p |\beta_j| \leq s,$$

e, no caso do Ridge,

$$\min_{\beta} \|\mathbf{Y} - \mathbf{X}\beta\|^2 \quad \text{sujeito a } \sum_{j=1}^p \beta_j^2 \leq s.$$

Para cada valor de $\lambda > 0$, existe um valor de $s > 0$ tal que as soluções das versões penalizadas e das versões com restrição são equivalentes.

Essas formas ajudam a entender o comportamento geométrico das soluções. No caso em que $p = 2$, a restrição do Lasso define um losango (por ser uma bola da norma ℓ_1), enquanto a do Ridge define um círculo (bola da norma ℓ_2). Como os cantos do losango coincidem com os eixos coordenados, é comum que a solução ótima ocorra exatamente em um desses cantos — ou seja, com algum $\beta_j = 0$. Isso explica por que o Lasso promove esparsidade e realiza implicitamente seleção de variáveis. Já a forma circular do Ridge penaliza todos os coeficientes de forma suave e simétrica, o que leva a coeficientes pequenos, mas raramente nulos.

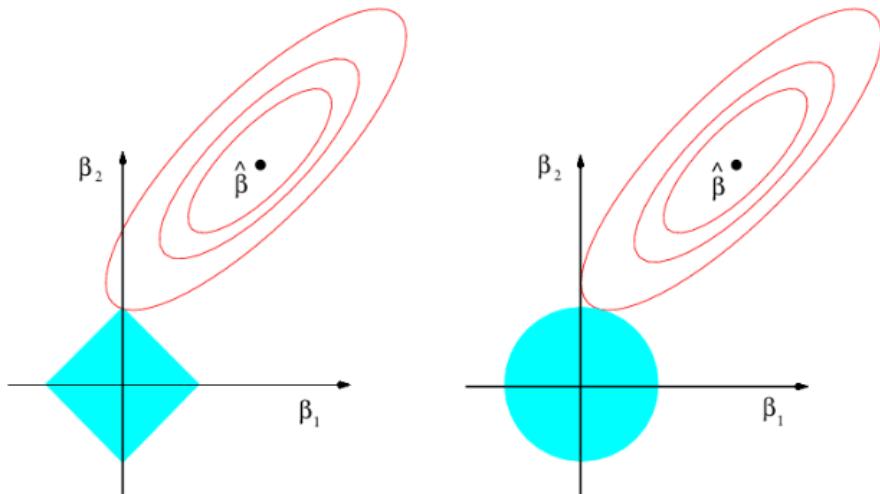


FIGURE 6.7. Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.

Essas formulações também nos permitem interpretar o Lasso como uma aproximação convexa ao problema de seleção de subconjunto. Esse problema pode ser formulado como:

$$\min_{\beta} \|\mathbf{Y} - \mathbf{X}\beta\|^2 \quad \text{sujeito a } \sum_{j=1}^p I(\beta_j \neq 0) \leq s,$$

onde $I(\cdot)$ é a função indicadora. Essa abordagem procura o menor erro possível utilizando no máximo s variáveis, mas é um problema combinatório e computacionalmente inviável para grandes p . O Lasso substitui essa restrição discreta por uma relaxação contínua via a norma ℓ_1 , tornando o problema convexamente tratável e eficiente de resolver com métodos numéricos.

Para entender essa equivalência, vamos começar com a formulação com restrição do Lasso:

$$\min_{\beta} \|\mathbf{Y} - \mathbf{X}\beta\|^2 \quad \text{sujeito a } \|\beta\|_1 \leq s$$

Formamos o Lagrangiano com multiplicador $\lambda \geq 0$:

$$\mathcal{L}(\boldsymbol{\beta}, \lambda) = \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda (\|\boldsymbol{\beta}\|_1 - s)$$

Como s é constante, minimizar \mathcal{L} em relação a $\boldsymbol{\beta}$ equivale a minimizar:

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1$$

Ou seja, obtemos a formulação penalizada do Lasso. O parâmetro λ atua como multiplicador de Lagrange: para cada valor de s , existe um $\lambda \geq 0$ tal que ambas as formulações têm a mesma solução ótima.

7.2.4 Interpretação via encolhimento

Para entender melhor o comportamento do Ridge e do Lasso, consideraremos um caso simples: $n = p$, a matriz de design \mathbf{X} é a identidade I , e não há intercepto. Nesse cenário, o problema de mínimos quadrados se torna:

$$\min_{\beta_1, \dots, \beta_p} \sum_{j=1}^p (y_j - \beta_j)^2,$$

cuja solução é simplesmente $\hat{\beta}_j = y_j$.

Aplicando a penalização do Ridge, o problema se torna:

$$\min_{\beta_1, \dots, \beta_p} \sum_{j=1}^p (y_j - \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2,$$

cuja solução analítica é:

$$\hat{\beta}_j^{\text{Ridge}} = \frac{y_j}{1 + \lambda}.$$

Ou seja, cada coeficiente é encolhido proporcionalmente em direção a zero, pela mesma razão.

Para o Lasso, temos:

$$\min_{\beta_1, \dots, \beta_p} \sum_{j=1}^p (y_j - \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|,$$

cuja solução é dada por *soft-thresholding*:

$$\hat{\beta}_j^{\text{Lasso}} = \begin{cases} y_j - \lambda/2 & \text{se } y_j > \lambda/2 \\ y_j + \lambda/2 & \text{se } y_j < -\lambda/2 \\ 0 & \text{se } |y_j| \leq \lambda/2 \end{cases}$$

Dessa forma, enquanto o Ridge encolhe todos os coeficientes pela mesma proporção, o Lasso aplica uma redução constante $\lambda/2$, podendo zerar coeficientes com valores absolutos pequenos. Esse mecanismo explica por que o Lasso realiza seleção de variáveis automaticamente, enquanto o Ridge não zera coeficientes, apenas os diminui.

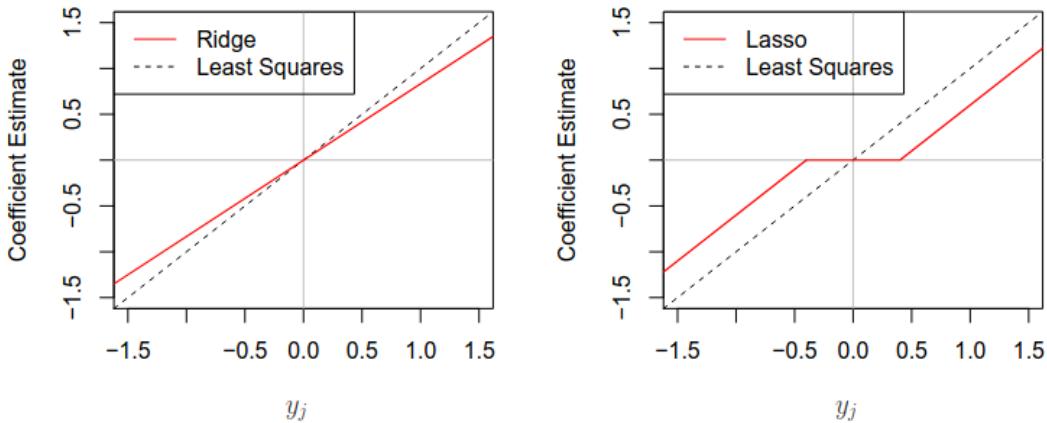


FIGURE 6.10. The ridge regression and lasso coefficient estimates for a simple setting with $n = p$ and \mathbf{X} a diagonal matrix with 1's on the diagonal. Left: The ridge regression coefficient estimates are shrunken proportionally towards zero, relative to the least squares estimates. Right: The lasso coefficient estimates are soft-thresholded towards zero.

Embora o cenário geral (com X não diagonal) seja mais complexo, a intuição permanece válida: o Ridge suaviza todos os coeficientes, enquanto o Lasso favorece soluções esparsas.

Exercício 14. Encontre o análogo para o caso do best-subset selection.

7.2.5 Interpretação Bayesiana de Ridge e Lasso

A regressão Ridge e o Lasso podem ser vistos como estimativas de *máxima a posteriori* (MAP) em um modelo Bayesiano. Assumimos o modelo linear:

$$Y = \mathbf{X}\beta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I),$$

onde a verossimilhança é dada por:

$$f(Y | \mathbf{X}, \beta) \propto \exp\left(-\frac{1}{2\sigma^2} \|Y - \mathbf{X}\beta\|^2\right).$$

Perceba que quando não impomos nenhum prior sobre β , a estimativa de **máxima verossimilhança** (MLE) é:

$$\hat{\beta}_{\text{MLE}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top Y,$$

que coincide com a solução de mínimos quadrados ordinários.

Agora, se assumirmos um prior sobre os coeficientes β , obtemos:

$$p(\beta | Y, \mathbf{X}) \propto f(Y | \mathbf{X}, \beta) \cdot p(\beta).$$

A solução MAP é então:

$$\hat{\beta}_{\text{MAP}} = \arg \max_{\beta} \log f(Y | \mathbf{X}, \beta) + \log p(\beta).$$

Duas escolhas clássicas para o prior levam às penalizações de Ridge e Lasso:

- **Ridge:** prior Gaussiano i.i.d., $\beta_j \sim \mathcal{N}(0, \tau^2)$. Isso implica:

$$\log p(\boldsymbol{\beta}) \propto -\frac{1}{2\tau^2} \|\boldsymbol{\beta}\|_2^2,$$

e a solução MAP se torna:

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \|Y - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_2^2, \quad \text{com } \lambda = \frac{\sigma^2}{\tau^2}.$$

- **Lasso:** prior Laplace (double-exponential), i.i.d., $\beta_j \sim \text{Laplace}(0, b)$. Isso implica:

$$\log p(\boldsymbol{\beta}) \propto -\frac{1}{b} \|\boldsymbol{\beta}\|_1,$$

e a solução MAP é:

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \|Y - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1, \quad \text{com } \lambda = \frac{\sigma^2}{b}.$$

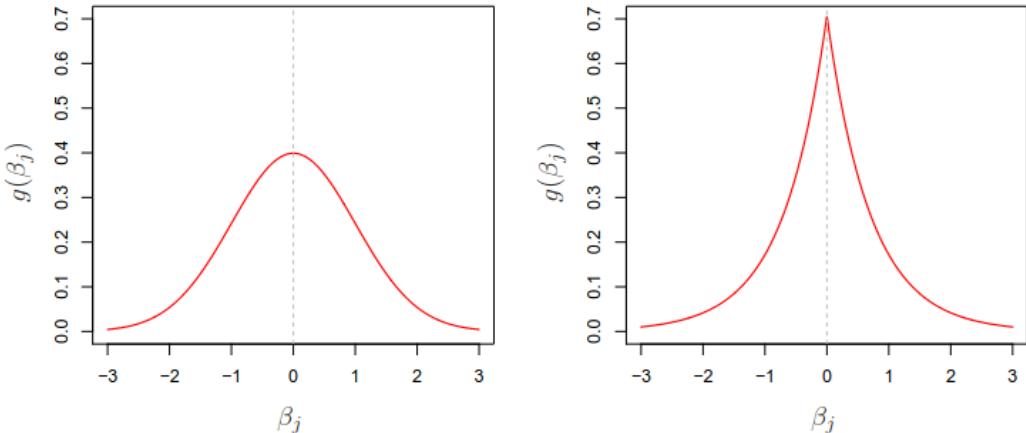


FIGURE 6.11. Left: *Ridge regression is the posterior mode for β under a Gaussian prior.* Right: *The lasso is the posterior mode for β under a double-exponential prior.*

Ambos os casos assumem erros Gaussianos, mas impõem diferentes estruturas a priori sobre os coeficientes:

- O prior Gaussiano (Ridge) é suave e disperso: tende a produzir coeficientes pequenos, mas raramente exatamente zero.
- O prior Laplace (Lasso) é pontudo no zero: concentra mais densidade ao redor do zero, promovendo esparsidade.

Assim, Ridge e Lasso são interpretações MAP sob diferentes priors, enquanto mínimos quadrados (ou MLE) assume apenas a verossimilhança sem prior.

Capítulo 8

Boosting

Em muitos problemas de aprendizado, pode ser difícil construir diretamente um modelo muito preciso. Por outro lado, é mais factível encontrar modelos simples que sejam capazes de errar menos do que um chute aleatório. Esse tipo de modelo é conhecido como *weak learner* ou aprendiz fraco.

Definição 1 (Aprendiz fraco). *Uma classe de problemas é dita ser aprendível por um weak learner se existe um algoritmo que, dado qualquer distribuição dos dados, consegue produzir uma hipótese com erro menor que 50% — ou seja, acerta um pouco mais do que o acaso.*

Mais formalmente, existe algum $\gamma > 0$ tal que, para qualquer distribuição dos dados e qualquer problema da classe, a hipótese h gerada satisfaz

$$\mathbb{P} \left(R(h) \leq \frac{1}{2} - \gamma \right) \geq 1 - \delta$$

para qualquer nível de confiança $\delta > 0$ e com um número de amostras suficientemente grande.

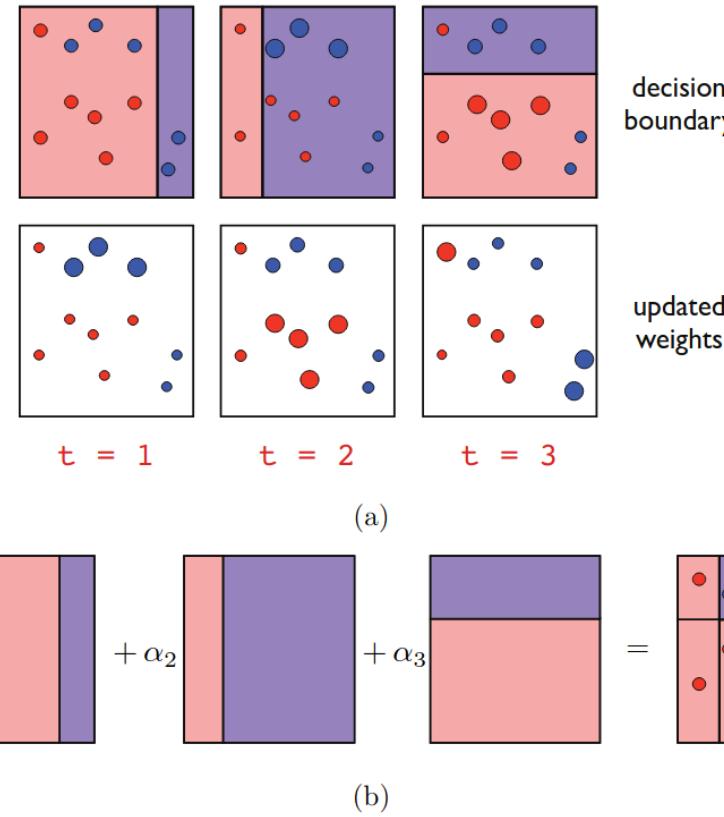
Aqui, $R(h)$ representa o erro da hipótese h . A condição significa que, com alta probabilidade, o erro do modelo é estritamente menor do que 50% por uma margem γ .

Os modelos gerados por um aprendiz fraco são chamados de *classificadores base* (*base classifiers*). A ideia central por trás do *boosting* é pegar esses classificadores fracos e combiná-los para formar um modelo forte, ou seja, um modelo que tenha alta precisão.

O boosting faz isso utilizando métodos de comitê: ele combina várias hipóteses fracas para construir um preditor mais preciso. Na prática, o boosting escolhe quais classificadores usar e como combiná-los, atribuindo pesos maiores aos modelos que erram menos e focando nas amostras onde os modelos anteriores erraram. Um dos algoritmos de boosting mais conhecidos é o *AdaBoost*.

8.1 AdaBoost

Denotamos por \mathcal{H} o conjunto de classificadores base, também chamado de *conjunto de hipóteses*, a partir do qual o algoritmo seleciona os modelos. Cada classificador é uma função que recebe uma entrada x e retorna um rótulo em $\{-1, +1\}$, ou seja, $\mathcal{H} \subseteq \{-1, +1\}^{\mathcal{X}}$.



O AdaBoost funciona mantendo uma distribuição de pesos sobre os exemplos do conjunto de treino. Inicialmente, essa distribuição é uniforme, ou seja, cada exemplo tem peso igual a $1/m$. A cada iteração, o algoritmo escolhe um classificador $h_t \in \mathcal{H}$ que minimiza o erro ponderado segundo a distribuição atual \mathcal{D}_t :

$$h_t = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^m \mathcal{D}_t(i) \cdot \mathbf{1}_{h(x_i) \neq y_i}$$

onde $\mathbf{1}_{h(x_i) \neq y_i}$ vale 1 se o classificador erra o exemplo i e 0 caso contrário.

O erro do classificador h_t na distribuição \mathcal{D}_t é dado por

$$\varepsilon_t = \sum_{i=1}^m \mathcal{D}_t(i) \cdot \mathbf{1}_{h_t(x_i) \neq y_i}$$

e o peso associado a esse classificador é calculado como

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

de forma que classificadores mais precisos (menor ε_t) recebem maior peso.

Após calcular α_t , a distribuição dos pesos sobre os exemplos é atualizada, aumentando o peso dos exemplos que foram classificados incorretamente:

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

onde Z_t é um fator de normalização que garante que a soma dos pesos seja igual a 1.

Ao final de T iterações, o classificador final é uma combinação linear dos classificadores base:

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

e a predição final é feita tomando o sinal dessa soma:

$$\text{sign}(f(x))$$

O procedimento completo do AdaBoost pode ser resumido no seguinte pseudocódigo:

Algorithm 6 AdaBoost

Require: Conjunto de treinamento $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, onde $y_i \in \{-1, +1\}$

Require: Número de iterações T

- 1: Inicialize $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$ para todo $i = 1, \dots, m$
- 2: **for** $t = 1$ até T **do**
- 3: Escolha $h_t \in \mathcal{H}$ que minimiza o erro ponderado:

$$h_t = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^m \mathcal{D}_t(i) \cdot \mathbf{1}_{h(x_i) \neq y_i}$$

- 4: Calcule o erro:
- $\varepsilon_t = \sum_{i=1}^m \mathcal{D}_t(i) \cdot \mathbf{1}_{h_t(x_i) \neq y_i}$

- 5: Calcule o peso do classificador:

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

- 6: Calcule o fator de normalização:

$$Z_t = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}$$

- 7: **for** $i = 1$ até m **do**

- 8: Atualize a distribuição:

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

- 9: **end for**

- 10: **end for**

- 11: Defina o classificador final:

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

- 12: **return** $\text{sign}(f(x))$
-

Esse processo permite que o AdaBoost combine sucessivamente vários classificadores fracos, dando mais foco aos exemplos difíceis, até construir um modelo final com desempenho muito superior aos modelos individuais.

8.1.1 Cálculo do erro empírico

Uma propriedade fundamental do AdaBoost é que seu erro empírico no conjunto de treinamento decai de forma exponencial à medida que o número de iterações T cresce. Esse resultado é uma consequência direta da forma como o algoritmo atualiza os pesos e da escolha dos classificadores base.

Seja $f = \sum_{t=1}^T \alpha_t h_t$ o classificador final, que é uma combinação linear dos classificadores base. O erro empírico desse classificador no conjunto de treinamento S é definido por

$$\hat{R}_S(f) = \frac{1}{m} \sum_{i=1}^m 1_{y_i f(x_i) \leq 0}$$

onde a função indicadora vale 1 quando a predição está incorreta, ou seja, quando o sinal de $f(x_i)$ não coincide com y_i .

Para obter uma cota superior para esse erro, usamos a desigualdade geral

$$1_{u \leq 0} \leq \exp(-u)$$

válida para qualquer $u \in \mathbb{R}$. Aplicando essa desigualdade, temos que

$$\hat{R}_S(f) = \frac{1}{m} \sum_{i=1}^m 1_{y_i f(x_i) \leq 0} \leq \frac{1}{m} \sum_{i=1}^m e^{-y_i f(x_i)}$$

Agora, usamos a identidade que relaciona a distribuição \mathcal{D}_{T+1} com a função f . Sabemos que

$$\mathcal{D}_{T+1}(i) = \frac{e^{-y_i f(x_i)}}{m \prod_{t=1}^T Z_t}$$

onde Z_t é o fator de normalização na iteração t .

Reorganizando essa equação, obtemos

$$e^{-y_i f(x_i)} = m \cdot \mathcal{D}_{T+1}(i) \cdot \prod_{t=1}^T Z_t$$

Substituindo isso na expressão para o erro empírico, temos

$$\hat{R}_S(f) \leq \frac{1}{m} \sum_{i=1}^m m \cdot \mathcal{D}_{T+1}(i) \cdot \prod_{t=1}^T Z_t$$

Cancelando o fator m ,

$$\hat{R}_S(f) \leq \left(\sum_{i=1}^m \mathcal{D}_{T+1}(i) \right) \cdot \prod_{t=1}^T Z_t$$

Como \mathcal{D}_{T+1} é uma distribuição, sua soma sobre todos os exemplos é igual a 1. Assim, obtemos

$$\hat{R}_S(f) \leq \prod_{t=1}^T Z_t$$

Portanto, o erro empírico é completamente controlado pelo produto dos fatores de normalização Z_t ao longo das iterações.

Vamos então calcular Z_t . Por definição,

$$Z_t = \sum_{i=1}^m \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(x_i)}$$

Dividimos os exemplos em dois grupos: aqueles corretamente classificados ($y_i h_t(x_i) = +1$) e aqueles incorretamente classificados ($y_i h_t(x_i) = -1$). Assim,

$$Z_t = \sum_{i:y_i h_t(x_i) = +1} \mathcal{D}_t(i) e^{-\alpha_t} + \sum_{i:y_i h_t(x_i) = -1} \mathcal{D}_t(i) e^{\alpha_t}$$

A soma das probabilidades dos exemplos corretamente classificados é $(1 - \varepsilon_t)$, e dos incorretamente classificados é ε_t . Portanto,

$$Z_t = (1 - \varepsilon_t) e^{-\alpha_t} + \varepsilon_t e^{\alpha_t}$$

Agora substituímos a definição de α_t , que é

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

Calculamos cada termo:

$$e^{\alpha_t} = \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}}$$

e

$$e^{-\alpha_t} = \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}}$$

Substituindo esses valores em Z_t :

$$Z_t = (1 - \varepsilon_t) \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} + \varepsilon_t \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}}$$

Multiplicamos cada termo:

$$Z_t = \sqrt{\varepsilon_t(1 - \varepsilon_t)} \left(\frac{1 - \varepsilon_t}{\sqrt{\varepsilon_t(1 - \varepsilon_t)}} + \frac{\varepsilon_t}{\sqrt{\varepsilon_t(1 - \varepsilon_t)}} \right)$$

Observe que os termos no parêntese somam exatamente 1:

$$(1 - \varepsilon_t) + \varepsilon_t = 1$$

Portanto,

$$Z_t = 2 \sqrt{\varepsilon_t(1 - \varepsilon_t)}$$

Agora calculamos o produto dos Z_t ao longo de $t = 1$ até T :

$$\prod_{t=1}^T Z_t = \prod_{t=1}^T 2 \sqrt{\varepsilon_t(1 - \varepsilon_t)}$$

Fatorando o 2:

$$= 2^T \prod_{t=1}^T \sqrt{\varepsilon_t(1 - \varepsilon_t)}$$

Reescrevemos cada termo da raiz como

$$\sqrt{\varepsilon_t(1-\varepsilon_t)} = \sqrt{\frac{1}{4} - \left(\frac{1}{2} - \varepsilon_t\right)^2}$$

Aplicamos então a desigualdade $1-x \leq e^{-x}$

$$\sqrt{1 - 4 \left(\frac{1}{2} - \varepsilon_t\right)^2} \leq \exp\left(-2 \left(\frac{1}{2} - \varepsilon_t\right)^2\right)$$

que é válida para qualquer número real.

Substituindo essa cota no produto, obtemos

$$\prod_{t=1}^T Z_t \leq \exp\left(-2 \sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t\right)^2\right)$$

Portanto, o erro empírico satisfaz

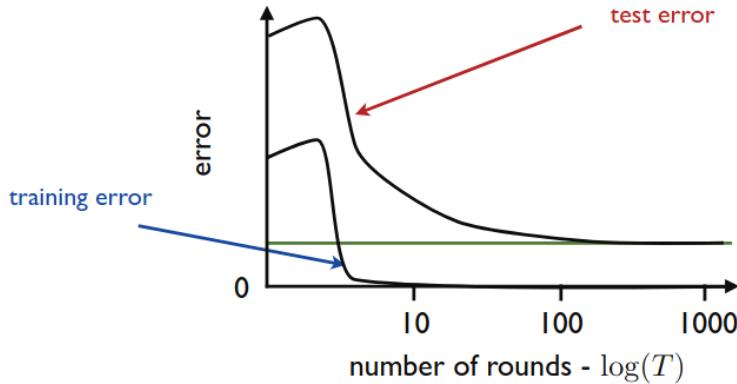
$$\hat{R}_S(f) \leq \exp\left(-2 \sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t\right)^2\right)$$

Este resultado mostra que o erro empírico decai exponencialmente rápido à medida que T aumenta, desde que cada classificador tenha erro ε_t ligeiramente inferior a 50%.

No caso em que todos os classificadores satisfazem $\varepsilon_t \leq \frac{1}{2} - \gamma$ para algum $\gamma > 0$ constante, então obtemos diretamente a cota

$$\hat{R}_S(f) \leq \exp(-2\gamma^2 T)$$

ou seja, o erro empírico decai exponencialmente em função do número de iterações T .



8.1.2 Um pouco de teoria

Um fenômeno interessante observado na prática com o AdaBoost é que, apesar do erro de treinamento frequentemente cair rapidamente até zero, o erro no conjunto de teste continua diminuindo por algum tempo mesmo após o treinamento já não apresentar mais erros. Esse comportamento é contraintuitivo do ponto de vista clássico de sobreajuste, onde se espera que, ao

atingir erro zero no treino, o modelo passe a superajustar os dados e o erro de teste comece a aumentar. No entanto, no AdaBoost, o que se observa é que, após alcançar erro zero no conjunto de treino, o algoritmo continua ajustando as margens — ou seja, continua aumentando a confiança nas previsões, especialmente afastando os exemplos corretamente classificados da fronteira de decisão. Esse aumento das margens tem efeito direto na melhora da generalização, e por isso o erro no teste continua decaindo por várias iterações mesmo quando o erro no treino já é zero. Esse fenômeno está intimamente relacionado ao fato de que o AdaBoost não apenas busca classificar corretamente, mas também maximizar as margens dos exemplos.

Esse comportamento, que inicialmente parece surpreendente, pode ser explicado matematicamente por uma cota de generalização baseada em margens. Esse resultado mostra que o erro verdadeiro de um classificador não depende apenas do erro no conjunto de treinamento, mas também da distribuição das margens — isto é, de quão "confiantes" são as previsões realizadas. Especificamente, vale o seguinte:

Teorema 3 (Cota de generalização via margem). *Seja \mathcal{H} um conjunto de funções de valores reais. Fixe $\rho > 0$. Então, para qualquer $\delta > 0$, com probabilidade ao menos $1 - \delta$, vale que, para todo $h \in \text{conv}(\mathcal{H})$,*

$$R(h) \leq \widehat{R}_{S,\rho}(h) + \frac{2}{\rho} \mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

e também

$$R(h) \leq \widehat{R}_{S,\rho}(h) + \frac{2}{\rho} \widehat{\mathfrak{R}}_S(\mathcal{H}) + 3 \sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

onde $R(h)$ é o erro verdadeiro, $\widehat{R}_{S,\rho}(h)$ é o erro empírico considerando apenas os exemplos cuja margem é menor que ρ , e $\mathfrak{R}_m(\mathcal{H})$ é a complexidade de Rademacher do conjunto \mathcal{H} .

De forma intuitiva, esse resultado nos diz que a generalização de modelos como AdaBoost não depende apenas do erro no conjunto de treinamento, mas também de como esse erro está distribuído em termos de margem. O termo $\widehat{R}_{S,\rho}(h)$ mede quantos exemplos estão próximos da fronteira de decisão — ou seja, com margem menor que ρ . Se quase todos os exemplos têm margens grandes, esse termo é pequeno, mesmo que o erro zero tenha sido alcançado.

O segundo termo, que é proporcional a $\frac{1}{\rho} \mathfrak{R}_m(\mathcal{H})$, reflete a influência da complexidade do modelo. Aqui, a complexidade de Rademacher mede o quanto o conjunto de classificadores \mathcal{H} é flexível, ou seja, capaz de se ajustar a diferentes padrões nos dados. Quanto maior \mathcal{H} , maior essa complexidade. No entanto, ela aparece dividida por ρ , o que significa que, quanto maior for a margem, menor será o efeito da complexidade no bound de generalização.

Por fim, o terceiro termo é puramente estatístico, decrescendo com $1/\sqrt{m}$, e representa o efeito do tamanho da amostra na confiança do bound.

No contexto do AdaBoost, esse teorema formaliza a intuição de que, após alcançar erro zero no treinamento, o algoritmo continua melhorando a generalização não por reduzir mais o erro empírico (que já é zero), mas por aumentar as margens — isto é, tornando as classificações mais confiantes, afastando os exemplos da fronteira de decisão. Esse mecanismo explica por que, empiricamente, o erro de teste muitas vezes continua caindo mesmo depois do erro de treino ter sido completamente eliminado.

8.2 Gradient Boosting

O Gradient Boosting é uma generalização do AdaBoost desenvolvida no início dos anos 2000 como uma abordagem mais ampla e flexível para o problema de boosting. Enquanto o AdaBoost foi originalmente proposto para classificação binária, o Gradient Boosting estende a ideia central — ajustar modelos sucessivos para corrigir os erros dos anteriores — para qualquer função de perda diferenciável, permitindo seu uso tanto em regressão quanto em classificação.

Historicamente, a conexão entre AdaBoost e otimização foi primeiramente identificada por Leo Breiman, que mostrou que o AdaBoost pode ser interpretado como um algoritmo de otimização que minimiza uma função de perda exponencial. Essa observação levou Jerome Friedman, em 2001, a formalizar o *Gradient Boosting* como um método geral de descida de gradiente no espaço de funções. Esse framework permite entender boosting como uma sequência de passos na direção oposta ao gradiente da função de risco, de maneira completamente análoga à descida de gradiente tradicional no espaço de vetores.

Antes de formalizarmos o Gradient Boosting, é útil revisar a intuição da descida de gradiente no contexto clássico de otimização.

Seja $F(\theta)$ uma função de custo que depende de um vetor de parâmetros $\theta \in \mathbb{R}^d$. Nosso objetivo é resolver

$$\min_{\theta} F(\theta)$$

A intuição da descida de gradiente surge da aproximação de Taylor de primeira ordem da função F ao redor de um ponto θ_0 :

$$F(\theta_0 + h) \approx F(\theta_0) + \nabla F(\theta_0)^{\top} h$$

Esse desenvolvimento nos diz que, para pequenos deslocamentos g , o valor da função varia aproximadamente de forma linear na direção de g . A direção do gradiente $\nabla F(\theta_0)$ aponta na direção de maior crescimento local de F . Portanto, se escolhermos

$$h = -\eta \cdot \nabla F(\theta_0)$$

com $\eta > 0$ pequeno, teremos

$$F(\theta_0 + h) \approx F(\theta_0) - \eta \cdot \|\nabla F(\theta_0)\|^2$$

ou seja, a função diminui, pois o termo subtraído é não-negativo.

Esse raciocínio leva à regra de atualização da descida de gradiente:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla F(\theta_{t-1})$$

onde o parâmetro η controla o tamanho do passo.

Se a função F for convexa e suave, a descida de gradiente converge para o mínimo global, desde que o *learning rate* seja escolhido de forma adequada. Na prática, mesmo em problemas não convexos, esse método é extremamente utilizado e serve como base para muitos algoritmos modernos de otimização.

A ideia central do Gradient Boosting é aplicar o princípio da descida de gradiente no espaço de funções. Assim como no caso paramétrico nos movemos na direção oposta ao gradiente para

reduzir uma função de custo, aqui buscamos, a cada iteração, uma função que aponta na direção que mais reduz o risco.

Sejam (X, Y) variáveis aleatórias conjuntas, com $X \in \mathcal{X}$ e $Y \in \mathbb{R}$ (ou $\{-1, +1\}$ na classificação). Seja $L(Y, z)$ uma função de perda que mede o custo de predizer z quando o verdadeiro valor observado é Y . O objetivo é encontrar uma função $f : \mathcal{X} \rightarrow \mathbb{R}$ que minimiza o risco populacional

$$\mathcal{R}(f) = \mathbb{E}[L(Y, f(X))].$$

O procedimento começa com uma função inicial

$$f_0 = 0$$

e realiza atualizações iterativas da forma

$$f_t(x) = f_{t-1}(x) + \nu \cdot h_t(x)$$

onde h_t aproxima a direção de descida e $\nu > 0$ é o *learning rate*.

O gradiente funcional no ponto x é dado por

$$g_t(x) = \mathbb{E} \left[\frac{\partial}{\partial z} L(Y, z) \Big|_{z=f_{t-1}(x)} \Big| X = x \right].$$

Na prática, como não conhecemos a distribuição dos dados, trabalhamos com uma amostra $\{(x_i, y_i)\}_{i=1}^n$ e minimizamos o risco empírico

$$\widehat{R}(f) = \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)).$$

O gradiente funcional se torna simplesmente o gradiente da perda no ponto (x_i, y_i) :

$$g_t(x_i) = \frac{\partial}{\partial z} L(y_i, z) \Big|_{z=f_{t-1}(x_i)}.$$

O modelo h_t é ajustado para aproximar os pseudo-resíduos

$$r_{i,t} = -g_t(x_i)$$

sobre os dados $\{x_i\}$, tipicamente via uma árvore de decisão.

A atualização segue

$$f_t(x) = f_{t-1}(x) + \nu \cdot h_t(x).$$

Exemplos de funções de perda e seus gradientes:

- **Perda quadrática (regressão):**

$$L(y, z) = (y - z)^2, \quad \frac{\partial}{\partial z} L = -2(y - z)$$

Pseudo-resíduo: $(y - f_{t-1}(x))$.

Note que nesse caso, o primeiro resíduo é $r = y - 0$ e portanto na primeira iteração do método, tentamos aprender y dado x , como numa tarefa usual de aprendizado supervisionado.

- Perda exponencial (classificação binária, AdaBoost):

$$L(y, z) = \exp(-yz), \quad y \in \{-1, +1\}$$

$$\frac{\partial}{\partial z} L = -y \exp(-yz)$$

Atribui mais peso a exemplos mal classificados.

- Log-loss (classificação binária, regressão logística):

$$L(y, z) = \log(1 + \exp(-2yz))$$

$$\frac{\partial}{\partial z} L = -\frac{2y}{1 + \exp(2yz)}$$

O pseudo-resíduo se comporta como o erro na probabilidade predita.

- Perda absoluta (regressão robusta):

$$L(y, z) = |y - z|$$

Gradiente subdiferencial:

$$\frac{\partial}{\partial z} L = \begin{cases} -1 & \text{se } y - z > 0 \\ +1 & \text{se } y - z < 0 \end{cases}$$

Leva à mediana condicional em vez da média.

O Gradient Boosting é, portanto, um framework geral de otimização que aplica descida de gradiente no espaço de funções. Escolher a função de perda define os pseudo-resíduos e, consequentemente, a dinâmica do algoritmo, permitindo sua aplicação tanto em regressão quanto em classificação.

Algorithm 7 Gradient Boosting via Pseudo-Resíduos

Require: Dados de treino $\{(x_i, y_i)\}_{i=1}^n$

Require: Número de iterações B

Require: Função de perda $L(y, z)$

Require: Learning rate $\nu > 0$

Require: Espaço de funções \mathcal{H} (ex.: árvores)

1: Inicialize $f_0(x) = 0$

2: **for** $b = 1$ até B **do**

3: Computar os pseudo-resíduos:

$$r_{i,b} = -\left. \frac{\partial}{\partial z} L(y_i, z) \right|_{z=f_{b-1}(x_i)}$$

4: Ajustar um modelo $h_b \in \mathcal{H}$ para aprender $r_{i,b}$ dado x_i

5: Atualizar o modelo:

$$f_b(x) = f_{b-1}(x) + \nu \cdot h_b(x)$$

6: **end for**

7: **return** Modelo final $f_B(x)$

Capítulo 9

SVM

Este capítulo apresenta um dos algoritmos de classificação mais bem fundamentados teoricamente e também um dos mais eficazes na prática: as Máquinas de Vetores de Suporte (SVMs). Começaremos com a formulação do problema de classificação linear, depois trataremos do caso em que os dados não são separáveis e, por fim, discutiremos a fundamentação teórica baseada na noção de margem.

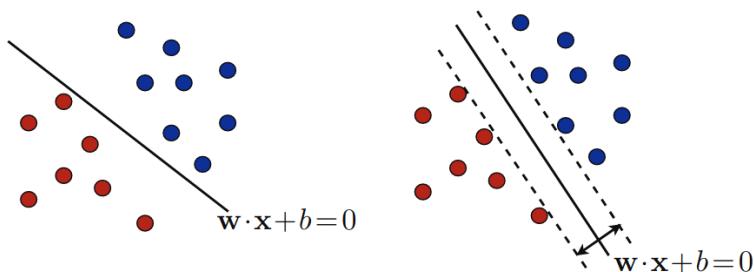
Considere um espaço de entrada $\mathcal{X} \subset \mathbb{R}^p$, com $p \geq 1$, e um espaço de saída $\mathcal{Y} = \{-1, +1\}$. Suponha que exista uma função desconhecida $f : \mathcal{X} \rightarrow \mathcal{Y}$ que associa rótulos às observações. Dado um conjunto de hipóteses \mathcal{H} , que contém funções que mapeiam \mathcal{X} em \mathcal{Y} , o objetivo da tarefa de classificação binária é escolher uma hipótese $h \in \mathcal{H}$, também chamada de classificador binário, de forma que seu erro de generalização seja pequeno.

Disponibiliza-se uma amostra de treinamento $S = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset (\mathcal{X} \times \mathcal{Y})^n$, com $y_i = f(x_i)$, onde os pares (x_i, y_i) são amostrados de forma i.i.d. a partir de uma distribuição desconhecida \mathcal{D} . O desempenho do classificador h é avaliado pelo erro de generalização, definido como

$$R_{\mathcal{D}}(h) = \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)].$$

A escolha do conjunto de hipóteses \mathcal{H} é fundamental. Resultados anteriores, como o princípio da navalha de Occam, sugerem que conjuntos com menor complexidade tendem a oferecer melhores garantias de aprendizado, assumindo todas as demais condições iguais. Uma classe de hipóteses naturalmente simples e bastante estudada é a dos classificadores lineares, também conhecidos como hiperplanos. Essa classe pode ser definida como:

$$\mathcal{H} = \{x \mapsto \text{sign}(\langle \mathbf{w}, x \rangle + b) : \mathbf{w} \in \mathbb{R}^p, b \in \mathbb{R}\}.$$



O problema de aprendizado com essa classe recebe o nome de problema de classificação linear. Geometricamente, a equação $\langle \mathbf{w}, x \rangle + b = 0$ define um hiperplano em \mathbb{R}^p , onde \mathbf{w} é um vetor normal não nulo ao hiperplano e b é um escalar. Um classificador da forma $x \mapsto \text{sign}(\langle \mathbf{w}, x \rangle + b)$ atribui o rótulo $+1$ a todos os pontos que estão de um lado do hiperplano e o rótulo -1 aos pontos que estão do outro lado.

9.1 Caso separável

9.1.1 Problema primal

Passamos agora ao caso em que a amostra de treinamento $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ pode ser perfeitamente separada por um hiperplano linear. Isso equivale a supor que existe um par $(\mathbf{w}, b) \in (\mathbb{R}^p \setminus \{0\}) \times \mathbb{R}$ tal que

$$\forall i \in [n], \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 0.$$

Ou seja, o hiperplano $\langle \mathbf{w}, x \rangle + b = 0$ separa corretamente todos os exemplos da amostra, classificando os pontos com $y_i = +1$ de um lado e os com $y_i = -1$ do outro.

No entanto, há infinitos hiperplanos que satisfazem essa propriedade. A questão é: qual deles devemos escolher? O critério adotado pelas SVMs é selecionar o hiperplano com **maior margem geométrica**.

A margem geométrica $\rho_h(x)$ de um classificador linear $h(x) = \langle \mathbf{w}, x \rangle + b$ em um ponto x é a distância euclidiana desse ponto ao hiperplano de decisão, dada por:

$$\rho_h(x) = \frac{|\langle \mathbf{w}, x \rangle + b|}{\|\mathbf{w}\|_2}.$$

Essa fórmula pode ser demonstrada observando que a projeção ortogonal de x sobre o hiperplano ocorre ao longo da direção \mathbf{w} . Se considerarmos a reta $r(t) = x + t\mathbf{w}$, o ponto de interseção com o hiperplano ocorre quando $\langle \mathbf{w}, x + t\mathbf{w} \rangle + b = 0$, o que nos dá

$$t' = -\frac{\langle \mathbf{w}, x \rangle + b}{\|\mathbf{w}\|^2}.$$

A distância entre x e $r(t')$ é então $\|t'\mathbf{w}\| = \frac{|\langle \mathbf{w}, x \rangle + b|}{\|\mathbf{w}\|}$, como desejado.

Exercício 15. Preencha os detalhes da dedução geométrica da fórmula da margem.

A margem geométrica do classificador h em relação à amostra $S = \{x_1, \dots, x_n\}$ é definida como:

$$\rho_h = \min_{i \in [n]} \rho_h(x_i) = \min_{i \in [n]} \frac{|\langle \mathbf{w}, x_i \rangle + b|}{\|\mathbf{w}\|}.$$

A SVM procura o hiperplano separador que maximiza essa margem mínima — o chamado hiperplano de margem máxima. Podemos então escrever o problema de otimização como:

$$\rho = \max_{\mathbf{w}, b : y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 0} \min_{i \in [n]} \frac{|\langle \mathbf{w}, x_i \rangle + b|}{\|\mathbf{w}\|}.$$

Note agora que, para qualquer (\mathbf{w}, b) que separa corretamente a amostra, temos $y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 0$. Além disso, para tais (\mathbf{w}, b) , o valor absoluto pode ser removido pela identidade:

$$|\langle \mathbf{w}, x_i \rangle + b| = y_i(\langle \mathbf{w}, x_i \rangle + b),$$

uma vez que o rótulo y_i garante o sinal positivo da expressão. Assim, podemos escrever:

$$\rho = \max_{\mathbf{w}, b} \min_{i \in [n]} \frac{y_i(\langle \mathbf{w}, x_i \rangle + b)}{\|\mathbf{w}\|}.$$

Agora, note que a expressão

$$\min_{i \in [n]} \frac{y_i(\langle \mathbf{w}, x_i \rangle + b)}{\|\mathbf{w}\|}$$

é invariante por multiplicação simultânea de \mathbf{w} e b por qualquer escalar positivo. De fato, se substituímos (\mathbf{w}, b) por $(\alpha \mathbf{w}, \alpha b)$, com $\alpha > 0$, tanto o numerador quanto o denominador da fração são multiplicados por α , e o fator se cancela.

Isso nos permite fixar a escala de (\mathbf{w}, b) da forma mais conveniente. Uma escolha natural é normalizar os parâmetros para que o menor valor de $y_i(\langle \mathbf{w}, x_i \rangle + b)$ seja igual a 1. Sob essa convenção, a margem geométrica passa a ser

$$\rho = \frac{1}{\|\mathbf{w}\|},$$

e o problema de maximização da margem pode ser reformulado como:

$$\rho = \max_{\mathbf{w}, b : \min_{i \in [n]} y_i(\langle \mathbf{w}, x_i \rangle + b) = 1} \frac{1}{\|\mathbf{w}\|} = \max_{\mathbf{w}, b : y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1, \forall i \in [n]} \frac{1}{\|\mathbf{w}\|}.$$

A equivalência entre as duas expressões decorre do fato de que, se (\mathbf{w}, b) satisfaz as restrições $y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1$ para todo i , então podemos considerar os parâmetros reescalados $(\mathbf{w}', b') = (\mathbf{w}/\lambda, b/\lambda)$, que satisfazem

$$y_i(\langle \mathbf{w}', x_i \rangle + b') = \frac{1}{\lambda} y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1,$$

com $\|\mathbf{w}'\| = \|\mathbf{w}\|/\lambda < \|\mathbf{w}\|$. Portanto, o par reescalado obedece às mesmas restrições com margem funcional mínima igual a 1, e resulta em uma norma menor — ou seja, uma margem maior. Isso mostra que, na formulação com ≥ 1 , a solução ótima sempre ocorre no caso em que o mínimo é igual a 1, justificando a troca da igualdade por desigualdade.

Como maximizar $\frac{1}{\|\mathbf{w}\|}$ é equivalente a minimizar $\|\mathbf{w}\|$, e mais convenientemente $\frac{1}{2}\|\mathbf{w}\|^2$, a solução da SVM no caso separável corresponde ao seguinte problema de otimização convexa:

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{sujeito a} \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1, \quad \forall i \in [n].$$

A função objetivo $F(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$ é infinitamente diferenciável. Seu gradiente é $\nabla F(\mathbf{w}) = \mathbf{w}$, e a matriz hessiana é $\nabla^2 F(\mathbf{w}) = I$, a matriz identidade. Como todos os autovalores da hessiana são estritamente positivos, segue que F é estritamente convexa.

9.1.2 Um pouco de otimização convexa

Para resolver problemas de otimização com restrições, uma ferramenta fundamental é o *Lagrangiano*. Dado um problema de minimização com restrições de desigualdade da forma

$$\min_{z \in \mathbb{R}^d} f(z) \quad \text{sujeito a} \quad g_i(z) \leq 0, \quad i = 1, \dots, m,$$

o Lagrangiano associado é definido como

$$\mathcal{L}(z, \lambda) = f(z) + \sum_{i=1}^m \lambda_i g_i(z),$$

onde $\lambda_i \geq 0$ são os multiplicadores de Lagrange. Intuitivamente, os termos $\lambda_i g_i(z)$ penalizam o descumprimento das restrições $g_i(z) \leq 0$.

As soluções ótimas desse problema satisfazem as chamadas condições de Karush-Kuhn-Tucker (KKT), que generalizam as condições de optimalidade de Lagrange para problemas com desigualdades. No caso em que f é convexa, g_i são funções convexas e as restrições são qualificadas (por exemplo, há um ponto estritamente viável), as condições KKT são necessárias e suficientes para a optimalidade global.

As condições de Karush-Kuhn-Tucker (KKT) exigem que existam $z^* \in \mathbb{R}^d$ e $\lambda^* \in \mathbb{R}^m$ tais que:

- **Estacionaridade:** $\nabla f(z^*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(z^*) = 0$;
- **Viabilidade primal:** $g_i(z^*) \leq 0$ para todo i ;
- **Viabilidade dual:** $\lambda_i^* \geq 0$ para todo i ;
- **Complementaridade:** $\lambda_i^* g_i(z^*) = 0$ para todo i .

9.1.3 Vetores de suporte

Voltando ao problema de otimização primal, notamos que as restrições são afins e, portanto, qualificadas. Tanto a função objetivo quanto as restrições são continuamente diferenciáveis e convexas, o que garante, pelo que as condições KKT são válidas no ponto ótimo.

Vamos introduzir variáveis de Lagrange $\alpha_i \geq 0$ associadas às n restrições, e denotar por $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^\top \in \mathbb{R}_+^n$. O Lagrangiano do problema primal é dado por:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i [y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1].$$

As condições KKT são obtidas anulando o gradiente do Lagrangiano em relação às variáveis primais \mathbf{w} e b , e impondo a condição de complementaridade. Especificamente:

- Derivando em relação a \mathbf{w} :

$$\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i;$$

- Derivando em relação a b :

$$\nabla_b \mathcal{L} = - \sum_{i=1}^n \alpha_i y_i = 0;$$

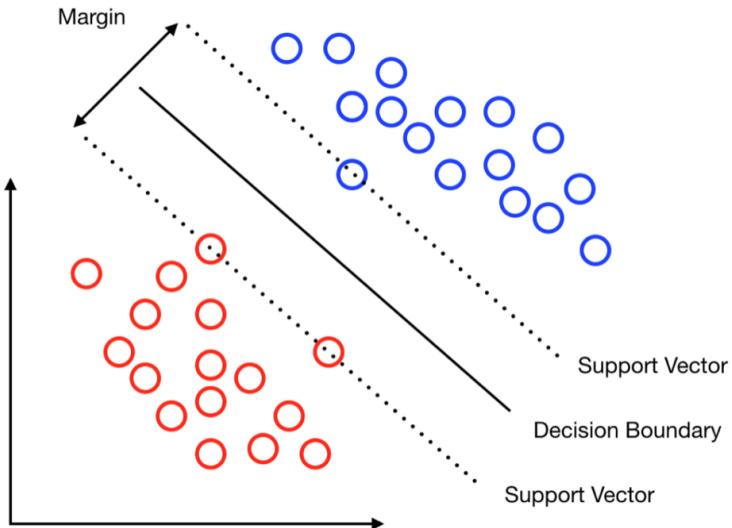
- Condição de complementaridade:

$$\alpha_i (y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) = 0, \quad \text{para todo } i.$$

A primeira equação mostra que o vetor \mathbf{w} na solução do problema é uma combinação linear dos vetores da amostra. Um vetor \mathbf{x}_i contribui para essa combinação apenas se $\alpha_i \neq 0$. Tais vetores são chamados de *vetores de suporte*.

Pela condição de complementaridade, sempre que $\alpha_i \neq 0$, temos $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$, ou seja, esses pontos estão exatamente sobre as *hiperplanos marginais* $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm 1$.

Os vetores de suporte determinam completamente a solução do problema. Vetores não suportes ($\alpha_i = 0$) não afetam a posição do hiperplano — sua presença ou ausência não altera a resposta da SVM. Apesar de a solução (\mathbf{w}, b) ser única (graças à convexidade estrita do problema), os vetores de suporte não são necessariamente únicos. Como qualquer hiperplano em \mathbb{R}^p é definido por p pontos em posição geral, bastam $p + 1$ vetores de suporte para determinar a solução — embora, em geral, mais de $p + 1$ pontos possam estar sobre a margem.



9.1.4 Um breve comentário sobre dualidade

Dualidade é um conceito fundamental em otimização. A ideia central é que, dado um problema de minimização (o chamado problema *primal*), podemos associar a ele um problema alternativo, chamado *dual*, cuja estrutura pode ser mais simples ou mais informativa. Resolver o problema dual, quando possível, fornece um limite inferior (ou superior, no caso de maximização) para a solução do primal.

Considere um problema de otimização com restrições de desigualdade da forma:

$$\min_{z \in \mathbb{R}^d} f(z) \quad \text{sujeito a} \quad g_i(z) \leq 0, \quad i = 1, \dots, m.$$

Definimos o *Lagrangiano* associado como:

$$\mathcal{L}(z, \lambda) = f(z) + \sum_{i=1}^m \lambda_i g_i(z), \quad \text{com } \lambda_i \geq 0.$$

Para cada vetor de multiplicadores $\lambda \geq 0$, a função

$$d(\lambda) := \inf_z \mathcal{L}(z, \lambda)$$

define o valor dual correspondente, e o problema *dual* consiste em maximizar essa função:

$$\max_{\lambda \geq 0} d(\lambda).$$

Esse valor dual fornece sempre um limite inferior para o ótimo do primal. De fato, se z é viável (isto é, satisfaz $g_i(z) \leq 0$) e $\lambda \geq 0$, então os termos $\lambda_i g_i(z)$ são não positivos, o que implica que $\mathcal{L}(z, \lambda) \leq f(z)$. Em particular, isso vale para o ponto ótimo z^* , o que nos dá

$$d(\lambda) = \inf_z \mathcal{L}(z, \lambda) \leq \mathcal{L}(z^*, \lambda) \leq f(z^*) = p^*.$$

Portanto, o valor ótimo dual d^* satisfaz sempre $d^* \leq p^*$, o que é conhecido como *dualidade fraca*. Isso mostra que o dual está sempre “por baixo” do primal, e justifica a estratégia de maximizar a função dual para obter bons limites inferiores. A diferença $p^* - d^*$ é chamada de *dual gap*, e mede o quanto estamos distantes da solução ótima do problema original.

Em problemas convexos com restrições bem comportadas — por exemplo, se f e os g_i são convexos e existe um ponto estritamente viável ($g_i(z) < 0$ para todo i) — temos que o dual gap é nulo, isto é, $d^* = p^*$. Esse resultado é conhecido como *dualidade forte*, e garante que podemos resolver o problema primal indiretamente, via sua formulação dual, sem perda de exatidão.

9.1.5 Problema dual

Vamos agora derivar a formulação dual do problema primal da SVM. Para isso, substituímos no Lagrangiano a expressão de \mathbf{w} em termos das variáveis dual α_i , conforme obtida pela condição KKT:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i x_i.$$

Substituindo essa expressão no Lagrangiano e aplicando a condição de viabilidade $\sum_{i=1}^n \alpha_i y_i = 0$, obtemos:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|^2 - \sum_{i=1}^n \alpha_i \left[y_i \left(\sum_{j=1}^n \alpha_j y_j \langle x_j, x_i \rangle + b \right) - 1 \right] \\ &= \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i. \end{aligned}$$

O segundo termo é igual a duas vezes o primeiro, e o termo com b é zero devido à restrição $\sum_i \alpha_i y_i = 0$. Assim, obtemos:

$$\mathcal{L} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle.$$

A formulação dual da SVM no caso separável é, portanto:

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle,$$

$$\text{sujeito a } \alpha_i \geq 0, \quad \sum_{i=1}^n \alpha_i y_i = 0.$$

A função objetivo do dual é uma função quadrática côncava de α , pois a hessiana $\nabla^2 G = -A$, com $A = (y_i y_j \langle x_i, x_j \rangle)$, é negativa semidefinida (por ser o negativo de uma matriz de Gram). Como as restrições são afins e qualificadas, temos que o problema dual é uma QP convexa com dualidade forte. Isso garante que a solução ótima do problema dual coincide com a do primal.

Uma vez encontrada a solução α , podemos determinar a hipótese aprendida substituindo a expressão de \mathbf{w} no classificador linear:

$$h(x) = \text{sign}(\langle \mathbf{w}, x \rangle + b) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b\right).$$

Além disso, o valor de b pode ser obtido a partir de qualquer vetor de suporte x_i tal que $\alpha_i > 0$, usando o fato de que para esses pontos $y_i(\langle \mathbf{w}, x_i \rangle + b) = 1$, ou seja:

$$b = y_i - \sum_{j=1}^n \alpha_j y_j \langle x_j, x_i \rangle.$$

As equações acima revelam uma característica importante das SVMs: a hipótese aprendida depende apenas de produtos internos entre exemplos de treinamento e o ponto de teste. Isso será crucial mais adiante, ao introduzirmos o método do kernel.

Também podemos usar a equação anterior para derivar uma expressão da margem geométrica ρ em função das variáveis α . Multiplicando ambos os lados por $\alpha_i y_i$ e somando sobre os vetores de suporte ($\alpha_i > 0$), obtemos:

$$\sum_{i=1}^n \alpha_i y_i b = \sum_{i=1}^n \alpha_i y_i^2 - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle.$$

Como $y_i^2 = 1$ e $\sum_i \alpha_i y_i = 0$, isso implica:

$$0 = \sum_{i=1}^n \alpha_i - \|\mathbf{w}\|^2,$$

ou seja,

$$\|\mathbf{w}\|^2 = \sum_{i=1}^n \alpha_i.$$

Como a margem geométrica é $\rho = \frac{1}{\|\mathbf{w}\|}$, temos:

$$\rho^2 = \frac{1}{\|\mathbf{w}\|^2} = \frac{1}{\sum_{i=1}^n \alpha_i} = \frac{1}{\|\alpha\|_1}.$$

Assim, a margem está inversamente relacionada à norma- L^1 do vetor α .

9.2 Caso não separável

No caso não separável, a suposição de que existe um hiperplano que separa perfeitamente os dados deixa de ser válida. Para contornar isso, introduzimos variáveis de folga (slack variables)

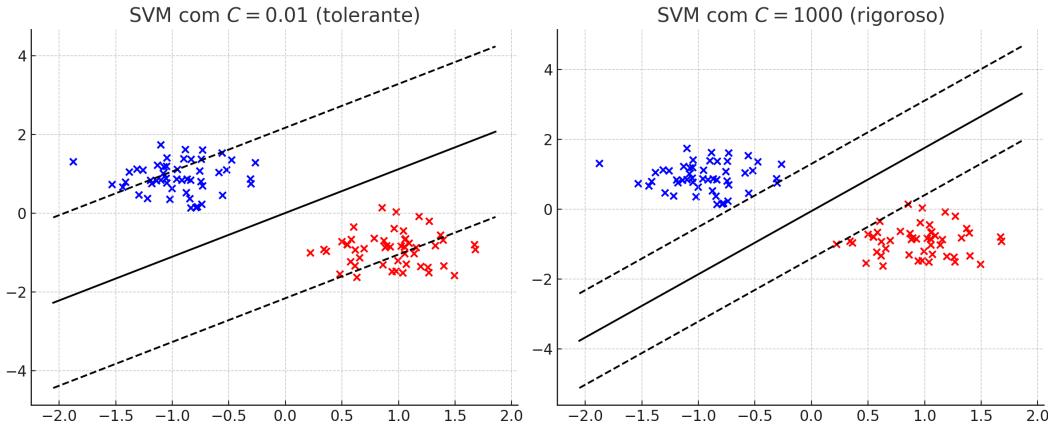
$\xi_i \geq 0$ que permitem violações marginais nas restrições. O problema primal torna-se:

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{sujeito a} \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

No caso não separável, introduzimos variáveis de folga $\xi_i \geq 0$ para permitir que alguns exemplos violem a margem, total ou parcialmente. O termo $\sum \xi_i$ penaliza essas violações, e o parâmetro $C > 0$ controla o equilíbrio entre maximizar a margem e permitir erros de classificação.

O parâmetro C atua como um peso para a penalização de pontos mal classificados ou que ficam dentro da margem. Valores altos de C impõem uma penalização severa a essas violações, forçando o modelo a buscar uma separação quase perfeita dos dados de treinamento — o que pode reduzir o viés, mas aumenta o risco de overfitting, especialmente na presença de ruído. Já valores pequenos de C tornam o modelo mais permissivo a erros, favorecendo margens mais largas e soluções mais regulares, que tendem a generalizar melhor.

Portanto, C é um hiperparâmetro crucial, pois regula diretamente o trade-off entre complexidade do modelo e erro de treinamento. Na prática, seu valor é escolhido por validação cruzada.



A construção do problema dual segue de forma análoga ao caso separável: escrevemos o Lagrangiano, aplicamos as condições KKT e eliminamos as variáveis primais \mathbf{w} , b e agora também as ξ_i . A única modificação estrutural importante no dual é a restrição $0 \leq \alpha_i \leq C$, que substitui a condição $\alpha_i \geq 0$ do caso separável. Isso reflete o fato de que agora há um custo controlado para violações de margem. A função objetivo, a restrição $\sum \alpha_i y_i = 0$, e a expressão da hipótese final mantêm a mesma forma.

9.3 O truque do kernel

Na formulação primal da SVM, o classificador é linear, ou seja, sua fronteira de decisão é um hiperplano no espaço original dos dados. No entanto, a formulação *dual* revela uma estrutura importante: a solução ótima depende apenas de produtos internos entre os dados de treinamento.

Mais precisamente, o classificador obtido a partir da solução dual é da forma:

$$h(x) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b \right),$$

onde os coeficientes α_i são determinados pela solução do problema dual. Note que os dados aparecem apenas por meio de produtos internos $\langle x_i, x \rangle$.

Essa observação permite generalizar a SVM para casos não lineares. A ideia é supor que os dados, embora não separáveis no espaço original \mathbb{R}^p , possam ser linearmente separáveis em um espaço de dimensão maior (possivelmente infinita), obtido por um mapeamento não linear $\phi : \mathbb{R}^p \rightarrow \mathcal{H}$. Em vez de trabalhar diretamente com $\phi(x)$, utilizamos uma função *kernel* $K(x, x')$ que satisfaz:

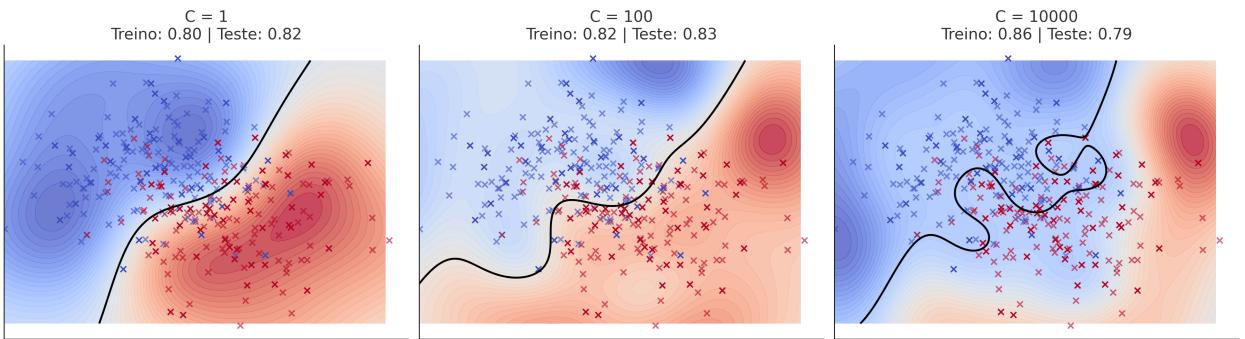
$$K(x, x') = \langle \phi(x), \phi(x') \rangle.$$

Dessa forma, substituímos os produtos internos $\langle x_i, x_j \rangle$ por $K(x_i, x_j)$ no problema dual. Como o dual depende apenas desses produtos, todo o algoritmo passa a operar implicitamente no espaço transformado \mathcal{H} , sem necessidade de calcular $\phi(x)$ explicitamente. Esse artifício é conhecido como **truque do kernel**.

Exemplos clássicos de funções kernel incluem:

- **Linear:** $K(x, x') = \langle x, x' \rangle$
- **Polinomial:** $K(x, x') = (\langle x, x' \rangle + c)^d$
- **Gaussiano (RBF):** $K(x, x') = \exp(-\gamma \|x - x'\|^2)$

Com isso, as SVMs passam a ser capazes de aprender fronteiras de decisão não lineares, mantendo as garantias teóricas e a robustez do caso linear.



9.3.1 Formulações primal e dual com kernel

O uso de kernels não altera conceitualmente o problema primal, mas transforma completamente a formulação dual. No espaço de características \mathcal{H} , o problema primal com slack é:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ \text{sujeito a} \quad & y_i(\langle \mathbf{w}, \phi(x_i) \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

Entretanto, como $\phi(x)$ pode ser de dimensão muito alta, resolvemos o problema dual, onde os dados aparecem apenas via produtos internos. A formulação dual com kernel é:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{sujeito a} \quad & 0 \leq \alpha_i \leq C, \quad \forall i, \\ & \sum_{i=1}^n \alpha_i y_i = 0. \end{aligned}$$

Por fim, o classificador aprendido é dado por:

$$h(x) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i K(x_i, x) + b \right),$$

onde apenas os vetores de suporte (aqueles com $\alpha_i > 0$) contribuem para a decisão final.

Capítulo 10

Redução de dimensão

Em muitos problemas de aprendizado de máquina e análise de dados, lidamos com conjuntos de dados de alta dimensão, nos quais cada observação é descrita por um grande número de variáveis (ou atributos). Embora alta dimensionalidade permita capturar muitos aspectos dos dados, ela também pode trazer dificuldades tanto computacionais quanto estatísticas.

- **Sobreajuste (overfitting):** Em espaços de alta dimensão, é mais fácil encontrar funções que se ajustem perfeitamente aos dados de treinamento, mas Análise de componentes principais que generalizam mal.
- **Custo computacional:** Métodos de aprendizado e visualização podem se tornar inviáveis conforme cresce a dimensionalidade.
- **Ruído e redundância:** Muitas variáveis podem ser irrelevantes ou fortemente correlacionadas, adicionando ruído ao modelo e dificultando sua interpretação.

A redução de dimensionalidade busca transformar os dados para um espaço de menor dimensão preservando, na medida do possível, suas características essenciais.

Neste capítulo, abordamos métodos não supervisionados de redução de dimensionalidade, com ênfase na Análise de Componentes Principais (PCA) e t-SNE.

10.1 Análise de componentes principais

Seja $A \in \mathbb{R}^{p \times p}$ uma matriz simétrica, isto é, tal que $A^\top = A$. Um resultado fundamental da álgebra linear garante que toda matriz simétrica admite uma **decomposição espectral** da forma

$$A = Q\Lambda Q^\top,$$

onde:

- $Q \in \mathbb{R}^{p \times p}$ é uma matriz ortogonal, ou seja, $Q^\top Q = I$;
- $\Lambda \in \mathbb{R}^{p \times p}$ é uma matriz diagonal contendo os autovalores reais de A ;
- As colunas de Q são os autovetores ortonormais de A .

Essa decomposição nos permite interpretar A como uma combinação linear de projeções ao longo de direções ortogonais, ponderadas por seus respectivos autovalores:

$$A = \sum_{i=1}^p \lambda_i q_i q_i^\top,$$

onde λ_i é o i -ésimo autovalor e q_i é o autovetor correspondente.

A decomposição espectral $A = Q\Lambda Q^\top$ de uma matriz simétrica $A \in \mathbb{R}^{p \times p}$ pode ser interpretada geometricamente como a composição de três transformações lineares: uma rotação (ou mudança de base ortogonal), seguida de uma dilatação ao longo dos eixos coordenados, seguida por outra rotação. Especificamente, a matriz Q^\top realiza uma rotação que alinha os vetores do espaço com os autovetores de A , a matriz diagonal Λ aplica dilatações (ou contrações) escalares ao longo dessas direções, e a matriz Q retorna ao sistema original de coordenadas.

Essa interpretação mostra que, a menos de rotações, o comportamento essencial da transformação associada a A é determinado por Λ , ou seja, pelos autovalores. Em particular, os autovalores descrevem quanto uma forma esférica é alongada ou comprimida ao longo das direções associadas a cada autovetor. Portanto, do ponto de vista geométrico, entender a ação de uma matriz simétrica sobre o espaço equivale a entender os valores em Λ , pois as rotações preservam ângulos e distâncias relativas, não alterando a natureza da deformação — apenas sua orientação.

A ideia central da análise de componentes principais (PCA) é aplicar a decomposição espectral a uma matriz simétrica construída a partir dos dados, com o objetivo de entender e simplificar sua estrutura de variabilidade.

Seja $X \in \mathbb{R}^{n \times p}$ a matriz de dados, onde cada linha representa uma observação e cada coluna corresponde a uma variável. Para aplicar o PCA corretamente, é necessário que os dados estejam centralizados — isto é, cada variável deve ter média zero. Na prática, isso é feito subtraindo a média de cada coluna da matriz X . Denotando por $\mu_j = \frac{1}{n} \sum_{i=1}^n X_{ij}$ a média da j -ésima coluna, construímos uma nova matriz $\tilde{X} \in \mathbb{R}^{n \times p}$ cujos elementos são dados por

$$\tilde{X}_{ij} = X_{ij} - \mu_j.$$

Essa centralização garante que a matriz de covariância empírica $S = \frac{1}{n} \tilde{X}^\top \tilde{X}$ capture apenas a variabilidade em torno da média, o que é essencial para que o PCA identifique corretamente as direções principais de variação dos dados.

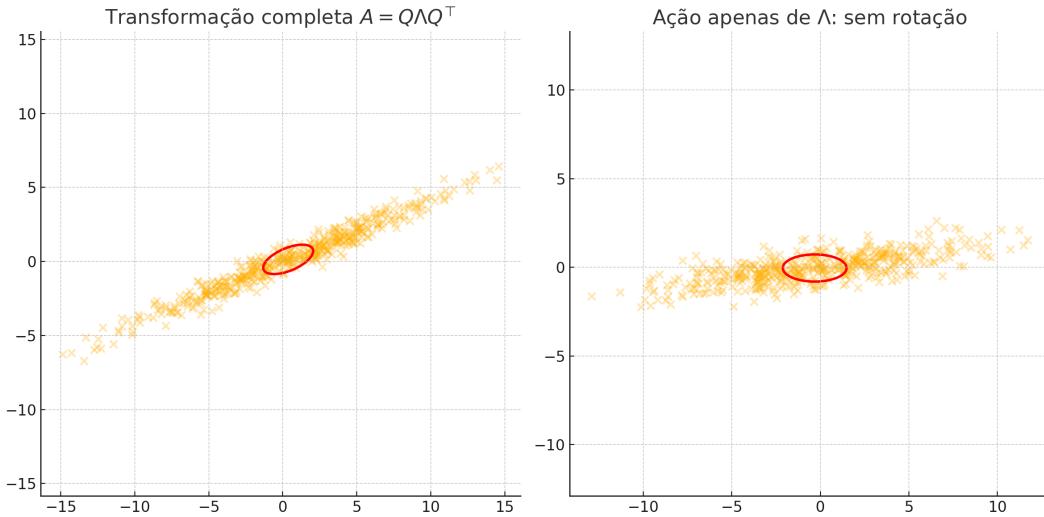
Aplicando a decomposição espectral à matriz S , obtemos

$$S = Q\Lambda Q^\top,$$

onde as colunas de Q são autovetores ortonormais (as *componentes principais*) e os valores em Λ são os autovalores não negativos que indicam a variação dos dados ao longo dessas direções.

Projetar os dados X nas direções dadas pelas colunas de Q resulta em novas variáveis não correlacionadas, ordenadas de forma que as primeiras carregam a maior parte da variação dos dados. Isso nos permite realizar redução de dimensionalidade: ao manter apenas os $k < p$ primeiros autovetores, obtemos uma representação aproximada dos dados que preserva a maior parte de sua variabilidade.

Assim, o PCA pode ser entendido como uma mudança de base ortogonal para uma nova coordenada onde as direções estão alinhadas com as elipses de nível da matriz de covariância. A deformação elíptica observada na decomposição espectral se torna, neste contexto, a ferramenta que revela quais são as direções de maior variação — e, portanto, de maior interesse estatístico — na distribuição dos dados.



Os autovalores obtidos na decomposição espectral da matriz de covariância $S = \frac{1}{n} \tilde{X}^\top \tilde{X}$ representam a quantidade de variância dos dados explicada por cada componente principal. Mais precisamente, se $\lambda_1, \dots, \lambda_p$ são os autovalores ordenados de forma decrescente, então λ_1 indica a variância dos dados ao longo da primeira direção principal (isto é, a direção que maximiza a variância), λ_2 a variância na segunda direção mais importante, e assim por diante.

10.1.1 Variância explicada

Seja $\tilde{X} \in \mathbb{R}^{n \times p}$ a matriz de dados centralizada, isto é, cada coluna tem média zero. A matriz de covariância empírica dos dados é então

$$S = \frac{1}{n} \tilde{X}^\top \tilde{X}.$$

Essa matriz S é simétrica e semi-definida positiva, e representa as covariâncias entre todas as variáveis. Em particular, sua entrada S_{jj} na diagonal é a variância da variável j -ésima:

$$S_{jj} = \frac{1}{n} \sum_{i=1}^n \tilde{X}_{ij}^2 \approx \text{Var}(X_j).$$

O **traço** de S , que é a soma dos elementos diagonais, satisfaz

$$\text{tr}(S) = \sum_{j=1}^p S_{jj} = \sum_{j=1}^p \text{Var}(X_j),$$

ou seja, o traço de S é a soma das variâncias das p variáveis.

Por outro lado, como S é simétrica, ela admite uma decomposição espectral $S = Q\Lambda Q^\top$, onde $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_p)$ contém os autovalores de S . Um fato fundamental da álgebra linear garante

que o traço de uma matriz é igual à soma de seus autovalores:

$$\text{tr}(S) = \sum_{j=1}^p \lambda_j.$$

Portanto, a soma dos autovalores de S é exatamente igual à soma das variâncias das variáveis originais. Isso mostra que os autovalores representam como a variância total dos dados é distribuída entre as diferentes direções principais.

O valor total da variância nos dados é dado pela soma dos autovalores,

$$\text{Var total} = \sum_{j=1}^p \lambda_j,$$

e a proporção da variância explicada pela k -ésima componente principal é

$$\text{Proporção explicada por } k = \frac{\lambda_k}{\sum_{j=1}^p \lambda_j}.$$

Essas proporções fornecem uma medida quantitativa de quanta informação (no sentido de variabilidade dos dados) está concentrada em cada direção principal. Isso permite, por exemplo, decidir quantas componentes manter em uma redução de dimensionalidade: basta escolher o menor k tal que a soma das k primeiras proporções seja suficientemente próxima de 1 (por exemplo, 90% ou 95%).

Essa interpretação dos autovalores como variância explicada torna o PCA não apenas uma ferramenta geométrica, mas também estatística, permitindo compreender e resumir conjuntos de dados de alta dimensão com base em sua estrutura de variação.

10.1.2 Decomposição em valores singulares

Seja $A \in \mathbb{R}^{n \times p}$ uma matriz qualquer. Consideramos a matriz simétrica $A^\top A \in \mathbb{R}^{p \times p}$, que é semi-definida positiva. Como tal, ela admite uma decomposição espectral da forma

$$A^\top A = V \Lambda V^\top,$$

onde $V \in \mathbb{R}^{p \times p}$ é ortogonal e $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_p)$ contém os autovalores reais e não negativos de $A^\top A$, que podemos ordenar como $\lambda_1 \geq \dots \geq \lambda_r > 0 = \lambda_{r+1} = \dots = \lambda_p$, com $r = \text{rank}(A)$.

Definimos os *valores singulares* de A como $\sigma_j = \sqrt{\lambda_j}$ para $j = 1, \dots, r$. Os vetores $v_j \in \mathbb{R}^p$ (colunas de V) satisfazem

$$A^\top A v_j = \lambda_j v_j.$$

Multiplicando ambos os lados por A , obtemos:

$$A A^\top (A v_j) = A (A^\top A v_j) = \lambda_j A v_j.$$

Ou seja, $A v_j$ é um autovetor de $A A^\top$ associado ao mesmo autovalor λ_j . Definimos

$$u_j = \frac{A v_j}{\sigma_j},$$

o que é bem definido pois $\sigma_j > 0$. Assim, os vetores $u_j \in \mathbb{R}^n$ têm norma unitária e satisfazem:

$$Av_j = \sigma_j u_j \quad \text{e} \quad A^\top u_j = \sigma_j v_j.$$

Tomando $U_r = [u_1 \ \cdots \ u_r] \in \mathbb{R}^{n \times r}$, $V_r = [v_1 \ \cdots \ v_r] \in \mathbb{R}^{p \times r}$ e $\Sigma_r = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{r \times r}$, temos a fatoração

$$A = U_r \Sigma_r V_r^\top.$$

Essa é a forma reduzida da decomposição em valores singulares (SVD). Podemos completá-la para obter a SVD completa estendendo U_r e V_r para bases ortonormais completas $U \in \mathbb{R}^{n \times n}$ e $V \in \mathbb{R}^{p \times p}$, e definindo $\Sigma \in \mathbb{R}^{n \times p}$ como a matriz retangular que contém Σ_r no canto superior esquerdo e zeros no restante. Com isso, temos:

$$A = U \Sigma V^\top.$$

Logo, a SVD pode ser rigorosamente derivada a partir da decomposição espectral de $A^\top A$ e da relação $AA^\top(Av_j) = \lambda_j Av_j$, o que mostra que os vetores Av_j também são autovetores de AA^\top .

Intuição geométrica

Consideramos uma matriz real $A \in \mathbb{R}^{n \times p}$. A imagem da esfera unitária $S = \{x \in \mathbb{R}^p : \|x\| = 1\}$ sob a transformação linear A é uma superfície chamada *iperelipse* em \mathbb{R}^n . Essa superfície é obtida ao esticar a esfera unitária em até $r = \text{rank}(A)$ direções ortogonais. As direções e os fatores de estiramento são determinados pelos *valores singulares* de A .

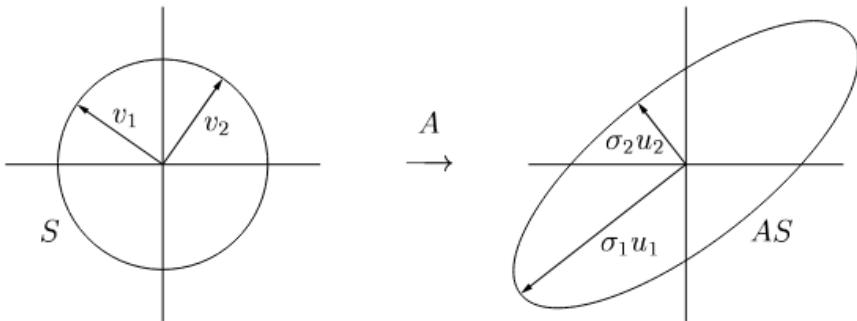


Figure 4.1. SVD of a 2×2 matrix.

Figura 10.1: Retirado de ([Trefethen and Bau, 1997](#)).

Mais precisamente, os valores singulares $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ correspondem aos comprimentos dos *semieixos principais* da hiperelipse. Para cada $j = 1, \dots, r$, temos que $Av_j = \sigma_j u_j$, onde:

- $v_j \in \mathbb{R}^p$ é o j -ésimo **vetor singular à direita**, ou seja, uma direção unitária na entrada que é alongada por A ;

- $u_j \in \mathbb{R}^n$ é o j -ésimo **vetor singular à esquerda**, que indica a direção de saída correspondente.

Essas relações podem ser organizadas matricialmente como

$$AV_r = U_r \Sigma_r,$$

onde:

- $V_r = [v_1 \ \cdots \ v_r] \in \mathbb{R}^{p \times r}$,
- $U_r = [u_1 \ \cdots \ u_r] \in \mathbb{R}^{n \times r}$,
- $\Sigma_r = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{r \times r}$.

Como V_r tem colunas ortonormais, podemos escrever

$$A = U_r \Sigma_r V_r^\top,$$

que é a forma reduzida da decomposição em valores singulares (SVD). A forma completa é obtida ao estender U_r e V_r para bases ortonormais completas:

$$A = U \Sigma V^\top,$$

onde:

- $U \in \mathbb{R}^{n \times n}$ é ortogonal (isto é, $U^\top U = I$);
- $V \in \mathbb{R}^{p \times p}$ é ortogonal;
- $\Sigma \in \mathbb{R}^{n \times p}$ é uma matriz diagonal retangular, com os valores singulares $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ nas primeiras entradas da diagonal, e zeros no restante.

Teorema 4 (Teorema da existência e unicidade da SVD). *Toda matriz $A \in \mathbb{R}^{n \times p}$ admite uma decomposição da forma $A = U \Sigma V^\top$. Os valores singulares σ_j são unicamente determinados. Se os valores singulares forem distintos, então os vetores singulares à esquerda u_j e à direita v_j também são unicamente determinados a menos de sinais.*

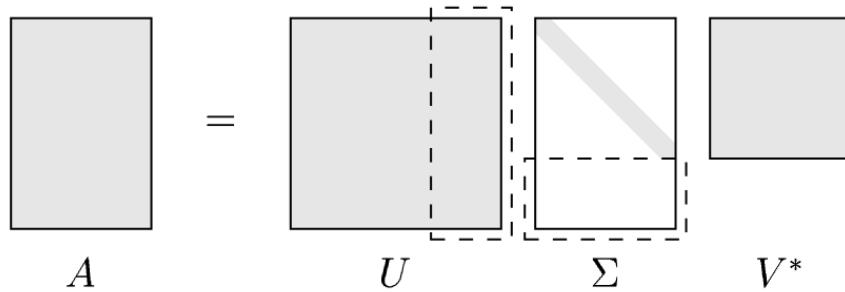


Figura 10.2: Ilustração do SVD. Retirado de ([Trefethen and Bau, 1997](#)).

Exercício 16. Prove que o posto de uma matriz A é igual ao número r de valores singulares não nulos.

Exercício 17. Prove que a imagem de uma matriz $\text{Im}(A) = \langle u_1, \dots, u_r \rangle$ e seu núcleo $N(A) = \langle v_{r+1}, \dots, v_r \rangle$.

Exercício 18. Encontre uma expressão para $N(A)^\perp$ e uma para $\text{Im}(A^\top)$ utilizando a decomposição SVD de A .

SVD e PCA

Seja $X \in \mathbb{R}^{n \times p}$ uma matriz de dados, onde cada linha corresponde a uma observação e cada coluna a uma variável. Suponha que X já foi centralizada, ou seja, a média de cada coluna é zero.

A matriz de covariância empírica dos dados é dada por

$$S = \frac{1}{n} X^\top X.$$

Como S é simétrica e semi-definida positiva, ela admite uma decomposição espectral da forma $S = V\Lambda V^\top$, onde as colunas de $V \in \mathbb{R}^{p \times p}$ são os autovetores de S , chamados de *componentes principais*, e $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_p)$ contém os autovalores não negativos correspondentes, que representam a variância dos dados ao longo dessas direções.

Por outro lado, podemos aplicar a decomposição em valores singulares (SVD) diretamente à matriz X :

$$X = U\Sigma V^\top,$$

com $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{n \times p}$, onde σ_j são os valores singulares de X , e $r = \text{rank}(X)$. Então:

$$X^\top X = V\Sigma^\top \Sigma V^\top,$$

de modo que:

$$S = \frac{1}{n} X^\top X = V \left(\frac{1}{n} \Sigma^\top \Sigma \right) V^\top.$$

Isso mostra que os autovalores de S são $\lambda_j = \sigma_j^2/n$, e os autovetores de S são exatamente os vetores singulares à direita de X , isto é, as colunas de V . Portanto, a análise de componentes principais pode ser realizada a partir da SVD da matriz de dados centralizada: os vetores principais são os v_j , e as variâncias explicadas correspondem aos quadrados dos valores singulares normalizados por n .

Aproximações de posto baixo

Seja $A \in \mathbb{R}^{n \times p}$ e considere sua decomposição em valores singulares:

$$A = \sum_{j=1}^r \sigma_j u_j v_j^\top,$$

onde $r = \text{rank}(A)$, os vetores $u_j \in \mathbb{R}^n$ e $v_j \in \mathbb{R}^p$ são ortonormais, e os valores singulares satisfazem $\sigma_1 \geq \dots \geq \sigma_r > 0$. Para cada $\nu \in \{0, 1, \dots, r\}$, define-se a aproximação de posto ν como

$$A_\nu = \sum_{j=1}^{\nu} \sigma_j u_j v_j^\top.$$

Teorema 5. Seja $A \in \mathbb{R}^{n \times p}$ uma matriz qualquer, e A_ν a aproximação de posto ν definida acima. Então, para todo ν , temos:

$$\|A - A_\nu\|_2 = \inf_{\substack{B \in \mathbb{R}^{n \times p} \\ \text{rank}(B) \leq \nu}} \|A - B\|_2 = \sigma_{\nu+1},$$

com a convenção de que $\sigma_{\nu+1} = 0$ se $\nu = \min(n, p)$.

Esse resultado mostra que a melhor aproximação de A por uma matriz de posto no máximo ν , no sentido da norma espectral, é obtida truncando sua decomposição em valores singulares e mantendo apenas os ν primeiros termos. Além disso, o erro de aproximação é exatamente o próximo valor singular descartado, ou seja, a maior deformação não capturada por A_ν .

Quando $A = X$ é uma matriz de dados centralizada, com $X \in \mathbb{R}^{n \times p}$, a decomposição $X = U\Sigma V^\top$ pode ser usada para obter as componentes principais (PCA). O truncamento da SVD em ν termos fornece a matriz $X_\nu = U_\nu \Sigma_\nu V_\nu^\top$, que é a projeção dos dados nas ν primeiras direções principais. O teorema acima garante que X_ν é a melhor representação possível dos dados entre todas as matrizes de posto ν , no sentido de minimizar a norma espectral do erro:

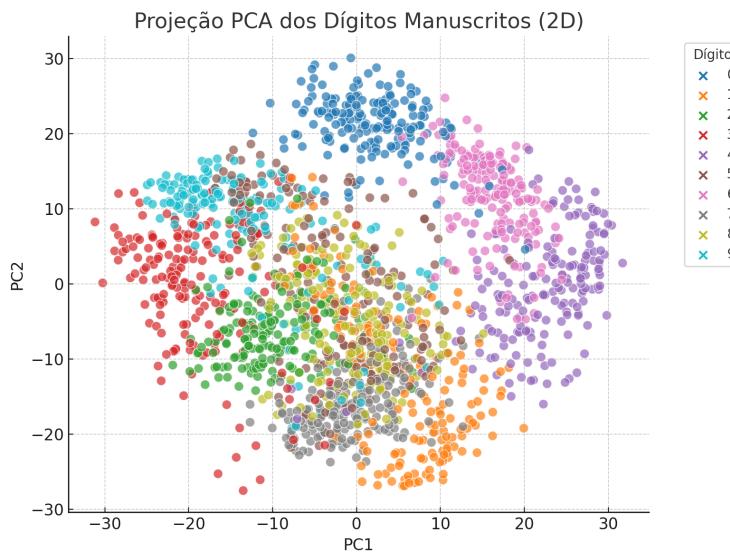
$$\|X - X_\nu\|_2 = \sigma_{\nu+1}.$$

Essa propriedade justifica o uso do PCA como uma técnica de redução de dimensionalidade com controle explícito sobre a perda de informação.

10.1.3 Aplicação: redução de dimensão

Como exemplo prático de aplicação da análise de componentes principais (PCA), consideramos o conjunto de dados `digits`, disponível na biblioteca `scikit-learn`. Esse conjunto contém imagens de dígitos manuscritos de 0 a 9, cada uma representada por uma matriz 8×8 de pixels, ou seja, um vetor em \mathbb{R}^{64} .

A matriz de dados $X \in \mathbb{R}^{1797 \times 64}$ contém $n = 1797$ observações, cada uma correspondente a uma imagem achatada com $p = 64$ variáveis. Aplicamos o PCA a X após centralizar suas colunas, e projetamos os dados nas duas primeiras componentes principais.



O resultado é uma nuvem de pontos em \mathbb{R}^2 , onde cada ponto corresponde a uma imagem original e está colorido de acordo com o dígito representado. Observamos que, mesmo após a redução para duas dimensões, as diferentes classes de dígitos ainda apresentam agrupamentos visivelmente distintos. Isso evidencia que as duas primeiras componentes principais capturaram uma quantidade substancial da estrutura de variabilidade dos dados.

Essa aplicação ilustra como o PCA pode ser utilizado para:

- Reduzir a dimensionalidade de dados de alta dimensão com mínima perda de informação;
- Visualizar a estrutura interna dos dados em duas ou três dimensões;
- Explorar padrões e possíveis agrupamentos em dados não rotulados.

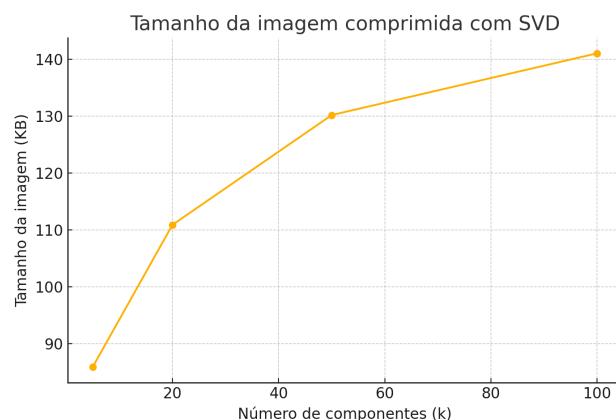
10.1.4 Aplicação: compressão de dados

Para ilustrar uma aplicação prática da decomposição SVD, utilizamos uma imagem em tons de cinza e reconstruímos aproximações de posto reduzido variando o número de componentes k . A cada valor de k , reconstruímos a imagem utilizando apenas os k maiores valores singulares e seus vetores associados. O objetivo é verificar até que ponto é possível reduzir a quantidade de informação armazenada sem comprometer significativamente a qualidade visual da imagem.



Cada imagem reconstruída foi salva no formato PNG, e em seguida medimos seu tamanho em kilobytes. Isso nos permite avaliar empiricamente o impacto de k no espaço ocupado em disco, levando em conta também a compactação realizada pelo formato PNG. Além disso, comparamos esses tamanhos com o da imagem original, também salva em PNG.

Os resultados mostram que para valores baixos de k , a imagem perde detalhes, mas o tamanho do arquivo é drasticamente reduzido — evidenciando compressão com perda. À medida que aumentamos k , a imagem se aproxima da original em termos visuais, e o tamanho do arquivo cresce progressivamente. A curva gerada revela o ponto a partir do qual adicionar mais componentes singulares tem impacto marginal na qualidade visual, mas aumenta o tamanho do arquivo.



Essa aplicação demonstra como a SVD pode ser usada como um método de compressão controlável: quanto menor o k , maior a compressão (com mais perda de informação), e quanto maior o k , melhor a qualidade visual, com custo maior de armazenamento.

10.2 Projeções aleatórias

Seja $U \in \mathbb{R}^{p \times p}$ uma matriz ortogonal, ou seja, satisfaça $U^\top U = I$. Nesse caso, para qualquer vetor $x \in \mathbb{R}^p$, temos:

$$\|Ux\|^2 = x^\top U^\top Ux = x^\top x = \|x\|^2.$$

Ou seja, transformações lineares ortogonais preservam normas e ângulos. Geometricamente, isso significa que U é uma rotação (ou rotação seguida de reflexão) do espaço, e não altera distâncias entre pontos.

Essa propriedade sugere que, ao projetar dados em uma base ortonormal (ou aproximadamente ortonormal), podemos manter a estrutura geométrica essencial. Mais ainda, em espaços de alta dimensão, ocorre um fenômeno conhecido como *concentração da medida*, que implica que, com alta probabilidade, dois vetores $x, y \in \mathbb{R}^p$ escolhidos ao acaso estarão quase ortogonais, ou seja, o cosseno do ângulo entre eles será próximo de zero. Intuitivamente, isso significa que, à medida que a dimensão cresce, o espaço se torna "mais esférico", e vetores aleatórios tendem a se distribuir de forma aproximadamente ortogonal.

A preservação de normas por matrizes ortogonais motiva a busca por projeções lineares que preservem, ao menos aproximadamente, distâncias euclidianas. De fato, se $U \in \mathbb{R}^{p \times p}$ é ortogonal, então $\|Ux\| = \|x\|$ para todo $x \in \mathbb{R}^p$, e $\|Ux - Uy\| = \|x - y\|$ para todos os pares x, y , ou seja, a transformação preserva completamente a geometria euclidiana.

Esse tipo de preservação exata só é possível quando a projeção acontece em todo o espaço \mathbb{R}^p , mas uma pergunta natural é se existe alguma forma de preservar aproximadamente as distâncias quando projetamos para um subespaço de dimensão $k \ll p$.

Intuitivamente, a resposta é positiva se nos restringirmos a um conjunto finito de vetores. A ideia central é que, em alta dimensão, vetores aleatórios tendem a ser quase ortogonais. Mais precisamente, se sorteamos dois vetores $x, y \in \mathbb{R}^p$ de maneira independente, com entradas $\sim \mathcal{N}(0, 1)$, então o ângulo entre eles tende a ser próximo de $\pi/2$, e a norma de $x - y$ concentra em torno de $\sqrt{2p}$. Esse fenômeno de *concentração da medida* sugere que vetores aleatórios bem escolhidos podem servir como base aproximadamente ortonormal com alta probabilidade.

Esse insight leva à ideia de gerar uma matriz aleatória $R \in \mathbb{R}^{k \times p}$ com entradas i.i.d. $\mathcal{N}(0, 1)$, e definir a aplicação linear

$$f(x) = \frac{1}{\sqrt{k}} Rx.$$

O fator $\frac{1}{\sqrt{k}}$ garante que a variância da norma de $f(x)$ não cresça artificialmente com k . Podemos então perguntar: essa projeção aleatória f preserva as distâncias entre vetores de maneira controlada?

A resposta é afirmativa, como garantido pelo seguinte resultado:

Teorema 6. Seja $0 < \varepsilon < 1$ e $n \in \mathbb{N}$. Para qualquer conjunto fixo de n vetores $x_1, \dots, x_n \in \mathbb{R}^p$, existe uma aplicação linear $f : \mathbb{R}^p \rightarrow \mathbb{R}^k$ com

$$k \geq C \frac{\log n}{\varepsilon^2}$$

para uma constante absoluta C , tal que, para todos os pares $i, j \in \{1, \dots, n\}$,

$$(1 - \varepsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \varepsilon) \|x_i - x_j\|^2.$$

Além disso, tal aplicação pode ser escolhida como uma projeção aleatória do tipo $f(x) = \frac{1}{\sqrt{k}} Rx$, onde $R \in \mathbb{R}^{k \times p}$ tem entradas independentes $\mathcal{N}(0, 1)$.

Esse resultado mostra que é possível reduzir a dimensionalidade de um conjunto finito de pontos de alta dimensão para um espaço de dimensão apenas $O(\log n)$, mantendo as distâncias aproximadamente preservadas com erro relativo controlado por ε . Essa técnica é conhecida como *projeção aleatória*, e tem aplicações em diversas áreas, como:

- Compressão de dados de alta dimensão;
- Aceleração de algoritmos baseados em distâncias, como k -NN;
- Redução de custo computacional em aprendizado de máquina;
- Preservação de estrutura geométrica em embeddings.

Na prática, a matriz R pode ser gerada uma única vez, aplicada a todos os vetores de interesse, e fornece uma transformação linear eficiente e universal. Embora a projeção seja aleatória, o resultado acima garante que, com alta probabilidade, ela funcionará corretamente — desde que a dimensão do subespaço k seja suficientemente grande em relação a $\log n$.

10.2.1 Aplicação

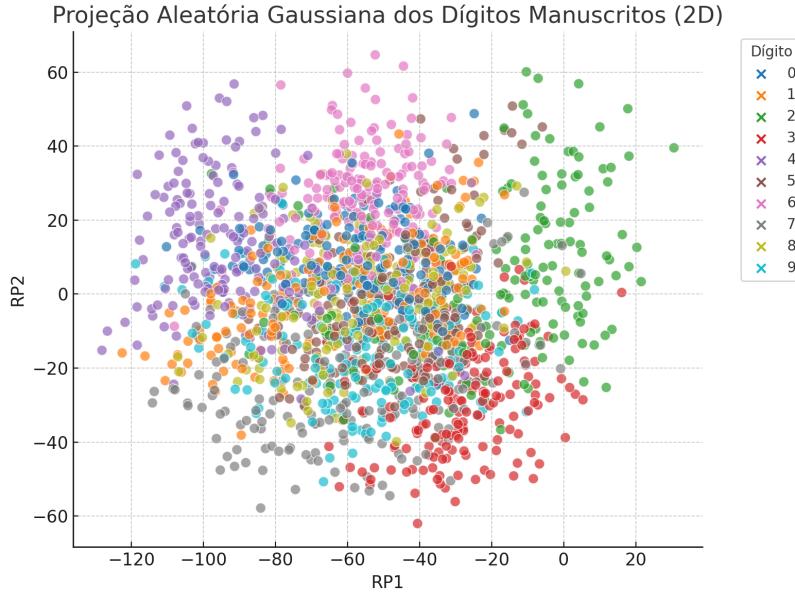
Como exemplo prático da aplicação do teorema de preservação de distâncias por projeções aleatórias, consideramos novamente o conjunto de dados *digits*, composto por imagens de dígitos manuscritos, cada uma representada por um vetor em \mathbb{R}^{64} . Nosso objetivo é reduzir a dimensionalidade desses dados para \mathbb{R}^2 , preservando aproximadamente a estrutura geométrica do conjunto, mas agora utilizando uma transformação *aleatória*.

Construímos uma projeção aleatória $f : \mathbb{R}^{64} \rightarrow \mathbb{R}^2$ da forma

$$f(x) = \frac{1}{\sqrt{2}} Rx,$$

onde $R \in \mathbb{R}^{2 \times 64}$ tem entradas independentes $R_{ij} \sim \mathcal{N}(0, 1)$. Cada vetor $x \in \mathbb{R}^{64}$ é então projetado em um vetor $f(x) \in \mathbb{R}^2$, e essa transformação é aplicada a todas as amostras do conjunto.

O resultado é uma nuvem de pontos em duas dimensões, onde cada ponto representa uma imagem projetada e é colorido de acordo com o dígito correspondente. Apesar da projeção ser inteiramente aleatória, ainda é possível observar agrupamentos e certa separação entre as diferentes classes, o que evidencia que as relações geométricas entre os dados foram, em parte, preservadas.



Esse experimento ilustra a aplicabilidade prática do teorema de preservação de distâncias: mesmo com uma redução drástica da dimensão de 64 para 2, a projeção aleatória preserva, com razoável fidelidade, a estrutura global dos dados. Isso torna tais projeções úteis em tarefas como visualização, aceleração de algoritmos e pré-processamento para métodos de aprendizado de máquina em ambientes de alta dimensão.

10.3 t-SNE

Uma técnica bastante utilizada para visualização de dados de alta dimensão é o *t-distributed Stochastic Neighbor Embedding* (t-SNE). Ao contrário do PCA e das projeções aleatórias, que são transformações lineares, o t-SNE é um método *não linear*, projetando dados de alta dimensão em duas ou três dimensões de modo que as relações de vizinhança local entre os pontos sejam preservadas.

A ideia central do t-SNE é transformar distâncias euclidianas entre pontos de alta dimensão em probabilidades que representam a semelhança entre pares de pontos. Mais precisamente, dados pontos $x_1, \dots, x_n \in \mathbb{R}^p$, o algoritmo define, para cada par (i, j) , a probabilidade condicional

$$P_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)},$$

onde σ_i é ajustado automaticamente de modo que a entropia da distribuição $P_{j|i}$ reflita uma perplexidade pré-definida. A matriz P de semelhanças simétricas é então definida por:

$$P_{ij} = \frac{P_{j|i} + P_{i|j}}{2n}.$$

No espaço de baixa dimensão, $y_1, \dots, y_n \in \mathbb{R}^2$, o t-SNE define uma distribuição de semelhança Q_{ij} entre pares usando uma distribuição de cauda mais pesada (distribuição de Student com um

grau de liberdade, também chamada de Cauchy):

$$Q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

O algoritmo então encontra os vetores $y_i \in \mathbb{R}^2$ que minimizam a divergência de Kullback–Leibler entre as distribuições P e Q :

$$\text{KL}(P\|Q) = \sum_{i \neq j} P_{ij} \log \left(\frac{P_{ij}}{Q_{ij}} \right).$$

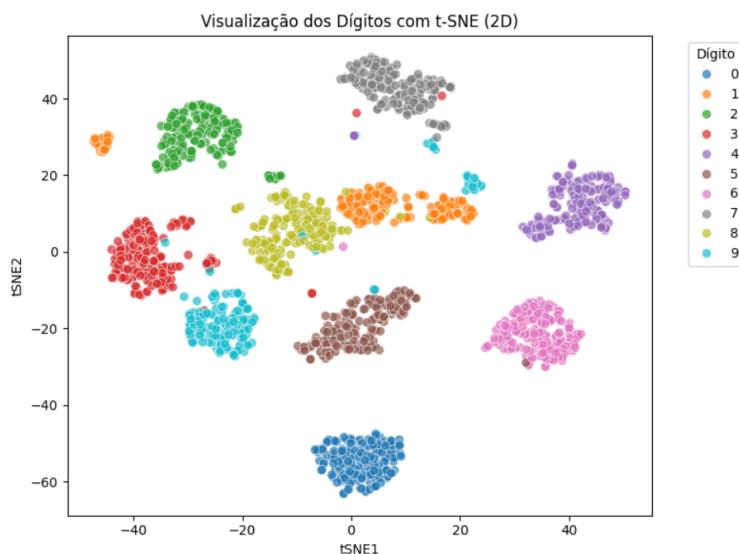
Como o foco do t-SNE é preservar *relações locais*, ou seja, quais pontos são próximos uns dos outros, ele é especialmente eficaz para revelar agrupamentos (clusters) e separações em dados que estão em variedades não lineares. Em contrapartida, o método não tenta preservar distâncias globais — por isso, as distâncias entre grupos no espaço reduzido podem não refletir relações verdadeiras no espaço original.

Dessa forma, o t-SNE é particularmente útil como ferramenta de visualização exploratória, ajudando a identificar estruturas internas nos dados, mesmo quando essas estruturas não são acessíveis por métodos lineares como o PCA.

10.3.1 Aplicação 1

Aplicamos o método t-SNE ao conjunto de dados digits, composto por imagens de dígitos manuscritos representadas por vetores em \mathbb{R}^{64} . O objetivo foi reduzir essa representação para duas dimensões de forma que as relações de vizinhança entre os pontos fossem preservadas.

Ao contrário de técnicas lineares como o PCA, o t-SNE busca manter as proximidades locais entre os dados. Ele constrói uma distribuição de probabilidades que mede a similaridade entre pares de pontos no espaço original e outra no espaço reduzido, e encontra uma projeção que minimiza a divergência entre essas duas distribuições.



O resultado é uma visualização bidimensional onde os pontos associados a diferentes dígitos tendem a formar agrupamentos bem definidos. Isso mostra que, mesmo sem conhecer os rótulos,

o t-SNE consegue capturar a estrutura interna dos dados e separar classes de forma qualitativa. Essa abordagem é especialmente útil para visualização exploratória de dados de alta dimensão.

Capítulo 11

K-Médias

O agrupamento por K -médias é uma abordagem simples e elegante para particionar um conjunto de dados em K grupos distintos e não sobrepostos. Para aplicar o K -médias, é necessário especificar previamente o número de grupos K ; então o algoritmo atribui cada observação exatamente a um dos K grupos. A Figura 12.7 mostra os resultados obtidos ao aplicar o método de K -médias em um conjunto simulado com 150 observações em duas dimensões, utilizando três valores diferentes de K .

O procedimento de agrupamento por K -médias decorre de um problema matemático simples e intuitivo. Começamos definindo a notação. Seja C_1, \dots, C_K o conjunto dos índices das observações pertencentes a cada grupo. Estes conjuntos satisfazem duas propriedades:

- $C_1 \cup C_2 \cup \dots \cup C_K = \{1, \dots, n\}$. Ou seja, cada observação pertence a pelo menos um dos K grupos.
- $C_k \cap C_{k'} = \emptyset$ para todo $k \neq k'$. Em outras palavras, os grupos são não sobrepostos: nenhuma observação pertence a mais de um grupo.

Se a i -ésima observação pertence ao grupo k , então $i \in C_k$. A ideia por trás do agrupamento por K -médias é que um bom agrupamento é aquele em que a variação intra-grupo é a menor possível. A variação intra-grupo para o grupo C_k é uma medida $W(C_k)$ da diferença entre as observações dentro do grupo. Assim, queremos resolver o seguinte problema de otimização:

$$\min_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K W(C_k) \right\}. \quad (11.1)$$

Em palavras, a fórmula acima diz que queremos partitionar as observações em K grupos de modo que a variação total intra-grupo, somada em todos os K grupos, seja a menor possível.

Resolver o problema acima parece uma ideia razoável, mas para torná-la executável, precisamos definir precisamente a variação intra-grupo. Existem diversas maneiras possíveis de fazer isso, mas a mais comum é usar a *distância euclidiana ao quadrado*. Isto é, definimos:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2, \quad (11.2)$$

onde $|C_k|$ denota o número de observações no grupo k . Em outras palavras, a variação intra-grupo para o grupo k é a soma de todas as distâncias euclidianas ao quadrado entre pares de observações dentro do grupo k , dividida pelo número total de observações no grupo.

Combinando as equações anteriores, obtemos o problema de otimização que define o agrupamento por K -médias:

$$\min_{C_1, \dots, C_K} \left\{ \sum_{k=1}^K \frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right\}. \quad (11.3)$$

Esse problema de otimização visa partitionar as observações em K grupos de modo que a soma total da variação intra-grupo (em todos os grupos) seja a menor possível. No entanto, resolver esse problema diretamente é muito difícil, pois há aproximadamente K^n maneiras de partitionar n observações em K grupos.

Felizmente, existe um algoritmo simples que encontra uma boa solução local — uma *boa solução*, embora não necessariamente ótima — para o problema. Esse método é apresentado no Algoritmo:

Algorithm 8 K-médias

1. Atribua aleatoriamente um número de 1 a K para cada observação (isto é, atribuições iniciais de grupo).
 2. Repita até que as atribuições de grupo não mudem mais:
 - (a) Para cada um dos K grupos, calcule o *centróide*, isto é, o vetor de médias das p variáveis dentro do grupo.
 - (b) Atribua cada observação ao grupo cujo centróide está mais próximo (usando distância euclidiana).
-

O algoritmo acima garante que o valor da função objetivo irá diminuir a cada passo. Isso pode ser entendido por meio da seguinte identidade:

$$\frac{1}{|C_k|} \sum_{i, i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^p (x_{ij} - \bar{x}_{kj})^2, \quad (11.4)$$

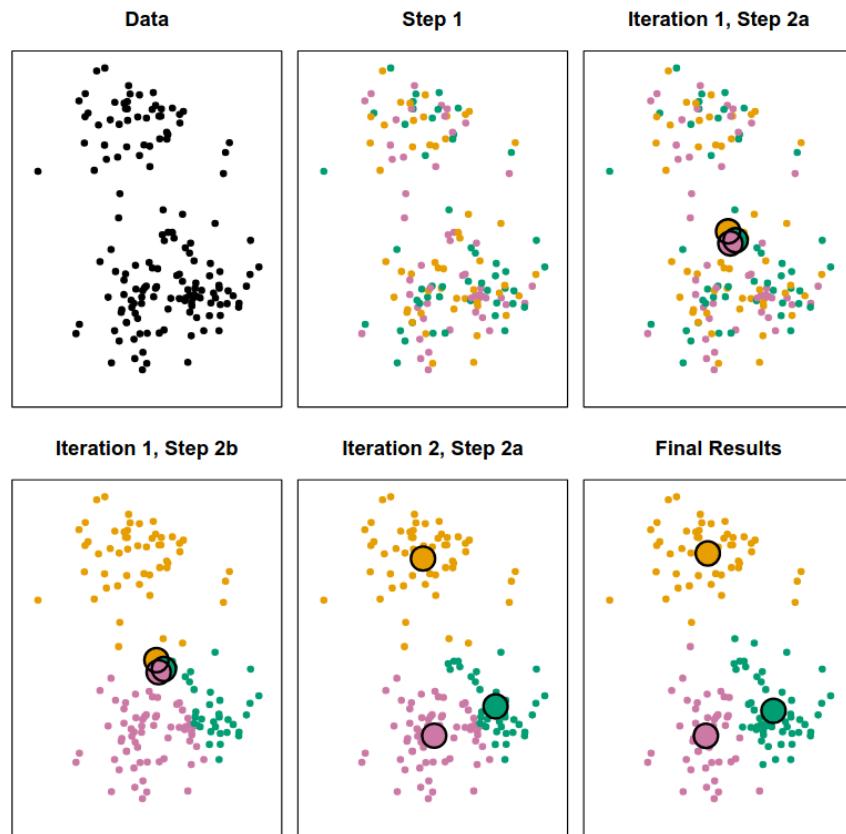
onde $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$ é a média da j -ésima variável no grupo C_k .

No passo 2(a), os centróides de cada grupo são essas médias que minimizam a soma dos desvios ao quadrado; e no passo 2(b), ao realocar as observações para os grupos mais próximos, o valor da função objetivo só pode diminuir.

Isso significa que, ao executar o algoritmo, o agrupamento obtido vai melhorar continuamente até que o resultado pare de mudar. Quando isso acontece, atingimos um *mínimo local*, ou seja, a função objetivo não irá mais diminuir.

O nome K -médias vem do fato de que, no passo 2(a), os centróides dos grupos são calculados como a média das observações atribuídas a cada grupo.

Como o algoritmo encontra apenas um mínimo local e não global, o resultado obtido pode depender bastante da atribuição inicial aleatória dos grupos (passo 1). Por isso, é importante executar o algoritmo várias vezes com diferentes configurações iniciais e escolher a melhor solução (aquele com menor valor da função objetivo).



Apêndice A

Revisão

Nesta seção, faremos uma breve revisão de alguns conceitos matemáticos importantes.

A.1 Álgebra linear

Ao longo deste material, adotaremos a seguinte notação:

- n : número de observações.
- p : número de variáveis preditoras.
- x_{ij} : valor da j -ésima variável na i -ésima observação, com $i = 1, \dots, n$ e $j = 1, \dots, p$.

Representamos os dados como uma matriz $X \in \mathbb{R}^{n \times p}$:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}.$$

Cada linha de X é um vetor $x_i \in \mathbb{R}^p$, representando as variáveis da i -ésima observação:

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}.$$

Também podemos considerar as colunas de X , escritas como $x_j \in \mathbb{R}^n$:

$$x_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}.$$

Assim, a matriz X pode ser expressa de duas formas:

$$X = (x_1 \ x_2 \ \cdots \ x_p).$$

Também podemos considerar as linhas de X , escritas como $x^i \in \mathbb{R}^p$:

$$x^i = \begin{pmatrix} x_{i1} & x_{i2} & \cdots & x_{ip} \end{pmatrix}.$$

Assim, a matriz X pode ser expressa de duas formas:

$$X = \begin{pmatrix} x^1 \\ x^2 \\ \vdots \\ x^n \end{pmatrix}.$$

O símbolo T representa a transposta de vetores ou matrizes, por exemplo:

$$X^T = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{np} \end{pmatrix}.$$

Denotamos a variável resposta (ou target) por y_i , para a i -ésima observação. O vetor completo de respostas é:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

O conjunto de dados observados é formado por pares $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$.

Exercício 19. Considere o conjunto de dados de salários, exemplificado abaixo:

	year	age	marital	race	education	region	jobclass	health	health_ins	logwage	wage
0	2006	18	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	75.043154
1	2004	24	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	70.476020
2	2003	45	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	130.982177
3	2003	43	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	154.685293
4	2005	50	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	75.043154
...
2995	2008	44	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	5.041393	154.685293
2996	2007	30	2. Married	1. White	2. HS Grad	2. Middle Atlantic	1. Industrial	2. >=Very Good	2. No	4.602060	99.689464
2997	2005	27	2. Married	2. Black	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.193125	66.229408
2998	2005	27	1. Never Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	4.477121	87.981033
2999	2009	55	5. Separated	1. White	2. HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.505150	90.481913

3000 rows × 11 columns

Descreva quem é a matriz de dados X , quem é n , quem é p , quem é o vetor resposta \mathbf{y} . Dica: tem uma pegadinha.

A.1.1 Multiplicações

Nessa seção, vamos estudar fatos importantes sobre multiplicações envolvendo matrizes. Para mais detalhes, o leitor pode ver o excelente livro [Trefethen and Bau \(1997\)](#).

Matriz-vetor

Seja x_j a j -ésima coluna de X , um n -vetor. Então, a equação $y = Xb$ pode ser reescrito como:

$$y = Xb = \sum_{j=1}^p x_j b_j. \quad (\text{A.1})$$

Essa equação pode ser representada esquematicamente da seguinte forma:

$$\begin{bmatrix} y \\ \vdots \\ y \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_p \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} = b_1 \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} + b_2 \begin{bmatrix} x_2 \\ \vdots \\ x_p \end{bmatrix} + \cdots + b_p \begin{bmatrix} x_p \\ \vdots \\ x_p \end{bmatrix}.$$

Na equação acima, y é expresso como uma combinação linear das colunas de X . Desse forma, podemos resumir essas diferentes descrições do produto matriz-vetor da seguinte forma. Como matemáticos, estamos acostumados a interpretar a fórmula $Xb = y$ como uma afirmação de que X age sobre b para produzir y . A forma acima, por outro lado, sugere a interpretação de que b age sobre X para produzir y .

Matriz-Matriz

Para o produto matriz-matriz $B = AC$, cada coluna de B é uma combinação linear das colunas de A . Para demonstrar esse fato, começamos com a fórmula usual para produtos de matrizes. Se A é uma matriz de dimensão $\ell \times n$ e C é de dimensão $n \times p$, então B será de dimensão $\ell \times p$, com entradas definidas por

$$B_{ij} = \sum_{k=1}^n A_{ik} C_{kj}. \quad (\text{A.2})$$

Aqui, B_{ij} , A_{ik} e C_{kj} são elementos de B , A e C , respectivamente. Escrito em termos de colunas, o produto é

$$\begin{bmatrix} B_1 & B_2 & \cdots & B_n \end{bmatrix} = A \begin{bmatrix} C_1 & C_2 & \cdots & C_n \end{bmatrix},$$

que implica em:

$$B_j = AC_j = \sum_{k=1}^m C_{kj} A_k. \quad (\text{A.3})$$

Note que isso é só uma generalização da multiplicação anterior, já que $B_j = AC_j$ e podemos utilizar a formulação Matriz-Vetor da seção anterior.

Um exemplo simples de um produto matriz-matriz é o *produto externo*. Este é o produto de um vetor coluna u de dimensão n com um vetor linha v de dimensão p ; o resultado é uma matriz $n \times p$ de posto 1. O produto externo pode ser escrito como:

$$\begin{bmatrix} u \\ v_1 & v_2 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} v_1 u & v_2 u & \cdots & v_n u \end{bmatrix} = \begin{bmatrix} v_1 u_1 & \cdots & v_n u_1 \\ \vdots & \ddots & \vdots \\ v_1 u_m & \cdots & v_n u_m \end{bmatrix}.$$

As colunas são todas múltiplos do mesmo vetor u e, da mesma forma, as linhas são todas múltiplos do mesmo vetor v .

A.1.2 Mudança de base

Ao escrever o produto $b = X^{-1}y$, é importante não deixar que a notação de matriz inversa obscureça o que realmente está acontecendo! Em vez de pensar em b como o resultado da aplicação de X^{-1} a y , devemos entendê-lo como o vetor único que satisfaz a equação $Xb = y$.

Uma coisa importante de se notar é que como $XX^{-1}y = y$, então se $z = X^{-1}y$, temos que:

$$y = \sum z_i x_i,$$

isto é, as coordenadas do vetor $z = X^{-1}y$ indicam os coeficientes necessários para escrever y na base dada pelas colunas de X .

Aplicações

Com as ideias desenvolvidas nessa seção, somos capazes de desenvolver várias transformações de forma rápida. Por exemplo, suponha que queremos uma matriz C cuja primeira coluna é a primeira coluna de A duplicada, e as outras colunas são iguais as de A . Pela Seção de multiplicação Matriz-Matriz, queremos então que

$$\begin{aligned} C_1 &= 2A_1 + 0A_2 + \dots + 0A_n = A[2, 0, \dots, 0]^T \\ &\vdots \\ C_i &= A_i = A[0, 0, \dots, 1, \dots, 0]^T, \end{aligned}$$

logo, $C = AB$ onde $B = \text{diag}(2, 1, \dots, 1)$.

Suponha agora que D é igual a M , porém com a linha 3 somada com a linha 1. Note que a gente só sabe trabalhar com operações nas colunas, então a primeira coisa é transformar linhas em colunas, fazendo A^T , logo

$$\begin{aligned} D_1 &= A_1^T + A_3^T = A^T[1, 0, 1, \dots, 0]^T \\ &\vdots \\ D_i &= A_i^T = A^T[0, 0, \dots, 1, \dots, 0]^T. \end{aligned}$$

Logo,

$$D = A^T \begin{pmatrix} 1, 0, \dots, 0 \\ 0, 1, \dots, 0 \\ 1, 0, \dots, 0 \\ \vdots \\ 0, 0, \dots, 1 \end{pmatrix} = A^T M$$

Como queremos uma expressão em termos de A , podemos fazer $D^T = M^T A$.

Ou seja, operações nas colunas de uma matriz são feitas à direita e operações com linhas são feitas à esquerda transposta.

Exercício 20. Considere: $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Verifique que as multiplicações definidas acima de fato tem o comportamento esperado descrito no texto.

Exercício 21. Considere: $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Calcule as multiplicações necessárias para dobrar a coluna 1 somada com menos a coluna 2 e fazer linha 2 mais o dobro da linha 1.

Faça os cálculos explícitos para amostrar que suas multiplicações estão corretas.

A.1.3 Produtos internos

Nos espaços de dimensão 2 ou 3, definimos o comprimento de um vetor \mathbf{x} (ou seja, a distância de sua extremidade até a origem) usando o teorema de Pitágoras. Por exemplo, no espaço \mathbb{R}^3 , temos:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Essa fórmula pode ser naturalmente estendida para qualquer dimensão n , definindo a *norma* de um vetor $\mathbf{x} \in \mathbb{R}^n$ como:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

O termo *norma* é frequentemente usado como uma forma mais técnica ou refinada de se referir ao comprimento de um vetor.

O produto interno que definimos para \mathbb{R}^n e \mathbb{C}^n satisfaz as seguintes propriedades fundamentais:

1. **Simetria (conjugada):** $\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}$; no caso real, isso equivale à simetria usual: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$;
2. **Linearidade:** $\langle \alpha \mathbf{x} + \beta \mathbf{y}, \mathbf{z} \rangle = \alpha \langle \mathbf{x}, \mathbf{z} \rangle + \beta \langle \mathbf{y}, \mathbf{z} \rangle$, para todos os vetores $\mathbf{x}, \mathbf{y}, \mathbf{z}$ e escalares α, β ;
3. **Não-negatividade:** $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ para todo \mathbf{x} ;
4. **Não-degenerescência:** $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ se, e somente se, $\mathbf{x} = \mathbf{0}$.

Seja V um espaço vetorial (real ou complexo). Um *produto interno* em V é uma função que associa a cada par de vetores $\mathbf{x}, \mathbf{y} \in V$ um escalar $\langle \mathbf{x}, \mathbf{y} \rangle$ que satisfaz as propriedades 1 a 4 acima.

No caso real, assumimos que $\langle \mathbf{x}, \mathbf{y} \rangle$ é sempre real. Já em espaços complexos, o produto interno pode assumir valores complexos.

Chamamos de *espaço com produto interno* o par $(V, \langle \cdot, \cdot \rangle)$ formado por um espaço vetorial V e um produto interno definido sobre ele. Dado um produto interno, podemos definir a norma de um vetor por:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}.$$

Exemplo 3. Seja $V = \mathbb{R}^n$ ou \mathbb{C}^n . Já vimos que o produto interno pode ser definido como

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}^* \mathbf{x} = \sum_{k=1}^n x_k \overline{y_k}.$$

Esse produto interno é conhecido como o produto interno padrão em \mathbb{R}^n ou \mathbb{C}^n .

Ao longo do texto, usaremos a letra \mathbb{F} para representar tanto \mathbb{R} quanto \mathbb{C} . Assim, qualquer afirmação sobre o espaço \mathbb{F}^n é válida para ambos os casos: \mathbb{R}^n e \mathbb{C}^n .

Exemplo 4. Recordemos que, para uma matriz quadrada A , o traço é definido como a soma dos elementos da diagonal principal:

$$\text{tr}(A) := \sum_{k=1}^n a_{k,k}.$$

Seja $M_{m \times n}$ o espaço das matrizes $m \times n$. Definimos o produto interno de Frobenius por:

$$\langle A, B \rangle = \text{tr}(B^* A).$$

É possível verificar que esse produto interno satisfaz as propriedades 1 a 4, ou seja, é de fato um produto interno.

Observe que:

$$\text{tr}(B^* A) = \sum_{j,k} A_{j,k} \overline{B_{j,k}},$$

o que mostra que esse produto interno coincide com o produto interno padrão em \mathbb{C}^{mn} .

Exemplo 5. Seja $V = L^2([a, b])$, o espaço das funções complexas mensuráveis ao quadrado integrável no intervalo $[a, b]$, ou seja:

$$L^2([a, b]) = \left\{ f : [a, b] \rightarrow \mathbb{C} \mid \int_a^b |f(t)|^2 dt < \infty \right\}.$$

Definimos o produto interno entre duas funções $f, g \in L^2([a, b])$ por:

$$\langle f, g \rangle = \int_a^b f(t) \overline{g(t)} dt.$$

Esse produto interno satisfaz as propriedades fundamentais (conjugada simetria, linearidade, não-negatividade e não-degenerescência), tornando $L^2([a, b])$ um espaço com produto interno.

A norma induzida por esse produto interno é:

$$\|f\| = \left(\int_a^b |f(t)|^2 dt \right)^{1/2}.$$

Exercício 22. Mostre que nos exemplos acima, os produtos acima de fato satisfazem as propriedades de produto-interno.

A seguir apresentamos alguns resultados importantes sobre produtos internos.

Lema 1. Seja \mathbf{x} um vetor em um espaço com produto interno V . Então $\mathbf{x} = 0$ se, e somente se,

$$\langle \mathbf{x}, \mathbf{y} \rangle = 0 \quad \text{para todo } \mathbf{y} \in V. \quad (1.1)$$

Corolário 1. Sejam \mathbf{x}, \mathbf{y} vetores em um espaço com produto interno V . A igualdade $\mathbf{x} = \mathbf{y}$ vale se, e somente se,

$$\langle \mathbf{x}, \mathbf{z} \rangle = \langle \mathbf{y}, \mathbf{z} \rangle \quad \text{para todo } \mathbf{z} \in V.$$

Corolário 2. Sejam $A, B : X \rightarrow Y$ dois operadores lineares. Suponha que

$$\langle A\mathbf{x}, \mathbf{y} \rangle = \langle B\mathbf{x}, \mathbf{y} \rangle \quad \text{para todo } \mathbf{x} \in X \text{ e } \mathbf{y} \in Y.$$

Então, $A = B$.

Um dos resultados mais importantes com relação ao produto interno é a desigualdade de Cauchy-Schwarz:

Teorema 7 (Desigualdade de Cauchy-Schwarz). Sejam \mathbf{x}, \mathbf{y} vetores em um espaço com produto interno. Então:

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \cdot \|\mathbf{y}\|.$$

Demonstração. Vamos apresentar uma demonstração que, embora não seja a mais curta, revela bem a origem das ideias principais.

Comecemos com o caso real. Se $\mathbf{y} = 0$, a desigualdade é trivial. Assim, podemos supor que $\mathbf{y} \neq 0$. Pelas propriedades do produto interno, para qualquer escalar t temos:

$$0 \leq \|\mathbf{x} - t\mathbf{y}\|^2 = \langle \mathbf{x} - t\mathbf{y}, \mathbf{x} - t\mathbf{y} \rangle = \|\mathbf{x}\|^2 - 2t\langle \mathbf{x}, \mathbf{y} \rangle + t^2\|\mathbf{y}\|^2.$$

Essa desigualdade vale para todo $t \in \mathbb{R}$, em particular para

$$t = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|^2},$$

o que nos leva a:

$$0 \leq \|\mathbf{x}\|^2 - \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|^2}{\|\mathbf{y}\|^2},$$

ou seja,

$$|\langle \mathbf{x}, \mathbf{y} \rangle|^2 \leq \|\mathbf{x}\|^2 \cdot \|\mathbf{y}\|^2.$$

Portanto, obtemos a desigualdade desejada.

Para o caso complexo, uma das estratégias é considerar o mesmo argumento acima com t complexo (escolhendo, por exemplo, $t = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|^2}$), ou então proceder de forma análoga usando:

$$\|\mathbf{x} - t\mathbf{y}\|^2 = \|\mathbf{x}\|^2 - t\langle \mathbf{y}, \mathbf{x} \rangle - \bar{t}\langle \mathbf{x}, \mathbf{y} \rangle + |t|^2\|\mathbf{y}\|^2.$$

A escolha de $t = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|^2}$ minimiza a expressão acima, o que nos leva novamente a:

$$0 \leq \|\mathbf{x}\|^2 - \frac{|\langle \mathbf{x}, \mathbf{y} \rangle|^2}{\|\mathbf{y}\|^2},$$

ou seja,

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \cdot \|\mathbf{y}\|.$$

Esse raciocínio mostra completamente a validade da desigualdade. A justificativa anterior serviu apenas para motivar a escolha do valor específico de t . \square

Lema 2 (Desigualdade triangular). *Para quaisquer vetores \mathbf{x}, \mathbf{y} em um espaço com produto interno, vale:*

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|.$$

Demonstração.

$$\begin{aligned} \|\mathbf{x} + \mathbf{y}\|^2 &= \langle \mathbf{x} + \mathbf{y}, \mathbf{x} + \mathbf{y} \rangle \\ &= \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{x} \rangle \\ &\leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2|\langle \mathbf{x}, \mathbf{y} \rangle| \\ &\leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\|\mathbf{x}\| \cdot \|\mathbf{y}\| = (\|\mathbf{x}\| + \|\mathbf{y}\|)^2. \end{aligned}$$

Tomando a raiz quadrada dos dois lados, obtemos a desigualdade desejada. \square

Lema 3 (Identidades de polarização). *Sejam $\mathbf{x}, \mathbf{y} \in V$. É possível recuperar o produto interno a partir da norma usando as seguintes fórmulas:*

- Se V é um espaço com produto interno real, então:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \frac{1}{4} (\|\mathbf{x} + \mathbf{y}\|^2 - \|\mathbf{x} - \mathbf{y}\|^2).$$

- Se V é um espaço com produto interno complexo, então:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \frac{1}{4} \sum_{\alpha \in \{1, -1, i, -i\}} \alpha \|\mathbf{x} + \alpha \mathbf{y}\|^2.$$

Exercício 23. Prove todos os resultados anteriores que não possuem provas.

A.1.4 Ortogonalidade

Definição 2. Dois vetores \mathbf{u} e \mathbf{v} são chamados ortogonais (ou também perpendiculares) se

$$\langle \mathbf{u}, \mathbf{v} \rangle = 0.$$

Usamos a notação $\mathbf{u} \perp \mathbf{v}$ para indicar que os vetores são ortogonais.

Note que, se os vetores \mathbf{u} e \mathbf{v} forem ortogonais, então vale a seguinte identidade, conhecida como identidade pitagórica:

$$\|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 \quad \text{se } \mathbf{u} \perp \mathbf{v}.$$

Definição 3. Dizemos que um vetor \mathbf{v} é ortogonal a um subespaço E se \mathbf{v} for ortogonal a todos os vetores $\mathbf{w} \in E$.

Analogamente, dizemos que dois subespaços E e F são ortogonais se todos os vetores de E são ortogonais a todos os vetores de F , ou seja, $\langle \mathbf{e}, \mathbf{f} \rangle = 0$ para todo $\mathbf{e} \in E$ e $\mathbf{f} \in F$.

O próximo lema mostra como verificar se um vetor é ortogonal a um subespaço gerado por um conjunto finito de vetores.

Lema 4. *Seja E o subespaço gerado pelos vetores $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$. Então, um vetor \mathbf{v} é ortogonal a E se, e somente se,*

$$\mathbf{v} \perp \mathbf{v}_k, \quad \text{para todo } k = 1, 2, \dots, r.$$

Definição 4. *Um sistema de vetores $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ é dito ortogonal se quaisquer dois vetores distintos do sistema forem ortogonais entre si, ou seja,*

$$\langle \mathbf{v}_j, \mathbf{v}_k \rangle = 0 \quad \text{para } j \neq k.$$

Se, adicionalmente, $\|\mathbf{v}_k\| = 1$ para todo k , então o sistema é chamado de ortonormal.

Lema 5 (Identidade de Pitágoras generalizada). *Sejam $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ um sistema ortogonal. Então:*

$$\left\| \sum_{k=1}^n \alpha_k \mathbf{v}_k \right\|^2 = \sum_{k=1}^n |\alpha_k|^2 \cdot \|\mathbf{v}_k\|^2.$$

Essa fórmula assume uma forma particularmente simples no caso de sistemas ortonormais, pois nesse caso $\|\mathbf{v}_k\| = 1$ para todo k .

Definição 5. *Um sistema ortogonal (ou ortonormal) $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$, que também forma uma base, é chamado de base ortogonal (ou base ortonormal).*

É claro que, se $\dim V = n$, então qualquer sistema ortogonal de n vetores não nulos em V é automaticamente uma base ortogonal.

Como vimos anteriormente, para encontrar as coordenadas de um vetor em uma base arbitrária, normalmente é necessário resolver um sistema linear. No entanto, no caso de uma base ortogonal, isso pode ser feito de forma muito mais simples.

Suponha que $\mathbf{v}_1, \dots, \mathbf{v}_n$ seja uma base ortogonal, e que

$$\mathbf{x} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_n \mathbf{v}_n = \sum_{j=1}^n \alpha_j \mathbf{v}_j.$$

Tomando o produto interno de ambos os lados com \mathbf{v}_1 , obtemos:

$$\langle \mathbf{x}, \mathbf{v}_1 \rangle = \sum_{j=1}^n \alpha_j \langle \mathbf{v}_j, \mathbf{v}_1 \rangle = \alpha_1 \langle \mathbf{v}_1, \mathbf{v}_1 \rangle = \alpha_1 \|\mathbf{v}_1\|^2.$$

(já que $\langle \mathbf{v}_j, \mathbf{v}_1 \rangle = 0$ para $j \neq 1$)

Portanto,

$$\alpha_1 = \frac{\langle \mathbf{x}, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2}.$$

De forma semelhante, multiplicando ambos os lados por \mathbf{v}_k , obtemos:

$$\langle \mathbf{x}, \mathbf{v}_k \rangle = \sum_{j=1}^n \alpha_j \langle \mathbf{v}_j, \mathbf{v}_k \rangle = \alpha_k \|\mathbf{v}_k\|^2,$$

então

$$\alpha_k = \frac{\langle \mathbf{x}, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2}. \quad (\text{A.4})$$

Para encontrar as coordenadas de um vetor em uma base ortogonal, não é necessário resolver um sistema linear — as coordenadas são dadas diretamente pela fórmula (A.4).

Retomando a definição de projeção ortogonal da geometria clássica no plano (bidimensional), podemos introduzir a seguinte definição. Seja E um subespaço de um espaço com produto interno V .

Definição 6. *Seja \mathbf{v} um vetor. A sua projeção ortogonal sobre o subespaço E , denotada por $P_E \mathbf{v}$, é o vetor \mathbf{w} tal que:*

1. $\mathbf{w} \in E$;
2. $\mathbf{v} - \mathbf{w} \perp E$.

Usaremos a notação $\mathbf{w} = P_E \mathbf{v}$ para representar a projeção ortogonal de \mathbf{v} sobre E .

Teorema 8. *Seja $\mathbf{w} = P_E \mathbf{v}$ a projeção ortogonal de \mathbf{v} sobre o subespaço E . Então, \mathbf{w} é o ponto de E mais próximo de \mathbf{v} , ou seja, para todo $\mathbf{x} \in E$:*

$$\|\mathbf{v} - \mathbf{w}\| \leq \|\mathbf{v} - \mathbf{x}\|.$$

Além disso, se existir $\mathbf{x} \in E$ tal que

$$\|\mathbf{v} - \mathbf{w}\| = \|\mathbf{v} - \mathbf{x}\|,$$

então $\mathbf{x} = \mathbf{w}$.

Demonstração. Seja $\mathbf{y} = \mathbf{w} - \mathbf{x}$. Então:

$$\mathbf{v} - \mathbf{x} = \mathbf{v} - \mathbf{w} + \mathbf{w} - \mathbf{x} = \mathbf{v} - \mathbf{w} + \mathbf{y}.$$

Como $\mathbf{v} - \mathbf{w} \perp E$ e $\mathbf{y} \in E$, segue que $\mathbf{v} - \mathbf{w} \perp \mathbf{y}$. Assim, pelo teorema de Pitágoras:

$$\|\mathbf{v} - \mathbf{x}\|^2 = \|\mathbf{v} - \mathbf{w}\|^2 + \|\mathbf{y}\|^2 \geq \|\mathbf{v} - \mathbf{w}\|^2.$$

A igualdade ocorre se, e somente se, $\|\mathbf{y}\| = 0$, ou seja, $\mathbf{y} = 0$, o que implica $\mathbf{x} = \mathbf{w}$. \square

Proposição 1. *Sejam $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$ uma base ortogonal do subespaço E . Então, a projeção ortogonal de um vetor \mathbf{v} sobre E é dada por:*

$$P_E \mathbf{v} = \sum_{k=1}^r \alpha_k \mathbf{v}_k, \quad \text{onde} \quad \alpha_k = \frac{\langle \mathbf{v}, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2}.$$

Em outras palavras:

$$P_E \mathbf{v} = \sum_{k=1}^r \frac{\langle \mathbf{v}, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2} \mathbf{v}_k. \quad (\text{A.5})$$

Note que a fórmula dos coeficientes α_k coincide com a da equação (A.4), isto é, essa fórmula continua válida mesmo se o sistema $\{\mathbf{v}_k\}$ for apenas ortogonal (e não base), pois ela projeta \mathbf{v} sobre o subespaço gerado pelos vetores.

Demonstração. Definimos:

$$\mathbf{w} := \sum_{k=1}^r \alpha_k \mathbf{v}_k, \quad \text{com} \quad \alpha_k = \frac{\langle \mathbf{v}, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2}.$$

Queremos mostrar que $\mathbf{v} - \mathbf{w} \perp E$. É suficiente mostrar que

$$\mathbf{v} - \mathbf{w} \perp \mathbf{v}_k, \quad \text{para todo } k = 1, 2, \dots, r.$$

Para isso, calculamos:

$$\begin{aligned} \langle \mathbf{v} - \mathbf{w}, \mathbf{v}_k \rangle &= \langle \mathbf{v}, \mathbf{v}_k \rangle - \langle \mathbf{w}, \mathbf{v}_k \rangle \\ &= \langle \mathbf{v}, \mathbf{v}_k \rangle - \left\langle \sum_{j=1}^r \alpha_j \mathbf{v}_j, \mathbf{v}_k \right\rangle \\ &= \langle \mathbf{v}, \mathbf{v}_k \rangle - \sum_{j=1}^r \alpha_j \langle \mathbf{v}_j, \mathbf{v}_k \rangle. \end{aligned}$$

Como o sistema $\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ é ortogonal, temos $\langle \mathbf{v}_j, \mathbf{v}_k \rangle = 0$ para $j \neq k$, e $\langle \mathbf{v}_k, \mathbf{v}_k \rangle = \|\mathbf{v}_k\|^2$.

Logo:

$$\langle \mathbf{v} - \mathbf{w}, \mathbf{v}_k \rangle = \langle \mathbf{v}, \mathbf{v}_k \rangle - \alpha_k \|\mathbf{v}_k\|^2 = \langle \mathbf{v}, \mathbf{v}_k \rangle - \langle \mathbf{v}, \mathbf{v}_k \rangle = 0.$$

Portanto, $\mathbf{v} - \mathbf{w} \perp \mathbf{v}_k$ para todo k , e segue que $\mathbf{v} - \mathbf{w} \perp E$. Assim, $\mathbf{w} = P_E \mathbf{v}$. \square

Observação 2. Retomando a definição de produto interno em \mathbb{C}^n e \mathbb{R}^n , podemos deduzir da fórmula (A.5) que a matriz da projeção ortogonal P_E sobre um subespaço $E \subseteq \mathbb{C}^n$ (ou \mathbb{R}^n) é dada por:

$$P_E = \sum_{k=1}^r \frac{1}{\|\mathbf{v}_k\|^2} \mathbf{v}_k \mathbf{v}_k^*, \tag{A.6}$$

onde os vetores coluna $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r$ formam uma base ortogonal de E .

Ortogonalização de Gram–Schmidt

Suponha que temos um conjunto linearmente independente de vetores $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$. O método de Gram–Schmidt constrói a partir desse conjunto um sistema ortogonal $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ tal que:

$$\text{span}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) = \text{span}(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n).$$

Além disso, para todo $r \leq n$, temos:

$$\text{span}(\mathbf{x}_1, \dots, \mathbf{x}_r) = \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_r).$$

O algoritmo segue os seguintes passos:

1. Defina $\mathbf{v}_1 := \mathbf{x}_1$. Seja $E_1 := \text{span}(\mathbf{v}_1)$.

2. Defina

$$\mathbf{v}_2 := \mathbf{x}_2 - P_{E_1} \mathbf{x}_2 = \mathbf{x}_2 - \frac{\langle \mathbf{x}_2, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1.$$

Seja $E_2 := \text{span}(\mathbf{v}_1, \mathbf{v}_2)$. Como $\mathbf{x}_2 \notin E_1$, temos $\mathbf{v}_2 \neq 0$.

3. Defina

$$\mathbf{v}_3 := \mathbf{x}_3 - P_{E_2} \mathbf{x}_3 = \mathbf{x}_3 - \frac{\langle \mathbf{x}_3, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1 - \frac{\langle \mathbf{x}_3, \mathbf{v}_2 \rangle}{\|\mathbf{v}_2\|^2} \mathbf{v}_2.$$

Seja $E_3 := \text{span}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$.

4. Suponha que já tenhamos construído vetores ortogonais $\mathbf{v}_1, \dots, \mathbf{v}_r$, tais que

$$E_r := \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_r) = \text{span}(\mathbf{x}_1, \dots, \mathbf{x}_r).$$

Defina:

$$\mathbf{v}_{r+1} := \mathbf{x}_{r+1} - \sum_{k=1}^r \frac{\langle \mathbf{x}_{r+1}, \mathbf{v}_k \rangle}{\|\mathbf{v}_k\|^2} \mathbf{v}_k.$$

Note que $\mathbf{x}_{r+1} \notin E_r$, então $\mathbf{v}_{r+1} \neq 0$.

Continuando esse processo até $r = n$, obtemos um sistema ortogonal de vetores $\mathbf{v}_1, \dots, \mathbf{v}_n$ tal que:

$$\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n) = \text{span}(\mathbf{x}_1, \dots, \mathbf{x}_n).$$

Exercício 24. Aplique o método de Gram-Schmidt para os vetores $\{(1, 1, 1), (0, 1, 2), (1, 0, 2)\}$.

3.3. Complemento ortogonal. Decomposição $V = E \oplus E^\perp$

Definição 7. Seja E um subespaço de um espaço com produto interno V . O complemento ortogonal de E , denotado por E^\perp , é o conjunto de todos os vetores ortogonais a E :

$$E^\perp := \{\mathbf{x} \in V : \mathbf{x} \perp E\}.$$

Se $\mathbf{x}, \mathbf{y} \perp E$, então qualquer combinação linear $\alpha\mathbf{x} + \beta\mathbf{y}$ também está em E^\perp (consegue ver por quê?). Logo, E^\perp é um subespaço.

Pela definição de projeção ortogonal, qualquer vetor $\mathbf{v} \in V$ admite uma decomposição única da forma:

$$\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2, \quad \text{com } \mathbf{v}_1 \in E \text{ e } \mathbf{v}_2 \perp E \text{ (ou seja, } \mathbf{v}_2 \in E^\perp\text{)}.$$

Neste caso, temos $\mathbf{v}_1 = P_E \mathbf{v}$.

Essa afirmação pode ser escrita simbolicamente como:

$$V = E \oplus E^\perp,$$

o que expressa precisamente que todo vetor admite a decomposição única acima.

A proposição a seguir mostra uma propriedade fundamental do complemento ortogonal:

Proposição 2. Seja E um subespaço. Então:

$$(E^\perp)^\perp = E.$$

Exercício 25. Prove todos os resultados anteriores que não possuem provas.

A.2 Probabilidade

Um *espaço de probabilidade* é uma tupla composta por três elementos: o *espaço amostral*, o *conjunto de eventos* e uma *distribuição de probabilidade*:

- **Espaço amostral Ω** : Ω é o conjunto de todos os eventos elementares ou resultados possíveis de um experimento. Por exemplo, ao lançar um dado, $\Omega = \{1, 2, 3, 4, 5, 6\}$.
- **Conjunto de eventos \mathcal{F}** : \mathcal{F} é uma σ -álgebra, ou seja, um conjunto de subconjuntos de Ω que contém Ω e é fechado sob complementação e união enumerável (e, consequentemente, também sob interseção enumerável). Um exemplo de evento é: “o dado mostra um número ímpar”.
- **Distribuição de probabilidade \mathbb{P}** : \mathbb{P} é uma função que associa a cada evento de \mathcal{F} um número em $[0, 1]$, tal que $\mathbb{P}[\Omega] = 1$, $\mathbb{P}[\emptyset] = 0$ e, para eventos mutuamente exclusivos A_1, \dots, A_n , temos:

$$\mathbb{P}[A_1 \cup \dots \cup A_n] = \sum_{i=1}^n \mathbb{P}[A_i].$$

A distribuição de probabilidade discreta associada ao lançamento de um dado justo pode ser definida como $\mathbb{P}[A_i] = 1/6$ para $i \in \{1, \dots, 6\}$, onde A_i é o evento “o dado mostra o valor i ”.

A.2.1 Variáveis aleatórias

Uma variável aleatória X é uma função $X : \Omega \rightarrow \mathbb{R}$ mensurável, ou seja, tal que para qualquer intervalo I , o subconjunto $\{\omega \in \Omega : X(\omega) \in I\}$ pertence ao conjunto de eventos.

A *função de massa de probabilidade* de uma variável aleatória discreta X é a função $x \mapsto \mathbb{P}[X = x]$.

Uma distribuição é dita *absolutamente contínua* quando possui uma *função densidade de probabilidade* f associada, tal que, para todo $a, b \in \mathbb{R}$:

$$\mathbb{P}[a \leq X \leq b] = \int_a^b f(x) dx.$$

Exemplo 6 (Binomial). Uma variável aleatória X segue uma distribuição binomial $B(n, p)$ com $n \in \mathbb{N}$ e $p \in [0, 1]$ se, para $k \in \{0, 1, \dots, n\}$,

$$\mathbb{P}[X = k] = \binom{n}{k} p^k (1 - p)^{n-k}.$$

Exemplo 7 (Uniforme). Uma variável aleatória X segue uma distribuição uniforme $U(a, b)$ no intervalo (a, b) se,

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b \\ 0 & \text{caso contrário.} \end{cases}$$

Exemplo 8 (Normal). Uma variável aleatória X segue uma distribuição normal $N(\mu, \sigma^2)$ com $\mu \in \mathbb{R}$ e $\sigma > 0$ se sua densidade for dada por:

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

A distribuição normal padrão é $N(0, 1)$, com média zero e variância unitária.

Exemplo 9 (Laplace). Uma variável aleatória X segue uma distribuição de Laplace com parâmetro de localização $\mu \in \mathbb{R}$ e parâmetro de escala $b > 0$ se sua densidade for:

$$f(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right).$$

A.2.2 Probabilidade condicional e independência

A probabilidade condicional do evento A dado o evento B é definida como a razão entre a probabilidade da interseção $A \cap B$ e a probabilidade de B , desde que $\mathbb{P}[B] \neq 0$:

$$\mathbb{P}[A | B] = \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}.$$

Dois eventos A e B são ditos independentes quando a probabilidade conjunta $\mathbb{P}[A \cap B]$ pode ser fatorada como o produto $\mathbb{P}[A]\mathbb{P}[B]$:

$$\mathbb{P}[A \cap B] = \mathbb{P}[A]\mathbb{P}[B].$$

De forma equivalente, a independência entre A e B pode ser expressa afirmando que $\mathbb{P}[A | B] = \mathbb{P}[A]$, sempre que $\mathbb{P}[B] \neq 0$.

Além disso, uma sequência de variáveis aleatórias é dita *i.i.d.* (independentes e identicamente distribuídas) quando todas as variáveis da sequência são mutuamente independentes e seguem a mesma distribuição de probabilidade.

A.2.3 Algumas fórmulas importantes

$$\mathbb{P}[A \cup B] = \mathbb{P}[A] + \mathbb{P}[B] - \mathbb{P}[A \cap B] \quad (\text{regra da soma})$$

$$\mathbb{P}\left[\bigcup_{i=1}^n A_i\right] \leq \sum_{i=1}^n \mathbb{P}[A_i] \quad (\text{desigualdade da união})$$

$$\mathbb{P}[A | B] = \frac{\mathbb{P}[B | A]\mathbb{P}[A]}{\mathbb{P}[B]} \quad (\text{fórmula de Bayes})$$

$$\mathbb{P}\left[\bigcap_{i=1}^n A_i\right] = \mathbb{P}[A_1]\mathbb{P}[A_2 | A_1] \cdots \mathbb{P}\left[A_n | \bigcap_{i=1}^{n-1} A_i\right] \quad (\text{regra da cadeia})$$

Exercício 26. Prove os resultados acima.

A.2.4 Esperança e desigualdade de Markov

A esperança ou valor esperado de uma variável aleatória X é denotada por $\mathbb{E}[X]$ e, no caso discreto, é definida como

$$\mathbb{E}[X] = \sum_x x \mathbb{P}[X = x]. \quad (\text{C.9})$$

No caso contínuo, quando X possui uma função densidade de probabilidade $f(x)$, a esperança é dada por

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x) dx.$$

Além disso, dado uma função qualquer g , temos que:

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)f(x) dx.$$

Uma propriedade fundamental da esperança é sua linearidade. Isto é, para quaisquer variáveis aleatórias X e Y e constantes $a, b \in \mathbb{R}$, temos:

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]. \quad (\text{C.10})$$

A seguir, apresentamos um limite superior simples para uma variável aleatória não-negativa em função de sua esperança, conhecido como a *Desigualdade de Markov*.

Teorema 9 (Desigualdade de Markov). *Seja X uma variável aleatória não-negativa ($X \geq 0$ quase certamente) com valor esperado $\mathbb{E}[X] < \infty$. Então, para todo $t > 0$, temos:*

$$\mathbb{P}(X \geq t) \leq \frac{\mathbb{E}[X]}{t}.$$

Exercício 27. Prove as desigualdades de Markov.

A.2.5 Variância e a desigualdade de Chebyshev

A variância de uma variável aleatória X é denotada por $\text{Var}[X]$ e definida como

$$\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2].$$

O desvio padrão de X é denotado por σ_X e definido como

$$\sigma_X = \sqrt{\text{Var}[X]}.$$

Para qualquer variável aleatória X e qualquer constante $a \in \mathbb{R}$, as seguintes propriedades básicas são válidas:

$$\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2,$$

$$\text{Var}[aX] = a^2 \text{Var}[X].$$

Além disso, se X e Y forem independentes, então

$$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y].$$

Exercício 28. Prove as identidades acima.

A seguinte desigualdade, conhecida como *Desigualdade de Chebyshev*, fornece um limite para a probabilidade de uma variável aleatória se desviar de sua esperança em função do seu desvio padrão.

Teorema 10 (Desigualdade de Chebyshev). *Seja X uma variável aleatória com valor esperado $\mu = \mathbb{E}[X]$ e variância finita $\text{Var}(X) = \sigma^2$. Então, para todo $\varepsilon > 0$, vale:*

$$\mathbb{P}(|X - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2}.$$

Exercício 29. Prove a desigualdade de Chebyshev.

A.2.6 Covariância

A covariância entre duas variáveis aleatórias X e Y é denotada por $\text{Cov}(X, Y)$ e definida por

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

Exercício 30. Prove que

$$\text{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

Dizemos que X e Y são *não correlacionadas* quando $\text{Cov}(X, Y) = 0$. Se X e Y forem independentes, então certamente são não correlacionadas, mas a recíproca nem sempre é verdadeira.

Exercício 31. Seja X uniforme no intervalo $[-1, 1]$ e seja $Y = X^2$. Mostre que $\text{Cov}(X, Y) = 0$ mas X, Y não são independentes.

Observação 3. Considere uma variável aleatória contínua X centrada em zero, ou seja, $\mathbb{E}[X] = 0$, com densidade de probabilidade par e definida em um intervalo do tipo $(-a, a)$, com $a > 0$. Seja $Y = g(X)$ para uma função g . A questão é: para quais funções $g(X)$ temos $\text{Cov}(X, g(X)) = 0$?

Sabemos que

$$\text{Cov}(X, g(X)) = \mathbb{E}[Xg(X)] - \mathbb{E}[X]\mathbb{E}[g(X)].$$

Como $\mathbb{E}[X] = 0$, segue que $\text{Cov}(X, g(X)) = \mathbb{E}[Xg(X)]$. Denotando a densidade de X por $f(x)$, temos

$$\text{Cov}(X, g(X)) = \int_{-a}^a xg(x)f(x)dx.$$

Uma maneira de garantir que $\text{Cov}(X, g(X)) = 0$ é exigir que $g(x)$ seja uma função par. Assim, $xg(x)f(x)$ será uma função ímpar e a integral em $(-a, a)$ se anulará, ou seja,

$$\int_{-a}^a xg(x)f(x)dx = 0.$$

Portanto, $\text{Cov}(X, f(X)) = 0$ e como $Y = g(X)$, teremos que ambas são dependentes.

Dessa forma, podemos concluir que a distribuição precisa de X não afeta a condição, desde que $p(x)$ seja simétrica em torno da origem. Qualquer função par $f(\cdot)$ satisfará $\text{Cov}(X, f(X)) = 0$.

A covariância é uma forma bilinear simétrica e semi-definida positiva, com as seguintes propriedades:

- **Simetria:** $\text{Cov}(X, Y) = \text{Cov}(Y, X)$ para quaisquer variáveis X e Y .
- **Bilinearidade:** $\text{Cov}(X + X', Y) = \text{Cov}(X, Y) + \text{Cov}(X', Y)$ e $\text{Cov}(aX, Y) = a\text{Cov}(X, Y)$ para qualquer $a \in \mathbb{R}$.
- **Semi-definida positiva:** $\text{Cov}(X, X) = \text{Var}[X] \geq 0$ para qualquer variável X .

Além disso, vale a desigualdade de Cauchy-Schwarz, que afirma que para variáveis X e Y com variância finita,

$$|\text{Cov}(X, Y)| \leq \sqrt{\text{Var}[X] \text{Var}[Y]}.$$

Perceba a semelhança do resultado acima com a desigualdade de Cauchy-Schwarz!

Exercício 32. Prove os resultados acima.

A matriz de covariância de um vetor de variáveis aleatórias $\mathbf{X} = (X_1, \dots, X_p)$ é a matriz em $\mathbb{R}^{n \times n}$ denotada por $\mathbf{C}(\mathbf{X})$ e definida por

$$\mathbf{C}(\mathbf{X}) = \mathbb{E} [(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top].$$

Portanto, $\mathbf{C}(\mathbf{X})$ é a matriz cujos elementos são $\text{Cov}(X_i, X_j)$. Além disso, é imediato mostrar que

$$\mathbf{C}(\mathbf{X}) = \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top.$$

A.2.7 Teoremas assintóticos

Em muitas aplicações de probabilidade e estatística, estamos interessados no comportamento de sequências de variáveis aleatórias quando o número de observações tende ao infinito. Os *teoremas assintóticos* fornecem resultados fundamentais que descrevem como certos estimadores ou somas de variáveis aleatórias se comportam no limite, ou seja, quando o tamanho da amostra n cresce indefinidamente.

Teorema 11 (Lei Fraca dos Grandes Números). *Seja $(X_n)_{n \in \mathbb{N}}$ uma sequência de variáveis aleatórias independentes, todas com a mesma esperança μ e variância $\sigma^2 < \infty$. Definindo a média amostral por*

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i,$$

então, para qualquer $\varepsilon > 0$,

$$\lim_{n \rightarrow \infty} \mathbb{P} (|\bar{X}_n - \mu| \geq \varepsilon) = 0.$$

Exercício 33. Prove a Lei Fraca dos Grandes números utilizando a desigualdade de Chebyshev.

Teorema 12 (Teorema Central do Limite). *Seja X_1, \dots, X_n uma sequência de variáveis aleatórias i.i.d. com esperança μ e desvio padrão σ . Definimos a média amostral como*

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$$

e a variância da média como $\sigma_n^2 = \sigma^2/n$. Então, a variável padronizada $(\bar{X}_n - \mu)/\sigma_n$ converge em distribuição para uma normal padrão $N(0,1)$. Mais precisamente, para todo $t \in \mathbb{R}$,

$$\lim_{n \rightarrow \infty} \mathbb{P} \left(\frac{\bar{X}_n - \mu}{\sigma_n} \leq t \right) = \int_{-\infty}^t \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

Observação 4. Apesar dos teoremas assintóticos, como a Lei Fraca dos Grandes Números e o Teorema Central do Limite, serem fundamentais para entender o comportamento de sequências de variáveis aleatórias quando $n \rightarrow \infty$, na prática, em aprendizado de máquina, o número de amostras n nem sempre é grande o suficiente para que esses resultados sejam aplicáveis com segurança. Por outro lado, desigualdades como as de Markov e Chebyshev fornecem limites válidos para qualquer valor finito de n . Essas desigualdades são exemplos de desigualdades de concentração, que nos permitem controlar a probabilidade de desvios em torno da média de uma variável aleatória. A teoria de concentração será crucial em tópicos futuros, pois fornece ferramentas importantes para analisar o desempenho de algoritmos em cenários onde o regime assintótico não pode ser garantido.

A.2.8 Função geradora de momentos

A esperança $\mathbb{E}[X^p]$ é chamada de p -ésimo momento da variável aleatória X . A função geradora de momentos de uma variável aleatória X é uma ferramenta importante, pois permite obter seus diferentes momentos por meio de diferenciação em zero. Essa função é crucial tanto para descrever a distribuição de X quanto para analisar suas propriedades.

A função geradora de momentos de uma variável aleatória X é a função $M_X : t \mapsto \mathbb{E}[e^{tX}]$, definida para os valores de $t \in \mathbb{R}$ tais que a expectativa exista (seja finita).

Exercício 34. Mostre que se M_X for diferenciável em zero, então o p -ésimo momento de X é dado por $\mathbb{E}[X^p] = M_X^{(p)}(0)$.

Exercício 35. Seja X uma variável aleatória com distribuição normal padrão, ou seja, $X \sim N(0,1)$. Mostre que a função geradora de momentos de X é dada por:

$$M_X(t) = e^{\frac{t^2}{2}}.$$

Bibliografia

Apêndice B

Guia de desigualdades

Apêndice C

Ferramentas computacionais

C.1 Git

O **Git** é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software. Ele permite que diversos desenvolvedores trabalhem simultaneamente em um projeto, acompanhando as alterações feitas no código, revertendo mudanças, criando ramificações (*branches*) e colaborando de forma eficiente.

Com o Git, o histórico de alterações de um projeto é armazenado localmente, o que possibilita o trabalho off-line e oferece grande flexibilidade na manipulação de versões.

Principais comandos do Git

A seguir, apresentamos os comandos mais básicos e úteis do Git:

- `git init`
Inicializa um novo repositório Git em um diretório.
- `git clone <URL>`
Clona um repositório remoto (por exemplo, do GitHub) para a máquina local.
- `git status`
Exibe o estado atual do repositório: arquivos modificados, não rastreados etc.
- `git add <arquivo>`
Adiciona um ou mais arquivos ao *staging area*, preparando-os para o commit.
- `git commit -m "mensagem"`
Registra as mudanças preparadas com uma mensagem descritiva.
- `git log`
Mostra o histórico de commits do repositório.
- `git diff`
Exibe as diferenças entre arquivos modificados e o último commit.

- `git branch`
Lista todas as branches do projeto.
- `git checkout <branch>`
Altera para outra branch existente.
- `git merge <branch>`
Mescla o conteúdo de uma branch à branch atual.
- `git pull`
Atualiza o repositório local com as alterações do repositório remoto.
- `git push`
Envia os commits locais para o repositório remoto.

Exemplo de fluxo básico

```
git init  
git add arquivo.txt  
git commit -m "Adiciona arquivo inicial"  
git remote add origin https://github.com/usuario/repositorio.git  
git push -u origin main
```

C.2 Python

C.3 Poetry

Referências Bibliográficas

- Bach, F. (2024). *Learning Theory from First Principles*. Adaptive Computation and Machine Learning series. MIT Press. [28](#)
- Breiman, L. (2001a). Random forests. *Machine Learning*, 45:5–32. [9](#)
- Breiman, L. (2001b). Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199 – 231. [9](#)
- Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA. [6](#), [18](#)
- Izbicki, R. and dos Santos, T. M. (2020). *Aprendizado de máquina: uma abordagem estatística*. [6](#), [8](#), [15](#), [19](#)
- James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer. [6](#), [7](#), [28](#)
- Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of Machine Learning*. The MIT Press, 2nd edition. [6](#), [10](#)
- Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning : from theory to algorithms*. [6](#), [7](#)
- Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM. [103](#), [104](#), [119](#)