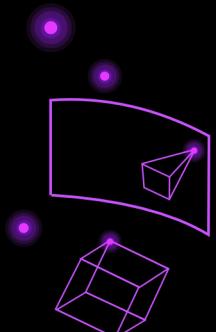


Introdução a *Machine Learning* e Redes Neurais

Autoria:

Rafael Divino Ferreira Feitosa
Danilo Turkievicz dos Santos
Rafael Teixeira Sousa



Organizadores:

Taciana Novo Kudo
Renata Dutra Braga
Deborah Silva Alves Fernandes
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho



Universidade Federal de Goiás

Reitora

Angelita Pereira de Lima

Vice-Reitor

Jesiel Freitas Carvalho

Diretora do Cegraf UFG

Maria Lucia Kons

Conselho Editorial da Coleção Formação no AKCIT

Anderson da Silva Soares

Arlindo Rodrigues Galvão Filho

Deborah Silva Alves Fernandes

Juliana Pereira de Souza Zinader

Renata Dutra Braga

Taciana Novo Kudo

Telma Woerle de Lima Soares

Equipe de produção:

Amanda Souza Vitor

Ana Laura de Sene Amâncio Zara Brisolla

Ana Luísa Silva Gonçalves

Caio Barbosa Dias

Daiane Souza Vitor

Dandra Alves de Souza

Davi Oliveira Gomes

Guilherme Correia Dutra

Iuri Vaz Miranda

Layane Grazielle Souza Dias

Luciana Dantas Soares Alves

Luis Felipe Ferreira Silva

Luiza de Oliveira Costa

Luma Wanderley de Oliveira

Suse Barbosa Castilho

Wanderley de Souza Alencar

Introdução a *Machine Learning* e Redes Neurais

Autoria:

Rafael Divino Ferreira Feitosa
Danilo Turkievicz dos Santos
Rafael Teixeira Sousa

Organizadores:

Taciana Novo Kudo
Renata Dutra Braga
Deborah Silva Alves Fernandes
Cristiane Bastos Rocha Ferreira
Arlindo Rodrigues Galvão Filho

Cegraf UFG
2024

© Cegraf UFG, 2024

© Taciana Novo Kudo

Renata Dutra Braga

Deborah Silva Alves Fernandes

Cristiane Bastos Rocha Ferreira

Arlindo Rodrigues Galvão Filho

© Universidade Federal de Goiás, 2024

© AKCIT, 2024

Revisão Técnica

Deborah Silva Alves Fernandes

Revisão Editorial

Ana Laura de Sene Amâncio Zara Brisolla

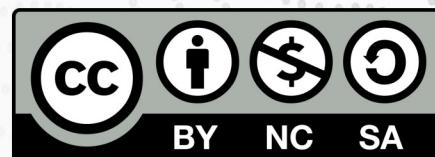
Capa

Iuri Vaz Miranda

Editoração Eletrônica

Luma Wanderley de Oliveira

Layane Grazielle Souza Dias



Esta obra é disponibilizada nos termos da Licença Creative Commons – Atribuição – Não Comercial – Compartilhamento pela mesma licença 4.0 Internacional. É permitida a reprodução parcial ou total desta obra, desde que citada a fonte.

<https://doi.org/10.5216/FEI.int.ebook.978-85-495-1018-1/2024>

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

Feitosa, Rafael Divino Ferreira
Introdução a Machine Learning e redes neurais
[livro eletrônico] / Rafael Divino Ferreira Feitosa,
Danilo Turkievicz de Souza, Rafael Teixeira Souza. --
Goiânia, GO : Cegraf UFG, 2024.

PDF

Bibliografia.

ISBN 978-85-495-1018-1

1. Ciência da computação 2. Inteligência
artificial 3. Redes neurais (Ciência da computação)
I. Souza, Danilo Turkievicz de. II. Souza, Rafael
Teixeira. III. Título.

24-241695

CDD-004

Índices para catálogo sistemático:

1. Ciência da computação 004

Eliete Marques da Silva - Bibliotecária - CRB-8/9380

Introdução a *Machine Learning* e Redes Neurais

Instituições responsáveis

Universidade Federal de Goiás (UFG)

Centro de Competência Embrapii em Tecnologias Imersivas, denominado AKCIT (Advanced Knowledge Center for Immersive Technologies)

Centro de Excelência em Inteligência Artificial (CEIA)

Instituições financiadoras

Empresa Brasileira de Pesquisa e Inovação Industrial (Embrapii)

Governo do Estado de Goiás

Empresas parceiras do AKCIT

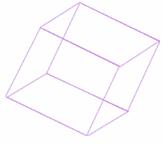
Apoio

Universidade Federal de Goiás (UFG)

Pró-Reitoria de Pesquisa e Inovação (PRPI-UFG)

Instituto de Informática (INF-UFG)





Lista de Abreviaturas e Siglas

1D	1 Dimensão
2D	2 Dimensões
3D	3 Dimensões
4D	4 Dimensões
AUC	Área Sob a Curva
BPTT	<i>Backpropagation Through Time</i> - Retropropagação Através do Tempo
CNN	<i>Convolutional Neural Network</i> - Rede Neural Convolucional
CUDA	<i>Compute Unified Device Architecture</i> - Arquitetura Unificada de Cálculo de Dispositivos
DBSCAN	<i>Density-Based Spatial Clustering of Applications with Noise</i> - Agrupamento de Aplicações Baseado em Densidade com Ruído
DL	<i>Deep Learning</i> - Aprendizado Profundo
GANs	<i>Generative Adversarial Networks</i> - Redes Generativas Adversárias
GPUs	<i>Graphics Processing Units</i> - Unidades de Processamento Gráfico
HOG	<i>Histograms of Oriented Gradients</i> - Histogramas de Gradientes Orientados
HSV	<i>Hue, Saturation and Value</i> - Matiz, Saturação e Valor
Hz	<i>Hertz</i>
IA	Inteligência Artificial
IID	<i>Independent and Identically Distributed</i> - Independentes e Identicamente Distribuídos

kNN	<i>k-Nearest Neighbors</i> - k-Vizinhos Mais Próximos
LASSO	<i>Least Absolute Shrinkage and Selection Operator</i> - Operador de Encolhimento e Seleção Absoluto Mínimo
LDA	<i>Linear Discriminant Analysis</i> - Análise Discriminante Linear
LightGBM	<i>Light Gradient Boosting Machine</i>
MAE	<i>Mean Absolute Error</i> - Erro Médio Absoluto
MFCC	<i>Mel-Frequency Cepstral Coefficients</i> - Coeficientes Cepstrais de Frequência Mel
ML	<i>Machine Learning</i> - Aprendizado de Máquina
MLP	<i>Multi-Layer Perceptron</i> - Perceptron de Múltiplas Camadas
MSE	<i>Mean Squared Error</i> - Erro Médio Quadrado
OCR	Reconhecimento Óptico de Caracteres - Reconhecimento Óptico de Caracteres
PCA	<i>Principal Component Analysis</i> - Análise de Componentes Principais
PLN	Processamento de Linguagem Natural
QDA	<i>Quadratic Discriminant Analysis</i> - Análise Discriminante Quadrática
RDA	<i>Regularized Discriminant Analysis</i> - Análise Discriminante Regularizada
ReLU	<i>Rectified Linear Unit</i> - Unidade Linear Retificada
RFE	<i>Recursive Feature Elimination</i> - Eliminação Recursiva de Atributos
RGB	<i>Red, Green and Blue</i> - Vermelho, Verde e Azul
RMSE	<i>Root Mean Squared Error</i> - Raiz do Erro Médio Quadrado
RNNAs	<i>Artificial Neural Networks</i> - Redes Neurais Artificiais
RNN	<i>Recurrent Neural Network</i> - Rede Neural Recorrente
ROC	<i>Receiver Operating Characteristic</i> - Curva de Característica de Operação do Receptor

SGBD

Database Management Systems - Sistemas de Gestão de Bancos de Dados

SSR

Sum of Squared Residuals - Soma dos Quadrados dos Resíduos

SST

Total Sum of Squares - Soma Total dos Quadrados

SUV

Sports Utility Vehicles - Veículos Utilitários Desportivos

SVM

Support Vector Machines - Máquinas de Vetores de Suporte

t-SNE

t-Distributed Stochastic Neighbor Embedding - Embutimento Estocástico de Vizinhos Distribuídos t

TF-IDF

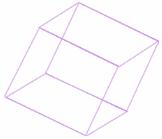
Term Frequency-Inverse Document Frequency - Frequência de Termos-Inversa Frequência de Documentos

TPUs

Tensor Processing Units - Unidades de Processamento de Tensor

XGBoost

Extreme Gradient Boosting



Lista de Figuras, Quadros e Tabelas

Figura 1 - Etapas de construção de um modelo de <i>machine learning</i>	24
Figura 2 - Aplicações de <i>machine learning</i> em diferentes setores	26
Figura 3 - Estreita relação entre inteligência artificial, <i>machine learning</i> e <i>deep learning</i>	27
Figura 4 - Série temporal com dados normais e uma anomalia em destaque	31
Figura 5 - Exemplo de reconhecimento de face	36
Figura 6 - Extração de características de uma imagem de íris	38
Figura 7 - Sequência de ácido desoxirribonucleico (DNA)	39
Figura 8 - Árvore de decisão para escolha de um carro	39
Figura 9 - Disposição de objetos em um espaço vetorial de três dimensões	42
Figura 10 - Imagem de classe, objeto e característica	42
Figura 11 - Processo de reconhecimento de padrões em <i>machine learning</i>	44
Figura 12 - Tipos de aprendizagem de máquina	48
Figura 13 - Treinamento de um algoritmo de aprendizagem supervisionada	49
Figura 14 - Treinamento de um algoritmo de aprendizagem não supervisionada	50
Figura 15 - Treinamento de um algoritmo de aprendizagem semi-supervisionada	51
Figura 16 - Treinamento de um algoritmo de aprendizagem por reforço	52
Figura 17 - Dados unidimensionais sobre altura (em cm)	57
Figura 18 - Dados bidimensionais que relacionam peso (em kg) e altura (em cm) de cinco pessoas	58
Figura 19 - Dados tridimensionais que relacionam peso (em kg), altura (em cm) e idade (em anos) de cinco pessoas	59

Figura 20 - Espaços de características bidimensional denso (à esquerda) e tridimensional esparsos (à direita)	60
Figura 21 - Espaço de características de peso e altura não normalizado (à esquerda) e normalizado (à direita)	62
Figura 22 - Transformação do espaço de características	63
Figura 23 - Exemplo tipo de fronteiras	70
Figura 24 - Na classificação binária utilizou-se uma função discriminante, na multi-classes, múltiplas funções discriminantes são utilizadas	70
Figura 25 - Matriz de covariância e as relações de co-dependência entre as variáveis	72
Figura 26 - Aplicação de transformações em vetor, (a) exemplo de autovalor e autovetor, (b) exemplo de um não autovetor e autovalor	73
Figura 27 - Gráfico com os dados do dataset iris	83
Figura 28 - Gráfico dos valores do X_train_lda	85
Figura 29 - Gráfico dos valores do X_train_lda	86
Figura 30 - Gráfico dos valores do X_train_lda	88
Figura 31 - Exemplo de dados estruturados e não estruturados	98
Figura 32 - Exemplo de tipos de dados para diferentes modelos	99
Figura 33 - Exemplo de técnicas para criação de variáveis categóricas	103
Figura 34 - Exemplo divisão do conjunto de dados	105
Figura 35 - Exemplo <i>k-fold</i>	106
Figura 36 - Exemplo de dados desbalanceados	107
Figura 37 - Exemplo do funcionamento do <i>Synthetic Minority Over-sampling Technique</i> (SMOTE)	109
Figura 38 - Exemplo de ajustes de curvas	111
Figura 39 - Exemplo de carregamento de imagem	115
Figura 40 - Exemplo de um conjunto de dados não supervisionados	120

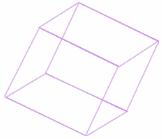
Figura 41 - Visualização de valores ausentes	126
Figura 42 - Tratamento de Outliers	129
Figura 43 - Distribuição de dados antes da normalização - MinMax Scaling	133
Figura 44 - Distribuição de dados após normalização - MinMaxScaling	135
Figura 45 - Distribuição de dados após normalização - ZCore	136
Figura 46 - Exemplo da utilização do SMOTE	147
Figura 47 - Underfitting vs Overfitting	153
Figura 48 - Underfitting: Modelo Polinomial de Grau 1	157
Figura 49 - Overfitting: Modelo Polinomial de Alto Grau	158
Figura 50 - Fitting Adequado: Modelo Polinomial de Grau 3	159
Figura 51 - Exemplo R ²	165
Figura 52 - Exemplo de erro quadrático médio (MSE)	166
Figura 53 - Exemplo de erro absoluto médio (MAE)	168
Figura 54 - Exemplo de raiz do erro quadrático médio (RMSE)	169
Figura 55 - Exemplo de árvore de regressão - dados	175
Figura 56 - Exemplo de árvore de regressão - árvore	175
Figura 57 - Comparação entre redes neurais, RNN e Feed-Forward	177
Figura 58 - Relação entre Horas Estudadas e Pontuação no Teste	180
Figura 59 - Relação entre Horas Estudadas e Pontuação no Teste - Regressão Linear	181
Figura 60 - Gráfico SSR / SST	184
Figura 61 - Gráfico dos Resíduos ao quadrado e MSE	186
Figura 62 - Gráfico dos Erros Absolutos e do MAE	188
Figura 63 - Gráfico do Quadrado dos Resíduos e do RMSE	191
Figura 64 - Comparação Dados da Corrida de Automóveis	195
Figura 65 - Comparação das Métricas entre conjuntos de dados	196

Figura 66 - Gráfico para exemplificar o processo de Regressão Linear	198
Figura 67 - Gráfico de Plotagem dos Dados	199
Figura 68 - Gráfico da Soma do Quadrado dos Resíduos	201
Figura 69 - Gráfico do SSR do Novo Modelo	202
Figura 70 - Gráfico de Regressão Linear treinado com dados apenas de bmi	206
Figura 71 - Gráfico de Regressão Ridge treinado com dados apenas de bmi	207
Figura 72 - Gráfico da Árvore de Regressão treinado com dados apenas de bmi	208
Figura 73 - Gráfico RNN treinado com dados apenas de bmi	209
Figura 74 - Exemplos de aplicações de classificação	214
Figura 75 - Exemplos de diferentes amostras de uma classe “gato”	217
Figura 76 - Mapa mental classificadores na literatura de aprendizado de máquina	217
Figura 77 - Exemplos das três espécies de flores Iris	218
Figura 78 - Distribuição das amostras de flores com base no comprimento e largura da sépala	219
Figura 79 - Exemplo de aplicação do KNN em um novo dado	219
Figura 80 - Fronteira de discriminação de um KNN	220
Figura 81 - Matriz de confusão e seus componentes	222
Figura 82 - Exemplos de Curva ROC de diferentes classificadores	227
Figura 83 - Exemplo de uma árvore de decisão sobre o problema de decidir sair de casa ou não	228
Figura 84 - Ilustração do efeito do viés e variância	230
Figura 85 - Gráfico de linha representando os valores das acurácia do modelo k-NN	238
Figura 86 - Mapa mental com os algoritmos de agrupamento	242
Figura 87 - Ilustração de um K-Means aplicado com três agrupamentos	243
Figura 88 - Exemplos de aplicação do K-means e DBSCAN	245

Figura 89 - Exemplos de agrupamentos e índice de silhueta com uso do KNN com diferentes valores de k	246
Figura 90 - Gráfico - Seaborn	250
Figura 91 - Gráfico - Visualização de Cluster	253
Figura 92 - Esquema de um neurônio artificial - Perceptron	257
Figura 93 - Funções de ativação step e sigmoide	258
Figura 94 - Esquema básico de um <i>Multi-Layer Perceptron</i>	259
Figura 95 - Esquema básico de um <i>Multi-Layer Perceptron</i> com indicação de fluxo de dados no <i>feedforward</i> e <i>backpropagation</i>	260
Figura 96 - Exemplo de vetor gradiente em uma função com duas variáveis	261
Figura 97 - Funções de ativação sigmoide e ReLU	262
Figura 98 - Esquema básico de um <i>Multi-Layer Perceptron</i> ao receber uma imagem	264
Figura 100 - Esquema da AlexNet	265
Figura 101 - Exemplo de imagens do ImageNet	266
Figura 102 - Visualização das convoluções obtidas ao treinar uma CNN	267
Figura 103 - Visualização da atenção em um problema de tradução	268
Quadro 1 - Detecção, previsão e reconhecimento de <i>machine learning</i>	32
Quadro 2 - Comparativo entre as métricas abordadas resume essa seção	170
Tabela 1 - Comparativo entre problemas de detecção, reconhecimento e previsão	32
Tabela 2 - Exemplos de classes e objetos pertencentes a elas	43
Tabela 3 - Resumo das vantagens e desvantagens dos algoritmos que usam funções discriminantes	94
Tabela 4 - Resumo dos conjuntos de dados Iris, MNIST e CIFAR-10	100



Sumário



Apresentação	19
Unidade I - Machine Learning	20
1.1 O Que é Machine Learning?	21
1.1.1 Etapas de Construção de um Modelo de Machine Learning	22
1.1.1.1 Coleta de Dados	22
1.1.1.2 Processamento de Dados	22
1.1.1.3 Treinamento de Modelos	23
1.1.1.4 Aplicação e Tomada de Decisão	23
1.2 Aplicações de Machine Learning	26
1.3 Inteligência Artificial x Machine Learning x Deep Learning	26
Unidade II - Detecção, reconhecimento e previsão	29
2.1 Detecção de padrões e anomalias	30
2.2 Detecção, Previsão e Reconhecimento	31
Unidade III - Reconhecimento de padrões	34
3.1 O Que é Reconhecimento?	35
3.1.1 Treinamento de Modelos	36
3.2 Padrões	37
3.2.1 Arranjos de Padrões	37
3.2.2 Objetos e Características	40
3.2.3 Classes	41

Unidade IV - Tipos de Aprendizagem de Máquina	46
4.1 Aprendizagem Supervisionada	48
4.2 Aprendizagem Não Supervisionada	49
4.3 Aprendizagem Semi-supervisionada	50
4.4 Aprendizagem por Reforço	51
Unidade V - Espaço de Características	54
5.1 Extração de Características	55
5.2 Seleção de características	56
5.3 Características vs. Dimensionalidade	57
5.4 Maldição da Dimensionalidade	60
5.5 Normalização do Espaço de Características	61
5.6 Transformação do Espaço de Características	62
5.7 Redução da Dimensionalidade	63
5.8 Seleção de Características vs. Redução da Dimensionalidade	64
Unidade VI - Funções Discriminantes	66
6.1 Introdução	67
6.2 Aplicações	67
6.3 Conceitos Importantes	68
6.3.1 Espaço de Características	69
6.3.2 Fronteiras de Decisão	69
6.3.3 Classificação Binária vs. Multiclasse	70
6.4 Principais Algoritmos	71
6.4.1 Conceitos Matemáticos Importantes	71
6.4.1.1 Matriz de Covariância ou de Dispersão	71
6.4.1.2 Autovetores e Autovalores	73
6.4.2 Análise Discriminante Linear (LDA)	74
6.4.2.1 Detalhamento do Código	75

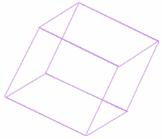
6.4.3. Análise Discriminante Quadrática (QDA)	77
6.4.4. Análise Discriminante Regularizada (RDA)	79
6.5 Notebook Colab	81
 Unidade VII - Conjuntos de dados	 95
7.1 A Importância dos Dados para a Construção de Modelos de ML	96
7.2 Dados Estruturados vs. Dados Não Estruturados	97
7.3 Conjuntos de Dados Supervisionados, Não Supervisionados e Semi-supervisionados	98
7.4 Limpeza de Dados (Tratamento de Valores Ausentes, <i>Outliers</i> , Erros de Digitação)	101
7.5 Pré-processamento (Normalização, Codificação de Variáveis Categóricas)	102
7.6 Seleção e Engenharia de Características (Feature Selection e Feature Engineering)	104
7.7 Divisão em Conjuntos de Treino, Validação e Teste	104
7.8 Dados Desbalanceados e Técnicas para Balanceamento	107
7.8.1 Técnica de <i>Oversampling</i>	108
7.8.2 Técnica de <i>Undersampling</i>	109
7.9 Subajuste (<i>Underfitting</i>) e Sobreajuste (<i>Overfitting</i>)	110
7.10 Notebook Colab:	112
 Unidade VIII - Regressão	 162
8.1 Métricas de Avaliação	164
8.1.1 Coeficiente de Determinação R ²	164
8.1.2 Erro Quadrático Médio (MSE)	165
8.1.3 Erro Absoluto Médio (MAE)	167
8.1.4 Raiz do Erro Quadrático Médio (RMSE)	168
8.2 Tipos de Regressão	170
8.2.1 Regressão Linear	170

8.2.2 Regressão <i>Ridge</i> , <i>Lasso</i> e <i>Elastic Net</i>	171
8.2.3 Árvores de Regressão	173
8.2.4 <i>Recurrent Neural Network</i> (RNN) ou Rede Neural Recorrente	176
8.3 Notebook Colab	178
Unidade IX - Classificação	213
9.1 Classificadores	216
9.1.1 KNN: <i>k-Nearest Neighbors</i>	218
9.2 Avaliação de Classificadores	220
9.2.1 Métricas para Classificação	221
9.2.1.1 Acurácia	222
9.2.1.2 Precisão	223
9.2.1.3 <i>Recall</i>	223
9.2.1.4 F1-score	224
9.2.1.5 Matriz de confusão	224
9.2.1.6 Acurácia em Detecção de Fraude	225
9.2.1.7 Precisão em Diagnóstico Médico (Detecção de Câncer)	225
9.2.1.8 F1-Score em Detecção de Spam	225
9.2.1.9 Curva ROC	226
9.3 Classificadores Modernos	227
9.3.1 XGBoost	228
9.3.2 LightGBM	229
9.4 Notebook Colab	230
Unidade X - Agrupamento	241
10.1 <i>K-Means</i>	243
10.2 DBSCAN	244
10.3 Avaliação de Agrupamentos	245
10.4 Notebook Colab	246

Unidade XI - Redes Neurais	255
11.1 Neurônio Artificial	256
11.2 <i>Multi-Layer Perceptron</i>	258
11.3 <i>Deep Learning</i>	262
11.3.1 CNNs: Redes Neurais Convolucionais	263
11.3.2 <i>Transformer</i>	267
11.3.3 Rede Neurais Generativas	268
11.3.4 <i>Frameworks</i> para DL	269
Unidade XII - Encerramento	271
12.1 Considerações Finais	272
Referências	274



Apresentação



Prezado(a) Participante,

Seja bem-vindo(a) ao Microcurso **Introdução a Machine Learning e Redes Neurais!**

Este microcurso faz parte da Coleção Formação e Capacitação do Centro de Competências Imersivas, uma parceria entre a Embrapii e a Universidade Federal de Goiás (UFG).

Neste Microcurso, você será introduzido aos conceitos fundamentais de *machine learning*, aprendendo a construir modelos, desde a coleta e o processamento de dados até a aplicação prática em problemas reais. Começaremos com uma visão geral do aprendizado de máquina, explorando suas principais etapas e diferenciando-o de outras áreas da inteligência artificial.

À medida que avançamos, aprofundaremos em técnicas de reconhecimento de padrões, essenciais para desenvolver modelos precisos. Abordaremos os diferentes tipos de aprendizado de máquina e exploraremos o espaço de características, discutindo como selecionar e transformar dados para otimizar o desempenho dos modelos.

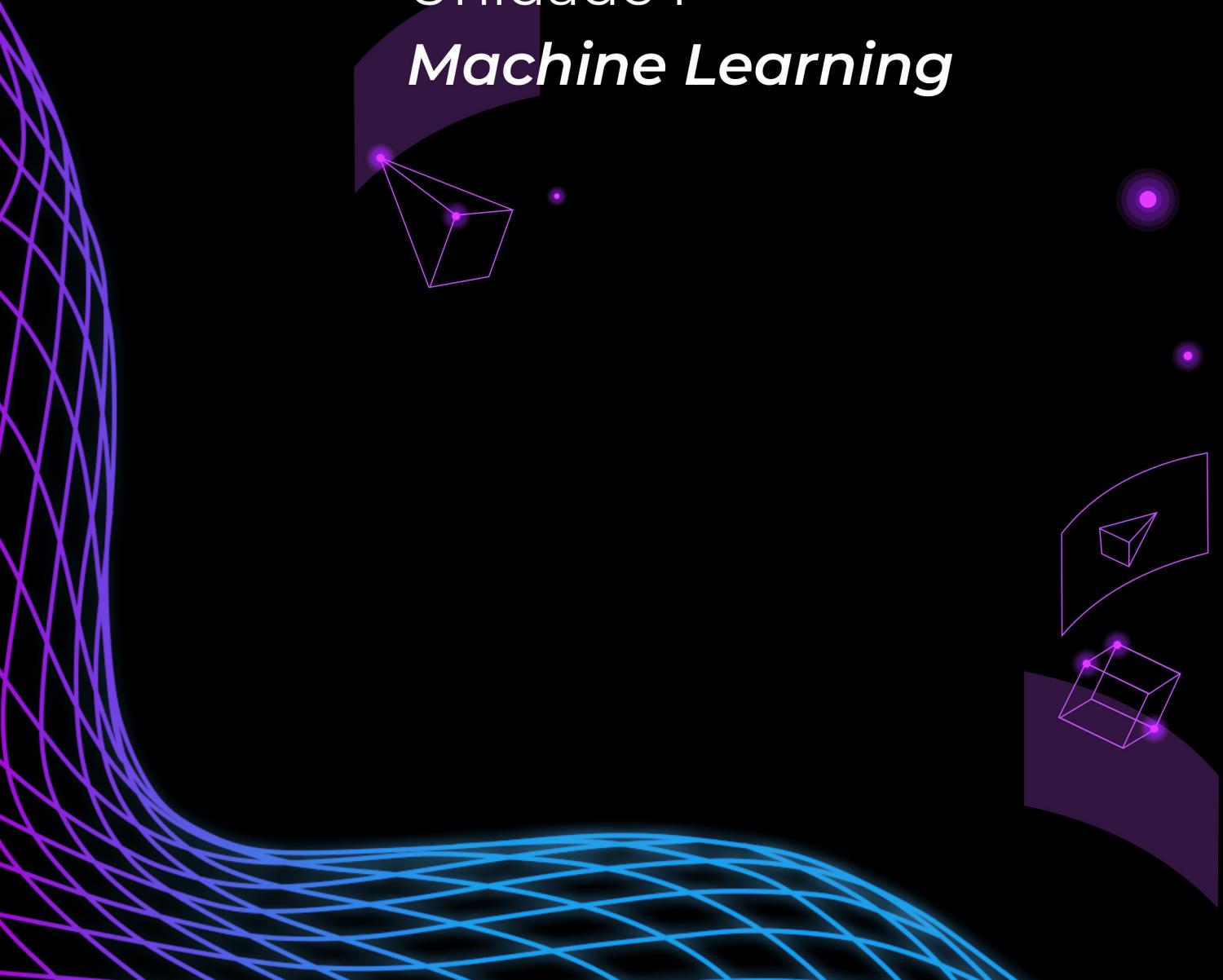
Na etapa final, exploraremos técnicas avançadas, como funções discriminantes e as aplicações práticas de regressão e classificação, além de uma introdução às redes neurais. Você terá a oportunidade de aprender sobre arquiteturas neurais modernas, compreender seus princípios e ver como aplicá-las em problemas que ilustram o potencial dessas tecnologias.

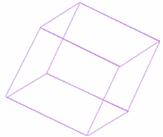
Este Microcurso oferece uma visão abrangente e prática que o prepara para os desafios do aprendizado de máquina, equipando você com as ferramentas e o conhecimento necessários para seguir em frente na sua jornada na inteligência artificial.



Desejamos a você uma excelente jornada de aprendizado!

Unidade I
Machine Learning





Unidade I - Machine Learning



1.1 O Que é Machine Learning?

Machine Learning (ML), ou aprendizado de máquina, é uma subárea da inteligência artificial (IA) que se concentra em desenvolver algoritmos e técnicas que permitem aos computadores aprender e fazer previsões ou tomar decisões com base em dados. Em vez de serem explicitamente programados para realizar uma tarefa, os sistemas de ML são treinados em grandes volumes de dados e são capazes de melhorar seu desempenho ao longo do tempo.

No contexto de ML, diferentemente de aplicações utilitárias, algoritmos são métodos ou conjuntos de regras que os sistemas de computador seguem para realizar tarefas específicas de aprendizado. Diferente da programação tradicional, onde o comportamento do software é definido por um conjunto de instruções explícitas codificadas por programadores, os algoritmos de ML permitem que os sistemas de computador aprendam padrões e tendências diretamente dos dados.

Para entendermos melhor os algoritmos utilizados em ML, para além de um simples ciclo de processamento de dados, listamos abaixo algumas características que os diferenciam:

1. **Aprendizado baseado em dados:** os algoritmos de ML utilizam dados para construir modelos matemáticos. Esses modelos são ajustados durante um processo chamado treinamento, onde o algoritmo analisa os dados de entrada e ajusta seus parâmetros internos para minimizar a diferença entre suas previsões e os resultados reais.
2. **Capacidade de generalização:** um bom algoritmo de ML deve ser capaz de generalizar o conhecimento adquirido durante o treinamento para novos dados, ou seja, deve fazer previsões precisas sobre dados que não foram usados durante o treinamento.
3. **Melhoria contínua:** os sistemas de ML podem melhorar seu desempenho ao longo do tempo conforme são expostos a mais dados. Essa melhoria contínua é alcançada através de técnicas de atualização de modelos, que ajustam continuamente os parâmetros do algoritmo conforme novos dados são disponibilizados.
4. **Tomada de decisões:** com base nos padrões aprendidos, os algoritmos de ML podem fazer previsões ou tomar decisões. Por exemplo, em um sistema de

recomendação, um algoritmo pode prever quais produtos um usuário pode gostar com base em seu histórico de compras e comportamento de navegação.

1.1.1 Etapas de Construção de um Modelo de *Machine Learning*

A transformação de dados brutos em decisões inteligentes utilizando ML é um processo sistemático que envolve várias etapas essenciais, desde a coleta inicial dos dados até a aplicação prática de modelos treinados. Cada etapa desempenha um papel crucial na construção de modelos que podem aprender com os dados e fazer previsões ou tomar decisões com base neles. A seguir, explicamos as principais etapas desse processo.

1.1.1.1 Coleta de Dados

A primeira etapa é a coleta de dados brutos, que são os dados não processados coletados diretamente da fonte. Esses dados podem vir de diversas fontes, como sensores, logs de atividades, interações de usuários, registros financeiros, dados de redes sociais, entre outros. A qualidade e a quantidade dos dados coletados são fundamentais, pois eles formam a base sobre a qual todo o processo de ML será construído. Essa etapa é importante pois:

- » A coleta adequada garante que os dados capturem as informações relevantes para o problema que se deseja resolver.
- » Dados variados e de alta qualidade aumentam a capacidade do modelo de generalizar e fazer previsões precisas.

1.1.1.2 Processamento de Dados

Após a coleta, os dados brutos passam por um processo de limpeza e preparação, conhecido como pré-processamento. Esta etapa envolve a remoção de ruídos, tratamento de valores ausentes, transformação de variáveis categóricas, normalização, e extração de características. O objetivo é transformar os dados brutos em um formato estruturado e adequado para a modelagem. Essa etapa é dividida nas seguintes sub-etapas:

- » **Limpeza de dados:** remoção de inconsistências, duplicatas e valores ausentes.
- » **Transformação de dados:** conversão de dados categóricos em numéricos, normalização e padronização.
- » **Extração de características:** identificação e seleção de atributos relevantes que ajudarão na construção do modelo.

Podemos destacar os seguintes pontos que tornam essa etapa igualmente importante:

- » Dados bem processados reduzem a complexidade do modelo e aumentam sua precisão.
- » Ajuda a prevenir problemas como o *overfitting* (sobreajuste) e melhora a performance do modelo.

1.1.1.3 Treinamento de Modelos

Nesta etapa, os dados processados são utilizados para treinar o modelo de ML. O treinamento envolve a aplicação de algoritmos que aprendem a partir dos dados, ajustando parâmetros internos para minimizar o erro de previsão. Durante o treinamento, o modelo é exposto a exemplos rotulados (em aprendizado supervisionado) ou a padrões nos dados (em aprendizado não supervisionado) para identificar as relações subjacentes. São sub-etapas a serem destacadas?

- » **Divisão dos dados:** Os dados são divididos em conjuntos de treinamento e teste para avaliar a performance do modelo.
- » **Treinamento:** O modelo é ajustado iterativamente para melhorar sua capacidade de prever os resultados corretos.
- » **Validação:** Técnicas como validação cruzada são usadas para garantir que o modelo não se ajuste excessivamente aos dados de treinamento (*overfitting*).

Algumas questões devem ser levadas em consideração para execução dessa etapa, sendo elas:

- » Um bom treinamento é crucial para que o modelo aprenda os padrões corretos nos dados.
- » O modelo deve ser capaz de generalizar bem para novos dados que ele não tenha visto durante o treinamento.

1.1.1.4 Aplicação e Tomada de Decisão

Depois que o modelo é treinado e validado, ele é aplicado em novos dados para fazer previsões ou tomar decisões. Nesta fase, o modelo é implementado em um ambiente de produção onde ele pode ser usado para automatizar processos, prever resultados futuros, ou tomar decisões baseadas em dados em tempo real. São exemplos de aplicações em ambientes de produção reais:

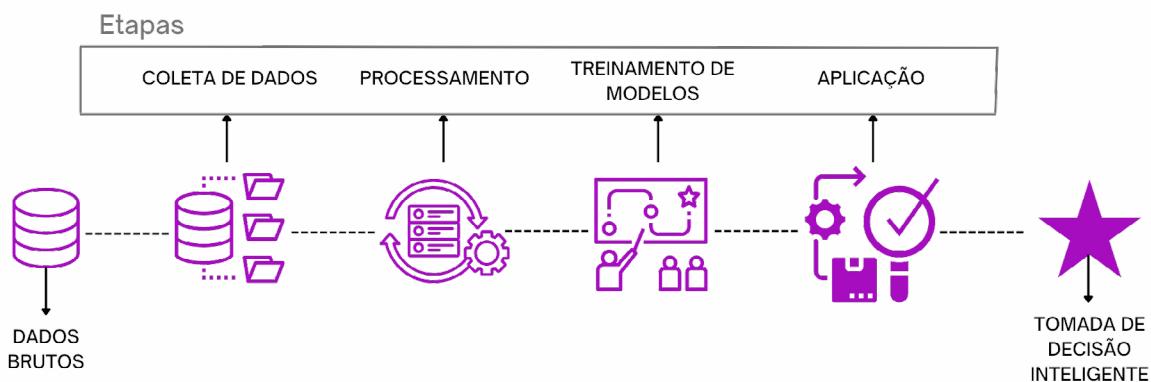
- » **Detecção de fraudes:** Identificação de transações suspeitas em sistemas financeiros.
- » **Recomendação de produtos:** Sugestão de produtos personalizados para usuários em plataformas de e-commerce (comércio digital).
- » **Previsão de demandas:** Estimativa de demanda futura para otimização de estoques.

Mesmo sendo as duas últimas etapas do processo de construção de um modelo de ML, ainda podemos destacar pontos de atenção visando garantir o sucesso do projeto:

- » A aplicação bem-sucedida do modelo permite que as organizações tomem decisões mais informadas, rápidas e precisas.
- » Modelos implementados em produção podem melhorar continuamente à medida que recebem novos dados e feedback.

Portanto, a evolução de dados brutos para a tomada de decisão inteligente em ML é um processo interativo e dinâmico, que exige atenção cuidadosa em cada etapa. Desde a coleta de dados até a aplicação do modelo, cada fase contribui para o desenvolvimento de soluções que não só aprendem com os dados, mas também melhoram a eficiência e a precisão na tomada de decisões. Este processo, ilustrado na Figura 1, é fundamental para transformar grandes volumes de dados em insights (percepções) acionáveis, permitindo que as organizações respondam de forma eficaz a desafios e oportunidades em tempo real.

Figura 1 - Etapas de construção de um modelo de machine learning



Fonte: adaptada de [Waltrick \(2020\)](#).

Também podemos compreender os algoritmos de ML dividindo-os quanto ao tipo de aprendizado, que discutiremos com mais detalhes adiante:

- » **Algoritmos supervisionados:** nestes algoritmos, o modelo é treinado com um conjunto de dados rotulados, ou seja, os dados de entrada são acompanhados por resultados desejados. Exemplos incluem regressão linear e classificadores de árvore de decisão.
- » **Algoritmos não-supervisionados:** o modelo é treinado com dados que não possuem rótulos. O objetivo é encontrar padrões ou estruturas subjacentes nos dados. Exemplos incluem algoritmos de agrupamentos como *k-means* e métodos de redução de dimensionalidade como Análise de Componentes Principais (PCA).
- » **Algoritmos semi-supervisionados:** estes algoritmos utilizam uma combinação de dados rotulados e não rotulados para o treinamento. Eles são particularmente úteis quando a rotulagem de dados é cara ou demorada.
- » **Algoritmos de aprendizado por reforço:** baseados em um sistema de recompensas e punições, esses algoritmos aprendem a tomar decisões sequenciais para maximizar uma recompensa cumulativa. Um exemplo comum é o treinamento de agentes de IA para jogar jogos de vídeo.

Citamos algumas técnicas como classificação, regressão e agrupamentos para explicar os algoritmos de ML com exemplos. Neste microcurso, você terá a oportunidade de desenvolver, testar e melhorar cada uma dessas técnicas no decorrer das unidades; até lá, abaixo citamos alguns exemplos de aplicações desses algoritmos:

- » **Classificação:** diferenciar emails em “spam” e “não spam”.
- » **Regressão:** prever o preço de uma casa com base em suas características.
- » **Agrupamentos:** agrupar clientes com comportamentos de compra semelhantes para campanhas de marketing direcionadas.
- » **Recomendações:** sugerir filmes ou produtos com base nas preferências do usuário.

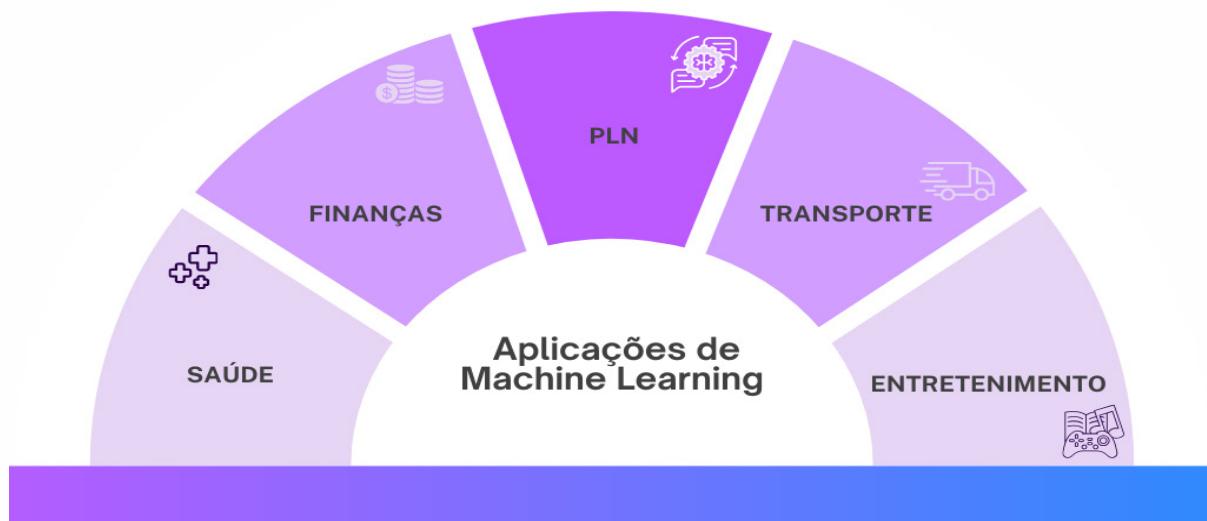
Os algoritmos são a base do aprendizado de máquina, permitindo que sistemas computacionais façam previsões e tomem decisões informadas a partir de grandes volumes de dados. Eles são fundamentais para a criação de sistemas inteligentes que podem evoluir e melhorar continuamente, ampliando as capacidades da IA.

1.2 Aplicações de Machine Learning

As aplicações de ML são vastas e se estendem por diversos setores (Figura 2). Alguns exemplos incluem:

- » **Reconhecimento de imagem:** utilizado em sistemas de segurança, diagnósticos médicos (radiologia) e redes sociais para reconhecimento facial;
- » **Processamento de linguagem natural (PLN):** aplicado em assistentes virtuais, tradução automática, análise de sentimentos e *chatbots*;
- » **Sistemas de recomendação:** utilizado por empresas como Netflix, Amazon e Spotify para sugerir produtos, filmes ou músicas com base nas preferências do usuário;
- » **Previsão de mercado:** usado em finanças para prever preços de ações, analisar risco de crédito e detectar fraudes;
- » **Veículos autônomos:** utilizam ML para processar informações sensoriais e tomar decisões em tempo real sobre navegação e segurança;
- » **Saúde:** aplicado em diagnósticos, personalização de tratamentos e predição de surtos de doenças.

Figura 2 - Aplicações de machine learning em diferentes setores



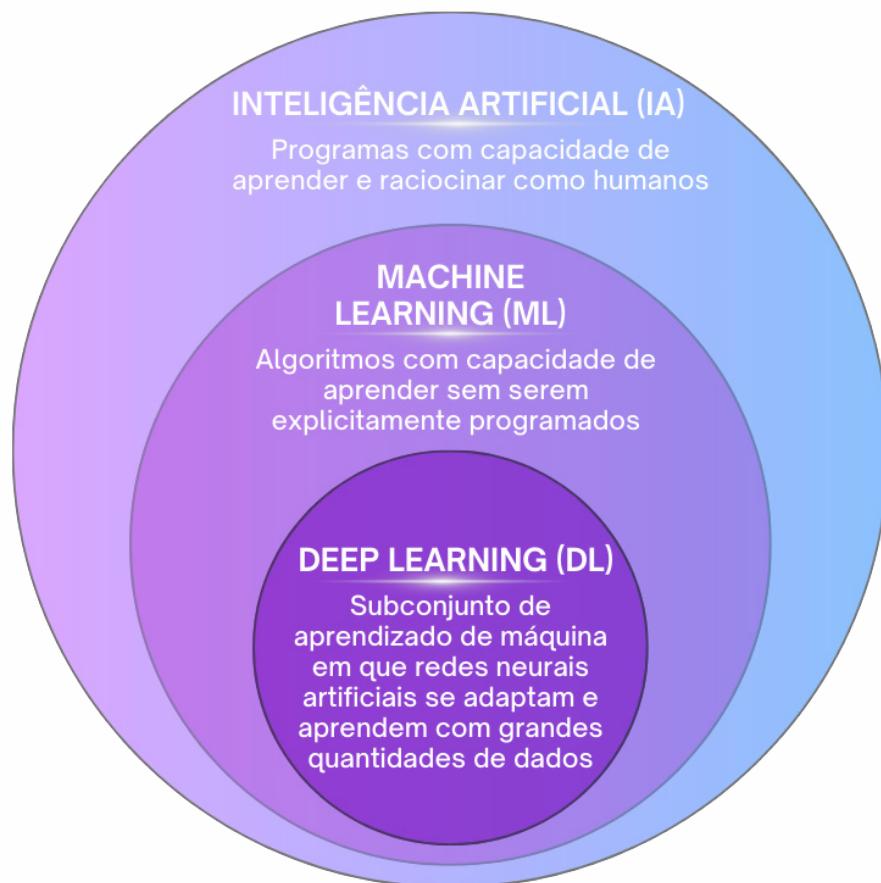
Fonte: autoria própria.

1.3 Inteligência Artificial x Machine Learning x Deep Learning

É importante entender a relação entre IA, ML e Deep Learning (DL) ou Aprendizagem Profunda. Apesar de apresentadas na Figura 3 como áreas independentes com interseções com suas correlatas, não temos um limite claro onde cada uma começa e termina. Para fins didáticos, vamos explicá-las separadamente:

1. **Inteligência artificial:** trata-se de um campo amplo da ciência da computação que visa criar sistemas capazes de realizar tarefas que normalmente requerem inteligência humana. Isso inclui raciocínio, aprendizado, percepção, e interação.
2. **Machine learning:** é um subcampo da IA focado no desenvolvimento de algoritmos que permitem às máquinas aprender a partir de dados e melhorar com a experiência. Em vez de serem explicitamente programadas para cada tarefa, as máquinas usam dados para identificar padrões e tomar decisões.
3. **Deep learning:** é uma subárea de ML que utiliza Redes Neurais Artificiais (RNAs) com muitas camadas (profundas) para modelar e aprender representações de dados complexos. DL tem sido particularmente eficaz em tarefas como reconhecimento de imagem, PLN e jogo autônomo.

Figura 3 - Estreita relação entre inteligência artificial, machine learning e deep learning



Fonte: adaptada de [Medeiros \(2019\)](#).



PARA RELEMBRAR...

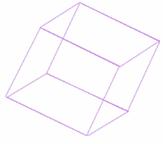
- » ML é uma subárea da IA que desenvolve algoritmos capazes de aprender e melhorar a partir de dados, em vez de serem explicitamente programados.
- » As aplicações de ML são vastas e incluem reconhecimento de imagem, PLN, sistemas de recomendação, previsão de mercado, veículos autônomos e diagnósticos médicos.
- » É crucial diferenciar entre IA, ML e DL: IA é o campo geral que busca criar sistemas inteligentes; ML é uma subárea de IA focada em algoritmos de aprendizado a partir de dados; e DL é uma subárea de ML que utiliza Redes Neurais Profundas (DL) para modelar dados complexos. Juntos, esses conceitos formam a base para a compreensão e aplicação de tecnologias inteligentes em diversas áreas.

SAIBA MAIS...

- » Artigo: “[*What is machine learning?*](#)” (IBM). Um artigo introdutório que explica o que é ML, como funciona e onde é aplicado.
- » Artigo: “[*A tour of machine learning algorithms*](#)”, de Jason Brownlee (*Machine Learning Mastery*). Um artigo abrangente que descreve diferentes tipos de algoritmos de ML, suas características e aplicações.

Unidade II
**Detecção,
reconhecimento
e previsão**





Unidade II - Detecção, reconhecimento e previsão

2.1 Detecção de padrões e anomalias

A detecção de padrões e anomalias é uma das principais tarefas no pré-processamento de dados e uma aplicação crucial em ML. Identificar padrões envolve encontrar regularidades ou estruturas repetitivas nos dados. No contexto de reconhecimento de padrões, “regularidades” ou “estruturas repetitivas” referem-se a padrões recorrentes e consistentes que podem ser identificados em conjuntos de dados. Essas regularidades são fundamentais para o aprendizado de máquina e o reconhecimento de padrões porque fornecem a base para a criação de modelos que podem generalizar para novos dados. Esses padrões recorrentes podem se manifestar como correlações entre variáveis, tendências ao longo do tempo, ou agrupamentos de dados com características semelhantes. Uma vez encontrados esses padrões, eles podem ser usados para criar modelos que façam previsões ou classifiquem novos dados. Exemplos de regularidades incluem:

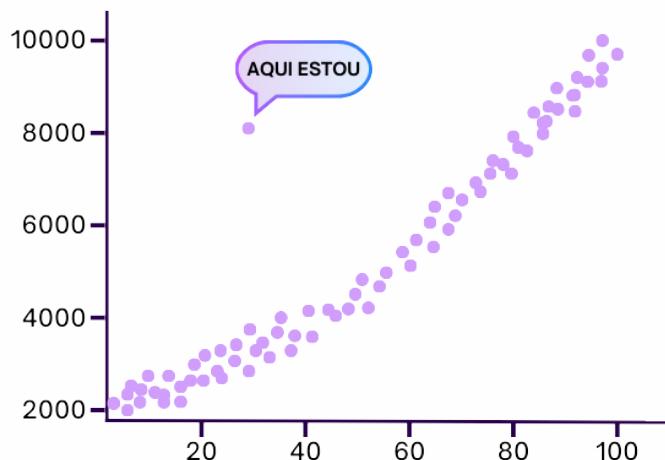
1. **Séries temporais:** em dados de séries temporais, como medições de temperatura ao longo do tempo, uma regularidade pode ser uma tendência de aumento ou diminuição. Padrões sazonais, como aumento de vendas de determinados produtos durante feriados, também são estruturas repetitivas.
2. **Imagens:** no reconhecimento de imagens, estruturas repetitivas podem ser características como bordas, texturas ou formas específicas que aparecem consistentemente em imagens de um determinado objeto ou classe de objetos.
3. **Texto:** em NLP, podem ser frequências de palavras, padrões de coocorrência de palavras, ou estruturas sintáticas que aparecem repetidamente em um corpus de texto.
4. **Dados tabulares:** em dados de clientes, por exemplo, podem ser comportamentos de compra, como a tendência de certos grupos demográficos a comprar produtos específicos. Padrões de uso, como horários de acesso a um serviço, também são estruturas repetitivas.

Como já discutimos, a identificação de regularidades ou estruturas repetitivas nos dados é fundamental para o reconhecimento de padrões. Essas regularidades são a base sobre a qual modelos de aprendizado são construídos, permitindo que os sistemas façam previsões, classifiquem dados e tomem decisões informadas. A seguir, citamos 3 situações em que a identificação de regularidades é aplicada:

- Modelagem:** identificar regularidades permite a criação de modelos que capturam relações consistentes. Por exemplo, um modelo de regressão linear pode capturar a relação entre a temperatura e a demanda por eletricidade.
- Generalização:** modelos que capturam regularidades nos dados de treinamento são capazes de generalizar para novos dados. Isso significa que o modelo pode fazer previsões precisas ou categorizações baseadas em dados não vistos, pois as regularidades subjacentes continuam presentes.
- Redução de ruído:** identificar estruturas repetitivas ajuda a separar os sinais relevantes do ruído nos dados. Isso permite que os modelos de reconhecimento de padrões se concentrem nas informações importantes e ignorem variações aleatórias, frequentemente chamadas de *outliers*.

Anomalias, por outro lado, são dados que não seguem o comportamento esperado ou padrão (veja Figura 4). Detectar anomalias é essencial para identificar erros, fraudes ou eventos raros. Por exemplo, na detecção de fraudes em transações financeiras, um sistema de ML pode aprender os padrões normais de gastos de um usuário e detectar transações que não se encaixam nesses padrões. Em manutenção preditiva, anomalias podem indicar falhas iminentes em máquinas ou equipamentos, permitindo intervenções antes que ocorram falhas catastróficas.

Figura 4 - Série temporal com dados normais e uma anomalia em destaque



Fonte: Adaptada de [Pandya, Jadeja, Degadwala \(2022\)](#).

2.2 Detecção, Previsão e Reconhecimento

Em ML, os termos “detecção”, “reconhecimento” e “previsão” referem-se a diferentes tipos de tarefas que os algoritmos podem executar. Cada um desses conceitos tem suas próprias características e aplicações específicas. Exploramos cada um deles em detalhes no Quadro 1.

Quadro 1 - Detecção, previsão e reconhecimento de machine learning



Fonte: autoria própria.

Para sintetizar essas diferenças, abaixo, na Tabela 1, estão estruturadas as características de cada abordagem.

Tabela 1 - Comparativo entre problemas de detecção, reconhecimento e previsão

Abordagem	Detecção	Reconhecimento	Previsão
Objetivo	Identificar a presença de algo	Identificar e classificar algo	Estimar valores ou eventos futuros
Foco	Existência	Identidade	Futuro
Exemplo	Detectar a presença de um carro em uma imagem	Identificar a marca e modelo do carro na imagem	Prever o número de carros que passarão por uma rua
Aplicação	Detecção de anomalias, detecção de eventos	Reconhecimento facial, reconhecimento de voz	Previsão de vendas, previsão do tempo

Fonte: autoria própria.



PARA RELEMBRAR...

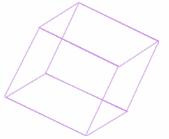
- » No pré-processamento de dados, a detecção de padrões e anomalias desempenha um papel crucial, permitindo identificar regularidades e *outliers* nos dados.
- » As tarefas de detecção, reconhecimento e previsão são fundamentais em ML, ajudando a identificar a presença de características específicas, reconhecer entidades e prever resultados futuros.
- » Essas tarefas são aplicadas em diversas áreas, desde a detecção de fraudes até a previsão de demanda, e são essenciais para a criação de modelos de ML precisos e confiáveis.

SAIBA MAIS...

- » [Artigos sobre diversos tópicos em ciência de dados e machine learning.](#)
- » [Um site que explora conceitos complexos de machine learning de forma visual e intuitiva.](#)

Unidade III Reconhecimento de padrões





Unidade III - Reconhecimento de padrões

3.1 O Que é Reconhecimento?

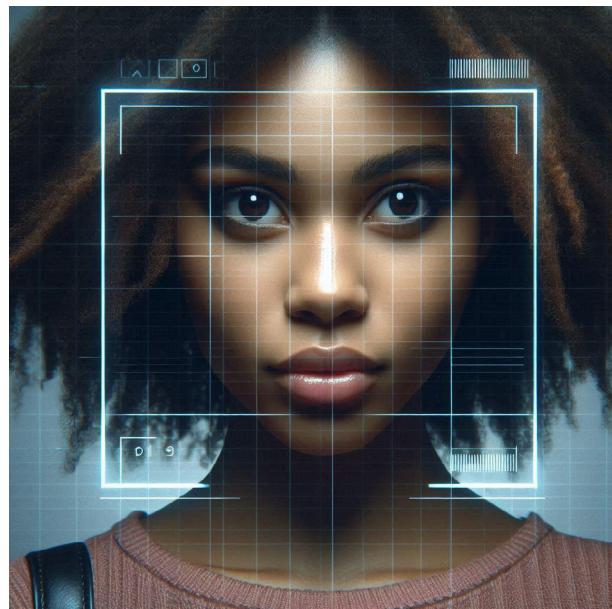
Reconhecimento é o processo de identificar ou perceber algo que já é conhecido ou familiar. Já o reconhecimento de padrões é definido por Theodoridis & Koutroumbas (2009) como a ciência do estudo da organização dos dados e pode ser realizada a partir de objetos representados em diversos tipos de dados como textos, áudios, sinais, vídeos, entre outros.

No contexto de ML e IA, reconhecimento refere-se à capacidade dos sistemas computacionais de identificar padrões específicos, objetos ou entidades em dados que correspondem a exemplos previamente aprendidos. Isso pode incluir: reconhecimento de imagens, onde um sistema identifica objetos ou pessoas em uma foto; reconhecimento de fala, onde uma máquina entende palavras e frases faladas; e reconhecimento de texto, onde um sistema identifica e extrai informações de documentos escritos. O processo de reconhecimento é fundamental para diversas aplicações, onde a habilidade de perceber semelhanças, mesmo na presença de variações, é crucial para o funcionamento eficaz do sistema.

Abaixo listamos alguns exemplos de aplicações do processo de reconhecimento de padrões:

- » **Reconhecimento facial:** sistemas que identificam indivíduos com base em características faciais que são semelhantes aos rostos presentes em um banco de dados, mesmo que haja pequenas variações como ângulos diferentes ou mudanças na expressão facial (Figura 5).
- » **Reconhecimento de voz:** assistentes virtuais que compreendem e respondem a comandos de voz ao identificar padrões de fala semelhantes aos exemplos treinados, mesmo que a entonação ou o ritmo da fala variem.
- » **Reconhecimento de texto:** ferramentas que extraem informações relevantes de documentos ao identificar padrões de letras e palavras semelhantes aos treinados em sistemas Reconhecimento Óptico de Caracteres (OCR), mesmo que a fonte ou a formatação do texto seja diferente.

Figura 5 - Exemplo de reconhecimento de face



Fonte: banco de imagens Canva.

3.1.1 Treinamento de Modelos

Mas você deve estar pensando: na prática, como ocorre o reconhecimento de algo? Vamos detalhar todo esse processo mas, descrevendo de uma forma direta, em ML, o reconhecimento é realizado por meio da aproximação de um modelo previamente treinado.

De forma simples, um modelo em ML é um conjunto de regras ou um sistema matemático que foi produzido durante a etapa de treinamento para fazer previsões ou tomar decisões com base em dados. Pense nele como uma fórmula que aprende a partir de exemplos fornecidos e é capaz de aplicar esse aprendizado a novos dados.

Para reconhecer novos dados, o sistema utiliza o modelo que foi treinado com dados conhecidos. Durante o treinamento, o modelo aprende a identificar padrões e características nos dados de entrada. Quando um novo dado é apresentado, o modelo compara esse dado com os padrões aprendidos e faz uma previsão ou classificação com base na similaridade. E essa similaridade é uma medida numérica de quão parecidos dois itens são entre si, determinando o grau de semelhança entre eles.

Por exemplo, em reconhecimento facial, um modelo é treinado com milhares de imagens de rostos diferentes. O modelo aprende as características faciais, como a distância entre os olhos, a forma do nariz e a linha do queixo. Quando uma nova imagem de rosto é apresentada, o modelo analisa essas características e tenta encontrar uma correspondência com os rostos conhecidos.

Os modelos são treinados com grandes volumes de dados históricos, e é essa quantidade de informação que torna possível fazer previsões e identificar padrões em novos dados.

Agora que você sabe o que é e como é realizado o reconhecimento, vamos introduzir o próximo conceito: padrões.

3.2 Padrões

Padrão, em um primeiro momento, pode ser entendido como uma sequência ou grupo de fenômenos que se repetem em forma, comportamento, frequência, etc. Entretanto, no contexto de ML, o padrão tem outro significado. Vamos entender o padrão e outros conceitos que permearão todo o curso.

Um padrão é uma descrição quantitativa ou estrutural de um objeto que permite identificar e reconhecer características recorrentes dentro de um conjunto de dados. O que antes poderíamos chamar de padrão, na verdade, chamamos de similaridade: explicando melhor, quando observamos algo que se repete de forma parecida, vemos uma similaridade de padrões; isso porque os padrões, no contexto de ML, são agrupamentos de valores de características dos objetos dessa observação; e quando dizemos que há uma similaridade entre esses padrões quer dizer que os valores das características desses objetos tendem a ser próximos.

Abaixo listamos situações em que podemos observar similaridade de valores e/ou comportamentos:

- » **Numérico:** Sequências de números que seguem uma determinada regra, como a sequência de Fibonacci.
- » **Visual:** Formas ou cores recorrentes em imagens, como a estrutura de folhas de uma planta ou características faciais em fotos.
- » **Comportamental:** Ações ou eventos que ocorrem repetidamente, como hábitos de compra de consumidores.

3.2.1 Arranjos de Padrões

Em todas essas situações descritas nos exemplos anteriores, as observações podem ser representadas por meio de um conjunto de valores, organizados em arranjos de padrões (Bishop, 2006), ou seja, estruturas de dados com valores de características (Theodoridis & Koutroumbas, 2009). Uma vez representadas em forma de arranjos de padrões, essas observações podem ser categorizadas em classes de padrões conforme suas propriedades em comum. E esse é um dos objetivos do reconhecimento de padrões: agrupar observações similares, baseado no valor de suas características.

Na prática, os padrões podem ser organizados de várias maneiras, dependendo das características e da natureza dos dados. Dois dos principais arranjos de padrões são vetores e descrições estruturais como cadeias e árvores:

Vetores são descrições quantitativas dos objetos, representando os dados em uma forma numérica e ordem fixa. Possuem 2 características principais:

- » **Dimensões:** cada vetor possui várias dimensões ou atributos, que são as características mensuráveis do objeto.

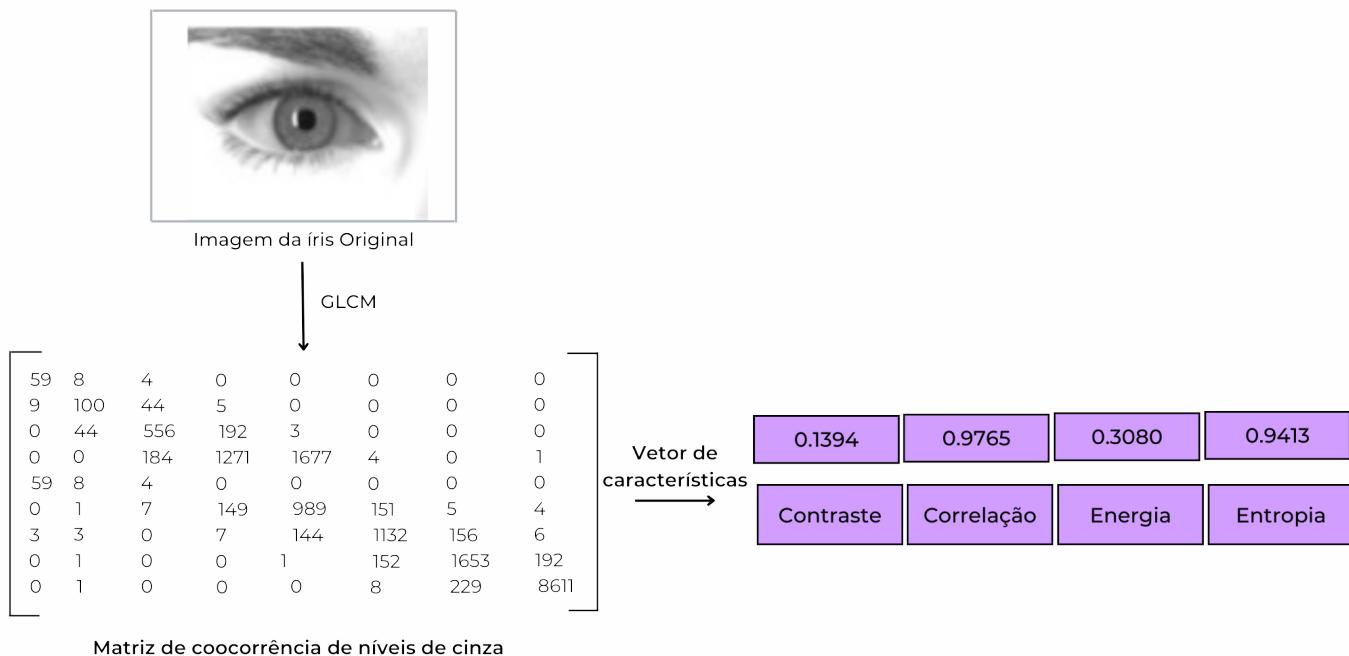
- » **Espaço vetorial:** os vetores são representados em um espaço multidimensional onde cada dimensão corresponde a um atributo.

Abaixo listamos alguns exemplos de tipos de dados e suas respectivas formas de representação em forma de vetor:

- » **Imagen:** um vetor pode representar os valores de intensidade de pixel de uma imagem.
- » **Texto:** um vetor de frequência de palavras, onde cada dimensão representa a contagem de uma palavra específica no texto.
- » **Áudio:** um vetor de características como frequência, amplitude e duração.

A Figura 6 ilustra o processo de criação de um vetor de 4 características a partir de uma imagem de uma íris.

Figura 6 - Extração de características de uma imagem de íris

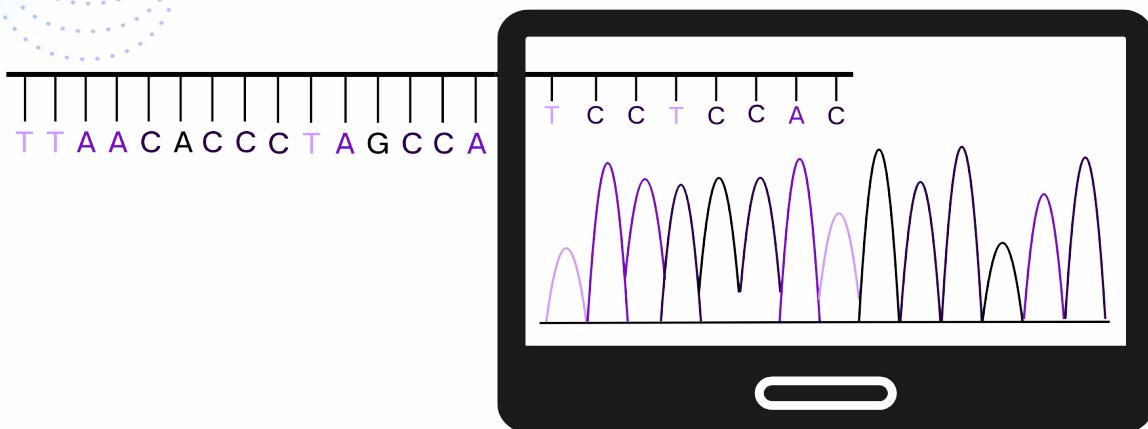


Fonte: adaptada de [Khade et al. \(2021\)](#).

Cadeias e árvores são descrições estruturais que capturam a relação entre diferentes partes dos dados. Cadeias são sequências lineares de elementos conectados de forma ordenada, por exemplo:

- » **Texto:** cadeias de caracteres ou palavras em uma frase.
- » **DNA:** sequências de nucleotídeos (Figura 7).

Figura 7 - Sequência de ácido desoxirribonucleico (DNA)



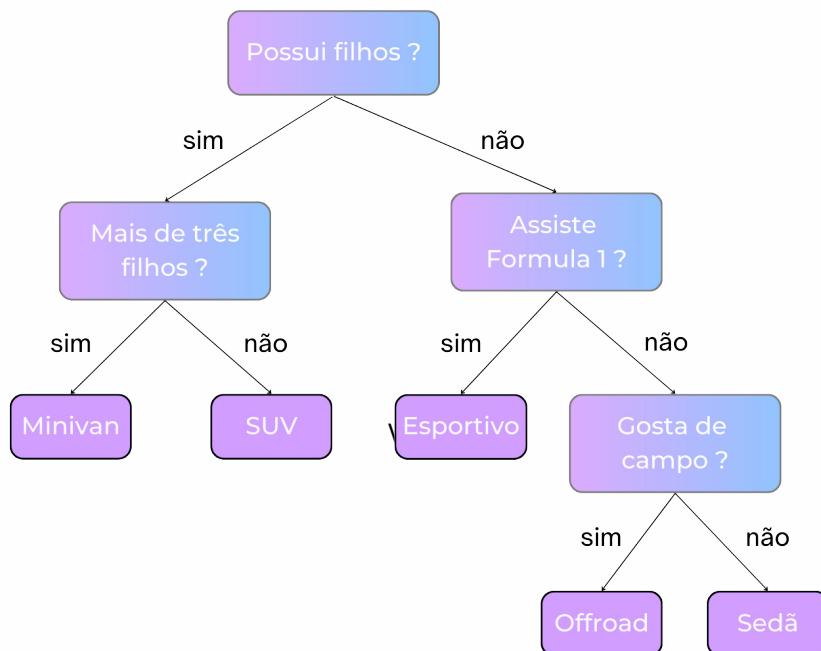
Fonte: adaptada de [Mendelikes \(2023\)](#).

Árvores são estruturas hierárquicas onde os dados são organizados em nós, com um nó raiz e ramificações para outros nós, por exemplo:

- » **Árvores de decisão:** utilizadas em algoritmos de classificação e regressão.
- » **Estruturas de dados:** árvores binárias utilizadas em bancos de dados e algoritmos de busca.

A Figura 8 ilustra um algoritmo de recomendação de tipo de carro baseado na composição familiar e preferências do usuário.

Figura 8 - Árvore de decisão para escolha de um carro



Fonte: adaptada de [Araujo \(2020\)](#).

3.2.2 Objetos e Características

Note que usamos o termo “observações” para se referir aos exemplos dados nas seções. Entretanto, vamos introduzir o termo usual: objeto. Um objeto em reconhecimento de padrões é uma entidade ou unidade de dados que possui características específicas (Mitchell, 1997) que podem ser analisadas para identificar regularidades. Esses objetos podem ser imagens, textos, sons, ou qualquer outra forma de dados estruturados ou não estruturados.

Explicando melhor, as características são propriedades mensuráveis ou categóricas de um objeto (Mitchell, 1997), como cor, forma, intensidade de pixel (em imagens), palavras ou frases (em textos), frequência e amplitude (em sinais de áudio). E quando falamos de classes (de padrões), estamos nos referindo a um grupo de objetos ou dados que possuem características ou atributos semelhantes. Esses padrões são agrupados porque compartilham propriedades que os tornam similares entre si. A seguir listamos alguns exemplos dessas propriedades nos diferentes domínios de imagem, texto e áudio:

» **Imagen:**

- » **Cor:** representada por valores de cores Red, Green and Blue (RGB), Hue, Saturation and Value (HSV) e etc. que indicam as cores vermelho, verde e azul e a tonalidade, saturação e brilho.
- » **Forma:** contornos e geometria que definem a estrutura de objetos, como círculos, quadrados ou polígonos.
- » **Intensidade de Pixel:** valores de brilho ou intensidade em cada pixel da imagem, que podem ser usados para detectar contrastes e detalhes.
- » **Textura:** padrões e variações na intensidade de pixel que indicam a superfície e a aparência de um objeto.

» **Texto:**

- » **Palavras:** conjuntos de letras ou caracteres que formam unidades de significado.
- » **Frases:** agrupamentos de palavras que constroem sentenças com significado completo.
- » **Repetições:** palavras ou termos que aparecem repetidamente em um texto, úteis para análise de tópicos ou sentimentos.

» **Áudio:**

- » **Frequência:** número de ciclos por segundo de uma onda sonora, medida em Hertz (Hz), essencial para identificar notas musicais e timbres.

- » **Amplitude:** intensidade ou volume da onda sonora, que determina a percepção de volume e potência do som.
- » **Tempo:** duração dos sons ou intervalos entre eventos sonoros, relevante para a análise de ritmo e tempo.

3.2.3 Classes

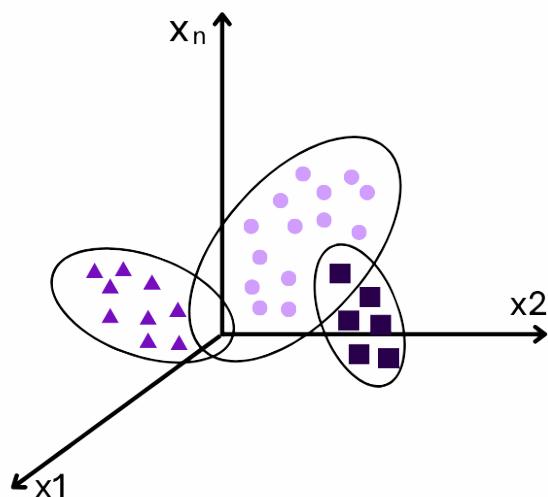
Uma classe é um rótulo ou categoria que define um grupo de objetos ou instâncias que compartilham características ou atributos comuns. As classes são usadas principalmente em tarefas de classificação, onde o objetivo é atribuir um rótulo de classe a uma nova instância com base em suas características.

Quando pretendemos determinar quais objetos pertencem à mesma classe, o compartilhamento dessas propriedades, características ou atributos similares muda conforme contexto que o algoritmo está atuando, por exemplo:

- » **Imagens de gatos:** têm propriedades comuns como orelhas pontudas, olhos grandes e pelos macios.
- » **Textos sobre tecnologia:** compartilham termos e jargões técnicos, como “algoritmo”, IA e ML.
- » **Músicas:** podem ter frequências, ritmos e estruturas semelhantes.

Na Figura 9, utilizamos um exemplo de representação dos objetos em um espaço vetorial e podemos entender como encontramos similaridade entre objetos baseado na proximidade de suas características. É possível identificar que se trata de um plano 3 Dimensões (3D): isso significa que cada objeto é representado por um vetor de características com 3 valores quaisquer. Também temos 3 agrupamentos de objetos e dizemos que estes pertencem a uma das classes: quadrado, triângulo ou círculo. Em cada uma dessas classes, os objetos estão mais próximos entre si do que de objetos de outras classes. Como esses objetos são representados por vetores com 3 valores, cada um deles indica um ponto no plano 3D. A partir desses pontos podemos calcular a distância entre eles e determinar quais estão mais próximos e, portanto, pertencem à mesma classe. Essa é uma das muitas formas de se reconhecer padrões e veremos algumas delas durante o curso. O objetivo de mostrar esse cenário é apresentar conceitos importantes que são aplicados em quaisquer algoritmos de ML. Vamos relembrá-los:

Figura 9 - Disposição de objetos em um espaço vetorial de três dimensões



Fonte: adaptada de [Lu, Wu e Kita \(2014\)](#).

- » **Objetos:** entidades ou instâncias de dados que possuem propriedades específicas e podem ser definidos em diferentes contextos, como imagens, textos, áudios, entre outros.
- » **Características:** propriedades ou atributos mensuráveis que descrevem os objetos e permitem diferenciá-los uns dos outros.
- » **Classes:** grupos ou conjuntos de objetos que compartilham um conjunto comum de características ou atributos. Representam categorias ou tipos específicos de objetos.

A Figura 10 exemplifica o caso de vários objetos de uma mesma classe Carro. Apesar de possuírem diferenças entre si, compartilham de um mesmo conjunto de características como quantidade de rodas, presença de farol, capacidade de passageiros, velocidade máxima, entre outras, mesmo com valores diferentes para cada uma dessas características.

Figura 10 - Imagem de classe, objeto e característica



Fonte: adaptada de [Scaler \(2024\)](#).

Abaixo, na Tabela 2, listamos alguns exemplos de classes e objetos. Esses exemplos ilustram como objetos diferentes podem compartilhar características semelhantes que os agrupam na mesma classe.

Tabela 2 - Exemplos de classes e objetos pertencentes a elas

Tema	Objetos	Características similares	Classe
Imagens de veículos	Um carro compacto, um sedã, e um SUV	Rodas, portas, janelas, faróis.	Veículos
Textos de notícias	Artigos sobre uma conferência internacional, uma reunião da ONU e um discurso de um chefe de estado.	Uso de termos como "diplomacia", "política internacional", "acordos".	Política internacional
Sons de instrumentos musicais	Som de um piano, um teclado eletrônico e um cravo	Frequências harmônicas, timbre percussivo	Instrumentos de tecla
Produtos eletrônicos	Um smartphone, um tablet e um e-reader	Tela sensível ao toque, conectividade à internet, bateria recarregável	Dispositivos móveis
Animais domésticos	Um gato persa, um gato siamês e um gato de rua	Bigodes, patas, comportamento felino	Gatos
Vegetais de folhas verdes	Espinafre, alface e couve	Folhas verdes, ricos em nutrientes	Vegetais de folhas verdes
Livros de ficção científica	"Duna" de Frank Herbert, "Fundação" de Isaac Asimov, e "Neuromancer" de William Gibson	Ambientação futurista, elementos de tecnologia avançada, exploração de temas científicos	Ficção científica
Aplicativos de redes sociais	Facebook, Instagram e Twitter	Perfis de usuários, compartilhamento de conteúdo, interações sociais	Redes sociais
Fotografias de paisagens naturais	Uma foto de uma montanha, uma floresta e uma praia	Elementos naturais, beleza cênica, ausência de construções humanas	Paisagens naturais
Documentos acadêmicos	Uma tese de mestrado, uma dissertação de doutorado e um artigo de conferência	Estrutura formal, revisão por pares, contribuições científicas	Publicações acadêmicas

Fonte: autoria própria.

Para que ocorra o reconhecimento de padrões com o objetivo de agrupar ou categorizar esses objetos, são necessárias algumas etapas a serem seguidas descritas na Figura 11:

- » **Pré-processamento:** preparação dos objetos para análise, como normalização e remoção de ruído (conceitos que discutiremos adiante), e extração de características relevantes.
- » **Seleção de características:** Identificação das propriedades mais significativas dos objetos que ajudarão na análise e reconhecimento.
- » **Classificação e agrupamento:** Uso de algoritmos para classificar ou agrupar os objetos com base nas características extraídas.

Figura 11 - Processo de reconhecimento de padrões em *machine learning*



Fonte: adaptada de [Marquês de Sá, 2000](#).

Então, resumindo, até agora aprendemos que, em ML, trabalhamos técnicas de reconhecimento de padrões, ou seja, podemos categorizar ou agrupar objetos em classes por meio da similaridade de suas características representadas em vetores de dados (padrões). Falaremos com mais detalhes sobre os tipos de algoritmos de aprendizagem de máquina e sobre o espaço vetorial, também conhecido como espaço de características nas próximas Unidades.



PARA RELEMBRAR...

- » O reconhecimento em ML refere-se à habilidade de identificar e categorizar objetos ou padrões com base em características previamente aprendidas. Este processo envolve a percepção de semelhanças entre os novos dados e os modelos já treinados, utilizando inferência baseada em conceitos aprendidos no passado. Em termos práticos, isso significa que o sistema de reconhecimento pode identificar rostos em imagens, detectar palavras em áudios ou classificar textos em categorias específicas, dependendo dos dados de treinamento e das características extraídas.
- » Além disso, discutimos a importância das propriedades ou características mensuráveis dos objetos, como cor, forma e intensidade de pixel em imagens, palavras ou frases em textos, e frequência e amplitude em sinais de áudio. Esses atributos são essenciais para a análise e classificação de dados. Também abordamos os principais arranjos de padrões utilizados na prática, como vetores para descrições quantitativas e cadeias e árvores para descrições estruturais.

SAIBA MAIS...

- » [Uma referência clássica para fundamentos em reconhecimento de padrões e aprendizado de máquinas.](#)
- » Site da Google DeepMind, que publica pesquisas inovadoras e artigos sobre avanços em IA, incluindo reconhecimento de padrões e previsão.

Unidade IV

Tipos de aprendizagem de máquina





Unidade IV - Tipos de Aprendizagem de Máquina

Nesta Unidade, vamos explorar os principais tipos de aprendizagem de máquina: supervisionada, não supervisionada, semi-supervisionada e por reforço. Vamos explicar cada um deles, utilizando metáforas para facilitar a compreensão, e apresentaremos exemplos para cada tipo.

No vasto campo da aprendizagem de máquina, conhecer os diferentes tipos de aprendizagem é crucial para aplicar a abordagem mais adequada a cada situação. Compreender as nuances entre a aprendizagem supervisionada, não supervisionada, semi-supervisionada e por reforço permite que você escolha a técnica que melhor se adapta aos dados disponíveis e ao problema que deseja resolver. Cada método possui suas próprias vantagens e limitações, e a escolha correta pode significar a diferença entre o sucesso e o fracasso de um projeto. Portanto, dominar esses conceitos não apenas aprimora sua capacidade de desenvolver soluções eficazes, mas também otimiza o uso dos recursos disponíveis, tornando suas análises e previsões mais precisas e eficientes.

Os diferentes tipos de aprendizagem de máquina possuem processos de treinamento distintos, refletindo suas abordagens únicas para lidar com dados. Vamos antecipar a diferença entre eles e discutiremos com mais detalhes cada um adiante.

Na aprendizagem supervisionada, o treinamento envolve dados rotulados que orientam o modelo para fazer previsões precisas. A aprendizagem não supervisionada, por outro lado, busca descobrir padrões ocultos em dados não rotulados sem orientação explícita. A aprendizagem semi-supervisionada combina ambos os métodos, utilizando um pequeno conjunto de dados rotulados e um grande conjunto de dados não rotulados para melhorar a precisão. Já a aprendizagem por reforço envolve a interação contínua com um ambiente, onde o modelo aprende através de recompensas e punições. Cada tipo de aprendizagem adapta-se a diferentes cenários e desafios, destacando a importância de entender suas características e aplicações específicas. No mapa mental apresentado na Figura 12, esses tipos de aprendizado são sintetizados.

Figura 12 - Tipos de aprendizagem de máquina



Fonte: autoria própria.

4.1 Aprendizagem Supervisionada

Na aprendizagem supervisionada, o modelo é treinado com um conjunto de dados rotulados, ou seja, cada entrada possui uma saída correspondente. Em outras palavras, cada objeto utilizado no treinamento possui a informação sobre qual classe ele pertence. O objetivo é fazer com que o modelo aprenda a mapear entradas (arranjos de padrões) para saídas corretas (classes), permitindo que ele faça previsões sobre novos dados.

Para entender melhor, imagine que você está ensinando uma criança a reconhecer frutas. Você mostra a ela várias frutas (entradas) e diz o nome de cada uma (saídas). Com o tempo, a criança aprende a associar a aparência da fruta ao seu nome. Se você mostrar uma maçã vermelha ou uma verde, a criança dirá “maçã” pois ambas compartilham de características que as tornam similares entre si, mesmo sendo de variedades distintas, e as diferencia de outras frutas.

De forma simplificada, o processo de treinamento de um modelo utilizando aprendizagem supervisionada pode ser descrito em 4 passos na Figura 13.

Figura 13 - Treinamento de um algoritmo de aprendizagem supervisionada



Fonte: autoria própria.

A classificação de emails como spam ou não-spam é um outro exemplo clássico de um problema de aprendizagem supervisionada. O objetivo é criar um modelo de ML que possa classificar automaticamente emails recebidos em duas classes ou categorias: **spam** (e-mail indesejado) e **não-spam** (e-mail legítimo):

- » **Dados de entrada:** texto dos emails.
- » **Dados de saída:** rótulos indicando se o email é spam ou não-spam.
- » **Modelo:** algoritmo de classificação ex.: *Naive Bayes* e *Support Vector Machines* (SVM).

4.2 Aprendizagem Não Supervisionada

Na aprendizagem não supervisionada não há uma medida de resultado (Hastie *et al.*, 2009), o modelo trabalha com dados não rotulados. O objetivo é encontrar padrões ou agrupamentos nos dados sem orientação explícita sobre o que procurar.

Na aprendizagem não supervisionada, não há uma medida de resultado, e o objetivo é descrever as associações e padrões entre um conjunto de medidas de entrada.

Pense em um biólogo em trabalho de campo para identificar espécies nativas. Ele não tem um mapa (rótulos), mas pode observar as árvores e plantas ao seu redor. Com o tempo, ele percebe que certas áreas da floresta têm árvores semelhantes e as agrupa mentalmente.

Com exceção dos dados não rotulados, as etapas para treinar um modelo utilizando um algoritmo de aprendizagem não supervisionado, em linhas gerais, são as mesmas do aprendizado supervisionado (Figura 14).

Figura 14 - Treinamento de um algoritmo de aprendizagem não supervisionada



Fonte: autoria própria.

Imagine que uma empresa deseja entender melhor seus clientes para oferecer produtos e serviços mais direcionados. Para isso, quer agrupar os clientes em segmentos de mercado distintos com base em suas características e comportamentos. Não se sabe quantos ou quais serão os segmentos de mercado. Esse tipo de problema é conhecido como agrupamento (ou *clustering*). Essa abordagem permite à empresa entender melhor seus clientes, resultando em estratégias mais eficazes e aumento da satisfação e lealdade do cliente:

- » **Dados de entrada:** informações sobre clientes (ex.: idade, renda, histórico de compras).
- » **Dados de saída:** agrupamentos de clientes com características semelhantes.
- » **Modelo:** algoritmo de *clustering* (ex.: *K-means* e *Density-Based Spatial Clustering of Applications with Noise [DBSCAN]*).

4.3 Aprendizagem Semi-supervisionada

A aprendizagem semi-supervisionada é um meio-termo entre a aprendizagem supervisionada e a não supervisionada (Chapelle *et al.*, 2006). Ela utiliza uma pequena quantidade de dados rotulados e uma grande quantidade de dados não rotulados. O modelo aproveita os dados rotulados para aprender e, em seguida, refinar seu conhecimento com os dados não rotulados.

Imagine um estudante que recebe algumas respostas corretas de um professor (dados rotulados) e, em seguida, tenta completar o restante da tarefa por conta própria usando essas respostas como referência.

Nas etapas de treinamento, incluímos o treinamento inicial e rodadas de refinamento para melhorar a acurácia do modelo (Figura 15).

Figura 15 - Treinamento de um algoritmo de aprendizagem semi-supervisionada



Fonte: adaptada de [Vecteezy \(2024\)](#).

Agora, veja a seguinte situação: imagine que temos uma coleção de documentos de texto de um fórum *online* onde os usuários discutem uma ampla variedade de tópicos. Queremos classificar esses documentos em categorias específicas, como “Tecnologia”, “Saúde”, “Educação”, etc. No entanto, só temos rótulos para um pequeno subconjunto dos documentos (talvez apenas 10% do total), enquanto a maioria dos documentos não está rotulada. A quantidade de documentos rotulados é pequena, o que torna difícil treinar um modelo de classificação tradicional com bom desempenho. Temos uma grande quantidade de documentos não rotulados que, se utilizados corretamente, podem melhorar a performance do modelo. Entretanto, a variação nos tópicos discutidos nos documentos pode ser ampla, tornando a classificação mais complexa.

- » **Dados de entrada:** texto dos documentos.
- » **Dados de saída:** etiquetas indicando a categoria dos documentos (rotulados).
- » **Modelo:** algoritmo semi-supervisionado (ex.: *Semi-Supervised SVM*).

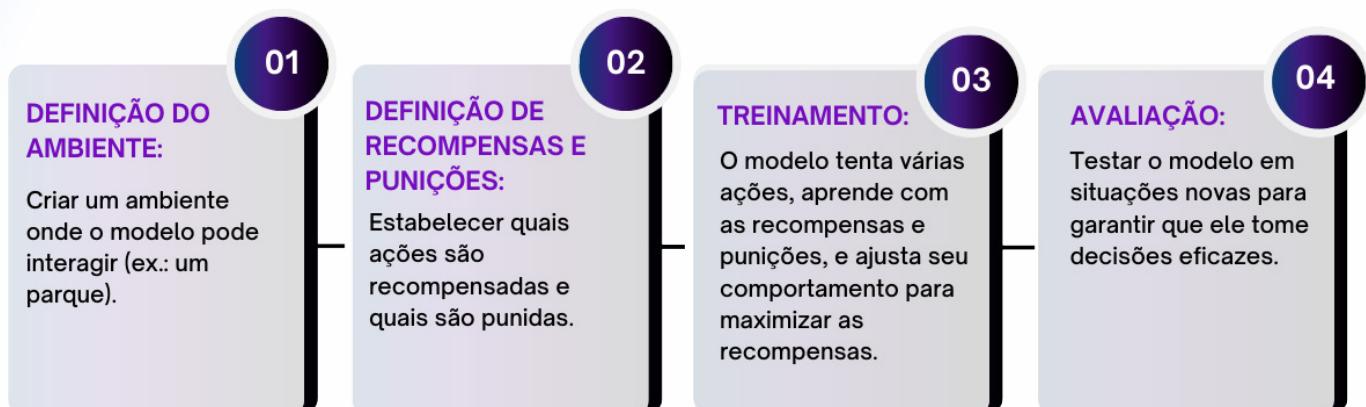
4.4 Aprendizagem por Reforço

Na aprendizagem por reforço, o modelo aprende a tomar decisões ao interagir com um ambiente. Ele recebe recompensas ou punições com base nas ações que realiza, e o objetivo é maximizar a recompensa total ao longo do tempo. Em outras palavras, problemas de aprendizagem por reforço envolvem aprender o que fazer - como mapear situações para ações - de modo a maximizar um sinal de recompensa numérica (Hastie *et al.*, 2009). É muito utilizado em jogos, robótica e sistemas autônomos.

Pense que você deseja treinar um cão a buscar uma bola. Quando o cão traz a bola de volta, você o recompensa com um petisco (recompensa). Se ele corre na direção errada, você não entrega o petisco (punição). Com o tempo, o cão aprende a buscar a bola corretamente para ganhar mais petiscos.

As etapas do processo de treinamento no caso de aprendizagem por reforço seguem outra lógica em relação às demais (Figura 16):

Figura 16 - Treinamento de um algoritmo de aprendizagem por reforço



Fonte: autoria própria.

Em um problema de aprendizagem por reforço aplicado a um jogo eletrônico de xadrez, o agente (um programa de IA) aprende a jogar xadrez interagindo com o ambiente (o tabuleiro de xadrez). O objetivo do agente é maximizar uma recompensa ao longo do tempo, que neste caso é ganhar a partida de xadrez. O agente é o jogador de xadrez controlado pelo algoritmo de aprendizagem por reforço. Esse agente deve aprender a tomar decisões sobre qual movimento fazer em cada estado do jogo. O ambiente é o próprio tabuleiro de xadrez e o estado atual do jogo, incluindo a posição de todas as peças e o histórico de movimentos. Cada estado representa uma configuração específica do tabuleiro de xadrez. Por exemplo, a posição inicial das peças no início do jogo é um estado, e qualquer configuração resultante de uma série de movimentos subsequentes também é um estado. As ações são todos os movimentos válidos que o agente pode fazer a partir de um determinado estado. No xadrez, isso inclui movimentar qualquer peça para qualquer posição permitida pelas regras do jogo. Por fim, a recompensa é um valor numérico que o agente recebe após realizar uma ação. No xadrez, uma recompensa positiva pode ser atribuída para capturar uma peça adversária, colocar o rei adversário em xeque-mate, ou ganhar a partida. Recompensas negativas podem ser atribuídas para perder uma peça, colocar o próprio rei em xeque, ou perder a partida:

- » **Dados de entrada:** estado atual do tabuleiro.
- » **Dados de saída:** ação do agente (ex.: movimento de uma peça).
- » **Modelo:** algoritmo de aprendizagem por reforço (ex.: *Q-Learning*, *Deep Q-Network*).



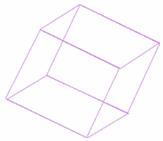
SAIBA MAIS...

- » [Artigo que revisa várias técnicas de aprendizagem de máquina aplicadas à classificação de documentos de texto.](#)
- » [Livro-texto fundamental para entender os conceitos e aplicações da aprendizagem por reforço.](#)

PARA RELEMBRAR...

- » Nesta Unidade, abordamos os diferentes tipos de aprendizagem: supervisória, não supervisionada, semi-supervisionada e por reforço. Para cada tipo, fornecemos uma explicação detalhada, metáforas para facilitar a compreensão e exemplos práticos. Discutimos também como os dados são manipulados e processados durante o treinamento de cada tipo de aprendizagem.
- » Também destacamos a importância de conhecer as diferentes abordagens de aprendizagem de máquina para escolher a melhor técnica para cada situação específica. Enfatizamos que cada tipo de aprendizagem possui um processo de treinamento único, que varia desde o uso de dados rotulados na aprendizagem supervisionada até a interação contínua com um ambiente na aprendizagem por reforço. Compreender essas nuances é essencial para desenvolver soluções eficazes e precisas em ML, otimizando assim os recursos disponíveis e garantindo melhores resultados nos projetos.

Unidade V
**Espaço de
características**



Unidade V - Espaço de Características

5.1 Extração de Características

A extração de características é uma etapa fundamental no processo de aprendizado de máquina, onde informações relevantes são derivadas dos dados brutos. Este processo visa transformar dados complexos e volumosos em um conjunto de características mais manejável e significativo para a modelagem. Por exemplo, em imagens, características como bordas, texturas e formas podem ser extraídas para facilitar a detecção ou o reconhecimento.

As características extraídas devem capturar a essência dos dados, preservando as informações críticas necessárias para as tarefas de análise subsequentes. Técnicas como Histogramas de Gradientes Orientados (HOG) para imagens, *Term Frequency-Inverse Document Frequency* (TF-IDF) para textos, e *Mel-Frequency Cepstral Coefficients* (MFCC) para áudios são exemplos comuns de métodos de extração de características. Cada técnica é escolhida com base na natureza dos dados e nos requisitos específicos da aplicação.

A extração manual de características é um processo no qual um especialista analisa os dados brutos para identificar e definir características relevantes que serão utilizadas em modelos de ML. Este processo exige um conhecimento profundo do domínio específico e das propriedades dos dados, permitindo ao especialista selecionar atributos significativos que capturam as informações mais importantes. Essas características, uma vez extraídas, são organizadas em um formato estruturado, facilitando a análise e o treinamento de algoritmos de aprendizado. A extração manual de características é essencial em muitos campos, pois garante que os modelos sejam treinados com base em dados bem definidos e altamente relevantes, aumentando assim a eficácia e a precisão das previsões. Falaremos mais sobre conjuntos de dados adiante.

Além disso, a extração de características pode ser automatizada utilizando DL. Em aplicações de DL, camadas convolucionais de Rede Neural Convolutional (CNN) extraem automaticamente características hierárquicas de imagens, enquanto Rede Neural Recorrente (RNN) ou *Transformers* (transformadores) podem capturar dependências temporais em dados sequenciais.

Não existe receita pronta: cada contexto deve ser criteriosamente estudado para se definir quais características serão extraídas manualmente pelo especialista ou qual método será empregado para a seleção automática, pois essas escolhas são cruciais para o desempenho do modelo de ML.

5.2 Seleção de características

A seleção de características é o processo realizado após a extração das características e tem como objetivo identificar e escolher um subconjunto das características mais relevantes para a construção de um modelo de ML (Guyon & Elisseeff, 2003). Esse processo é essencial para melhorar a eficiência do modelo, reduzir a complexidade computacional e prevenir o (sobreajuste) (falaremos sobre isso adiante). De forma direta, podemos entender que técnicas de seleção de características ajudam a eliminar dados redundantes ou irrelevantes que não contribuem para a precisão do modelo.

Existem vários métodos de seleção de características, incluindo métodos de filtro, wrapper¹ e baseados em modelos. Métodos de filtro utilizam estatísticas para avaliar a relevância de cada característica de forma independente do modelo. Exemplos incluem correlação, qui-quadrado e testes de informação mútua. Métodos de wrapper, como *Recursive Feature Elimination* (RFE), avaliam combinações de características utilizando o desempenho do modelo. Já os métodos baseados em modelos, como *Least Absolute Shrinkage and Selection Operator* (LASSO)², incorporam a seleção de características diretamente no processo de treinamento do modelo.

Ao eliminar características irrelevantes ou redundantes, a seleção de características simplifica o modelo, reduzindo o número de parâmetros que precisam ser ajustados durante o treinamento. Isso resulta em algoritmos mais rápidos, pois há menos dados a serem processados. Modelos simplificados necessitam de menos iterações para convergir durante o treinamento, acelerando o processo de aprendizado. Isso é particularmente vantajoso em algoritmos de otimização que dependem de múltiplas iterações, como gradiente descendente. Além disso, com menos características (mantendo, é claro, as mais relevantes), o tempo necessário para a realização dos cálculos durante o treinamento é significativamente reduzido. Isso é crucial em aplicações práticas onde o tempo de processamento é uma limitação.

Outra vantagem que determina a importância do processo de seleção de características é a redução da demanda de recursos computacionais. Menos características significam que menos dados precisam ser armazenados na memória durante o processamento. Isso é particularmente importante em sistemas com recursos limitados, como dispositivos embarcados ou aplicações móveis. A quantidade de operações matemáticas necessárias para processar os dados é reduzida, diminuindo a carga computacional e permitindo que modelos sejam treinados e executados em hardwares menos potentes. Por fim, a seleção de características facilita a escalabilidade dos modelos para grandes conjuntos de dados. Ao trabalhar com um subconjunto mais gerenciável de características, torna-se mais fácil lidar com volumes massivos de dados sem sobrecarregar os recursos computacionais.

¹ Wrapper é uma abordagem para a seleção de características em *machine learning* onde a eficácia de diferentes subconjuntos de características é avaliada diretamente pelo desempenho do modelo de aprendizado. Em vez de utilizar medidas estatísticas simples para selecionar características, o método wrapper usa o próprio algoritmo de aprendizado para determinar quais características são mais relevantes. Isso é feito testando iterativamente diferentes combinações de características e observando como a inclusão ou exclusão de cada uma afeta o desempenho do modelo.

² LASSO é uma técnica de regularização e seleção de características que adiciona uma penalidade baseada na soma dos valores absolutos dos coeficientes de regressão. Essa penalidade força a redução de alguns coeficientes a zero, efetivamente selecionando um subconjunto de características durante o treinamento do modelo.

A seleção adequada de características pode levar a modelos mais interpretáveis e eficientes, facilitando a compreensão dos dados e a tomada de decisões. Também pode ajudar a reduzir a dimensionalidade dos dados, abordando desafios associados a conjuntos de dados de alta dimensão.

5.3 Características vs. Dimensionalidade

A relação entre características e dimensionalidade é um aspecto crucial em ML. Cada característica adicionada a um conjunto de dados aumenta sua dimensionalidade, o que pode ter implicações significativas no desempenho e na complexidade do modelo. Embora a inclusão de mais características possa fornecer mais informações ao modelo, também pode levar a desafios como a maldição da dimensionalidade (abordaremos essa questão adiante). Vamos utilizar alguns exemplos para entendermos melhor como é a relação característica vs. dimensionalidade.

a) Uma dimensão (1D ou unidimensional):

- » Imagine que temos uma característica, como a altura de uma pessoa.
- » Cada pessoa pode ser representada como um ponto em uma linha, onde a posição na linha indica a altura.

Figura 17 - Dados unidimensionais sobre altura (em cm)

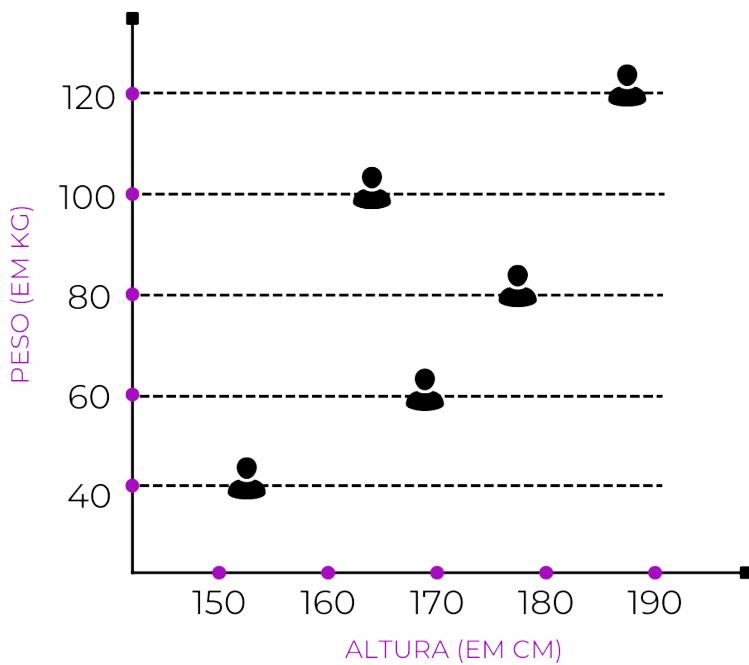


Fonte: autoria própria.

b) Duas dimensões (2D ou bidimensional):

- » Agora, adicionamos uma segunda característica, como o peso.
- » Cada pessoa agora é representada como um ponto em um plano bidimensional, onde a posição é determinada pela altura e pelo peso.

Figura 18 - Dados bidimensionais que relacionam peso (em kg) e altura (em cm) de cinco pessoas



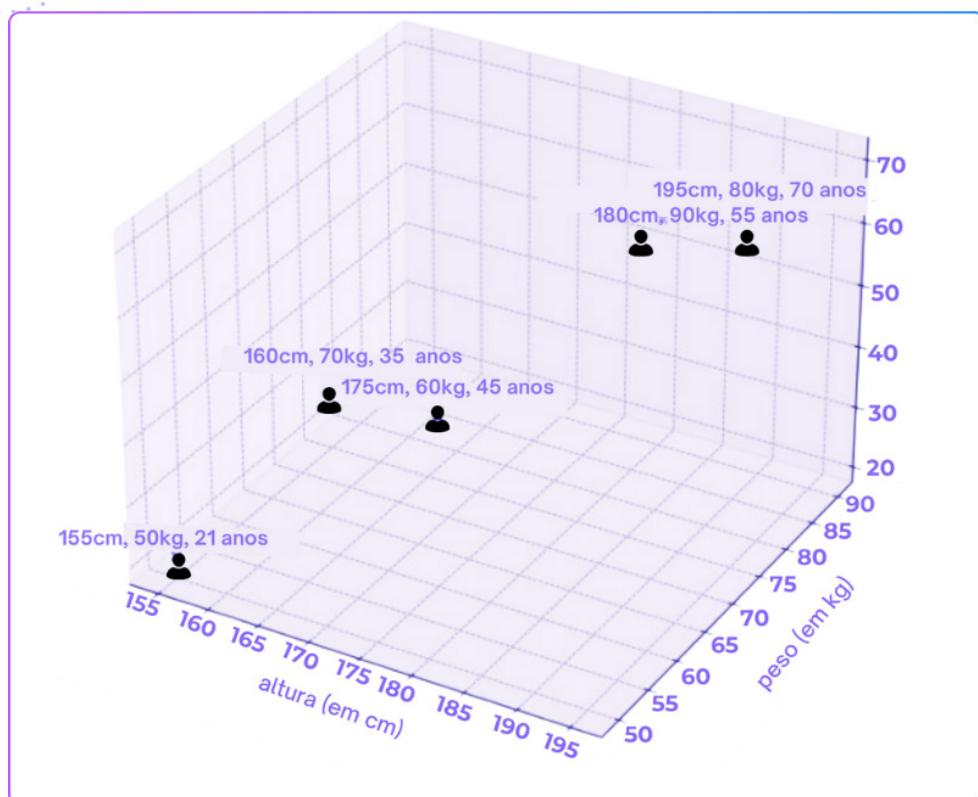
Fonte: autoria própria.

Cada ponto no gráfico da Figura 18 representa uma pessoa com uma combinação respectiva de altura e peso

c) **Três dimensões (3D ou tridimensional):**

- » Adicionando uma terceira característica, como a idade, movemos para um espaço tridimensional.
- » Cada pessoa agora é representada como um ponto em um espaço 3D, com altura, peso e idade determinando sua posição. Observe o gráfico da Figura 19 a seguir.

Figura 19 - Dados tridimensionais que relacionam peso (em kg), altura (em cm) e idade (em anos) de cinco pessoas



Fonte: autoria própria.

Cada nova característica adicionada representa uma nova dimensão no espaço dos dados. Assim:

- » 1 característica = 1 dimensão (linha);
- » 2 características = 2 dimensões (plano);
- » 3 características = 3 dimensões (espaço 3D);
- » N características = N dimensões (hiperespaço).

Em espaços de alta dimensão (hiperespaços), visualizar os dados se torna impossível, mas a matemática e os algoritmos de ML ainda podem operar nesses espaços. Cada ponto de dado em um espaço N-dimensional representa uma instância com N características, e os algoritmos de ML utilizam essas coordenadas para encontrar padrões, realizar classificações e fazer previsões.

Por exemplo, se adicionarmos uma nova característica como a “pressão arterial” ao nosso exemplo de pessoas, estariamos aumentando a dimensionalidade do nosso espaço de dados para quatro dimensões (4D) (altura, peso, idade, pressão arterial). Embora não possamos visualizar um espaço 4D, a adição dessa característica fornece mais informações que podem ser úteis para melhorar a precisão do modelo de ML.

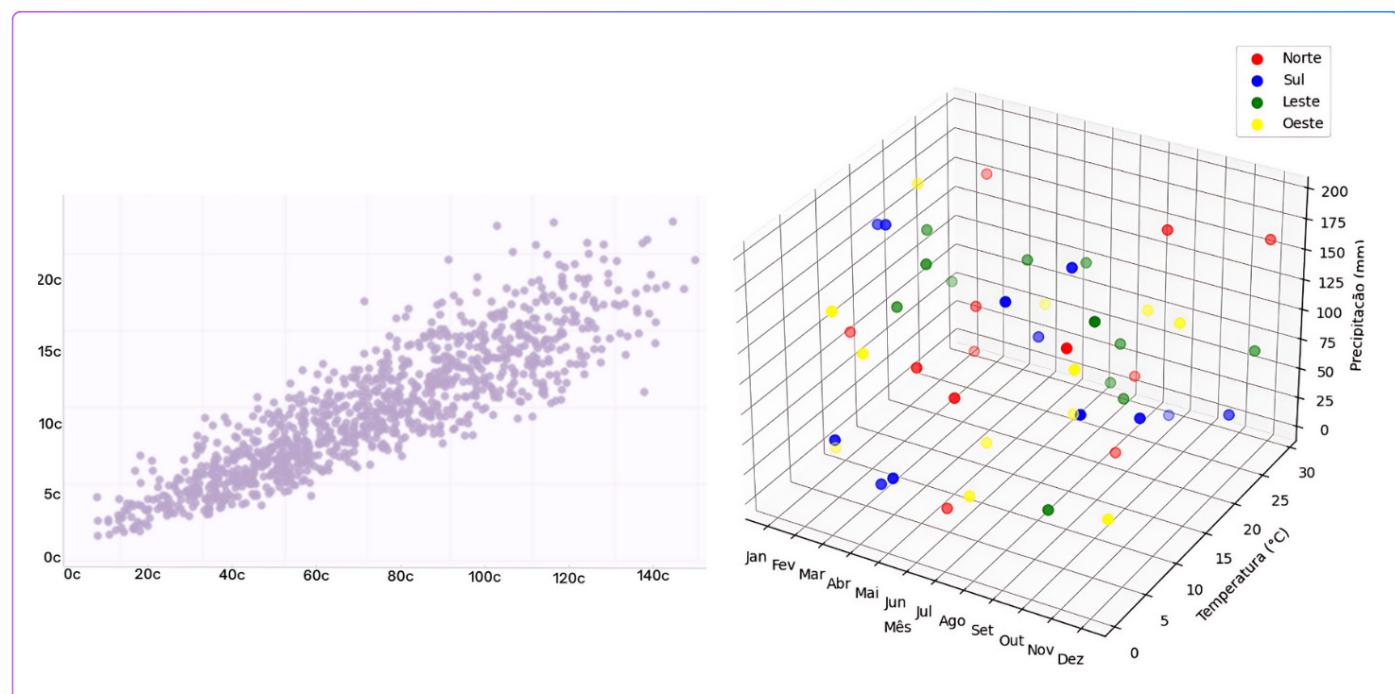
Em um espaço de alta dimensão, a densidade dos dados tende a diminuir, tornando mais difícil encontrar padrões significativos. Isso ocorre porque, à medida que a dimensionalidade aumenta, o volume do espaço de características cresce exponencialmente, e os dados se tornam esparsos. Consequentemente, os algoritmos de ML podem ter dificuldades para generalizar a partir dos dados de treinamento, resultando em *overfitting* (sobreajuste).

Portanto, equilibrar o número de características é fundamental. A seleção cuidadosa de características relevantes e a redução da dimensionalidade são práticas importantes para manter um conjunto de dados manejável e para garantir que o modelo seja capaz de aprender e generalizar de forma eficaz.

5.4 Maldição da Dimensionalidade

A maldição da dimensionalidade (Bellman, 1961) refere-se aos vários fenômenos que surgem quando se trabalha com dados em espaços de alta dimensão. À medida que a dimensionalidade dos dados aumenta, o volume do espaço aumenta de tal forma que os pontos de dados se tornam esparsos. Em consequência, muitas das intuições e técnicas que funcionam bem em espaços de baixa dimensão deixam de ser aplicáveis. Na Figura 20, há 2 gráficos representando um plano bidimensional denso e um plano tridimensional esparso. Veja que, no gráfico da Figura 22, os objetos estão distantes entre si de forma generalizada, o que dificulta encontrar similaridades.

Figura 20 - Espaços de características bidimensional denso (à esquerda) e tridimensional esparso (à direita)



Fonte: adaptada de [Pragmatic \(2023\)](#).

Uma das principais dificuldades associadas à maldição da dimensionalidade é que a distância entre os pontos de dados tende a se tornar uniforme, dificultando a distinção entre eles. Isso pode prejudicar a eficácia de algoritmos de aprendizado de máquina baseados em distância, como kNN (*k*-Nearest Neighbors) e de *clustering* (agrupamento).

Para mitigar os efeitos da maldição da dimensionalidade, técnicas de redução de dimensionalidade (aprofundaremos neste assunto adiante), como PCA e t-Distributed Stochastic Neighbor Embedding (t-SNE), são frequentemente utilizadas. Essas técnicas ajudam a projetar os dados em um espaço de menor dimensão, preservando as características mais importantes e facilitando a análise e a modelagem.

5.5 Normalização do Espaço de Características

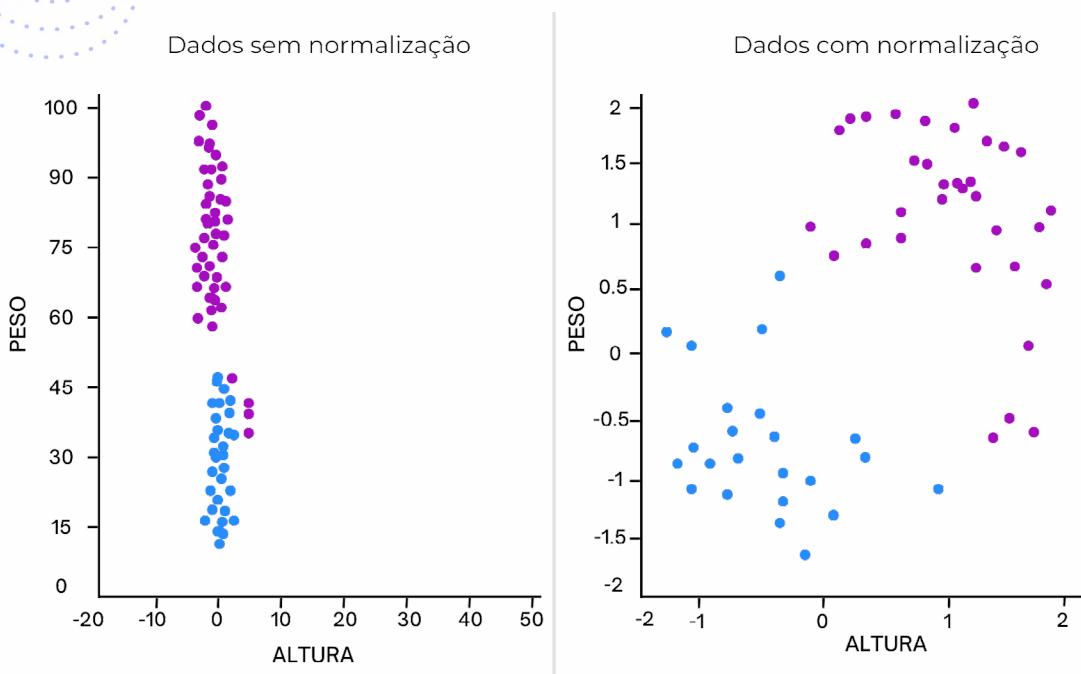
A normalização do espaço de características é uma etapa crucial no pré-processamento de dados em ML. Consiste em ajustar as escalas das características para que todas contribuam igualmente para o modelo, evitando que características com valores maiores dominem o processo de aprendizado. Isso é particularmente importante em algoritmos baseados em distância, como kNN e SVM.

Existem várias técnicas de normalização, como a padronização (*standardization*), que transforma as características para terem média zero e desvio padrão um, e a normalização min-max, que reescalas os valores para um intervalo específico, geralmente entre 0 e 1. A escolha da técnica de normalização depende da natureza dos dados e do algoritmo utilizado.

Retomando o exemplo anterior, em que apresentamos um espaço de características bidimensionais com os eixos de altura (em cm) e peso (em kg), se os valores de altura estivessem em metros, a amplitude dessa característica em relação ao peso das pessoas seria bem menor. Isso quer dizer que, sem normalização, as diferenças de escala entre essas características poderiam causar problemas nos algoritmos de aprendizado de máquina. Por exemplo, os valores de peso podem variar entre 50 e 150 kg, enquanto os valores de altura variam entre 1,5 e 2 metros. Esta diferença de escala pode fazer com que a característica de peso domine os cálculos de distância, levando a modelos que não consideram a altura de forma adequada. A normalização transforma os dados para que todas as características tenham a mesma escala, tipicamente ajustando os valores para um intervalo padrão, como 0 a 1, por exemplo. Isso garante que cada característica contribua igualmente para o modelo, melhorando a precisão, a eficiência do treinamento e a capacidade de generalização do modelo para novos dados.

A Figura 21 apresenta um mesmo espaço de características com os objetos representados por círculos. Na Figura com dados sem normalização percebemos que a escala dos dados do eixo de altura (x_1) é bem menor em relação à peso (x_2). Enquanto na outra, após a normalização dos valores entre -2 e 2, os objetos aparecem mais distribuídos, com a classe 0 (cor azul) e classe 1 (cor vermelha) mais distantes entre si, ao passo que respectivos elementos mantém a proximidade dentro da mesma classe.

Figura 21 - Espaço de características de peso e altura não normalizado (à esquerda) e normalizado (à direita)



Fonte: adaptada de [Kedarps \(2017\)](#).

Portanto, a normalização melhora a convergência de algoritmos de otimização e pode levar a um melhor desempenho geral do modelo. Além disso, facilita a comparação de diferentes características e garante que os modelos treinados sejam mais robustos e interpretáveis. Falaremos mais sobre normalização de dados mais adiante.



5.6 Transformação do Espaço de Características

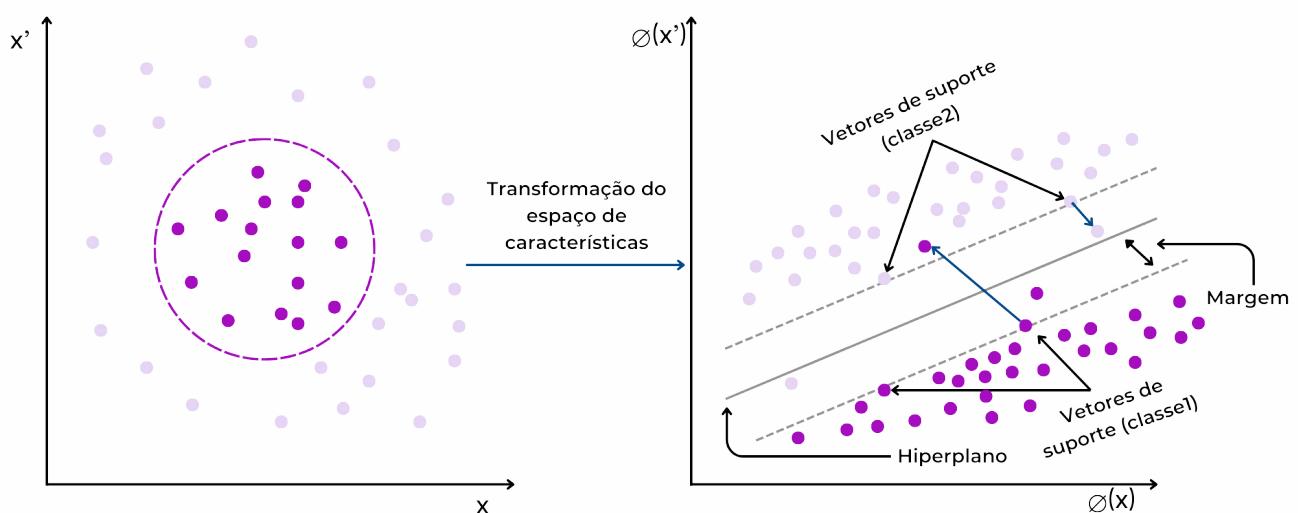
A transformação do espaço de características envolve a aplicação de técnicas para alterar a representação das características, de forma a destacar padrões ocultos nos dados ou tornar as características mais apropriadas para o modelo de ML. Apesar das técnicas de transformação do espaço de características alterarem a escala dos valores das características, não confunda com normalização. O processo de normalizar reescalas as características para mitigar possíveis desbalanceamentos nos valores que podem priorizar algumas características em detrimento de outras. Já as transformações re-escalam os valores das características para destacar padrões mais significativos e melhorar o desempenho do modelo. Transformações podem incluir a aplicação de funções matemáticas, como logaritmos, exponenciais, ou polinômios, bem como a utilização de técnicas avançadas como PCA.

PCA é uma técnica amplamente utilizada que transforma um conjunto de características originais em um novo conjunto de características ortogonais (componentes principais), que capturam a maior parte da variação presente nos dados originais. Essa transformação pode reduzir a dimensionalidade dos dados, ao mesmo tempo que preserva as informações essenciais, facilitando a visualização e a modelagem.

Outra técnica importante é o uso de *embeddings*, particularmente em PLN. *Embeddings*, como Word2Vec ou BERT, transformam palavras em vetores de características densos que capturam contextos semânticos, permitindo que os modelos de PLN lidem com a semântica de maneira mais eficaz. A transformação correta do espaço de características pode melhorar significativamente o desempenho dos modelos de ML.

Na Figura 22, é exemplificado o processo de transformação do espaço de características usando o *kernel* do SVM. De forma simplificada, o *kernel* transforma os dados de entrada para um espaço de características de maior dimensão. Esta transformação pode tornar dados que são não linearmente separáveis em seu espaço original (de baixa dimensão) linearmente separáveis no novo espaço (de alta dimensão). Nesse caso, aumentar a dimensionalidade, efeito colateral da transformação, favorece a separação das classes.

Figura 22 - Transformação do espaço de características



Fonte: adaptada de [Niyogisubizo et al. \(2023\)](#).

5.7 Redução da Dimensionalidade

A redução da dimensionalidade é o processo de diminuir o número de características em um conjunto de dados, mantendo a maior quantidade possível de informação relevante. Essa técnica é crucial para lidar com a maldição da dimensionalidade e melhorar a eficiência dos algoritmos de ML. Métodos como PCA e *Linear Discriminant Analysis* (LDA) são frequentemente utilizados para esse fim.

PCA reduz a dimensionalidade projetando os dados em um espaço de menor dimensão que capture a maior variância possível. LDA, por outro lado, busca maximizar a separabilidade entre diferentes classes ao projetar os dados em um espaço que preserva as informações discriminantes. Ambas as técnicas ajudam a simplificar o modelo, reduzindo o ruído e a complexidade computacional.

A redução da dimensionalidade não apenas melhora a eficiência do processamento, mas também pode levar a modelos mais robustos e generalizáveis. Ao focar nas características mais importantes, os modelos são menos propensos a *overfitting* (sobreajuste) e mais capazes de capturar os padrões subjacentes nos dados.

5.8 Seleção de Características vs. Redução da Dimensionalidade

A seleção de características e a redução da dimensionalidade são técnicas relacionadas, mas distintas, em ML. A seleção de características envolve a escolha de um subconjunto das características originais que são mais relevantes para a tarefa de aprendizado. Como vimos anteriormente, isso pode ser feito utilizando métodos de filtro, *wrapper* (empacotagem) ou baseados em modelos, que avaliam a importância de cada característica de forma individual ou em combinação.

Por outro lado, a redução da dimensionalidade cria novas características a partir das originais, projetando os dados em um espaço de menor dimensão. Técnicas como PCA e LDA transformam as características existentes em novas combinações que capturam a maior parte da variabilidade ou informação discriminante dos dados. Essas novas características não são um simples subconjunto das originais, mas uma reconfiguração que pode melhorar a análise.

Ambas as abordagens são valiosas para lidar com conjuntos de dados de alta dimensão, mas a seleção de características mantém a interpretabilidade das características originais, enquanto a redução da dimensionalidade pode revelar estruturas e padrões mais profundos. A escolha entre essas técnicas depende do objetivo específico e das características do conjunto de dados.



PARA RELEMBRAR...

- » Nessa Unidade, discutimos o processamento e transformação de características em ML, abordando temas como extração e seleção de características, relação entre características e dimensionalidade, maldição da dimensionalidade, normalização e transformação do espaço de características, além da distinção entre seleção de características e redução de dimensionalidade. A extração de características foi apresentada como o processo de derivar informações relevantes dos dados brutos, com exemplos de técnicas como HOG para imagens e TF-IDF para textos. A seleção de características, por sua vez, envolve identificar e escolher o subconjunto mais relevante de características, utilizando métodos como filtros, *wrappers* e técnicas baseadas em modelos.
- » Exploramos a relação entre características e dimensionalidade, destacando como cada nova característica adiciona uma nova dimensão ao espaço dos dados. Isso pode aumentar a complexidade do modelo e levar à maldição da dimensionalidade, onde a alta dimensionalidade faz com que os dados se tornem esparsos e os algoritmos de ML tenham dificuldades para generalizar. A

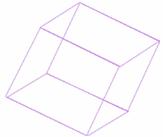
normalização do espaço de características foi enfatizada como uma prática essencial para ajustar as escalas das características, garantindo que todas contribuam igualmente para o modelo, o que é crucial para algoritmos baseados em distância, como kNN e SVM.

- » Além disso, discutimos a transformação do espaço de características, que pode incluir técnicas como PCA para redução da dimensionalidade, ajudando a simplificar os dados e melhorar a eficiência computacional. Foi feita uma distinção clara entre seleção de características e redução de dimensionalidade, onde a primeira mantém a interpretabilidade das características originais e a segunda transforma os dados em um novo espaço de menor dimensão. Abordamos também a importância de evitar o *overfitting* (sobreajuste), onde modelos complexos se ajustam excessivamente aos dados de treinamento, e como técnicas como validação cruzada e regularização podem ajudar a mitigar esse problema.

SAIBA MAIS...

- » [Artigo de revisão abrangente que revisa técnicas de seleção e extração de características.](#)
- » Revisão comparativa de diversas técnicas de redução de dimensionalidade, como PCA, t-SNE, e LDA.

Unidade VI
**Funções
discriminantes**



Unidade VI - Funções Discriminantes

6.1 Introdução

As funções discriminantes desempenham um papel fundamental no campo da aprendizagem de máquina e reconhecimento de padrões. Estas funções matemáticas são utilizadas para classificar ou categorizar objetos em diferentes classes com base em suas características ou atributos observáveis. O conceito de funções discriminantes tem suas raízes na estatística e na teoria da decisão, remontando ao início do século XX.

Um dos pioneiros no desenvolvimento de funções discriminantes foi o estatístico britânico Ronald Fisher, que introduziu a LDA em 1936. Fisher aplicou esta técnica para classificar espécies de íris com base em medidas de suas pétalas e sépalas. Seu trabalho estabeleceu as bases para muitas das abordagens modernas de classificação em aprendizagem de máquina.

Ao longo das décadas, outros pesquisadores contribuíram significativamente para o desenvolvimento das funções discriminantes e suas aplicações em ML. Entre esses estão Vladimir Vapnik, co-criador do SVM e Cedric Smith que introduziu uma variação do LDA com a Análise Discriminante Quadrática (QDA).

Formalmente, uma função discriminante é uma função matemática que mapeia um vetor de características de entrada para um valor escalar, que é então usado para tomar uma decisão de classificação. Para um problema de classificação com K classes, geralmente são definidas K funções discriminantes, uma para cada classe. A classe predita para um novo exemplo é determinada pela função discriminante que produz o maior valor para esse exemplo.

Esta abordagem fornece uma base teórica sólida para muitos algoritmos de classificação em aprendizagem de máquina, permitindo a criação de modelos que podem aprender a distinguir entre diferentes classes de objetos ou eventos com base em suas características observáveis.

6.2 Aplicações

As funções discriminantes são amplamente utilizadas em diversas aplicações de aprendizagem de máquina para solucionar problemas do cotidiano. Alguns exemplos notáveis incluem:

- » **Reconhecimento de caracteres:** em sistemas de OCR, funções discriminantes são empregadas para classificar caracteres individuais em letras ou números. Isso é crucial para a digitalização de documentos, leitura automática de placas de veículos e processamento de formulários escritos à mão.
- » **Diagnóstico médico:** em sistemas de diagnóstico, como a detecção de câncer, as funções discriminantes podem ser utilizadas para classificar se uma célula é cancerosa ou não com base em características extraídas de imagens médicas. O LDA, por exemplo, pode ser aplicado para distinguir entre células benignas e malignas, analisando parâmetros como a textura e a forma das células.
- » **Detecção de spam:** sistemas de filtragem de e-mails frequentemente utilizam funções discriminantes para classificar mensagens como spam ou não-spam. O modelo aprende a distinguir entre as duas classes com base em características do e-mail, como palavras-chave, estrutura do texto e informações do remetente.
- » **Reconhecimento de fala:** em sistemas de reconhecimento de voz, funções discriminantes são aplicadas para classificar segmentos de áudio em fonemas ou palavras específicas. Isso é fundamental para a criação de assistentes virtuais e sistemas de transcrição automática.
- » **Análise de sentimentos:** na análise de mídias sociais e avaliações de produtos, funções discriminantes são usadas para classificar o sentimento expresso em textos como positivo, negativo ou neutro. Isso auxilia empresas a entender a percepção do público sobre seus produtos ou serviços.
- » **Visão computacional:** em sistemas de reconhecimento facial, as funções discriminantes ajudam a identificar indivíduos com base em características faciais. Utilizando técnicas como o SVM, o sistema pode aprender a distinguir entre diferentes rostos, mesmo em condições de iluminação variada e ângulos diferentes.
- » **Previsão de risco de crédito:** instituições financeiras utilizam funções discriminantes para classificar clientes em diferentes níveis de risco de crédito, baseando-se em seu histórico financeiro, renda e outros fatores relevantes. Modelos como o QDA podem ser aplicados para identificar padrões em dados financeiros que indicam um risco elevado de inadimplência ou atividades fraudulentas.

6.3 Conceitos Importantes

Para compreender como as funções discriminantes podem ser utilizadas em aplicações é necessário conhecer um pouco mais. As subseções seguintes tratarão de conceitos importantes para esse entendimento.

6.3.1 Espaço de Características

O espaço de características é um conceito fundamental em ML. Ele representa um espaço multidimensional onde cada dimensão corresponde a uma característica ou atributo dos dados. Refere-se ao conjunto de todas as características ou variáveis independentes que são usadas para descrever as amostras de dados. Cada ponto nesse espaço representa uma amostra específica, e suas coordenadas correspondem aos valores das características dessa amostra.

Para ilustrar, considere um problema de classificação de flores, onde cada flor é descrita por quatro características: comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala. Aqui, o espaço de características é \mathbb{R}^4 , e cada flor é um ponto nesse espaço.

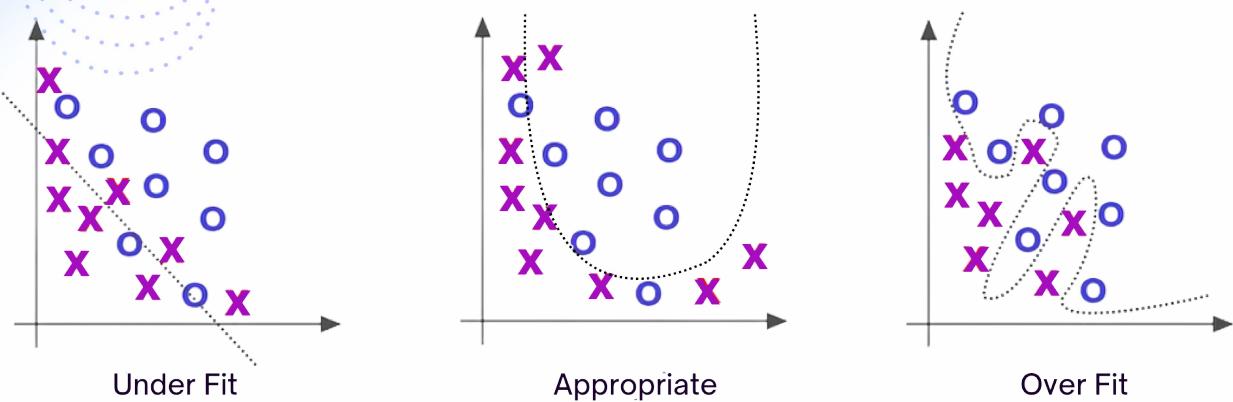
6.3.2 Fronteiras de Decisão

As fronteiras de decisão referem-se às regiões que separam o espaço de características em áreas associadas a diferentes classes. Na figura abaixo temos alguns exemplos de fronteiras de decisão para classificar X e O. O X e O podem ser, por exemplo, pessoas que foram diagnosticadas com uma doença ou não , respectivamente. Os eixos x, pode ser o valor dos resultados da contagem de leucócitos e o eixo y pode ser a quantidade de dias que a pessoa faz exercícios físicos.

As fronteiras de decisão podem ser:

- » **Fronteiras lineares:** hiperplanos que separam as classes de forma linear no espaço de características, como ilustrado na figura abaixo e a esquerda.
- » **Fronteiras quadráticas:** superfícies quadráticas que permitem separação não linear entre classes, sendo a mais simples fronteira não linear. Ela segue uma equação do segundo grau, como por exemplo . Um exemplo dessa curva aplicada à fronteira de decisão é ilustrada na figura abaixo e ao centro.
- » **Fronteiras não lineares:** superfícies de decisão complexas que podem assumir formas arbitrárias para melhor separar as classes. A figura abaixo ilustra uma complexa fronteira de decisão que separa perfeitamente X de O, contudo veremos na unidade Conjunto de Dados que nem sempre esse tipo de “perfeição” é vantajosa.

Figura 23 - Exemplo tipo de fronteiras



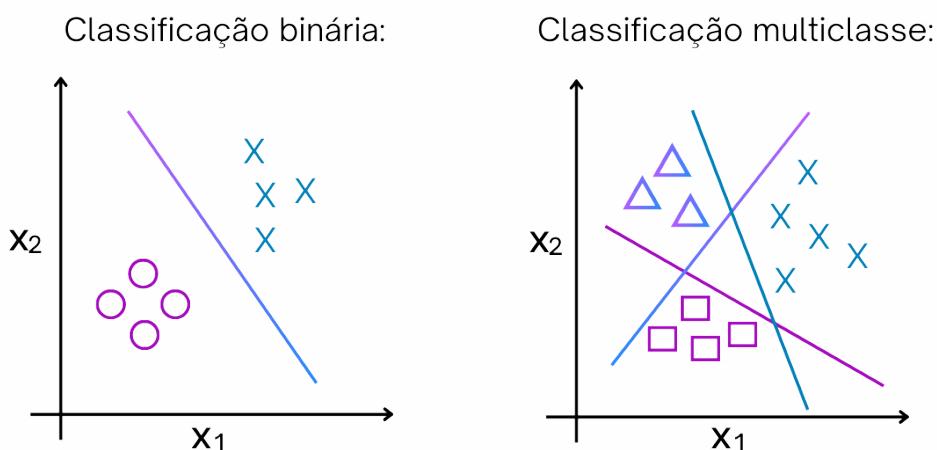
Fonte: adaptada de [Towards Data Science, 2018](#).

6.3.3 Classificação Binária vs. Multiclasse

Existem dois grandes grupos de classificação: binária e multiclasse. A classificação binária, como o próprio nome diz, refere-se apenas a duas classes, como por exemplo os casos ilustrados na figura abaixo e a direita. Nesse caso se resumem a uma única função (reta em verde) que separa as duas classes (O de X). Exemplos de métodos de classificação binária incluem Regressão Logística e SVM.

Em problemas de classificação com mais de duas classes, como classificação de imagens em diferentes categorias, são requeridas múltiplas funções discriminantes , uma para cada par de classes, como o ilustrado abaixo (Figura 24) e a direita. A reta vermelha separa o X dos quadrados e triângulos. A reta em verde separa os triângulos dos X e quadrados, por fim, a reta em azul separa os quadrados dos triângulos e X. Alguns exemplos de métodos multiclasse são a LDA e QDA.

Figura 24 - Na classificação binária utilizou-se uma função discriminante, na multi-classes, múltiplas funções discriminantes são utilizadas



Fonte: adaptada de [Medium, 2021](#).

6.4 Principais Algoritmos

Os algoritmos de aprendizagem de máquina que utilizam funções discriminantes variam em complexidade e capacidade de modelagem, os principais são:

- » **LDA:** é o algoritmo mais simples e amplamente utilizado. Ele assume que os dados seguem uma distribuição normal em cada classe e que as matrizes de covariância são iguais. É ideal para situações com baixo número de variáveis e alta dimensionalidade.
- » **QDA:** essa variante da LDA relaxa a suposição de igualdade das matrizes de covariância, permitindo que elas sejam diferentes para cada classe. É mais robusto que a LDA, mas pode ser computacionalmente mais caro.
- » **Análise discriminante regularizada (RDA):** essa técnica introduz regularização para lidar com problemas de alta dimensionalidade, onde o número de características é maior do que o número de observações.

6.4.1 Conceitos Matemáticos Importantes

Para entendimento dos algoritmos é bom termos uma noção de alguns conceitos matemáticos que são utilizados.

6.4.1.1 Matriz de Covariância ou de Dispersão

Suponha que você esteja conduzindo um estudo sobre o crescimento de diferentes espécies de plantas em um jardim. Para cada planta, você coleta dados sobre três características: altura, número de folhas e largura das folhas. Essas características podem variar de acordo com fatores como exposição ao sol, quantidade de água e tipo de solo.

A matriz de dispersão é uma ferramenta matemática que permite compreender como essas características variam em relação umas às outras. Por exemplo:

- » **Altura x Número de Folhas:** Plantas mais altas podem ter um maior número de folhas, ou essa correlação pode ser inexistente.
- » **Altura x Largura das Folhas:** Pode haver uma relação entre a altura da planta e a largura das folhas, indicando que plantas mais altas têm folhas mais largas.
- » **Número de Folhas x Largura das Folhas:** O número de folhas pode influenciar ou não a largura média das folhas.

A matriz de dispersão organiza essas relações de forma sistemática, permitindo uma análise clara de como cada par de características está relacionado. Se as características (altura, número de folhas e largura das folhas) tendem a aumentar ou diminuir juntas,

isso sugere uma correlação positiva entre elas. Caso contrário, a matriz indicará uma falta de correlação significativa.

A matriz de dispersão é composta por elementos que representam a variância e a covariância entre as diferentes características:

- » **Variância:** Refere-se à medida de quanto uma única característica (por exemplo, a altura) varia em relação à média.
- » **Covariância:** Refere-se à medida de como duas características (por exemplo, altura e número de folhas) variam juntas.

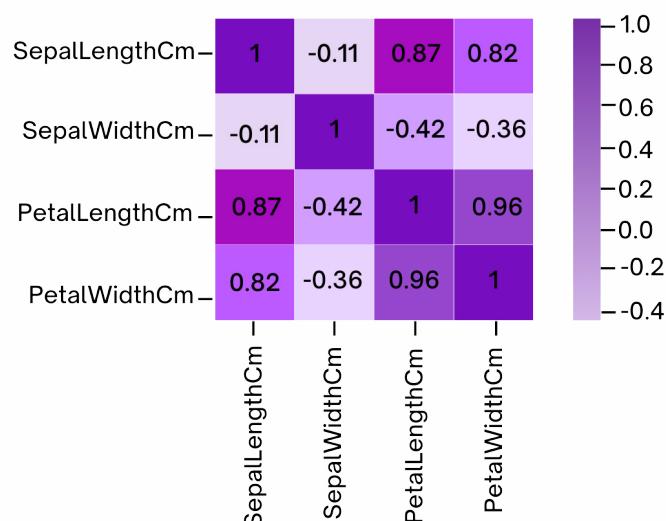
Por meio da matriz de dispersão, é possível identificar padrões nos dados que podem fornecer tendências valiosas sobre os fatores que mais influenciam o crescimento das plantas.

Se uma covariância entre duas variáveis for **positiva**, isso indica que, à medida que uma das variáveis aumenta, a outra também tende a aumentar. Por exemplo, na figura abaixo, o valor positivo na interseção de “Sepal.Length” e “Petal.Length” indica que, conforme o comprimento da sépala aumenta, o comprimento da pétala também tende a aumentar. Isso sugere uma relação linear direta entre essas duas variáveis.

Por outro lado, se uma covariância entre duas variáveis for **negativa**, isso indica que, à medida que uma das variáveis aumenta, a outra tende a diminuir. Por exemplo, o valor negativo na interseção de “Sepal.Length” e “Sepal.Width”, sugere que à medida que o comprimento da sépala aumenta, a largura da sépala tende a diminuir, indicando uma relação inversa entre essas duas variáveis.

A matriz de covariância (veja Figura 25 abaixo) permite entender as relações de co-dependência entre as variáveis, e a presença de valores positivos ou negativos ajuda a identificar o tipo de relação linear entre elas.

Figura 25 - Matriz de covariância e as relações de co-dependência entre as variáveis



Fonte: adaptada de [Hackers Realm \(2022\)](#).

6.4.1.2 Autovetores e Autovalores

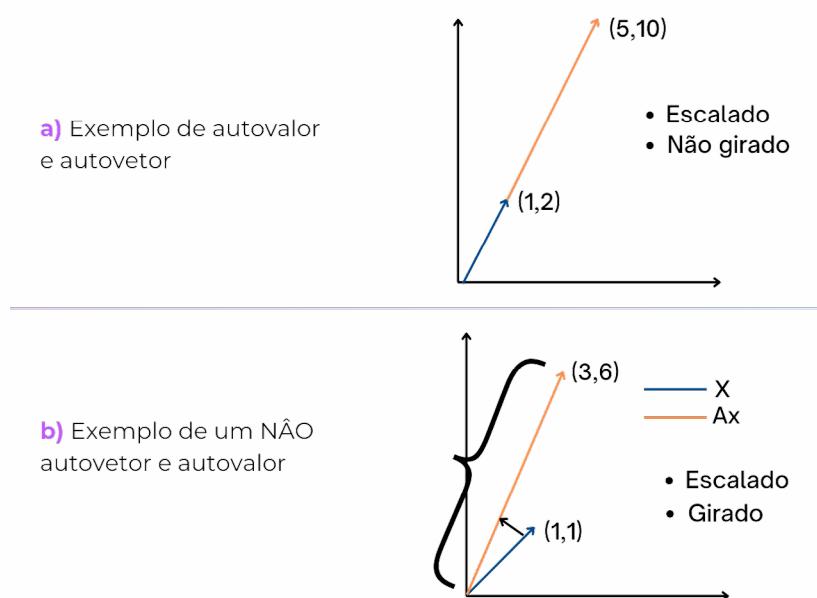
Os conceitos de autovalores e autovetores são fundamentais em diversas áreas da matemática aplicada, incluindo a análise de dados e o aprendizado de máquina. Apesar de serem conceitos avançados, eles podem ser compreendidos de maneira mais intuitiva por meio de uma analogia visual.

Imagine que você tem um vetor representado por uma flecha em um plano. Esse vetor aponta em uma determinada direção e possui um certo comprimento. Agora, suponha que aplicamos uma transformação matemática a esse vetor, como uma rotação, esticamento ou compressão. Após essa transformação, o vetor pode mudar de direção, de comprimento, ou ambos. Entretanto, em alguns casos especiais, o vetor mantém sua direção original, embora seu comprimento possa ter sido alterado. Quando isso ocorre, esse vetor é denominado **autovetor**.

O fator pelo qual o comprimento do vetor é alterado durante a transformação é chamado de **autovalor**. Ou seja, o autovalor representa o quanto o autovetor foi esticado ou comprimido, enquanto o autovetor é a direção que permanece inalterada.

Na figura abaixo, imagine que o vetor em azul passa por uma transformação matemática que resulta nos vetores em laranja. A transformação matemática aplicada a ambos os vetores em azul é a mesma (tanto no primeiro quanto no segundo gráfico da figura), porém os resultados são diferentes. No primeiro gráfico da figura 26, temos um vetor $(1,1)$ que após a transformação matemática teve rotação e alteração de tamanho, portanto pode-se dizer que o vetor azul **não é** um autovetor. Já no segundo gráfico, temos o vetor $(1,2)$ que após a transformação matemática **não teve** rotação porém teve alteração de tamanho, portanto pode-se dizer que o vetor azul **é** um autovetor e possui um autovalor de 5, que é o fator de escala quando comparados os vetores azul e laranja.

Figura 26 - Aplicação de transformações em vetor, (a) exemplo de autovalor e autovetor, (b) exemplo de um não autovetor e autovalor



Fonte: adaptada de [Solid Mechanics Classroom \(2017\)](#).

Em termos mais formais:

- » **Autovetor:** Um vetor que, após uma transformação linear, mantém sua direção original.
- » **Autovalor:** O escalar pelo qual o autovetor é multiplicado após a transformação.

Esses conceitos são amplamente utilizados na decomposição de matrizes, uma técnica que permite simplificar problemas complexos.

6.4.2 Análise Discriminante Linear (LDA)

A LDA é um método de classificação que busca encontrar uma combinação linear das características que melhor separa duas ou mais classes. O LDA assume que as classes têm a mesma matriz de covariância, mas médias diferentes.

Funcionamento do algoritmo LDA:

1. Calcula-se a média de cada classe;
2. Calcula-se a matriz de dispersão entre classes e dentro das classes;
3. Calcula-se os autovetores e autovalores da matriz de dispersão;
4. Seleciona-se os k melhores autovetores para projeção;
5. Projeta-se os dados no novo espaço e realiza a classificação.

Para ilustrar na prática uma aplicação do LDA usando o famoso conjunto de dados Iris, o código abaixo faz todo o processo do algoritmo descrito acima, cujo objetivo é classificar o tipo da flor com base nas suas características (largura e comprimento) de pétalas e sépalas.

Código 1 - Processamento do algoritmo de análise discriminante linear (LDA), usando o conjunto de dados Íris

```
# Importação das bibliotecas necessárias
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Carrega o conjunto de dados Iris
iris = load_iris()
X, y = iris.data, iris.target

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Cria e treina o modelo LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Realiza previsões e avalia a acurácia
accuracy = lda.score(X_test, y_test)
print(f"Acurácia do LDA: {accuracy:.2f}")
```

6.4.2.1 Detalhamento do Código

Código 2 - Importa as bibliotecas necessárias

```
# Importação das bibliotecas necessárias
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

- `from sklearn.discriminant_analysis import LinearDiscriminantAnalysis`: Este comando importa a classe *Linear Discriminant Analysis* (LDA) da biblioteca *scikit-learn* (comumente abreviada como `sklearn`).

- `from sklearn.datasets import load_iris`: Aqui, importamos a função `load_iris`, que carrega o conjunto de dados Iris, um dos *datasets* mais conhecidos e frequentemente utilizados em exemplos de aprendizado de máquina.

- `from sklearn.model_selection import train_test_split`: Este comando importa a função `train_test_split`, que facilita a divisão do conjunto de dados em subconjuntos de treinamento e de teste.

Código 3 - Carrega o conjunto de dados Íris

```
iris = load_iris()  
X, y = iris.data, iris.target
```

- `iris = load_iris()`: Nesta linha, o conjunto de dados Iris é carregado e armazenado na variável `iris`. Este conjunto de dados contém informações sobre três espécies de flores: setosa, versicolor e virginica; com quatro características mensuradas para cada amostra (comprimento e largura das pétalas e sépalas).

- `X, y = iris.data, iris.target`: Aqui, o conjunto de dados é dividido em duas partes:

- `X`: Representa as características das amostras: comprimento e largura das pétalas e sépalas (as variáveis independentes).

- `y`: Corresponde às classes ou rótulos das amostras (a variável dependente), indicando a qual espécie de flor cada amostra pertence (setosa, versicolor ou virginica).

Código 4 - Divide o conjunto de dados em dois subconjuntos

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- `train_test_split(X, y, test_size=0.3, random_state=42)`: Esta função divide o conjunto de dados em dois subconjuntos, seu objetivo mais claro será abordado na unidade 8:

- `X_train` e `y_train`: Contêm as amostras que serão utilizadas para treinar o modelo.

- `X_test` e `y_test`: Contêm as amostras que serão utilizadas para testar o modelo, avaliando seu desempenho.

- `test_size=0.3`: Especifica que 30% das amostras serão reservadas para o teste, enquanto os 70% restantes serão utilizados para o treinamento.

- `random_state=42`: Define um valor fixo para o gerador de números aleatórios, garantindo que a divisão dos dados seja reproduzível em diferentes execuções do código.

Código 5 - Cria o modelo de análise discriminante linear (LDA)

```
lda = LinearDiscriminantAnalysis()  
lda.fit(X_train, y_train)
```

- `lda = LinearDiscriminantAnalysis()`: Esta linha cria um modelo LDA que será utilizado para a classificação das amostras.

- `lda.fit(X_train, y_train)`: O método fit treina o modelo LDA utilizando os dados de treinamento `X_train` e `y_train`. Durante este processo, o modelo aprende as relações en-

tre as características das amostras e suas respectivas classes, ajustando os parâmetros internos para maximizar a separação entre as diferentes classes. É nessa fase que o algoritmo apresentado anteriormente é de fato aplicado.

Código 6 - Realiza previsões e avalia a acurácia do modelo

```
# Realiza previsões e avalia a acurácia
accuracy = lda.score(X_test, y_test)
print(f"Acurácia do LDA: {accuracy:.2f}")
```

- `accuracy = lda.score(X_test, y_test)`: Este método, chamado `score`, é aplicado ao modelo `lda` previamente treinado. Ele calcula a acurácia do modelo, ou seja, a proporção de previsões corretas que o modelo fez ao ser testado com o conjunto de dados `X_test` em comparação com os rótulos verdadeiros `y_test`. O resultado é guardado na variável `accuracy`, que representará a porcentagem de amostras corretamente classificadas pelo modelo.

- `print(f"Acurácia do LDA: {accuracy:.2f}")`: Esta função `print` imprime uma mensagem na tela. `f"Acurácia do LDA: {accuracy:.2f}"` é uma string formatada que inclui o valor da acurácia calculada. A expressão `{accuracy:.2f}` formata o valor da acurácia para que seja exibido com duas casas decimais, facilitando a leitura e a interpretação do resultado. Isso só é possível com o uso do `f` antes da string, que permite inserir o valor da variável `accuracy` diretamente no texto.

6.4.3 Análise Discriminante Quadrática (QDA)

A QDA é uma extensão do LDA que não assume que as classes têm a mesma matriz de covariância. Isso permite que o QDA capture relações não lineares entre as características.

Funcionamento do Algoritmo QDA:

1. Calcula a média e a matriz de covariância para cada classe;
2. Calcula a função discriminante quadrática para cada classe;
3. Classifica novos pontos de dados com base na função discriminante que produz o maior valor.

Para ilustrar na prática uma aplicação do QDA abaixo é proposto um código muito similar ao utilizado no LDA, com pequenas diferenças que serão detalhadas a seguir.

Código 7 - Processamento do algoritmo de análise discriminante quadrática (QDA)

```
# Importação das bibliotecas necessárias
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Carrega o conjunto de dados Iris
iris = load_iris()
X, y = iris.data, iris.target

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Cria e treina o modelo QDA
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)

# Realiza previsões e avalia a acurácia
accuracy = qda.score(X_test, y_test)
print(f"Acurácia do QDA: {accuracy:.2f}")
```

Nessa primeira parte do código, somente a primeira linha possui alguma diferença.

Código 8 - Importa as bibliotecas necessárias

```
# Importação das bibliotecas necessárias
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

- `from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis`: Este comando importa a classe referente ao QDA da biblioteca *scikit-learn*.

Outra parte que foi necessária adaptação é o emprego dessa classe mais a frente no código.

Código 9 - Cria e treina o modelo de análise discriminante quadrática (QDA)

```
# Cria e treina o modelo QDA
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
```

De forma similar ao visto no QDA:

- `qda = QuadraticDiscriminantAnalysis()`: Esta linha cria um modelo QDA que será utilizado para a classificação das amostras.

- `qda.fit(X_train, y_train)`: O método fit treina o modelo LDA utilizando os dados de treinamento `X_train` e `y_train`.

Por fim, a apresentação dos resultados de acurácia também precisam ser adaptados.

Código 10 - Realiza previsões e avalia a acurácia

```
# Realiza previsões e avalia a acurácia  
accuracy = qda.score(X_test, y_test)  
print(f"Acurácia do QDA: {accuracy:.2f}")
```

Em comparação com o LDA em `accuracy = qda.score(X_test, y_test)`, substitui-se `lda` por `qda` para que sejam capturados os `score` do modelo `qda`. De forma análoga, em `print(f"Acurácia do QDA: {accuracy:.2f}")` é alterado o texto da impressão de LDA para QDA.

6.4.4 Análise Discriminante Regularizada (RDA)

A RDA é uma técnica que combina LDA e QDA, permitindo um equilíbrio entre os dois métodos através de um parâmetro de regularização.

O parâmetro de regularização é uma variável crucial que controla a complexidade do modelo e a forma como a regularização é aplicada. A regularização é uma técnica utilizada para evitar o sobreajuste (*overfitting*), que ocorre quando um modelo se ajusta excessivamente aos dados de treinamento e perde capacidade de generalização para novos dados.

Ele equilibra entre a matriz de covariância empírica (calculada diretamente a partir dos dados) e uma matriz diagonal (uma matriz onde apenas as variâncias das variáveis são consideradas, e as covariâncias são ignoradas) com valores variando entre 0 e 1. Com valor 0, não é aplicada nenhuma regularização, portanto a matriz de covariância regularizada é igual à matriz de covariância empírica. Com valor 1, temos o máximo de regularização, ou seja, a matriz de covariância regularizada é igual à matriz de covariância diagonal. Valores intermediários de regularização a matriz de covariância regularizada é uma combinação ponderada entre a matriz de covariância empírica e a matriz de covariância diagonal.

Funcionamento do algoritmo RDA:

1. Calcula a média de cada classe;
2. Estima as matrizes de covariância de cada classe;
3. Aplica regularização para ajustar as matrizes de covariância;
4. Calcula a função discriminante regularizada;
5. Classifica novos pontos de dados com base na função discriminante que produz o maior valor.

Código 11 - Processamento do algoritmo de análise discriminante regularizada (RDA)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Carrega o conjunto de dados Iris
iris = load_iris()
X, y = iris.data, iris.target

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
state=42)

# Cria e treina o modelo RDA (usando LDA com shrinkage)
rda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
rda.fit(X_train, y_train)

# Realiza previsões e avalia a acurácia
accuracy = rda.score(X_test, y_test)
print(f"Acurácia do RDA: {accuracy:.2f}")
```

Se compararmos o código proposto ao apresentado no LDA notam-se pequenas diferenças. Não foi necessário a importação de um nova classe, sendo utilizada a mesma `LinearDiscriminantAnalysis` aplicada no LDA. A principal mudança está no uso de alguns argumentos no momento da criação do modelo `rda`.

Código 12 - Cria e treina o modelo RDA (usando LDA com shrinkage)

```
# Cria e treina o modelo RDA (usando LDA com shrinkage)
rda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
rda.fit(X_train, y_train)
```

- `rda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')`: cria o modelo `rda`, utilizando a regularização automática através do emprego do argumento `shrinkage='auto'`.

- `shrinkage` é o parâmetro de regularização, visto anteriormente. O valor `'auto'` encontrará o melhor valor de regularização, porém pode ser alterado manualmente para valores entre 0 e 1.

- `solver` é um parâmetro para escolher o algoritmo para encontrar a solução. Para o uso com o RDA é necessário ajustá-lo para `'lsqr'`, para utilizar o método dos mínimos quadrados; ou `'eigen'`, quando quer utilizar decomposição de autovalores. Somente com o uso desses 2 valores é calculada a matriz de covariância e o pode-se utilizar em conjunto o parâmetro `shrinkage`.

6.5 Notebook Colab

Nessa seção nós vamos utilizar o [Notebook Colab](#) para a implementação dos algoritmos LDA, QDA e RDA explicados anteriormente, bem como alguns exercícios.

Objetivos de Aprendizagem

- » Fazer uma análise exploratória inicial no conjunto de dados Iris
- » Entender o funcionamento da Análise Discriminante Linear (LDA)
- » Entender o funcionamento da Análise Discriminante Quadrática (QDA)
- » Entender o funcionamento da Análise Discriminante Regularizada (RDA)

Análise Exploratória do dataset Iris

O conjunto de dados **Iris** é extremamente reconhecido e utilizado em estatística e aprendizagem de máquina, especialmente como exemplo introdutório para a aplicação de técnicas de classificação. Este conjunto de dados consiste em 150 amostras de flores pertencentes à espécie Iris , distribuídas igualmente entre três subespécies: **Iris setosa**, **Iris versicolor** e **Iris virginica**.

Cada amostra no conjunto de dados é caracterizada por quatro variáveis numéricas que descrevem atributos morfológicos das flores:

- » **Comprimento da sépala** (em centímetros);
- » **Largura da sépala** (em centímetros);
- » **Comprimento da pétala** (em centímetros);
- » **Largura da pétala** (em centímetros).

Devido à sua simplicidade, equilíbrio e bem-definidas classes, o conjunto de dados Iris é frequentemente empregado para ilustrar algoritmos de classificação e técnicas de análise exploratória de dados.

```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Carrega o conjunto de dados Iris, incluindo um dataframe
# Essa função retorna um dicionário, com vários itens
iris = load_iris(as_frame=True)

# Guarda o item 'frame' do dicionario como df
df = iris.frame
```

continua

```
# Mostra os primeiros dados do dataset  
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
# Note que a coluna target está em função de valores numéricos  
# Para melhor entender, será adicionado uma coluna que converte esse número para a  
espécie da flor
```

```
# Acessando os nomes das espécies  
# Segundo a documentação do sklearn, os valores dos rótulos estão contidos no  
dicionário do load_iris()  
# Guarda o item 'target_names' do dicionario como df  
target_names = iris.target_names
```

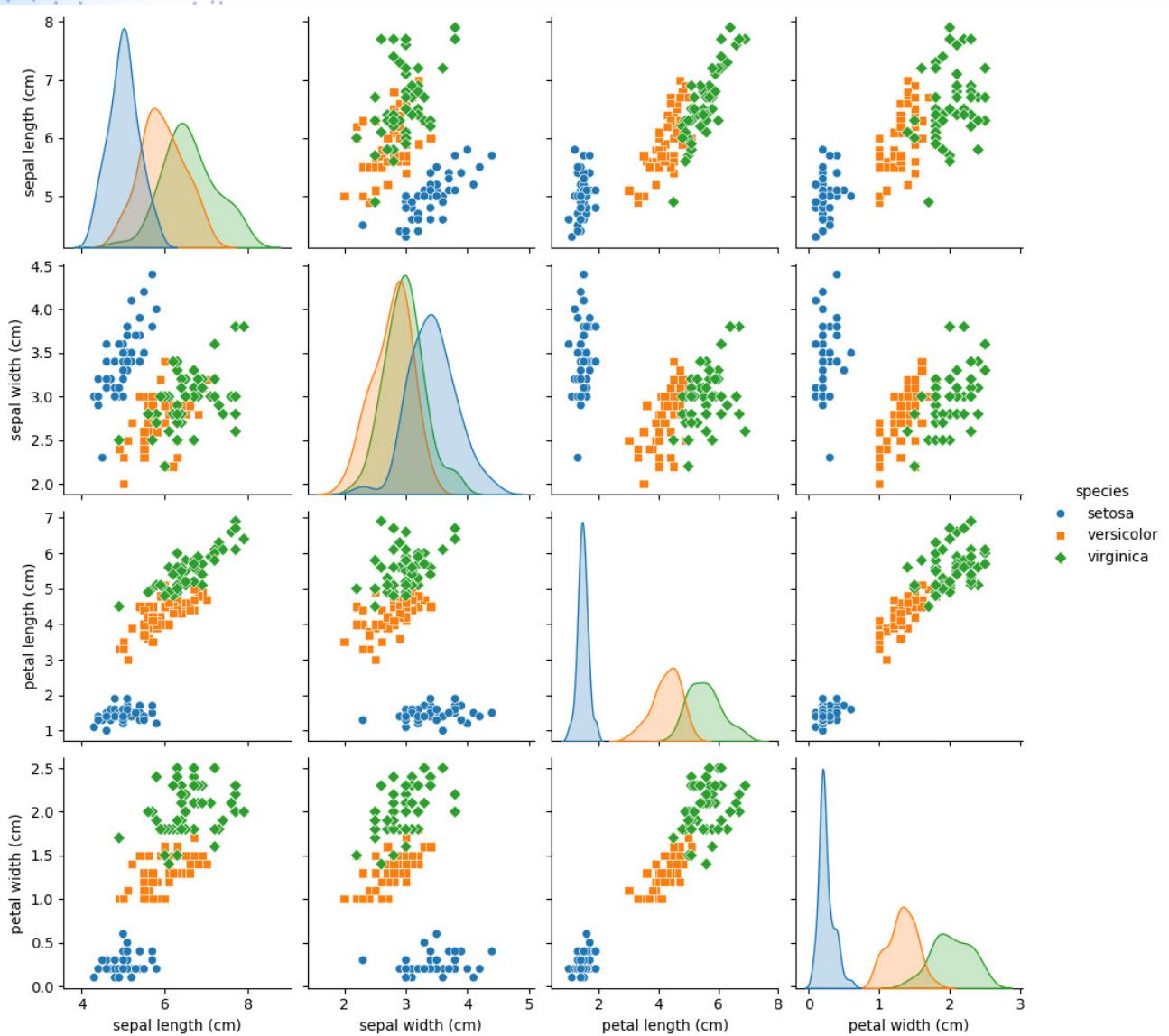
```
# Adicionando uma nova coluna com os nomes das espécies  
df['species'] = target_names[df['target']]
```

```
# Visualizando o DataFrame  
# Observe a última coluna com o nome das espécies  
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

```
# Apresenta um gráfico com os dados do dataset iris  
# Cada gráfico é composto de 16 minigráficos  
# Cada minigráfico possui uma característica no eixo x e outra no eixo y  
# Essa apresentação facilita o entendimento de todo o dataset e dá um panorama geral  
dos dados  
ax = sns.pairplot(df.drop(columns='target'), hue='species', markers=["o", "s", "D"])
```

Figura 27 - Gráfico com os dados do dataset iris



Fonte: autoria própria.

Análise Discriminante Linear (LDA)

Análise Discriminante Linear (LDA) é um poderoso método de classificação que opera projetando um conjunto de dados em um espaço de menor dimensionalidade com boa separabilidade de classes. É utilizada em estatística, reconhecimento de padrões e aprendizado de máquina para encontrar uma combinação linear de características que caracterizam ou separam duas ou mais classes de objetos ou eventos. A combinação resultante pode ser usada como um classificador linear ou, mais comumente, para redução de dimensionalidade antes da classificação posterior. A LDA é particularmente eficaz para problemas de classificação multiclasse.

Vantagens :

Algumas vantagens do LDA residem no fato de que é um método linear, ou que o torna computacionalmente eficiente e fácil de implementar, especialmente em grandes conjuntos de dados.

Ele é capaz de reduzir a dimensionalidade dos dados, projetando-os em um espaço de menor dimensão, onde as classes são mais separáveis. Isso é particularmente útil para visualizar e melhorar o desempenho de outros algoritmos de aprendizado.

Ao maximizar a separação entre as classes, o LDA é eficaz em situações onde as classes são linearmente separáveis.

O LDA funciona bem em cenários onde as classes são aproximadamente balanceadas e com distribuições gaussianas, oferecendo bom desempenho.

Objetivo :

Nesse exemplo o objetivo é criar 2 novas características que representem as 4 já existentes de forma que melhor separem as 3 espécies.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Separar em X os dados com as características
X = df.iloc[:, :-2].values
# Separar em y os dados com as espécies
y = df.iloc[:, -1].values

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Cria o modelo LDA para redução em 2 componentes
lda = LinearDiscriminantAnalysis(n_components=2)

# Treina com os valores de X_train e y_train
# obtém um novo X (X_train_lda) de dimensionalidade 2 com base no X_train
X_train_lda = lda.fit_transform(X_train, y_train)
# Com o modelo treinado co X_train e y_train
# obtém um novo X (X_test_lda) de dimensionalidade 2, com base no X_test
X_test_lda = lda.transform(X_test)
```

```
# Observe como o novo X (X_train_lda) agora possui apenas 2 características
print(f"Tamanho do X_train: {X_train.shape[1]}")
print(f"Tamanho do X_train_lda: {X_train_lda.shape[1]}")

# O mesmo ocorre com o conjunto X_test
print(f"Tamanho do X_test: {X_test.shape[1]}")
print(f"Tamanho do X_test_lda: {X_test_lda.shape[1]}")
```

```
→ Tamanho do X_train: 4
Tamanho do X_train_lda: 2
Tamanho do X_test: 4
Tamanho do X_test_lda: 2
```

O LDA criou essas duas características visando a máxima separação entre as classes, mas resta saber se foi efetivo, ou seja, será que com apenas essas duas classes é possível identificar as espécies?

Abaixo veremos se isso ocorre.

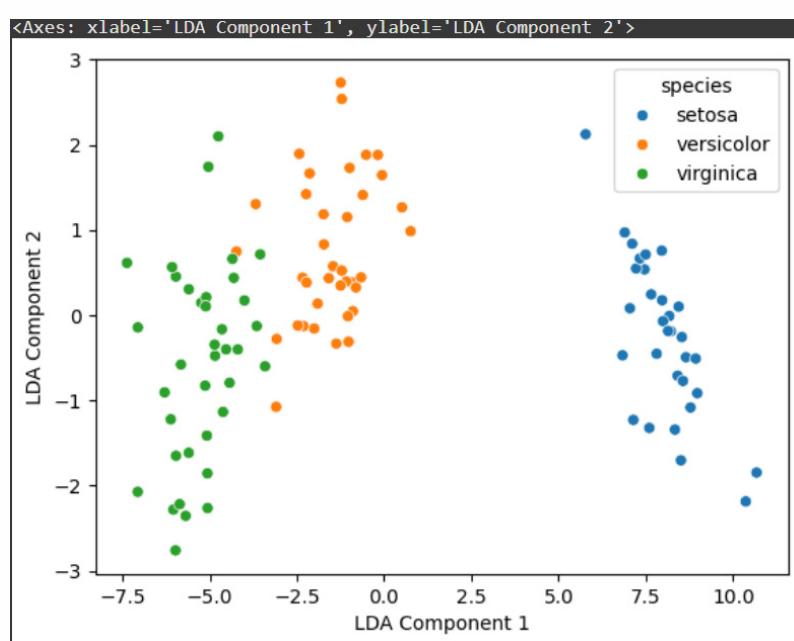
```
# Coloca os dados do X_train_lda e y_train em um dataframe para melhor visualização
df_lda_train = pd.DataFrame(X_train_lda,columns=['LDA Component 1','LDA Component 2'])
df_lda_train['species']=y_train

df_lda_train.head()
```

→	LDA Component 1	LDA Component 2	species
0	-0.514612	1.882729	versicolor
1	-3.549638	0.715341	virginica
2	-4.527751	-0.400175	virginica
3	-1.036480	-0.011181	versicolor
4	-5.975968	-2.759810	virginica

```
# Plota os valores do X_train_lda
# o sort_values é apenas para que as espécies apareçam em uma determinada ordem
sns.scatterplot(df_lda_train.sort_values(by='species'), x='LDA Component 1', y='LDA Component 2', hue='species')
```

Figura 28 - Gráfico dos valores do X_train_lda



Fonte: autoria própria.

```

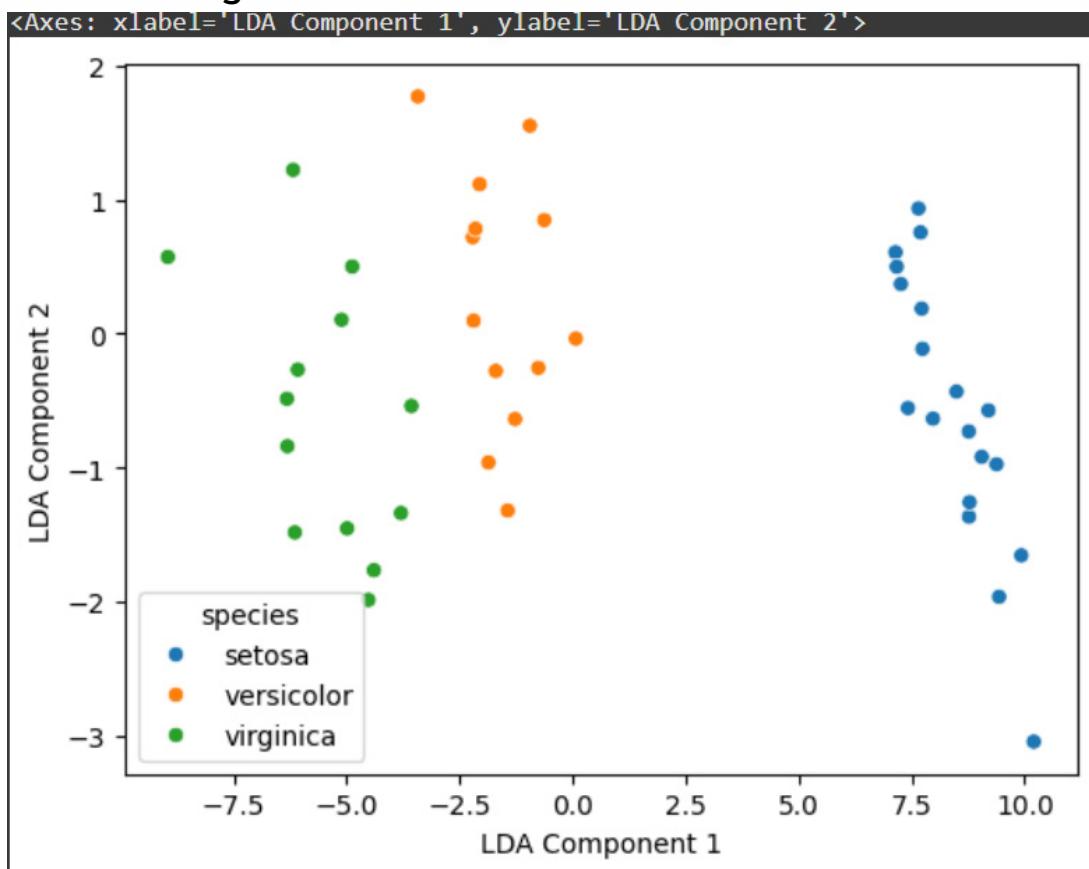
# Repetindo para o X_test

# Coloca os dados do X_test_lda e y_test em um dataframe para melhor visualização
df_lda_test = pd.DataFrame(X_test_lda, columns=['LDA Component 1','LDA Component 2'])
df_lda_test['species']=y_test

# Plota os valores do X_train_lda
# o sort_values é apenas para que as spécies apareçam em uma determinada ordem
sns.scatterplot(df_lda_test.sort_values(by='species'), x='LDA Component 1', y='LDA Component 2', hue='species')

```

Figura 29 - Gráfico dos valores do X_train_lda



Fonte: autoria própria.

Análise dos gráficos

Compare ambos os gráficos (Treino e Test).

Note como é praticamente possível separar as 3 espécies apenas com linhas verticais a partir do LDA Component 1. Isso é mais visível na classe setosa, porém a fronteira entre as classes virginica e versicolor está bem definida, apesar de, no caso do uso de uma regressão linear, alguns poucos pontos do conjunto de treino podem ser classificados equivocadamente. Já há caso do teste há clara divisão.

Os gráficos apresentados são o resultado da LDA, no entanto não é possível dizer se ele foi efetivo em classificação corretamente. É necessário comparar com o "gabarito".

```
# Obtem as previsões dos dados com base nos modelos treinados
y_pred_lda_train = lda.predict(X_train)
y_pred_lda_test = lda.predict(X_test)

# Adicionar aos dataframes
df_lda_train[‘species_predicted’] = y_pred_lda_train
df_lda_test[‘species_predicted’] = y_pred_lda_test

df_lda_train.head()
```

	LDA Component 1	LDA Component 2	species	species_predicted
0	-0.514612	1.882729	versicolor	versicolor
1	-3.549638	0.715341	virginica	versicolor
2	-4.527751	-0.400175	virginica	virginica
3	-1.036480	-0.011181	versicolor	versicolor
4	-5.975968	-2.759810	virginica	virginica

Notem como o LDA fez um excelente trabalho reduzindo a dimensionalidade do dataset e ainda mantendo um excelente performance (elevada acurácia).

- » Acertou 97% dos dados de TREINO
- » Acertou 100% dos dados de TESTE

```
# Calculando a acurácia do modelo, comparando o gabarito (y_train) com os valores do
# modelo y_pred_lda_train
lda_train_acc = accuracy_score(y_train, y_pred_lda_train)
print(f"Acurácia do LDA no dataset de Treino: {lda_train_acc}")
```

```
# Calculando a acurácia do modelo, comparando o gabarito (y_test) com os valores do
# modelo y_pred_lda_test
lda_test_acc = accuracy_score(y_test, y_pred_lda_test)
print(f"Acurácia do LDA no dataset de Teste: {lda_test_acc}")
```

→ Acurácia do LDA no dataset de Treino: 0.9714285714285714
Acurácia do LDA no dataset de Teste: 1.0

```
# Identificando quais foram os casos de erros
df_lda_train_erros = df_lda_train.query("species != species_predicted")

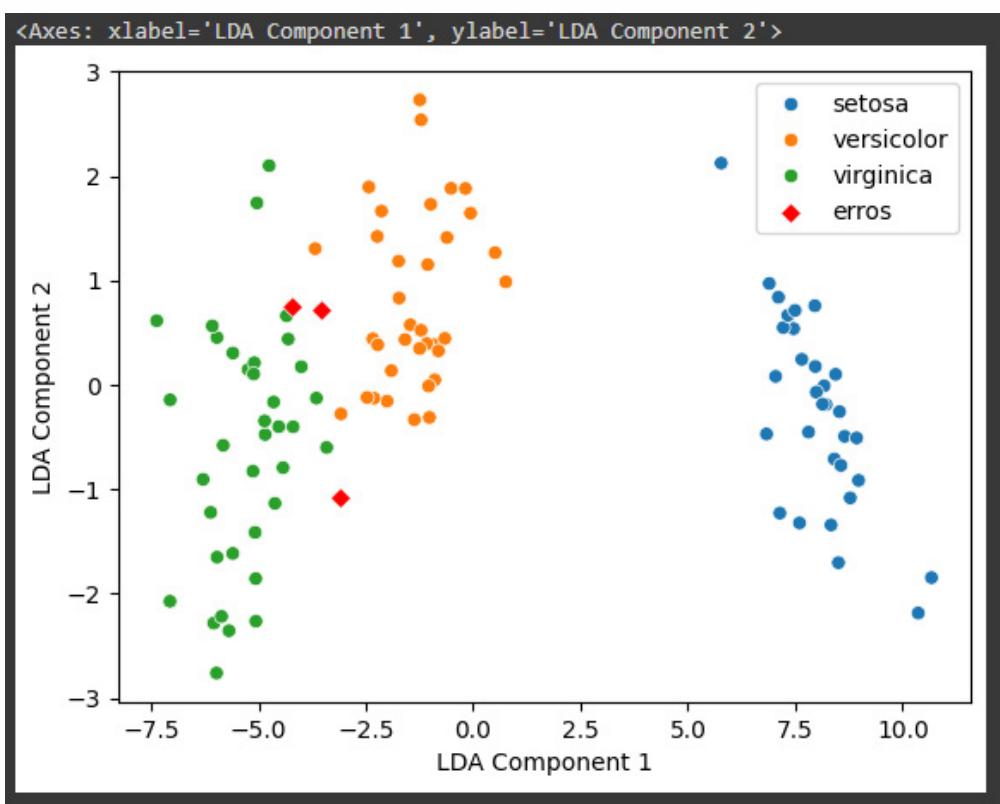
# Foram apenas 3 erros e estão listados abaixo
df_lda_train_erros.head()
```

	LDA Component 1	LDA Component 2	species	species_predicted
1	-3.549638	0.715341	virginica	versicolor
44	-3.094060	-1.075211	versicolor	virginica
47	-4.228972	0.748766	versicolor	virginica

Observando o gráfico abaixo, fica claro que os erros ocorreram mesmo na fronteira entre virginica e versicolor.

```
# Gráficos mostrando onde foram os erros
sns.scatterplot(df_lda_train.sort_values(by='species'), x='LDA Component 1', y='LDA Component 2', hue='species')
sns.scatterplot(df_lda_train_erros.sort_values(by='species'), x='LDA Component 1', y='LDA Component 2', color='red', marker='D', label='erros')
```

Figura 30 - Gráfico dos valores do X_train_lda



Análise Discriminante Quadrática (QDA)

A Análise Discriminante Quadrática (QDA) é um algoritmo de classificação que, assim como o LDA, busca separar classes de dados, mas difere na suposição de que as variâncias das classes não são iguais. Enquanto o LDA assume que todas as classes compartilham a mesma matriz de covariância e, portanto, traça limiares de decisão lineares, o QDA permite que cada classe tenha sua própria matriz de covariância, resultando em limites de decisão quadráticos. Essa flexibilidade torna o QDA mais adequado para problemas onde as fronteiras de decisão entre as classes são não-lineares e as classes têm distribuições com diferentes variâncias.

Vantagens :

As principais vantagens do QDA incluem a sua capacidade de modelar limites de decisão mais complexos, o que é especialmente útil em situações onde os dados não seguem distribuições gaussianas idênticas entre as classes. O QDA pode capturar padrões mais complexos, o que pode levar a um melhor desempenho em classificações onde as classes são não-linearmente separáveis. Além disso, o QDA é mais flexível do que o LDA, o que permite uma adaptação mais precisa aos dados, embora essa maior flexibilidade também possa aumentar o risco de overfitting, especialmente em conjuntos de dados pequenos ou com muitas dimensões.

Objetivo :

Nesse exemplo o objetivo é identificar corretamente através das 4 características as 3 espécies.

Nesse exemplo será utilizado o mesmo dataset Iris importado anteriormente.

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Separar em X os dados com as características
X = df.iloc[:, :-2].values
# Separar em y os dados com as espécies
y = df.iloc[:, -1].values

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Cria e treina o modelo QDA
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)

# Obtem as predições dos dados com base nos modelos treinados
y_pred_qda_train = qda.predict(X_train)
y_pred_qda_test = qda.predict(X_test)
```

Exercícios :

Exercício 1

Utilizando os dados gerados anteriormente, verifique a acurácia do QDA nos conjuntos de Treino e Validação. Quais são esses valores?

Resposta: os valores são 98% no Treino e 100% no teste. O código abaixo traz a solução:

```
# Calculando a acurácia do modelo, comparando o gabarito (y_train) com os valores do
# modelo y_pred_qda_train
qda_train_acc = accuracy_score(y_train, y_pred_qda_train)
print(f"Acurácia do QDA no dataset de Treino: {qda_train_acc}")

# Calculando a acurácia do modelo, comparando o gabarito (y_test) com os valores do
# modelo y_pred_qda_test
qda_test_acc = accuracy_score(y_test, y_pred_qda_test)
print(f"Acurácia do QDA no dataset de Teste: {qda_test_acc}")
```

→ Acurácia do QDA no dataset de Treino: 0.9809523809523809
Acurácia do QDA no dataset de Teste: 1.0

Exercício 2

Com base no `df_qda_train` e nos dados criados nessa seção, liste os index do `df_qda_train` em que o QDA errou na classificação.

Resposta: Os index são 1 e 47. Os códigos abaixo esclarecem esse valor:

```
# Para verificarmos onde foi o erro, é criado um dataframe para melhor visualização
# df.iloc[:, :-2].columns utiliza os rótulos do df eliminando as últimas 2 colunas
df_qda_train = pd.DataFrame(X_train, columns=df.iloc[:, :-2].columns)
df_qda_train['species']=y_train

# Adicionar aos dataframes
df_qda_train['species_predicted'] = y_pred_qda_train

df_qda_train.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_predicted
0	5.5	2.4	3.7	1.0	versicolor	versicolor
1	6.3	2.8	5.1	1.5	virginica	versicolor
2	6.4	3.1	5.5	1.8	virginica	virginica
3	6.6	3.0	4.4	1.4	versicolor	versicolor
4	7.2	3.6	6.1	2.5	virginica	virginica

continua

```
# Identificando quais foram os casos de erros
df_qda_train_erros = df_qda_train.query("species != species_predicted")

# Foram apenas 2 erros e estão listados abaixo
# Note pelo número do index, que eles também tinham sido classificados erroneamente no
LDA
df_qda_train_erros.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_predicted
1	6.3	2.8	5.1	1.5	virginica	versicolor
47	6.0	2.7	5.1	1.6	versicolor	virginica

Análise Discriminante Regularizada (RDA)

A Análise Discriminante Regularizada (RDA) é uma extensão tanto do Linear Discriminant Analysis (LDA) quanto do Quadratic Discriminant Analysis (QDA). Enquanto o LDA assume que todas as classes têm a mesma matriz de covariância e o QDA permite diferentes matrizes de covariância para cada classe, o RDA introduz um parâmetro de regularização que ajusta o grau de suavização entre essas duas abordagens. Essencialmente, o RDA permite um trade-off controlável entre a simplicidade do LDA e a flexibilidade do QDA, tornando-o adequado para situações em que se deseja evitar o overfitting (comum no QDA) ou underfitting (potencial no LDA).

Vantagens:

As principais vantagens do RDA incluem sua flexibilidade e capacidade de adaptação. O uso do parâmetro de regularização permite que o RDA se ajuste melhor aos dados, especialmente em cenários onde as suposições estritas do LDA (matrizes de covariância iguais) são demasiado simplistas, e o QDA, com sua abordagem mais flexível, pode ser propenso ao overfitting. Essa regularização permite que o RDA ofereça um desempenho mais robusto, equilibrando a capacidade de generalização e a modelagem precisa dos dados. Isso é particularmente útil em contextos com conjuntos de dados limitados ou com muitas variáveis, onde escolher o modelo correto entre LDA e QDA pode ser desafiador.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Separar em X os dados com as características
X = df.iloc[:, :-2].values
# Separar em y os dados com as espécies
y = df.iloc[:, -1].values

# Divide os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Cria e treina o modelo RDA (usando LDA com shrinkage)
rda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
```

continua

```

rda.fit(X_train, y_train)

# Obtem as predições dos dados com base nos modelos treinados
y_pred_rda_train = rda.predict(X_train)
y_pred_rda_test = rda.predict(X_test)

```

```

# Calculando a acurácia do modelo, comparando o gabarito (y_train) com os valores do
# modelo y_pred_lda_train
rda_train_acc = accuracy_score(y_train, y_pred_rda_train)
print(f"Acurácia do RDA no dataset de Treino: {rda_train_acc}")

# Calculando a acurácia do modelo, comparando o gabarito (y_test) com os valores do
# modelo y_pred_lda_test
rda_test_acc = accuracy_score(y_test, y_pred_rda_test)
print(f"Acurácia do RDA no dataset de Teste: {rda_test_acc}")

```

→ Acurácia do RDA no dataset de Treino: 0.9714285714285714
Acurácia do RDA no dataset de Teste: 1.0

```

# Para verificarmos onde foi o erro, é criado um dataframe para melhor visualização
# df.iloc[:, :-2].columns utiliza os rótulos do df eliminando as últimas 2 colunas
df_rda_train = pd.DataFrame(X_train, columns=df.iloc[:, :-2].columns)
df_rda_train['species']=y_train

```

```

# Adicionar aos dataframes
df_rda_train['species_predicted'] = y_pred_rda_train

```

```
df_rda_train.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_predicted
0	5.5	2.4	3.7	1.0	versicolor	versicolor
1	6.3	2.8	5.1	1.5	virginica	versicolor
2	6.4	3.1	5.5	1.8	virginica	virginica
3	6.6	3.0	4.4	1.4	versicolor	versicolor
4	7.2	3.6	6.1	2.5	virginica	virginica

```

# Identificando quais foram os casos de erros
df_rda_train_erros = df_rda_train.query("species != species_predicted")

```

```

# Foram apenas 3 erros e estão listados abaixo
# Note pelo número do index, que eles também tinham sido classificados erroneamente no
# LDA
df_rda_train_erros.head()

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_predicted
1	6.3	2.8	5.1	1.5	virginica	versicolor
44	5.9	3.2	4.8	1.8	versicolor	virginica
47	6.0	2.7	5.1	1.6	versicolor	virginica



SAIBA MAIS...

- » LDA:
 - » Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179-188.
 - » <https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/>
 - » <https://www.youtube.com/watch?v=azXCzI57Yfc>
 - » <https://www.youtube.com/watch?v=julEqA2ozcA>
- » QDA: Smith, C. A. B. (1947). Some examples of discrimination. *Annals of Eugenics*, 13(1), 272-282.
- » RDA: Friedman, J. H. (1989). Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405), 165-175.

PARA RELEMBRAR...

- » Cada algoritmo possui suas próprias características e é mais adequado para determinados tipos de problemas. A escolha do algoritmo depende das características dos dados, do objetivo da tarefa de classificação e dos requisitos de desempenho e interpretabilidade do modelo.

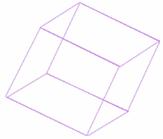
Tabela 3 - Resumo das vantagens e desvantagens dos algoritmos que usam funções discriminantes

ALGORITMO	VANTAGENS	DESVANTAGENS
LDA	<ul style="list-style-type: none"> • Simples e rápido • Funciona bem com dados lineares separáveis • Robusto a ruído 	<ul style="list-style-type: none"> • Assume distribuição normal • Não funciona bem com dados não lineares • Sensível a outliers
QDA	<ul style="list-style-type: none"> • Pode modelar fronteiras de decisão não lineares • Não assume igualdade das matrizes de covariância 	<ul style="list-style-type: none"> • Requer mais dados para estimar as matrizes de covariância • Mais suscetível a overfitting
RDA	<ul style="list-style-type: none"> • Lida bem com alta dimensionalidade • Mais robusto a overfitting 	<ul style="list-style-type: none"> • Requer escolha do parâmetro de regularização • Pode perder informações importantes com regularização excessiva

Fonte: autoria própria.

Unidade VII
**Conjuntos
de dados**





Unidade VII - Conjuntos de dados

7.1 A Importância dos Dados para a Construção de Modelos de ML

No ML, a qualidade dos dados é fundamental para o sucesso dos modelos preditivos. Dados robustos e bem preparados são a base para que os algoritmos possam extrair padrões relevantes e generalizar de maneira eficaz para novos casos. Um conjunto de dados rico e diversificado permite que os modelos sejam mais precisos e confiáveis, enquanto dados incompletos ou tendenciosos podem levar a resultados distorcidos e ineficazes. Portanto, a coleta, o processamento e a análise crítica dos dados são etapas cruciais na construção de sistemas inteligentes.

No contexto do ML, os dados são frequentemente referidos como a “matéria prima” que alimenta os modelos. A qualidade e a quantidade dos dados utilizados são fatores determinantes para o desempenho e a eficácia dos algoritmos. Para que um modelo de ML seja capaz de identificar padrões complexos e realizar previsões precisas, é essencial que os dados estejam devidamente coletados, limpos e preparados.

A qualidade dos dados se refere à precisão, completude, consistência e relevância das informações coletadas. Dados de alta qualidade possuem as seguintes características:

- » **Precisão:** Os dados devem refletir a realidade de forma exata, sem erros ou distorções.
- » **Completude:** Um conjunto de dados completo sobre todas as variáveis relevantes para o problema que se deseja resolver, sem lacunas significativas.
- » **Consistência:** Dados consistentes mantêm a coerência entre diferentes fontes e ao longo do tempo, sem contradições.
- » **Relevância:** Apenas os dados que são diretamente aplicáveis ao problema em questão devem ser utilizados, evitando o “ruído” de informações irrelevantes.

Quando a qualidade dos dados é alta, os modelos de ML são capazes de aprender de forma mais eficiente, resultando em previsões e classificações mais precisas.

Além da qualidade, a diversidade dos dados desempenha um papel crucial na robustez dos modelos. Dados diversos são aqueles que capturam uma ampla gama de variabilidades possíveis dentro do conjunto de informações. Isso inclui variações em termos de demografia, condições operacionais, comportamentos e cenários. Um modelo treinado em um conjunto de dados diversificado é mais provável de generalizar bem para novos casos que não foram observados durante o treinamento, reduzindo o risco de sobreajuste. Modelos treinados em dados que representam diversas populações e

situações têm maior probabilidade de serem justos e precisos para todos os grupos, evitando vieses indesejados.

Por outro lado, dados incompletos ou tendenciosos podem comprometer significativamente a performance dos modelos de ML. Dados incompletos, com valores ausentes ou insuficientes para determinadas variáveis, podem levar a falhas na aprendizagem do modelo. Da mesma forma, dados tendenciosos, que não representam adequadamente a realidade ou que refletem vieses implícitos, podem resultar em modelos que perpetuam essas distorções.

Por exemplo, se um conjunto de dados para um modelo de classificação de crédito é composto majoritariamente por exemplos de uma única classe social ou região geográfica, o modelo pode aprender a associar características irrelevantes com o resultado desejado, prejudicando sua capacidade de generalização.

Dada a importância dos dados no ML, o processamento e a análise crítica desses dados são etapas que demandam atenção e rigor. O processamento envolve tarefas como limpeza dos dados, normalização, tratamento de valores ausentes e redução de dimensionalidade, todas elas voltadas a melhorar a qualidade dos dados antes de serem utilizados para treinar um modelo.

Além disso, a análise crítica dos dados requer uma compreensão profunda do domínio em questão para identificar possíveis vieses, *outliers* e inconsistências. Esta análise permite a correção de problemas antes que os dados sejam usados na construção do modelo, garantindo que o modelo seja treinado em dados que verdadeiramente representem o problema que se deseja resolver.

7.2 Dados Estruturados vs. Dados Não Estruturados

Dados estruturados são aqueles organizados em uma estrutura predefinida, como em tabelas com linhas e colunas, facilitando a análise por meio de ferramentas de processamento de dados tradicionais. Já os dados não estruturados abrangem informações que não seguem um padrão de organização, como textos, imagens e vídeos, exigindo técnicas mais avançadas para serem analisados, como PLN e visão computacional. A capacidade de trabalhar com ambos os tipos de dados é essencial para aproveitar ao máximo o potencial da aprendizagem de máquina.

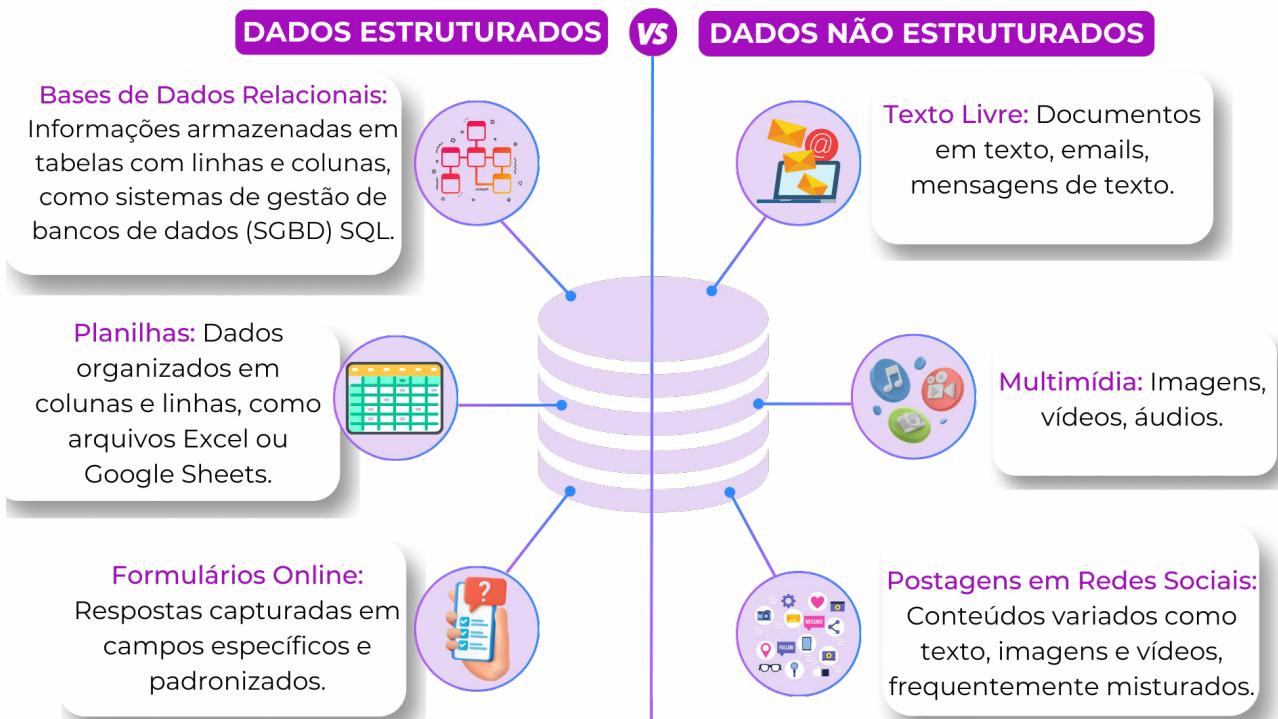
Os **dados estruturados** são organizados em um formato específico, como tabelas, facilitando a sua análise e processamento. Exemplos incluem:

- » **Bases de Dados Relacionais:** Informações armazenadas em tabelas com linhas e colunas, como Sistemas de Gestão de Bancos de Dados (SGBD).
- » **Planilhas:** Dados organizados em colunas e linhas, como arquivos Excel ou Google Sheets.
- » **Formulários Online:** Respostas capturadas em campos específicos e padronizados.

Os dados não estruturados não possuem um formato específico e são mais difíceis de organizar e analisar. Exemplos incluem:

- » **Texto Livre:** Documentos em texto, e-mails, mensagens de texto.
- » **Multimídia:** Imagens, vídeos, áudios.
- » **Postagens em Redes Sociais:** Conteúdos variados como texto, imagens e vídeos, frequentemente misturados.

Figura 31 - Exemplo de dados estruturados e não estruturados



Fonte: Adaptada de [Alam, 2024](#).

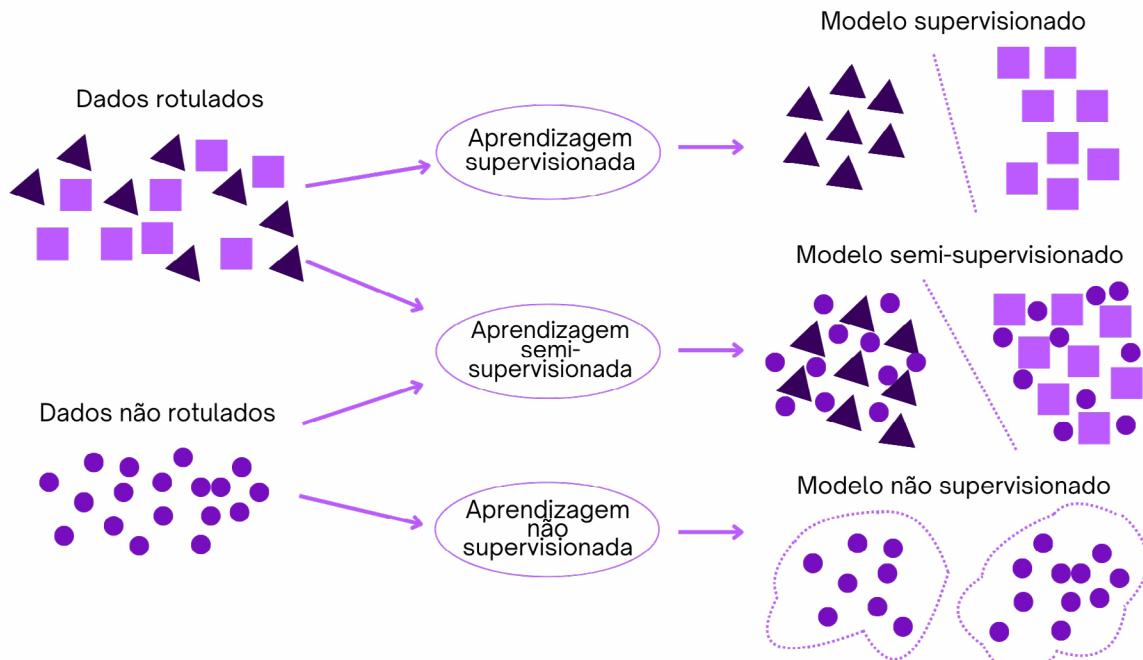
7.3 Conjuntos de Dados Supervisionados, Não Supervisionados e Semi-supervisionados

Nos conjuntos de dados supervisionados, cada exemplo de treinamento inclui uma entrada e uma saída desejada, permitindo ao modelo aprender a relação entre elas. Em contraste, os conjuntos não supervisionados contêm apenas entradas, cabendo ao algoritmo descobrir padrões sem uma resposta direta para guiar o aprendizado. Já os conjuntos semi-supervisionados combinam uma pequena quantidade de dados rotulados com uma grande quantidade de dados não rotulados, aproveitando o melhor de ambas as abordagens para melhorar a precisão dos modelos.

Na figura abaixo (Figura 28) é ilustrado o mesmo conjunto de dados apresentado de duas maneiras: rotulados (em triângulos e quadrados) e não rotulados (em círculos). Os métodos supervisionados necessitam que todos os dados possuam rótulos, como em Modelo Supervisionado da Figura 28. Já os métodos não supervisionados não necessitam de rótulo e tentarão identificar apenas pelas características, como apresentado da

porção Modelo Não Supervisionado. Por último, quando há parte dos dados com rótulo e parte sem rótulos, os métodos semi-supervisionados podem ser a melhor opção, como ilustrado na porção Modelo Semi-supervisionado da figura.

Figura 32 - Exemplo de tipos de dados para diferentes modelos



Fonte: adaptada de [Trambaiolli, Biazoli, Sato \(2022\)](#).

É possível encontrar publicamente uma série de conjuntos de dados, sejam eles supervisionados, semi-supervisionados ou não supervisionados para uso em conjunto com modelos de classificação, regressão, clusterização, etc. Alguns conjuntos de dados se tornaram icônicos no campo da aprendizagem de máquina pela sua relevância e frequente uso em estudos e aplicações. Conjuntos de dados padronizados são fundamentais para o desenvolvimento e a avaliação de algoritmos de aprendizado de máquina, pois permitem comparar o desempenho de diferentes abordagens em condições controladas. Dentre os conjuntos de dados mais amplamente utilizados estão o **Iris**, o **MNIST** e o **CIFAR-10**.

O conjunto **Iris**, por exemplo, é composto por 150 amostras de flores pertencentes à espécie *Iris*, distribuídas igualmente entre três subespécies. Cada amostra é caracterizada por quatro atributos numéricos que descrevem propriedades morfológicas das flores, e é utilizado principalmente em tarefas de classificação para prever a subespécie com base nessas características.

O **MNIST** é um extenso conjunto de 70,000 imagens de dígitos manuscritos, cada uma rotulada com um valor correspondente entre 0 e 9. Este conjunto é amplamente empregado em tarefas de reconhecimento de caracteres e serve como referência para avaliar o desempenho de algoritmos de classificação de imagens.

Por fim, o **CIFAR-10** consiste em 60,000 imagens coloridas de 32x32 pixels, distribuídas igualmente entre 10 categorias de objetos, como aviões, automóveis e animais. Este

conjunto de dados é utilizado para testar e comparar algoritmos de classificação de imagens, especialmente em tarefas que envolvem reconhecimento de objetos em imagens coloridas.

A Tabela abaixo a seguir resume as principais características dos conjuntos de dados mencionados, fornecendo uma visão geral sobre seu tamanho, quantidade de dados rotulados e aplicação principal.

Tabela 4 - Resumo dos conjuntos de dados Iris, MNIST e CIFAR-10

CONJUNTO DE DADOS	ATRIBUTOS	DESCRIÇÃO	LINK
Iris	<ul style="list-style-type: none"> Tipo de Dados: Atributos de flores (numéricos) Tamanho da Base: 150 amostras Quantidade de Dados Rotulados: 150 Quantidade de Rótulos: 3 	Conjunto de dados contendo medidas de flores Iris, utilizado para classificação de espécies com base em características morfológicas.	Iris
MNIST	<ul style="list-style-type: none"> Tipo de Dados: Imagens de dígitos manuscritos Tamanho da Base: 70,000 imagens Quantidade de Dados Rotulados: 70,000 Quantidade de Rótulos: 10 	Conjunto de imagens de dígitos manuscritos (0-9) utilizado para tarefas de reconhecimento de caracteres.	MNIST
CIFAR-10	<ul style="list-style-type: none"> Tipo de Dados: Imagens coloridas de objetos Tamanho da Base: 60,000 imagens Quantidade de Dados Rotulados: 60,000 Quantidade de Rótulos: 10 	Conjunto de imagens coloridas 32x32 pixels, categorizadas em 10 classes, usado para tarefas de classificação de objetos.	CIFAR-10

Fonte: Autoria própria

7.4 Limpeza de Dados (Tratamento de Valores Ausentes, *Outliers*, Erros de Digitação)

A limpeza de dados é uma etapa essencial no processamento e análise de dados, visando a correção e aprimoramento da qualidade dos dados para garantir a eficácia das análises e a confiabilidade dos resultados. Abaixo, são abordados os principais aspectos da importância da limpeza de dados, bem como os conceitos fundamentais envolvidos nesse processo:

- » **Confiabilidade:** Dados limpos e bem preparados são cruciais para garantir a confiabilidade das análises. Quando os dados estão isentos de erros e inconsistências, é possível obter percepções mais precisas e confiáveis, que refletem com maior exatidão a realidade que se pretende analisar. A presença de valores ausentes, duplicados ou errôneos pode comprometer a validade dos resultados e a tomada de decisões.
- » **Qualidade:** A qualidade dos dados é um fator determinante para a obtenção de percepções precisas. Dados bem limpos e estruturados fornecem uma base sólida para a realização de análises avançadas e a construção de modelos preditivos. A qualidade dos dados afeta diretamente a capacidade do modelo de aprender padrões significativos e generalizar para novos dados.
- » **Eficiência:** Dados bem estruturados e limpos facilitam a análise, economizando tempo e recursos. Ao resolver problemas como valores ausentes, duplicações e *outliers* de forma sistemática, a análise pode ser conduzida de maneira mais eficiente, permitindo que os analistas e cientistas de dados concentrem seus esforços na interpretação dos resultados em vez de corrigir problemas de dados.
- » **Valores Ausentes:** Valores ausentes referem-se a dados faltantes ou nulos em um conjunto de dados. Esses valores podem surgir por diversos motivos, como falhas na coleta de dados ou erros de entrada. A abordagem para tratar valores ausentes inclui a imputação, onde valores ausentes são substituídos por estimativas baseadas em outros dados, ou a interpolação, que utiliza os dados existentes para prever os valores faltantes.
- » **Valores Duplicados:** Valores duplicados são registros idênticos dentro do conjunto de dados que podem distorcer a análise e levar a resultados enviesados. A identificação e remoção de duplicatas são importantes para garantir que cada entrada seja considerada apenas uma vez, evitando redundâncias que podem afetar a precisão dos resultados analíticos.
- » **Outliers:** São valores que se desviam significativamente da maioria dos dados. Esses valores extremos podem influenciar desproporcionalmente os resultados das análises e dos modelos estatísticos. A detecção e tratamento de *outliers* são essenciais para evitar que eles distorçam a interpretação dos dados e a construção de modelos preditivos.

- » **Formatação:** A uniformidade na formatação dos dados refere-se à consistência nos formatos e na estrutura dos dados. Dados com formatos inconsistentes, como diferentes padrões de data ou variações na representação de números, podem causar erros e dificultar a análise. A padronização da formatação garante que todos os dados estejam no mesmo formato, facilitando a análise e a integração dos dados.

A limpeza de dados é uma etapa fundamental que assegura a precisão e a confiabilidade das análises. Ao tratar valores ausentes, duplicados e *outliers* e garantir a uniformidade na formatação, é possível obter dados de alta qualidade que são essenciais para a construção de modelos eficazes e a realização de análises precisas.

7.5 Pré-processamento (Normalização, Codificação de Variáveis Categóricas)

O pré-processamento de dados é uma etapa crucial que prepara os dados para o treinamento do modelo. A normalização, como visto na unidade 5, ajusta a escala dos dados para garantir que todas as características contribuam igualmente para o aprendizado. Já a codificação de variáveis categóricas transforma essas variáveis em um formato numérico que os algoritmos podem processar, como a codificação *one-hot* ou a utilização de variáveis *dummy*.

A normalização ajusta a escala dos dados para garantir que todas as características tenham igual importância no processo de aprendizado. As duas técnicas comuns de normalização são: *Min-Max Scaling* e *Z-score Standardization*.

O *Min-Max Scaling*, também conhecido como normalização de intervalo, transforma os dados para um intervalo específico, geralmente $[0, 1]$. A fórmula utilizada é:

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

Onde (X) é o valor original da característica, (X_{\min}) é o valor mínimo da característica, e (X_{\max}) é o valor máximo. Essa técnica é particularmente útil quando é necessário que os dados estejam em um intervalo fixo. No entanto, pode ser sensível a *outliers*, pois os valores extremos podem distorcer o intervalo de normalização.

A padronização pelo *Z-score*, também conhecida como normalização padrão, transforma os dados para que tenham média zero e desvio padrão um. A fórmula é:

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

Onde (X) é o valor original, (μ) é a média dos valores da característica, e (σ) é o desvio padrão. Este método é útil para dados que não possuem uma distribuição uniforme ou quando se deseja que as características tenham a mesma escala, independentemente das unidades originais. É menos sensível a *outliers* em comparação ao *Min-Max Scaling*.

Algoritmos de aprendizado de máquina geralmente exigem que os dados estejam em formato numérico. Portanto, variáveis categóricas, que representam categorias ou

rótulos, precisam ser convertidas em um formato que os algoritmos possam processar. As técnicas comuns para codificação de variáveis categóricas incluem:

- » **One-Hot Encoding** que transforma uma variável categórica em uma série de colunas binárias, onde cada coluna representa uma categoria específica. Se uma variável categórica tem (k) categorias, será criada uma nova coluna para cada categoria, preenchida com 0s e 1s indicando a presença ou ausência daquela categoria. Esta técnica é adequada para variáveis categóricas sem ordem intrínseca e evita a introdução de uma ordem implícita. No entanto, pode aumentar significativamente a dimensionalidade dos dados, especialmente se o número de categorias for grande.
- » **Label Encoding** que atribui um valor numérico a cada categoria de uma variável categórica. Cada categoria é substituída por um número inteiro único. *Label Encoding* é útil para variáveis categóricas ordinais, onde a ordem das categorias é importante. No entanto, pode introduzir uma ordem artificial para variáveis categóricas nominais, o que pode ser inadequado para alguns algoritmos de aprendizado de máquina.

Na figura abaixo é possível compreender melhor como funciona cada uma das técnicas apresentadas. *Label Encoding* simplesmente atribui um número único a cada categoria, como 1 para “Apple”, 2 para “Chicken” e 3 para “Broccoli”. *One-Hot Encoding*, por outro lado, cria uma nova coluna para cada categoria e atribui 1 se a observação pertence àquela categoria e 0 caso contrário.

Figura 33 - Exemplo de técnicas para criação de variáveis categóricas

Label Encoding			One Hot Encoding			
Nome do alimento	Categórico #	Calorias	Maça	Frango	Brócolis	Calorias
Maça	1	95	1	0	0	95
Frango	2	231	0	1	0	231
Brócolis	3	50	0	0	1	50

Fonte: Adaptada de [Medium, 2018](#).

O pré-processamento de dados é fundamental para garantir que as características estejam na mesma escala e em um formato que os algoritmos de aprendizado de máquina possam processar efetivamente. A normalização, por meio do *Min-Max Scaling* e da *Z-score Standardization*, ajusta a escala dos dados para melhorar a convergência e o desempenho do modelo. A codificação de variáveis categóricas, utilizando técnicas como *One-Hot Encoding* e *Label Encoding*, converte categorias em formatos numéricos, preparando os dados para análise e modelagem. Estas práticas são essenciais para a construção de modelos precisos e robustos em aprendizado de máquina.

7.6 Seleção e Engenharia de Características (Feature Selection e Feature Engineering)

A seleção de características envolve escolher as variáveis mais relevantes para o modelo, o que pode aumentar sua eficácia e reduzir a complexidade. Por outro lado, a engenharia de características é a arte de criar novas variáveis a partir das existentes para capturar relações mais complexas entre os dados, que possam influenciar a predição e melhorar a capacidade preditiva do modelo. Ambos os processos são essenciais para otimizar o desempenho do modelo.

A seleção de características visa identificar e escolher o subconjunto mais relevante de atributos que melhor explicam a variabilidade da variável alvo. Ao reduzir a dimensionalidade dos dados, a seleção de características contribui para:

- » **Melhora na performance:** A remoção de características irrelevantes ou redundantes pode reduzir o *overfitting* (sobreajuste) e aumentar a generalização do modelo.
- » **Redução da complexidade:** Modelos com menos características são mais simples de treinar, interpretar e manter.
- » **Diminuição do tempo de treinamento:** Menos características implicam em menor tempo de computação durante o treinamento do modelo.

A engenharia de características é uma etapa que exige um profundo conhecimento do domínio do problema e criatividade. Abaixo algumas técnicas de engenharia de características:

- » **Transformações:** Aplicação de funções matemáticas como logaritmo, exponencial ou normalização para ajustar a distribuição dos dados.
- » **Combinações:** Criação de novas características a partir da combinação de duas ou mais características existentes, como a razão entre duas variáveis.
- » **Interações:** Consideração de interações entre características, como a multiplicação de duas variáveis.
- » **Criação de features binárias:** Transformação de variáveis categóricas em variáveis binárias (*one-hot encoding*).
- » **Extração de features:** Aplicação de técnicas como PCA para reduzir a dimensionalidade e extrair as características mais relevantes.

7.7 Divisão em Conjuntos de Treino, Validação e Teste

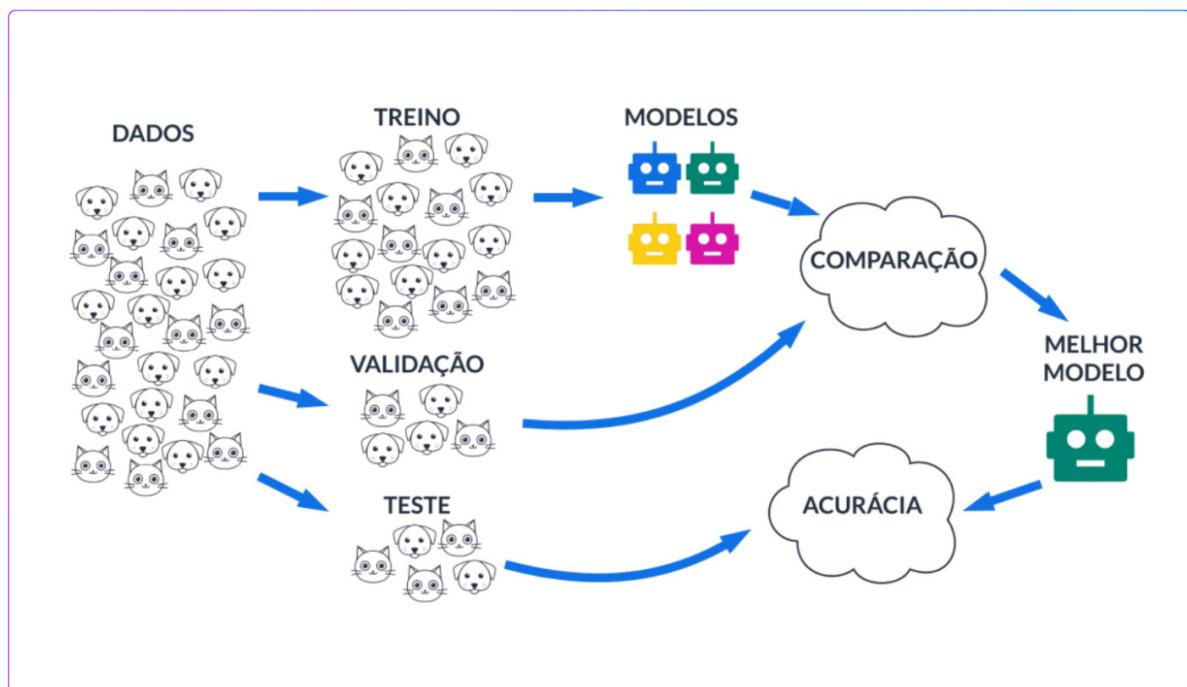
A divisão dos dados em conjuntos de treino, validação e teste é fundamental para a avaliação objetiva dos modelos de ML. O conjunto de treino é utilizado para ajustar os

parâmetros do modelo, o de validação é usado para ajustar hiperparâmetros e evitar *overfitting* (um ajuste excessivo do modelo), e o conjunto de teste fornece uma estimativa final da performance do modelo em dados não vistos, garantindo a generalização do modelo. Normalmente as proporções dos conjuntos são 60% para treino, 20% para validação e 20% para teste.

Para ilustrar, na Figura 32 abaixo temos um conjunto de dados de fotos de cachorros e gatos, cujo objetivo é treinar modelos de classificação para diferenciar fotos de gatos das de cachorros. Uma parte dos dados originais é alocada para o conjunto de treino. O modelo de aprendizado de máquina “aprende” a partir desses dados, ajustando seus parâmetros internos para realizar a tarefa desejada (por exemplo, classificar imagens como gatos ou cachorros). Outra parte dos dados é reservada para o conjunto de validação. Esse conjunto é utilizado para ajustar os hiperparâmetros do modelo, como a taxa de aprendizado ou a complexidade do modelo. A ideia é encontrar a configuração que leva ao melhor desempenho do modelo no conjunto de validação, sem que ele “veja” esses dados durante o treinamento.

Os modelos treinados são avaliados no conjunto de validação. A avaliação permite comparar o desempenho de cada modelo e escolher o que apresenta os melhores resultados. O modelo que obtém o melhor desempenho no conjunto de validação é selecionado como o melhor modelo. Esse melhor modelo é finalmente avaliado em um conjunto de dados totalmente independente, chamado conjunto de teste. Esse conjunto nunca foi utilizado durante o treinamento ou a validação. A avaliação no conjunto de teste fornece uma estimativa mais realista do desempenho do modelo em dados nunca vistos antes. A acurácia é uma métrica que mede o desempenho do modelo no conjunto de teste e indica a porcentagem de exemplos que o modelo classificou corretamente.

Figura 34 - Exemplo divisão do conjunto de dados

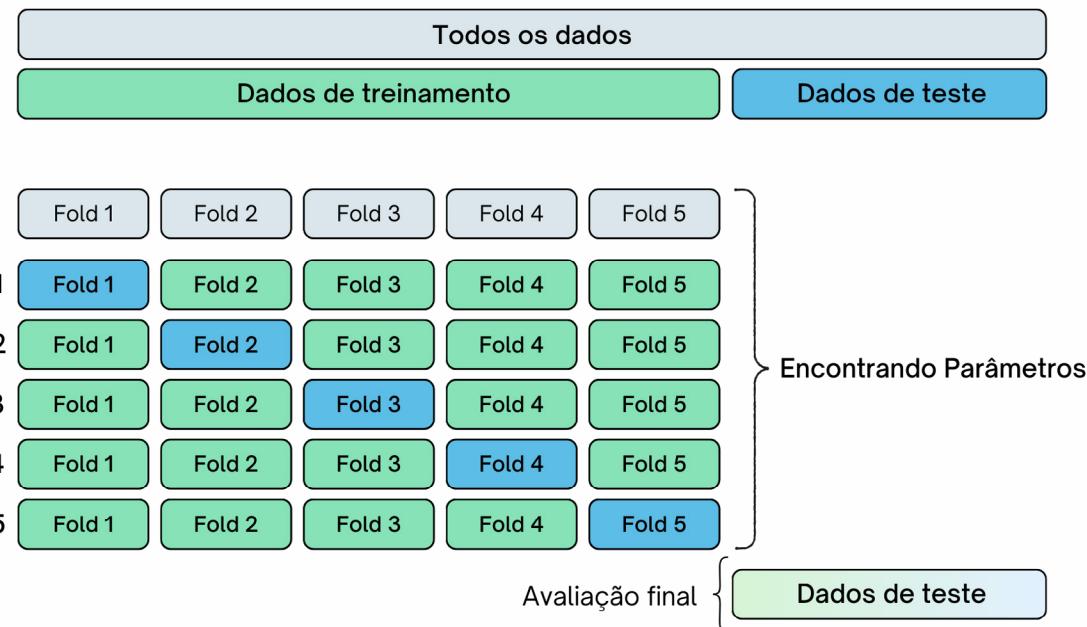


Fonte: adaptada de [Silva \(2023\)](#).

Frequentemente utilizado na fase de divisão dos conjuntos de dados, o Cross-validation (validação cruzada *k-fold*) é uma técnica estatística que permite avaliar a robustez do modelo ao dividir o conjunto de treino em várias partições e realizar treinamentos e validações múltiplas. Isso ajuda a garantir que o modelo não esteja sobreajustado (sofra *overfitting*) a um conjunto de dados específico favorecendo melhor desempenho do modelo para dados desconhecidos. Ao contrário da divisão tradicional em conjuntos de treino, validação e teste, a validação cruzada *k-fold* permite que todos os dados sejam utilizados tanto para treinamento quanto para teste, mitigando o impacto da aleatoriedade na divisão inicial dos dados.

Com base na Figura 31 a seguir, inicialmente o conjunto de dados é dividido em K partes (normalmente 5 ou 10), chamadas de *folds*, de tamanho aproximadamente igual. O processo é repetido K vezes (nesse caso 5), K-1 *folds* são combinados para formar o conjunto de treinamento (*Folds* em verde) e o fold restante é utilizado como conjunto de teste (*fold* em azul). O modelo é treinado em cada iteração (*split*) e avaliado no conjunto de teste correspondente (*fold* azul). As métricas de desempenho (acurácia, precisão, *recall*, etc.) obtidas em cada iteração são calculadas e uma média é tomada. Essa média representa a estimativa do desempenho do modelo.

Figura 35 - Exemplo *k-fold*



Fonte: adaptada de [Scikit-learn \(2024\)](#).

Ao utilizar todos os dados tanto para treinamento quanto para teste, o *k-fold* reduz o viés na estimativa do desempenho do modelo, que poderia ocorrer devido a uma divisão inicial aleatória não representativa. A média das métricas obtidas em todas as iterações fornece uma estimativa mais robusta e confiável do desempenho do modelo. O *k-fold* pode ajudar a identificar modelos que estão sofrendo de *overfitting*, ou seja, modelos que se ajustam muito bem aos dados de treinamento, mas apresentam um desempenho pobre em dados não vistos.

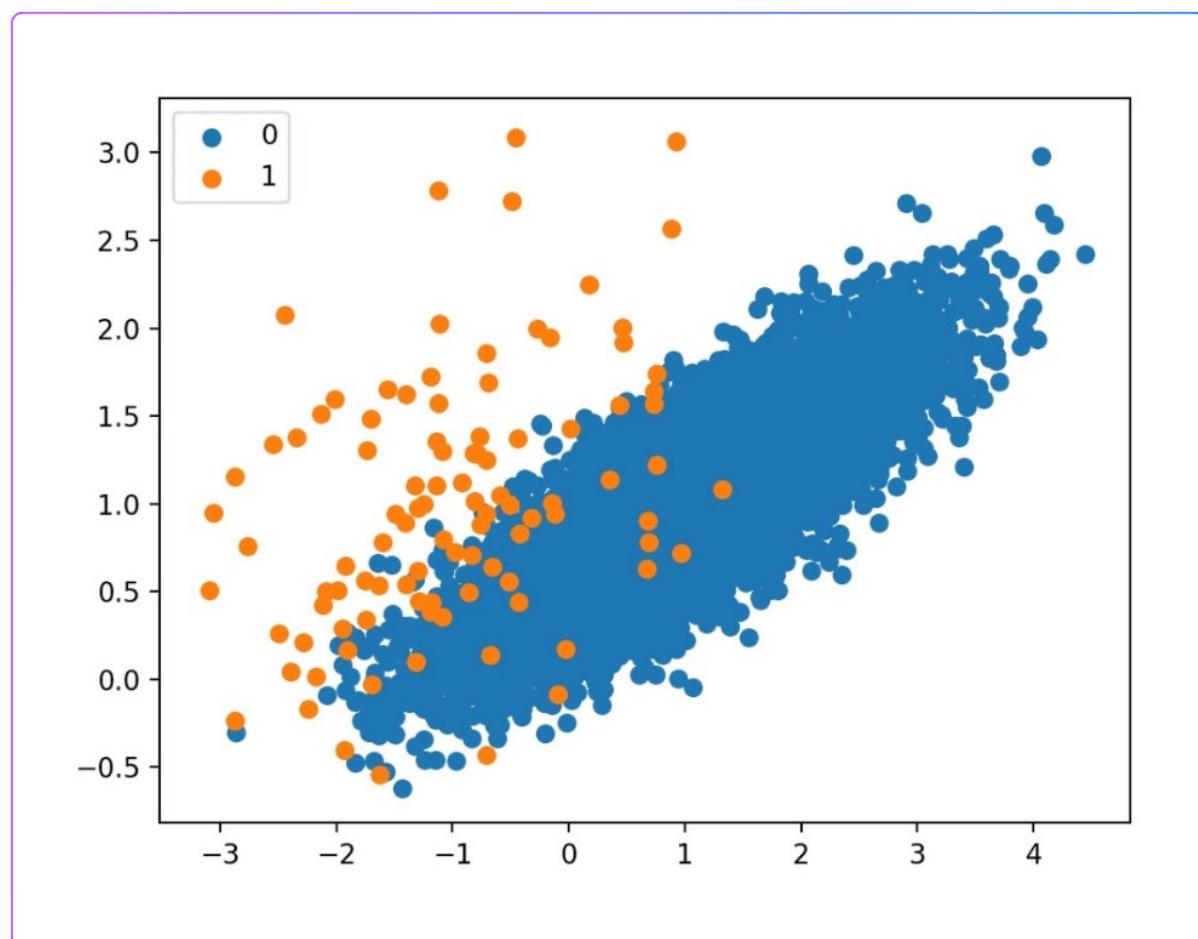
A validação cruzada *k-fold* é uma técnica poderosa e versátil para avaliar o desempenho de modelos de aprendizado de máquina. Ao garantir uma utilização mais eficiente dos dados e fornecer uma estimativa mais robusta do desempenho, o *k-fold* é uma ferramenta essencial no processo de desenvolvimento de modelos.

7.8 Dados Desbalanceados e Técnicas para Balanceamento

Conjuntos de dados desbalanceados ocorrem quando algumas classes têm muito mais exemplos do que outras, o que pode prejudicar o aprendizado do modelo. Técnicas como *oversampling* (aumentar a representatividade das classes minoritárias) e *undersampling* (reduzir a representatividade das classes majoritárias) são usadas para balançar o conjunto de dados e melhorar o desempenho do modelo.

Na figura abaixo há um exemplo de dados desbalanceados, nota-se a evidente diferença na quantidade de dados da classe 0 e 1, em azul e laranja respectivamente.

Figura 36 - Exemplo de dados desbalanceados



Fonte: adaptada de [Medium \(2022\)](#).

Modelos treinados com dados desbalanceados tendem a ser mais precisos na classificação da classe majoritária, ignorando ou classificando incorretamente os exemplos da classe minoritária. Isso ocorre porque o algoritmo busca minimizar o erro global, e a

classe majoritária, por ter mais exemplos, exerce maior influência na função de custo.

A acurácia, uma métrica comumente utilizada para avaliar o desempenho de modelos de classificação, pode ser enganosa em casos de desbalanceamento. Um modelo que simplesmente classifica todos os exemplos como pertencentes à classe majoritária pode obter uma alta acurácia, mesmo que tenha um desempenho muito ruim na classificação da classe minoritária. Métricas como precisão, *recall* e F1-score são mais indicadas para avaliar o desempenho em problemas de classificação com dados desbalanceados. Essas métricas serão detalhadas na unidade de Classificação.

Existem diversas técnicas para lidar com o desbalanceamento de dados, buscando equilibrar a distribuição das classes e melhorar o desempenho dos modelos. As duas principais abordagens são: *Oversampling* e *Undersampling*. A primeira consiste em aumentar a representatividade da classe minoritária, gerando novos exemplos sintéticos. Na segunda busca-se reduzir a representatividade da classe majoritária, removendo aleatoriamente exemplos dessa classe.

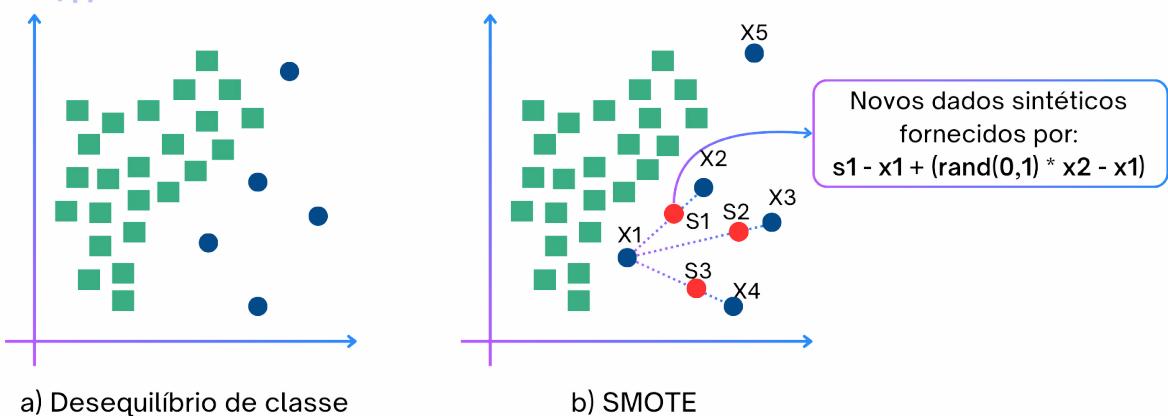
7.8.1 Técnica de *Oversampling*

○ **SMOTE (Synthetic Minority Over-sampling Technique)** é uma das técnicas mais populares de *oversampling*. Ele cria novos exemplos sintéticos para a classe minoritária, interpolando os dados existentes em um espaço multidimensional. Essa técnica é eficaz em lidar com desbalanceamentos significativos e pode melhorar significativamente o desempenho do modelo na classificação da classe minoritária. Nela, dados artificiais são criados com base nos K vizinhos mais próximos. Observe a figura abaixo na qual são apresentadas duas classes, quadrados verdes e bolas azuis. O SMOTE segue os seguintes passos:

1. Seleciona a classe minoritária (bolas em azul).
2. Utiliza o KNN na classe minoritária com algum valor de K. Em outras palavras, seleciona os K vizinhos mais próximos de bola azul x_1 . Nesse caso o K é 3, por isso, os vizinhos mais próximos são x_2 , x_3 e x_4 .
3. Desenha-se linhas entre uma amostra da classe minoritária e seus K vizinhos.
4. Escolhe um valor aleatório sobre essa linha. Nesse caso as bolas em vermelho s_1 , s_2 e s_3 são os valores aleatórios sobre essas linhas. Elas são os dados criados artificialmente para o balanceamento da classe de bolas azuis.

O processo se repete até que a quantidade de bolas (vermelhas + azuis) seja igual a quantidade de quadrados verdes.

Figura 37 - Exemplo do funcionamento do *Synthetic Minority Over-sampling Technique* (SMOTE)



Fonte: adaptada de [Yang \(2022\)](#).

7.8.2 Técnica de *Undersampling*

» **RandomUnderSampler** é uma técnica simples de *undersampling* que remove aleatoriamente exemplos da classe majoritária até que as classes estejam balanceadas. Embora seja fácil de implementar, pode levar à perda de informações importantes, especialmente se a classe majoritária contiver padrões relevantes.

A escolha da técnica de balanceamento depende de diversos fatores, como:

- » **Grau de desbalanceamento:** Para desbalanceamentos extremos, o *oversampling* pode ser mais eficaz.
- » **Tamanho do conjunto de dados:** Em conjuntos de dados pequenos, o *oversampling* pode levar ao *overfitting* (sobreajuste).
- » **Natureza dos dados:** A complexidade dos dados e a relação entre as *features*

O uso prático se estende a diversas aplicações, porém um exemplo didático é a detecção de fraudes em cartões de crédito, onde a classe “fraude” é significativamente menor do que a classe “não fraude”. Ao aplicar o SMOTE, novos exemplos de fraudes sintéticas seriam gerados, tornando a distribuição das classes mais equilibrada. Isso permitiria que o modelo aprendesse a identificar padrões associados a fraudes com maior precisão.

O tratamento de dados desbalanceados é um desafio comum em aprendizado de máquina. A escolha da técnica de balanceamento adequada depende das características do conjunto de dados e dos objetivos da análise. Ao utilizar técnicas como *oversampling* e *undersampling*, é possível melhorar significativamente o desempenho dos modelos de classificação em problemas com classes desbalanceadas.

7.9 Subajuste (*Underfitting*) e Sobreajuste (*Overfitting*)

Subajuste, ou mais comumente conhecido como *Underfitting*, ocorre quando um modelo é muito simples e não consegue capturar as tendências dos dados de treino, resultando em baixo desempenho tanto no treino quanto no teste. Sobreajuste, ou melhor conhecido como *Overfitting*, por outro lado, acontece quando um modelo se ajusta excessivamente aos dados de treino, memorizando ruídos e peculiaridades, e falha em generalizar para novos dados. Técnicas como validação cruzada, regularização e seleção de modelos são usadas para encontrar um equilíbrio ideal e evitar esses problemas. Ambos representam situações em que o modelo não consegue capturar adequadamente as relações entre as variáveis, resultando em um desempenho insatisfatório.

O *underfitting* ocorre quando um modelo é muito simples para capturar a complexidade dos dados. Ele não consegue aprender as relações subjacentes entre as *features* (características) e o *target* (variável alvo), resultando em um alto erro tanto no conjunto de treinamento quanto no conjunto de teste. Um exemplo é o ajuste de uma linha reta a dados que seguem um padrão não linear, como o ilustrado na figura abaixo e a esquerda. As principais causas são:

- » **Modelo subespecificado:** O modelo escolhido não possui a complexidade necessária para representar os dados. Por exemplo, utilizar uma regressão linear para modelar dados não lineares.
- » **Poucos dados de treinamento:** A quantidade de dados disponíveis é insuficiente para o modelo aprender as relações complexas entre as *features*.
- » **Características irrelevantes:** A inclusão de características que não possuem relação com a variável alvo pode dificultar o aprendizado do modelo.

Existem algumas maneiras de mitigação desses problemas, algumas técnicas são:

- » **Aumentar a complexidade do modelo:** Utilizar modelos com mais parâmetros, como redes neurais com mais camadas ou polinômios de maior grau.
- » **Adicionar mais *features*:** Incluir características relevantes que possam ajudar o modelo a capturar as relações subjacentes.

O *overfitting* ocorre quando um modelo se ajusta excessivamente aos dados de treinamento, memorizando o ruído presente nos dados e perdendo a capacidade de generalizar para novos dados. É como ajustar uma curva muito complexa a um conjunto de pontos, capturando até mesmo as pequenas variações aleatórias, como o ilustrado na figura abaixo e a direta. As principais causas são:

- » **Modelo superparametrizado:** O modelo possui muitos parâmetros, o que permite que ele memorize os dados de treinamento em vez de aprender as relações gerais.

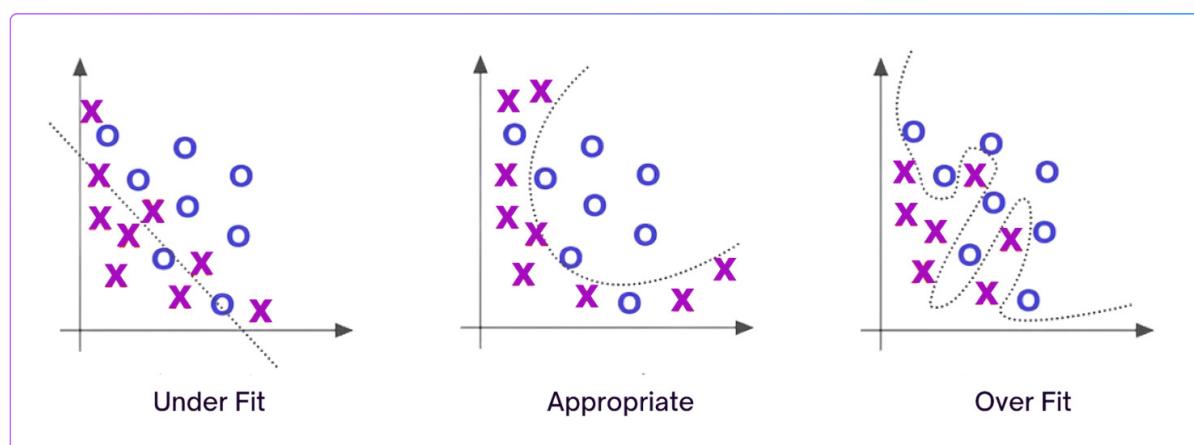
- » **Poucos dados de treinamento em relação à complexidade do modelo:** Quando o modelo é muito complexo em relação à quantidade de dados, ele tende a sobreajustar.
- » **Ruído nos dados:** A presença de ruído nos dados pode levar o modelo a ajustar-se a padrões aleatórios, em vez das relações subjacentes.

Existem algumas maneiras de mitigação desses problemas, algumas técnicas são:

- » **Reducir a complexidade do modelo:** Utilizar modelos com menos parâmetros, como redes neurais com menos camadas ou polinômios de menor grau.
- » **Aumentar a regularização:** A regularização penaliza modelos complexos, evitando que eles se ajustem demais aos dados de treinamento.
- » **Coleta de mais dados:** Aumentar a quantidade de dados de treinamento pode ajudar a reduzir o *overfitting* (sobreajuste), pois o modelo terá mais exemplos para aprender.
- » **Validação cruzada:** A validação cruzada permite avaliar o desempenho do modelo em diferentes subconjuntos dos dados, ajudando a identificar o *overfitting* (sobreajuste).
- » **Dropout:** Uma técnica utilizada em redes neurais que desativa aleatoriamente neurônios durante o treinamento, forçando o modelo a não depender excessivamente de nenhum neurônio em particular.

O objetivo do aprendizado de máquina é encontrar um modelo que seja capaz de generalizar bem para novos dados, evitando tanto o *underfitting* quanto o *overfitting* (sobreajuste), como o ilustrado na figura abaixo e ao centro. A escolha da técnica adequada para mitigar esses problemas depende das características dos dados e do modelo utilizado.

Figura 38 - Exemplo de ajustes de curvas



Fonte: adaptada de [Medium \(2022\)](#).

7.10 Notebook Colab:

No [link do Colab](#) serão abordados exemplos práticos e exercícios de dados estruturados e não estruturados, tipos conjunto de dados; limpeza dos dados; pré processamento; divisão do conjuntos de dados em treino, validação e teste; dados balanceados e desbalanceados; *underfitting* e *overfitting* (sobreajuste).

Objetivos de Aprendizagem

- » Entender a diferença entre dados estruturados e não estruturados;
- » Entender a diferença entre conjunto de dados supervisionados, não supervisionados e semi-supervisionados;
- » Como importar alguns datasets famosos
- » Como fazer a limpeza dos dados
- » Como fazer o pré-processamento
- » Entender a importância da seleção das caraterísticas
- » Como fazer a divisão dos dados em treino, validação e teste
- » Como fazer a validação cruzada
- » Como tratar dados desbalanceados
- » Entender o sobreajuste ou overfitting

Dados Estruturados e Não Estruturados

Dados Estruturados

Os dados estruturados são organizados em um formato específico, como tabelas, **facilitando a sua análise e processamento.**

Dados Não Estruturados

Os dados não estruturados não possuem um formato específico e **são mais difíceis de organizar e analisar.**

Pacotes Python para Dados Estruturados e Não Estruturados

Dados Estruturados

Como exemplo de pacotes python para uso com dados estruturados podem ser citados:

1. **Pandas:** Pandas é um pacote essencial para manipulação e análise de dados

estruturados. Ele fornece estruturas de dados como DataFrames e Series, que permitem operações eficientes de filtragem, agregação e transformação de dados tabulares.

2. **NumPy:** NumPy é uma biblioteca fundamental para computação numérica em Python, oferecendo suporte a arrays multidimensionais e uma ampla gama de funções matemáticas para operações rápidas e eficientes em grandes conjuntos de dados.
3. **Scikit-learn:** Scikit-learn é uma biblioteca robusta para aprendizado de máquina, que inclui diversas ferramentas para pré-processamento de dados, como normalização, imputação de valores ausentes e codificação de variáveis categóricas, facilitando a preparação de dados estruturados para modelos preditivos.
4. **Dask:** Dask é uma biblioteca que permite a manipulação de grandes conjuntos de dados que não cabem na memória, dividindo-os em blocos e distribuindo as operações em paralelo. Ele estende a funcionalidade do Pandas para trabalhar com dados maiores do que a memória disponível.
5. **SQLAlchemy:** SQLAlchemy é uma ferramenta poderosa para a integração de bancos de dados relacionais em Python, facilitando a execução de consultas SQL e a manipulação de dados estruturados diretamente do banco de dados, mantendo a compatibilidade com diferentes sistemas de gerenciamento de banco de dados (SGBDs).
6. **Polars:** Polars é uma biblioteca de processamento de dados altamente eficiente e de alta performance, projetada para manipulação e análise de grandes conjuntos de dados tabulares. Ele utiliza uma abordagem baseada em colunas e oferece APIs semelhantes ao Pandas, mas com desempenho significativamente mais rápido devido ao seu uso de Rust como backend.
7. **PySpark:** PySpark é a interface Python para o Apache Spark, um framework de computação distribuída. Ele permite a manipulação e análise de grandes volumes de dados de forma paralela e distribuída, sendo ideal para processamento de Big Data em clusters, com suporte para SQL, machine learning e streaming em tempo real..

Pandas:

```
# Importação da biblioteca
import pandas as pd

# Definição dos dados de forma estruturada Colunas e Linhas
d = {'col1': [1, 2], 'col2': [3, 4]}

# Importação do dicionário de dados para dentro do DataFrame do pandas
df = pd.DataFrame(data=d)

df
```

continua

	col1	col2
0	1	3
1	2	4

Dados Não Estruturados

Como exemplo de pacotes python para uso com dados não estruturados podem ser citados:

1. **Scikit-learn:** Scikit-learn é uma biblioteca versátil, já abordada também para uso com dados estruturados. Ela oferece uma ampla gama de algoritmos para aprendizado não supervisionado, como clustering, redução de dimensionalidade e análise de agrupamentos. Ela fornece implementações eficientes de algoritmos como k-means, PCA, e DBSCAN, permitindo a descoberta de padrões em dados não rotulados.
2. **OpenCV:** OpenCV é uma biblioteca poderosa para processamento de imagens e visão computacional, frequentemente utilizada em tarefas não supervisionadas como segmentação de imagem e detecção de bordas. Sua vasta coleção de ferramentas permite a análise e transformação de imagens, além de suportar operações avançadas de processamento de vídeo.
3. **NLTK (Natural Language Toolkit):** NLTK é uma biblioteca essencial para o processamento de linguagem natural (NLP), oferecendo ferramentas para análise de texto não supervisionada, como tokenização, stemming, e análise de frequências. Ela também suporta tarefas como clustering de documentos e análise semântica, facilitando o trabalho com grandes corpora de texto.
4. **HDBSCAN:** HDBSCAN é uma biblioteca especializada em clustering hierárquico baseado em densidade, que pode identificar clusters de diferentes densidades em dados não supervisionados. Ele é particularmente útil para encontrar agrupamentos em conjuntos de dados complexos e de forma não paramétrica, sem necessidade de definir o número de clusters previamente.
5. **Gensim:** Gensim é uma biblioteca focada em modelagem de tópicos e análise de grandes corpora de texto. Ela é conhecida por suas implementações de modelos como LDA (Latent Dirichlet Allocation) e Word2Vec, que permitem a extração de tópicos e representações semânticas de documentos em cenários não supervisionados.

NLTK (Natural Language Toolkit):

```
# Importação das bibliotecas
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

# Exemplo de texto
```

continua

```

texto = "Olá, como você está hoje?"

# Tokenizar o texto
tokens = word_tokenize(texto, language='portuguese')

print(tokens)

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]  Unzipping tokenizers/punkt.zip.
['Olá', ',', 'como', 'você', 'está', 'hoje', '?']

```

OpenCV:

```

# Importação das bibliotecas
from skimage import io
import matplotlib.pyplot as plt
import cv2

# Carrega a imagem da internet
image = io.imread('https://akcit.ufg.br/_next/image?url=%2Flogo_akcit.png&w=1080&q=75')

# Salva a imagem
io.imsave('image.jpg', image)

# Carrega a imagem salva utilizando o OpenCV

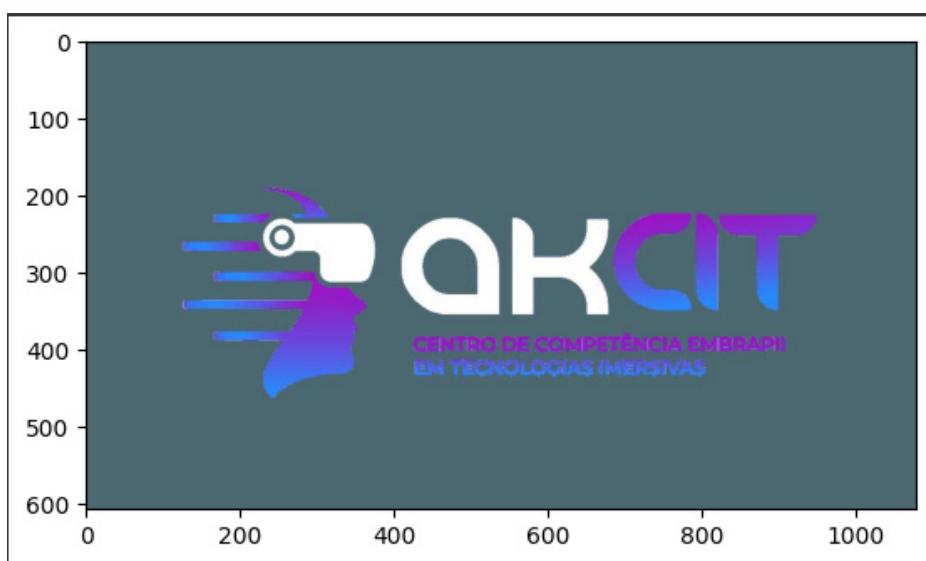
image = cv2.imread('image.jpg')

# Corrige as cores
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
plt.show()

```

Figura 39 - Exemplo de carregamento de imagem



Fonte: autoria própria.

Conjuntos de Dados Supervisionados, Não Supervisionados e Semi-Supervisionados

Dados Supervisionados

Os dados supervisionados são aqueles em que cada exemplo de treinamento é composto por uma entrada e a saída desejada (rótulo). Exemplos incluem:

- » **MNIST**: Conjunto de dados de dígitos manuscritos com rótulos indicando os dígitos de 0 a 9.
- » **Iris**: Conjunto de dados de flores com características como comprimento da pétala e rótulos indicando a espécie da flor.
- » **CIFAR-10**: Conjunto de imagens divididas em 10 classes, como aviões, carros, gatos, etc.

Exemplo de um conjunto de dados supervisionado

O conjunto de dados Iris é amplamente reconhecido e utilizado em estatística e aprendizado de máquina, especialmente como exemplo introdutório para a aplicação de técnicas de classificação. Este conjunto de dados consiste em 150 amostras de flores pertencentes à espécie Iris, distribuídas igualmente entre três subespécies: **Iris setosa**, **Iris versicolor** e **Iris virginica**.

Cada amostra no conjunto de dados é caracterizada por quatro variáveis numéricas que descrevem atributos morfológicos das flores:

- » **Comprimento da sépala** (em centímetros);
- » **Largura da sépala** (em centímetros);
- » **Comprimento da pétala** (em centímetros);
- » **Largura da pétala** (em centímetros).

Devido à sua simplicidade, equilíbrio e bem-definidas classes, o conjunto de dados Iris é frequentemente empregado para ilustrar algoritmos de classificação e técnicas de análise exploratória de dados.

```
# Importa dados do scikitlearn
from sklearn.datasets import load_iris
data = load_iris(as_frame=True)
```

```
# Imprime a descrição do dataset
print(data.DESCR)
.. _iris_dataset:
Iris plants dataset
-----
```

continua

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
- Iris-Setosa
- Iris-Versicolour
- Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None

:Class Distribution: 33.3% for each of 3 classes.

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

|details-start|
References
|details-split|

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions

- on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

```
|details-end|
```

```
# Lista de características  
data.feature_names
```

```
['sepal length (cm)',  
'sepal width (cm)',  
'petal length (cm)',  
'petal width (cm)']
```

```
# Lista de rótulos  
data.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
# Apresenta os dados como Pandas DataFrame  
data.frame
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Dados Não Supervisionados

Os dados não supervisionados são aqueles que não possuem rótulos, e o objetivo é encontrar padrões ou estrutura nos dados. Exemplos incluem:

- » **Google News Word Vectors:** O conjunto de dados consiste em vetores de palavras (word embeddings) treinados em um grande corpus de texto extraído do Google News. Esses vetores foram gerados utilizando o modelo Word2Vec, uma técnica de aprendizado não supervisionado que captura relações semânticas entre palavras ao mapeá-las em um espaço vetorial contínuo. Este conjunto é amplamente utilizado em tarefas de análise semântica, como busca por similaridade de palavras, análise de relações entre palavras, e construção de representações vetoriais para uso em modelos de aprendizado de máquina. Contém cerca de 3 milhões de vetores de palavras e frases, cada um com 300 dimensões. Sua aplicação inclui: Análise semântica, processamento de linguagem natural (NLP), busca por similaridade de palavras. Este conjunto de dados pode ser baixado publicamente em <https://code.google.com/archive/p/word2vec/>
- » **Análise de Sentimentos de Tweets:** consiste em uma coleção de tweets não rotulados, que podem ser utilizados para tarefas como clustering, redução de dimensionalidade ou identificação de padrões sem supervisão. Embora o conjunto de dados original seja não rotulado, ele é frequentemente utilizado em pesquisas de aprendizado não supervisionado para agrupar tweets com base em sua similaridade ou para extrair temas latentes. Dependendo da coleção possui tamanho variável, mas normalmente composto por milhares ou milhões de tweets. É normalmente aplicado em: Clustering, análise de tópicos, redução de dimensionalidade, detecção de tendências em redes sociais. Os conjuntos de dados de tweets podem ser extraídos via API ou encontrados em repositórios públicos como o Kaggle em várias competições ou projetos, como por exemplo <https://www.kaggle.com/code/leandrodoze/sentiment-analysis-in-portuguese>.
- » **Digits Dataset:** é composto por 1,797 imagens de dígitos manuscritos, cada uma com uma resolução de 8x8 pixels, em escala de cinza. Cada imagem representa um dígito de 0 a 9, e é acompanhada por um rótulo que indica o dígito correspondente. Esse conjunto de dados é amplamente utilizado para desenvolver e avaliar algoritmos de aprendizado de máquina em tarefas de reconhecimento de caracteres e classificação de imagens.

Exemplo de um conjunto de dados não supervisionados

Como exemplo, será carregado o conjunto de dados Digits de dígitos manuscritos utilizando a função `load_digits()` do Scikit-learn e, em seguida, visualiza uma amostra de imagens desse conjunto.

Primeiro, o conjunto de dados é carregado e os rótulos são removidos para simular um cenário não supervisionado. Em seguida, utiliza-se a biblioteca **Matplotlib** para criar uma grade de subplots e exibir as primeiras 10 imagens do conjunto de dados, cada uma representando um dígito manuscrito em escala de cinza..

```
# Importar as bibliotecas necessárias
from sklearn import datasets
import matplotlib.pyplot as plt

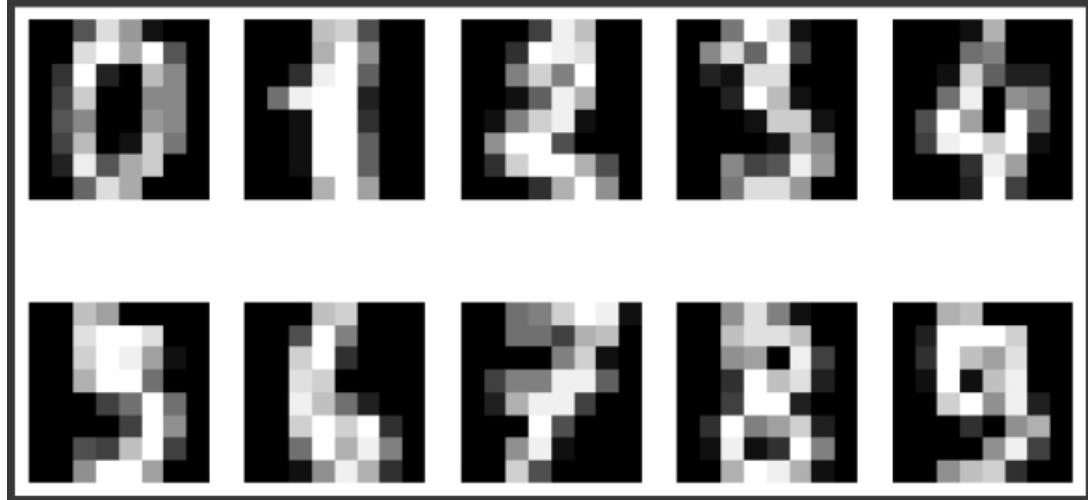
# Carregar o conjunto de dados de Digits
digits = datasets.load_digits()

# Remover os rótulos para simular um cenário não supervisionado
data = digits.data

# Visualizar algumas imagens do conjunto de dados
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.ravel()):
    ax.imshow(data[i].reshape(8, 8), cmap='gray')
    ax.axis('off')

plt.show()
```

Figura 40 - Exemplo de um conjunto de dados não supervisionados



Fonte: autoria própria.

Dados Semi-Supervisionados

Os dados semi-supervisionados combinam uma pequena quantidade de dados rotulados com uma grande quantidade de dados não rotulados. Na prática esses datasets podem ser adaptados dos dataset supervisionados, retirando os rótulos de alguns caos e assim podendo ser empregado para teste em algoritmos que visam performance em

conjunto de dados semi-supervisionados. Exemplos incluem:

- » **Cifar-10 com Rótulos Parciais:** Apenas uma fração das imagens possui rótulos, enquanto o restante é utilizado para melhorar a generalização do modelo.
- » **Dados de Textos Anotados:** Pequeno subconjunto de documentos anotados manualmente, com o restante não anotado, usado em tarefas de classificação de texto.
- » **Vídeos com Quadros Anotados:** Alguns quadros de um vídeo são rotulados, enquanto a maioria não possui rótulos, utilizado para treinamento de modelos de reconhecimento de ações.

Exemplo de um conjunto de dados semi-supervisionado

Para criar um dataset semi-supervisionado artificialmente será utilizado o mesmo dataset **Digits** utilizado para o caso não supervisionado.

Inicialmente, carregam-se os dados e os rótulos associados, depois embaralha aleatoriamente os índices dos exemplos para garantir uma divisão não sistemática. Em seguida, apenas 10% dos rótulos originais são mantidos, sendo que o restante dos exemplos como são rotulados com o valor -1. A partir dessas etapas, pode-se observar como transformar um conjunto de dados rotulado em um formato semi-supervisionado.

Em **X** estarão os valores dos 64 pixels (imagem de 8x8 pixels) que representam a imagem de um numeral. Em **y** qual é o número de 0 a 9 que aquela imagem representa.

```
# Importação das bibliotecas
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt

# Carregar o conjunto de dados de dígitos
digits = datasets.load_digits()
# X possui as características da imagem (valores dos 64 pixels --> imagem 8x8)
X = digits.data
# y possui o valor do rótulo (0 a 9) que representa aquela imagem (conjunto de pixels)
y = digits.target

print("Dados das imagens")
print(X)
print("Dados dos rótulos")
print(y)

print("Tamanho dos Dados das imagens:", X.shape)
print("Tamanho dos rótulos:", y.shape)
```

continua

```
Dados das imagens
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
Dados dos rótulos
[0 1 2 ... 8 9 8]
Tamanho das Dados das imagens: (1797, 64)
Tamanho dorótulos: (1797,)
```

```
# Quantidade de exemplos por rótulo
y_ori = pd.DataFrame(data=y, columns=['y_ori'])
y_ori.groupby('y_ori')['y_ori'].count()
```

y_ori	
y_ori	
0	178
1	182
2	177
3	183
4	181
5	182
6	181
7	179
8	174
9	180

dtype: int64

```
# Embaralhar o dataset
random_state = np.random.RandomState(42)
indices = np.arange(len(y))
random_state.shuffle(indices)
```

continua

```
print("Amostra dos valores de indices: ", indices)
print("Comprimento do indice: ", len(indices))
```

```
Amostra dos valores de indices: [1245 220 1518 ... 860 1459 1126]
Comprimento do indice: 1797
```

```
# Manter rótulos somente de 10% dos dados iniciais
n_labeled_points = int(len(y) * 0.1)
print("Quantidade de rótulos que serão mantidos: ", n_labeled_points)
```

```
# Marcar a maioria dos rótulos como -1 (não rotulados)
unlabeled_indices = indices[n_labeled_points:]
y[unlabeled_indices] = -1
```

```
Quantidade de rótulos que serão mantidos: 179
```

```
# Quantidade de exemplos por rótulo
y_semi = pd.DataFrame(data=y, columns=['y_semi'])
y_semi.groupby('y_semi')['y_semi'].count()
```

y_semi	
-1	1618
0	17
1	11
2	16
3	17
4	25
5	22
6	19
7	19
8	8
9	25

dtype: int64

Limpeza dos dados

Objetivos

- » Compreender a importância da limpeza de dados.
- » Aprender técnicas básicas de limpeza de dados.
- » Implementar essas técnicas utilizando bibliotecas populares do Python.
- » Praticar a aplicação das técnicas de limpeza de dados em conjuntos de dados reais.

1. Introdução

A limpeza de dados é uma etapa crucial no processo de análise de dados. Dados sujos podem levar a resultados enganosos e comprometer a qualidade da análise. Este notebook irá guiá-lo através dos conceitos e técnicas fundamentais para a limpeza de dados utilizando Python.

2. Explicação Teórica

Importância da Limpeza de Dados

- » **Confiabilidade:** Dados limpos aumentam a confiabilidade das análises.
- » **Qualidade:** Dados de alta qualidade são essenciais para obter insights precisos.
- » **Eficiência:** Dados bem estruturados facilitam a análise e economizam tempo.

Conceitos Principais

- » **Valores ausentes:** Dados faltantes ou nulos.
- » **Valores duplicados:** Registros idênticos que podem distorcer a análise.
- » **Outliers:** Valores extremos que podem influenciar desproporcionalmente os resultados.
- » **Formatação:** Uniformidade nos formatos de dados.

3. Exemplos Práticos

Nesse exemplo ilustra-se um processo abrangente de limpeza de dados, aplicando técnicas práticas para lidar com valores ausentes, duplicatas e outliers em um conjunto de dados. Primeiramente, o carrega-se um conjunto de dados do Titanic e visualiza-se a presença de valores ausentes utilizando a biblioteca **missingno**, substituindo os valores ausentes na coluna **age** pela média da coluna. Em seguida, verifica-se e remove regis-

tos duplicados para garantir a integridade dos dados. Para a detecção e tratamento de outliers na coluna **fare**, o utiliza-se um boxplot para visualizar os outliers e aplica o método do intervalo interquartil (IQR) para remover valores extremos. Finalmente, compara-se a média da coluna **fare** antes e depois da remoção dos outliers, demonstrando o impacto desses valores extremos nos resultados analíticos.

Carregando Bibliotecas e Dados

```
import pandas as pd
import numpy as np

# Carregar um conjunto de dados de exemplo
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv'
df = pd.read_csv(url)
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Tratamento de Valores Ausentes

```
# Visualizar valores ausentes
df.isnull().sum()
```

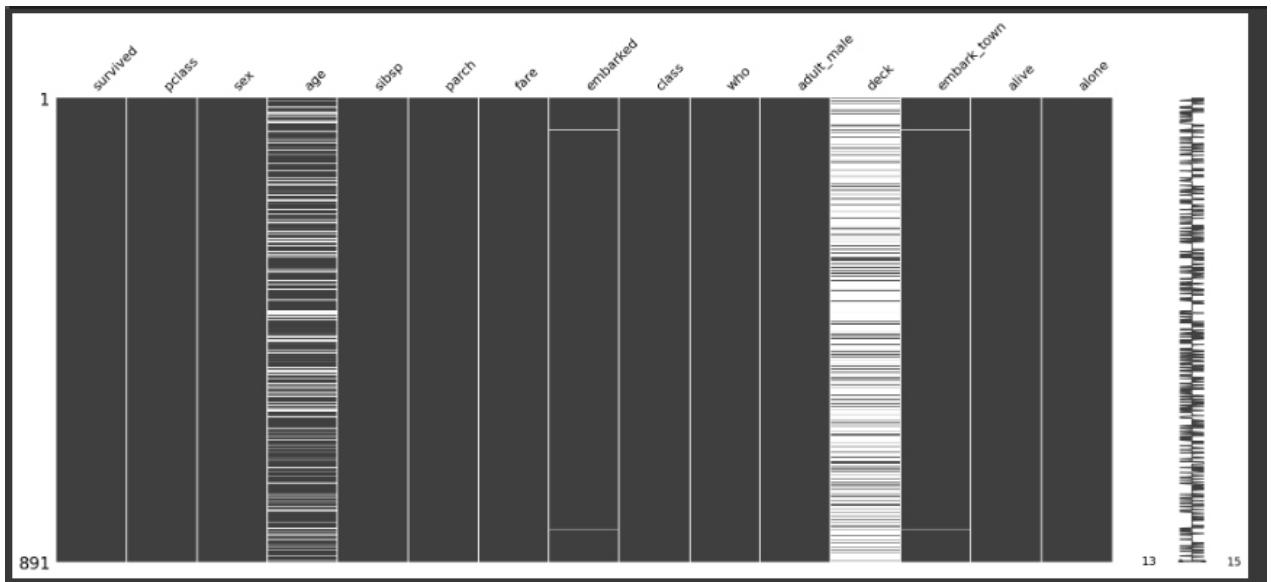
	0
survived	0
pclass	0
sex	0
age	177
sibsp	0
parch	0
fare	0
embarked	2
class	0
who	0
adult_male	0
deck	688
embark_town	2
alive	0
alone	0
	dtype: int64

Outra maneira de verificar os valores ausentes é visualizando graficamente. Abaixo o código apresenta graficamente os valores ausentes em cada coluna do dataset.

Em cor escura, os dados preenchidos, na cor branca os valores ausentes. Em alguns casos reais, pode ser que em determinados períodos não houve o registro de certa característica. Porém para chegar a essa conclusão pode não ser tão direto assim. Nesse caso, a visualização gráfica apontaria um grande bloco branco em meio a grandes blocos escuros, e facilitaria essa conclusão.

```
import missingno as msno  
  
# Visualizar valores ausentes  
msno.matrix(df)  
  
plt.show()
```

Figura 41 - Visualização de valores ausentes



Fonte: autoria própria.

```
# Preencher valores ausentes  
df['age'].fillna(df['age'].mean(), inplace=True)
```

```
# Visualizar valores ausentes  
df.isnull().sum()
```

continua

	0
survived	0
pclass	0
sex	0
age	0
sibsp	0
parch	0
fare	0
embarked	2
class	0
who	0
adult_male	0
deck	688
embark_town	2
alive	0
alone	0
dtype: int64	

Remoção de Valores Duplicados

```
# Dimensões do dataset antes da remoção de duplicatas
```

```
df.shape
```

```
(891, 15)
```

```
# Verificar as duplicidades
```

```
df.loc[df.duplicated()]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
47	1	3	female	29.699118	0	0	7.7500	Q	Third	woman	False	NaN	Queenstown	yes	True
76	0	3	male	29.699118	0	0	7.8958	S	Third	man	True	NaN	Southampton	no	True
77	0	3	male	29.699118	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
87	0	3	male	29.699118	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
95	0	3	male	29.699118	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...
870	0	3	male	26.000000	0	0	7.8958	S	Third	man	True	NaN	Southampton	no	True
877	0	3	male	19.000000	0	0	7.8958	S	Third	man	True	NaN	Southampton	no	True
878	0	3	male	29.699118	0	0	7.8958	S	Third	man	True	NaN	Southampton	no	True
884	0	3	male	25.000000	0	0	7.0500	S	Third	man	True	NaN	Southampton	no	True
886	0	2	male	27.000000	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True

107 rows × 15 columns

continua

```
# Remover duplicatas
df.drop_duplicates(inplace=True)

# Dimensões do dataset antes da remoção de duplicatas
df.shape

(784, 15)
```

Tratamento de Outliers

A técnica apresentada chama-se Tukey's Fence, ela e outras técnicas podem ser encontradas em <https://en.wikipedia.org/wiki/Outlier>

O Tukey's Fence é um método estatístico utilizado para identificar e remover outliers em um conjunto de dados, particularmente em conjunto com a análise de boxplots. Este método baseia-se em extensões do intervalo interquartil (IQR) e utiliza um critério de "cerca" para determinar quais observações são consideradas outliers.

O processo envolve calcular o IQR, que é a diferença entre o terceiro quartil (Q3) e o primeiro quartil (Q1), e então aplicar um fator multiplicativo, geralmente 1.5 ou 3, para estabelecer limites superior e inferior para a identificação de outliers. Dados que estão além desses limites são considerados como outliers.

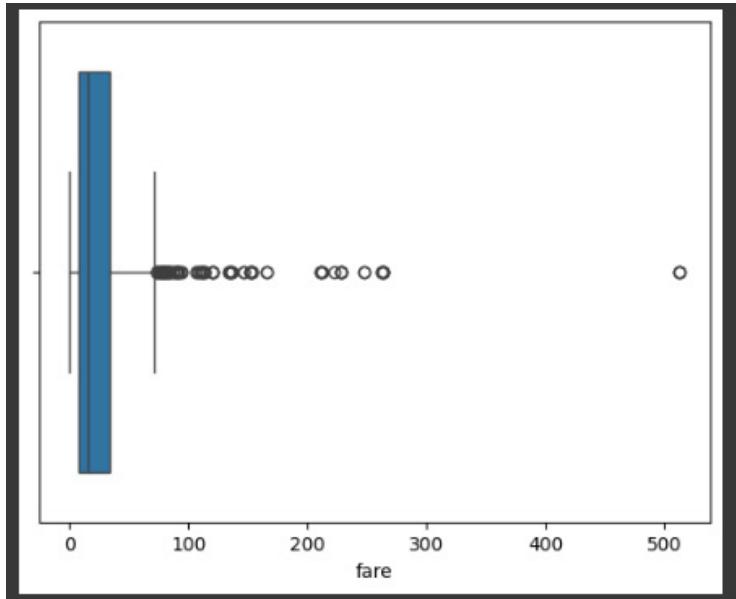
Em um boxplot, os outliers são visualmente representados como pontos fora das "cercas" do boxplot, facilitando a identificação e a decisão sobre quais valores devem ser removidos para melhorar a qualidade dos dados e a precisão das análises subsequentes.

```
# Visualizar outliers
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=df['fare'])
plt.show()

# Remover outliers
Q1 = df['fare'].quantile(0.25)
Q3 = df['fare'].quantile(0.75)
IQR = Q3 - Q1
df_sem_outlier = df[~((df['fare'] < (Q1 - 1.5 * IQR)) | (df['fare'] > (Q3 + 1.5 * IQR)))]
```

Figura 42 - Tratamento de Outliers



Fonte: autoria própria.

```
# Comparaçao do efeito dos outliers
print("Média dos valores de fare originais: ", df['fare'].mean())
print("Média dos valores de fare sem outliers: ", df_sem_outlier['fare'].mean())

Média dos valores de fare originais: 34.71173966836735
Média dos valores de fare sem outliers: 19.50749002932551
```

4. Exercícios e Desafios

Exercício 1: Identificação e Tratamento de Valores Ausentes

Rode o código abaixo que introduz alguns valores ausentes ao conjunto de dados de habitação da Califórnia.

Com base no df gerado, utilize o que aprendeu e identifique as colunas com valores ausente com preenchendo com os valores da mediana.

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing

# Carregar o conjunto de dados California Housing
data = fetch_california_housing(as_frame=True)
df = data.frame

# Introduzir valores ausentes artificialmente para fins de demonstração
df.loc[::10, 'MedInc'] = np.nan # Introduzir valores ausentes na coluna 'MedInc'
df.loc[::15, 'HouseAge'] = np.nan # Introduzir valores ausentes na coluna 'HouseAge'

df.head()
```

continua

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	NaN	NaN	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Exercício resolvido

O código realiza a identificação de colunas numéricas que contêm valores ausentes, preenche esses valores com a mediana de cada coluna e, finalmente, verifica se ainda há valores ausentes no DataFrame.

```
# Exercício Resolvido
# Identificar colunas numéricas com valores ausentes
# Seleciona todas as colunas do DataFrame df que possuem dados numéricos.
numeric_columns = df.select_dtypes(include=[np.number]).columns

# Identifica quais dessas colunas numéricas possuem valores ausentes.
# O método isnull() é usado para verificar valores ausentes,
# any() verifica se há pelo menos um valor ausente em cada coluna.
# O resultado é uma série de colunas que contêm valores ausentes,
# armazenadas na variável missing_numeric_columns.
missing_numeric_columns = numeric_columns[df[numeric_columns].isnull().any()]

# Apresenta as colunas com valores ausentes
print("Essas são as colunas com valores ausentes: ", missing_numeric_columns)

# Preencher valores ausentes com a mediana
# Percorre cada coluna identificada como contendo valores ausentes.
for column in missing_numeric_columns:
    # Para cada coluna, o métodofillna() é utilizado para preencher os valores
    # ausentes da coluna.
    # O método median() calcula a mediana dos valores não ausentes da coluna.
    # O parâmetro inplace=True garante que as mudanças sejam aplicadas diretamente ao
    # DataFrame df,
    # sem a necessidade de criar uma cópia.
    df[column].fillna(df[column].median(), inplace=True)

# Imprime na tela o número de valores ausentes restantes em cada coluna.
# A função isnull() cria uma matriz booleana indicando a presença de valores ausentes,
# sum() calcula o total de valores ausentes por coluna.
print(df.isnull().sum())
```

continua

```
Essas são as colunas com valores ausentes: Index(['MedInc', 'HouseAge'],
dtype='object')
MedInc      0
HouseAge    0
AveRooms   0
AveBedrms  0
Population  0
AveOccup   0
Latitude   0
Longitude  0
MedHouseVal 0
dtype: int64
```

Pré Processamento

Objetivos

- » Compreender a importância do pré-processamento de dados.
- » Aprender técnicas básicas de normalização e codificação de variáveis categóricas.
- » Implementar essas técnicas utilizando bibliotecas populares do Python.
- » Praticar a aplicação das técnicas de pré-processamento em conjuntos de dados reais.

1. Introdução

O pré-processamento de dados é uma etapa essencial no *pipeline* (sequência de etapas) de *machine learning*. Ele prepara os dados para a modelagem, melhorando a eficiência e a precisão dos modelos. Esta aula abordará duas técnicas fundamentais de pré-processamento: normalização e codificação de variáveis categóricas.

Normalização

A normalização é o processo de ajustar a escala dos dados para que estejam em um intervalo específico, geralmente entre 0 e 1. Isso é importante porque muitos algoritmos de *machine learning* se beneficiam de dados normalizados, resultando em melhor desempenho e convergência mais rápida.

Métodos de Normalização

Saiba mais: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#plot-all-scaling-minmax-scaler-section

Aqui serão abordados os 2 principais métodos:

- » **Min-Max Scaling**: Transforma os dados para que seus valores fiquem entre 0 e 1.
- » **Z-score Standardization**: Transforma os dados para que tenham média 0 e desvio padrão 1.

Exemplo de aplicação de Normalização

O exemplo abaixo tem como objetivo principal demonstrar técnicas de pré-processamento, especificamente no contexto de um conjunto de dados de imóveis da Califórnia. As etapas envolvem:

- » **Carregamento dos dados**: O conjunto de dados da California Housing é carregado e armazenado em um DataFrame Pandas para facilitar a manipulação.
- » **Introdução de variáveis categóricas**: Uma nova coluna categórica é criada aleatoriamente para simular uma situação real onde os dados podem conter variáveis desse tipo.
- » **Normalização dos dados**: As variáveis numéricas contínuas 'MedInc' e 'AveOccup' são normalizadas utilizando os métodos MinMaxScaler e StandardScaler, com o objetivo de escalar os valores para um intervalo específico ou com média zero e desvio padrão unitário, respectivamente.
- » **Visualização dos dados**: Gráficos de dispersão são utilizados para visualizar a distribuição dos dados antes e após a normalização.

Carregando Bibliotecas e Dados

```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder,
LabelEncoder

# Carregar o conjunto de dados California Housing
data = fetch_california_housing(as_frame=True)
df = data.frame

# Introduzir variáveis categóricas artificiais

# A função np.random.choice do NumPy é utilizada para gerar amostras aleatórias
# a partir de uma determinada população.
# Ela é usada para criar a coluna 'ocean_proximity', atribuindo aleatoriamente
# uma das três categorias possíveis ('<1H OCEAN', 'INLAND', 'NEAR OCEAN') a cada
# observação do DataFrame.
# em size: O número de amostras a serem geradas (igual ao número de linhas do
# DataFrame).
```

continua

```
# Essa etapa será necessária quando do tratamento de variáveis categóricas.
df['ocean_proximity'] = np.random.choice(['<1H OCEAN', 'INLAND', 'NEAR OCEAN'],
size=len(df))

df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	ocean_proximity
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526	NEAR OCEAN
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585	NEAR OCEAN
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521	INLAND
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413	<1H OCEAN
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422	<1H OCEAN

Min-Max Scaling

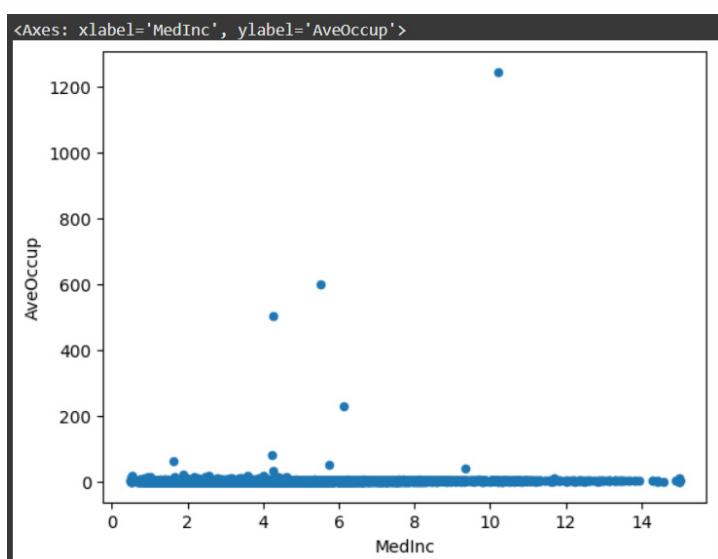
- » **MedInc**: Representa a renda média dos blocos censitários. Essa variável numérica contínua indica o nível de renda da região onde o imóvel está localizado.
- » **AveOccup**: Representa a ocupação média por unidade habitacional. Essa variável numérica contínua fornece informações sobre a densidade populacional nos imóveis.

Utilizaremos essas colunas para testar as técnicas de normalização.

Para melhor compreender, será visto a distribuição desses dados.

```
# Distribuição dos dados antes da normalização
# Atente-se aos valores do eixo x, variando aproximadamente entre 0 a 14
# Atente-se aos valores do eixo y, variando aproximadamente entre 0 e 1200
df.plot.scatter('MedInc', 'AveOccup')
```

Figura 43 - Distribuição de dados antes da normalização - MinMax Scaling



Fonte: autoria própria.

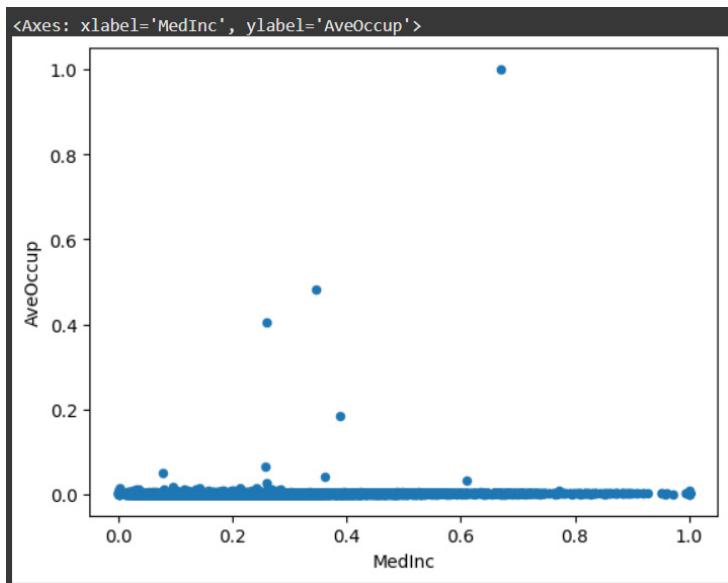
O **min-max scaling** escala os valores de cada característica para o intervalo [0, 1]. É útil quando se deseja que todas as características tenham a mesma escala.

```
# Cria um objeto da classe MinMaxScaler.  
# Esse objeto armazenará os parâmetros de transformação que serão aprendidos dos  
dados.  
scaler = MinMaxScaler()  
  
# Combina duas etapas em uma:  
# fit: Calcula os valores mínimo e máximo das colunas MedInc e AveOccup para determinar  
a escala.  
# transform: Aplica a transformação aos dados, ajustando cada valor para o intervalo  
[0, 1]  
# Os valores transformados substituem os valores originais nas colunas especificadas.  
df[['MedInc', 'AveOccup']] = scaler.fit_transform(df[['MedInc', 'AveOccup']])  
  
# Exibe as primeiras linhas do DataFrame após a transformação, mostrando os novos  
valores normalizados.  
df[['MedInc', 'AveOccup']].head()
```

	MedInc	AveOccup
0	0.539668	0.001499
1	0.538027	0.001141
2	0.466028	0.001698
3	0.354699	0.001493
4	0.230776	0.001198

```
# Distribuição dos dados após a normalização  
# Compare esse gráfico com o primeiro, note como os valores dos eixos x e y  
# agora estão entre 0 e 1  
df.plot.scatter('MedInc', 'AveOccup')
```

Figura 44 - Distribuição de dados após normalização - MinMaxScaling



Fonte: autoria própria.

Z-score Standardization

De forma análoga ao que foi realizado com o **Min-max Scaling**, agora será aplicado outra forma de normalização, o **Z-score Standardization**.

StandardScaler: Centraliza os dados (média = 0) e escala para ter variância unitária.

```
# Cria um objeto da classe StandardScaler.  
# Esse objeto armazenará os parâmetros de transformação que serão aprendidos dos  
dados.  
scaler = StandardScaler()  
  
# Combina duas etapas em uma:  
# fit: Calcula os valores mínimo e máximo das colunas MedInc e AveOccup para determinar  
a escala.  
# transform: Aplica a transformação aos dados, ajustando cada valor para que ao final  
# o conjunto possua a média 0 e variância unitária.  
# Os valores transformados substituem os valores originais nas colunas especificadas.  
df[['MedInc', 'AveOccup']] = scaler.fit_transform(df[['MedInc', 'AveOccup']])  
  
# Exibe as primeiras linhas do DataFrame após a transformação, mostrando os novos  
valores normalizados.  
df[['MedInc', 'AveOccup']].head()
```

	MedInc	AveOccup
0	2.344766	-0.049597
1	2.332238	-0.092512
2	1.782699	-0.025843
3	0.932968	-0.050329
4	-0.012881	-0.085616

continua

```
# Imprime os valores da média
# Nota-se que os valores obtidos são muito proximo de 0
print("Média dos valores após a normalização: ")
print(df[['MedInc', 'AveOccup']].mean())
```

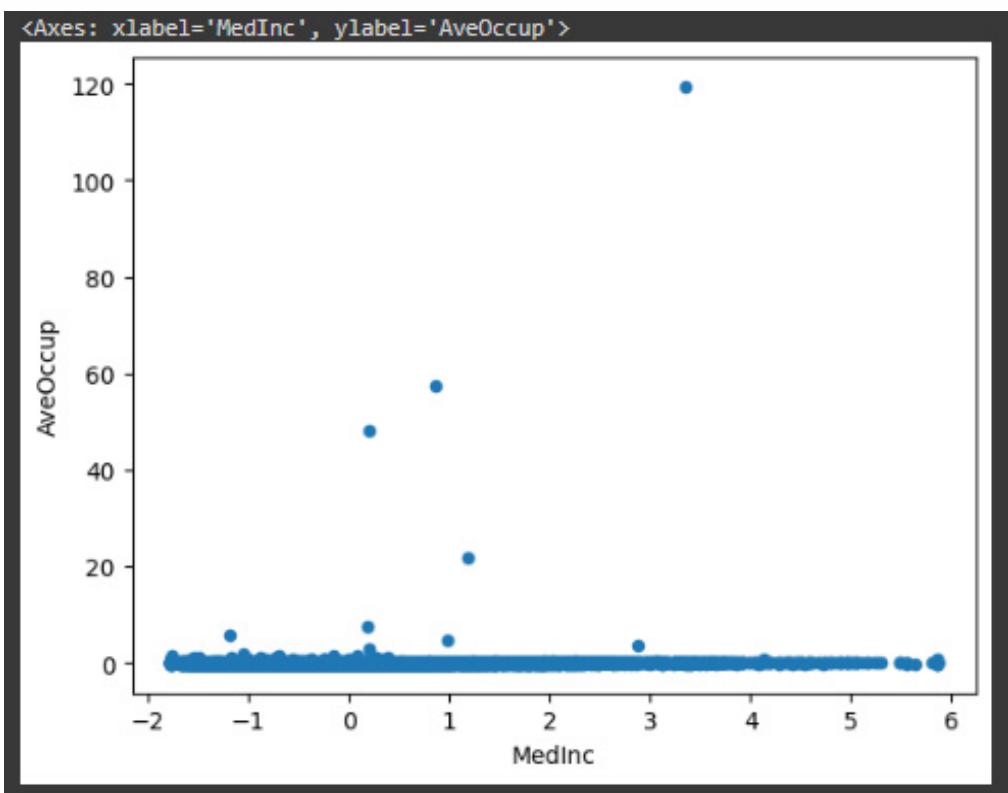
```
Média dos valores após a normalização:
MedInc      -1.652425e-17
AveOccup   -1.377021e-18
dtype: float64
```

```
# Imprime os valores do desvio padrão
# Nota-se que os valores obtidos são muito proximo de 1
print("Desvio padrão dos valores após a normalização: ")
print(df[['MedInc', 'AveOccup']].std())
```

```
Desvio padrão dos valores após a normalização:
MedInc      1.000024
AveOccup   1.000024
dtype: float64
```

```
# Distribuição dos dados antes da normalização
# Compare esse gráfico com o primeiro e o do min max scaling,
# Note como os valores dos eixos x e y não estão entre 0 e 1
# nem estão nos valores originais, porém, como visto anteriormente
# mantém média 0 e desvio padrão 1
df.plot.scatter('MedInc', 'AveOccup')
```

Figura 45 - Distribuição de dados após normalização - ZCore



Fonte: autoria própria.

Codificação de Variáveis Categóricas

Variáveis categóricas são aquelas que contêm valores discretos e não numéricos. Para usar essas variáveis em algoritmos de *machine learning*, precisamos convertê-las em uma forma numérica.

Métodos de Codificação

- » **One-Hot Encoding:** Cria colunas binárias para cada categoria.
- » **Label Encoding:** Atribui um número inteiro único a cada categoria.

One-Hot Encoding

Cria colunas binárias para cada categoria.

Serão utilizados os mesmos dados da etapa de normalização, por isso é importante não pular as etapas anteriores.

```
# Vizualização dos dados  
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	ocean_proximity
0	2.344766	41.0	6.984127	1.023810	322.0	-0.049597	37.88	-122.23	4.526	NEAR OCEAN
1	2.332238	21.0	6.238137	0.971880	2401.0	-0.092512	37.86	-122.22	3.585	NEAR OCEAN
2	1.782699	52.0	8.288136	1.073446	496.0	-0.025843	37.85	-122.24	3.521	INLAND
3	0.932968	52.0	5.817352	1.073059	558.0	-0.050329	37.85	-122.25	3.413	<1H OCEAN
4	-0.012881	52.0	6.281853	1.081081	565.0	-0.085616	37.85	-122.25	3.422	<1H OCEAN

Como abordado no ebook, muitos os algoritmos necessitam que colunas categóricas possuam números ao invés de textos.

No trecho abaixo do código o objetivo é transformar os rótulos a coluna `ocean_proximity` em números. Relembrando, esses rótulos foram criados artificialmente no começo da etapa de pré procesamento com rótulos do formato **string** podendo assumir valores: `<1H OCEAN`, `INLAND` ou `NEAR OCEAN`. No entanto, em alguns casos reais esses rótulos já podem vir nativamente no dataset.

Abaixo será visto:

- » Criação de um one-hot encoder.
- » Aplicação do one-hot encoding à coluna `ocean_proximity`.
- » A agregação do DataFrame codificado ao original.
- » Remoção da coluna original codificada.
- » Exibição das primeiras linhas do DataFrame resultante.

O resultado final é um DataFrame onde a coluna categórica ocean_proximity foi substituída por novas colunas binárias, representando cada categoria possível.

```
# Cria um codificador one-hot
encoder = OneHotEncoder(sparse_output=False)

# Aplica o one-hot encoding à coluna 'ocean_proximity'
#   - fit_transform: Ajusta o encoder aos dados e realiza a transformação de uma só vez
encoded_features = encoder.fit_transform(df[['ocean_proximity']])

# Cria um novo DataFrame com as features codificadas
#   - get_feature_names_out: Obtém os nomes das novas colunas geradas

encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(['ocean_proximity']))

# Combina o DataFrame original com o DataFrame codificado
#   - drop: Remove a coluna original 'ocean_proximity'
df_onehot = df.join(encoded_df).drop(columns=['ocean_proximity'])

# Visualiza as primeiras linhas do DataFrame com as features codificadas
# Note as últimas 3 colunas e os valores que elas assumem (0 ou 1)
df_onehot.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_NEAR OCEAN
0	2.344766	41.0	6.984127	1.023810	322.0	-0.049597	37.88	-122.23	4.526	0.0	0.0	1.0
1	2.332238	21.0	6.238137	0.971880	2401.0	-0.092512	37.86	-122.22	3.585	0.0	0.0	1.0
2	1.782699	52.0	8.288136	1.073446	496.0	-0.025843	37.85	-122.24	3.521	0.0	1.0	0.0
3	0.932968	52.0	5.817352	1.073059	558.0	-0.050329	37.85	-122.25	3.413	1.0	0.0	0.0
4	-0.012881	52.0	6.281853	1.081081	565.0	-0.085616	37.85	-122.25	3.422	1.0	0.0	0.0

Label Encoding

De forma análoga ao visto com o **OneHotEncoder** aqui é utilizado o **LabelEncoder**, que atribui um número inteiro único a cada categoria.

Cada categoria da coluna ocean_proximity receberá um número correspondente. Esse número substituirá o rótulo da coluna ocean_proximity, retirando permitindo que os algoritmos de machine learning possam funcionar adequadamente.

```
# Cria um codificador LabelEncoder
encoder = LabelEncoder()

# Aplica o LabelEncoder à coluna 'ocean_proximity', substituindo os valores existentes
#   - fit_transform: Ajusta o encoder aos dados e realiza a transformação de uma só vez
df['ocean_proximity'] = encoder.fit_transform(df['ocean_proximity'])

# Visualiza as primeiras linhas do DataFrame com as features codificadas
# Note as últimas 3 colunas e os valores que elas assumem (0 a 2)
df.head()
```

continua

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	ocean_proximity
0	2.344766	41.0	6.984127	1.023810	322.0	-0.049597	37.88	-122.23	4.526	2
1	2.332238	21.0	6.238137	0.971880	2401.0	-0.092512	37.86	-122.22	3.585	2
2	1.782699	52.0	8.288136	1.073446	496.0	-0.025843	37.85	-122.24	3.521	1
3	0.932968	52.0	5.817352	1.073059	558.0	-0.050329	37.85	-122.25	3.413	0
4	-0.012881	52.0	6.281853	1.081081	565.0	-0.085616	37.85	-122.25	3.422	0

```
# Tabela das classes criadas
# Com base nela é possível decodificar a coluna ocean_proximity
encoder.classes_
array([0, 1, 2])
```

Exercício 1: Normalização

Normalize a coluna **AveRooms** utilizando o Min-Max Scaling.

```
# Exercício resolvido
# Normalizar a coluna AveRooms utilizando Min-Max Scaling
scaler = MinMaxScaler()
df['AveRooms'] = scaler.fit_transform(df[['AveRooms']])
print(df[['AveRooms']].head())
```

```
AveRooms
0  0.043512
1  0.038224
2  0.052756
3  0.035241
4  0.038534
```

Conjuntos de Treino, Validação e Teste

Objetivos

- » Compreender a importância de dividir os dados em conjuntos de treino, validação e teste.
- » Aprender a dividir um conjunto de dados utilizando bibliotecas populares do Python.
- » Implementar e avaliar modelos de machine learning utilizando esses conjuntos de dados.

1. Introdução

Dividir os dados em conjuntos de treino, validação e teste é uma prática essencial em *machine learning*. Essa divisão permite avaliar o desempenho do modelo de forma objetiva e evitar problemas como o *overfitting*.

Importância da Divisão dos Dados

- » **Treino:** Usado para ajustar os parâmetros do modelo.

- » **Validação:** Usado para ajustar os hiperparâmetros do modelo e selecionar o melhor modelo.
- » **Teste:** Usado para avaliar o desempenho final do modelo.

Exemplo de aplicação da divisão dos dados

Nesse exemplo utilizaremos o dataset do Titanic. O dataset do Titanic é um conjunto de dados bastante popular na comunidade de ciência de dados, especialmente para aqueles que estão iniciando seus estudos em machine learning. Ele contém informações sobre os passageiros do Titanic, incluindo:

- » **Informações demográficas:** Idade, sexo, classe social.
- » **Informações sobre a viagem:** Porto de embarque, número de familiares a bordo.
- » **Resultado:** Se o passageiro sobreviveu ou não ao naufrágio.

Nesse parte, o objetivo é dividir esse dataset em treino, validação e teste utilizando duas técnicas distintas Holdout e K-fold.

Importação dos dados

```
# Importa-se as bibliotecas
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Carregar p conjunto de dados do Titanic para trabalhar como exemplo
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv'
df = pd.read_csv(url)

# Visualizar os primeiros dados
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Holdout

Aqui será apresentado a divisão dos dados em Treino, Validação e Teste. E o treino de um modelo simples para acompanhar a importância dessa divisão.

Algumas etapas importantes são:

- » **Seleciona as features relevantes:** Idade (Age) e tarifa (fare), como tentativa para prever se o passageiro sobreviveu ou não.
- » **Remove dados faltantes:** Garante que o modelo seja treinado com dados completos.
- » **Divide os dados:** Cria conjuntos de treinamento (60%), validação (20%) e teste (20%) para avaliar o desempenho do modelo e evitar overfitting.
- » **Verifica as dimensões:** Confirma que os conjuntos foram divididos corretamente.

```
# Definir features e target

# Remove as linhas com valores faltantes (NaN) das colunas 'age', 'fare', 'survived' utilizando o método dropna()
df_model = df[['age', 'fare', 'survived']].dropna()
# As colunas 'age' e 'fare' serão utilizadas como features (variáveis explicativas) para o modelo.
X = df_model[['age', 'fare']]
# A coluna 'survived' será o target (variável a ser prevista), indicando se o passageiro sobreviveu ou não.
y = df_model['survived']

# Dividir os dados em treino (60%), validação (20%) e teste (20%)
# O random_state é importante para garantir a reproduzibilidade dos resultados, especialmente quando se deseja comparar diferentes modelos ou realizar experimentos.
# Essa primeira chamada divide os dados em 60% para treinamento e 40% para validação e teste.
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)

# Essa segunda chamada divide os 40% restantes em partes iguais para validação e teste.
# O parâmetro random_state=42 garante que a divisão seja sempre a mesma, permitindo reproduzibilidade dos resultados.
# X_train, X_val, X_test: Contêm as features (idade e tarifa) para os conjuntos de treinamento, validação e teste, respectivamente.
# y_train, y_val, y_test: Contêm os valores correspondentes da variável target (sobreviveu ou não) para os conjuntos de treinamento, validação e teste, respectivamente.
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Impressão do tamanhos dos conjuntos de dados
# Note que os tamanhos obedecem 60% para treino, 20% para validação e 20% para teste
print(f"Tamanho do conjunto de treino: {X_train.shape}")
print(f"Tamanho do conjunto de validação: {X_val.shape}")
print(f"Tamanho do conjunto de teste: {X_test.shape}")

Tamanho do conjunto de treino: (428, 2)
Tamanho do conjunto de validação: (143, 2)
Tamanho do conjunto de teste: (143, 2)
```

Treinando o Modelo

O código apresentado define uma função **treina_modelo** que encapsula o processo de treinamento e avaliação de um modelo de regressão logística.

A função **treina_modelo** recebe como entrada os dados de treinamento e validação, treina um modelo de regressão logística e avalia seu desempenho nos conjuntos de validação e teste, fornecendo como saída as respectivas acurárias.

Entradas:

- » **X_train**: Um DataFrame contendo as features (nesse caso, 'age' e 'fare') dos passageiros do conjunto de treinamento. Essas features serão utilizadas pelo modelo para aprender a prever se um passageiro sobreviveria ao naufrágio do Titanic.
- » **y_train**: Um Series contendo os valores da variável target (sobreviveu ou não) correspondentes aos dados de treinamento.
- » **X_val**: Um DataFrame contendo as features do conjunto de validação, utilizado para ajustar os hiperparâmetros do modelo e avaliar seu desempenho em dados não vistos durante o treinamento.
- » **y_val**: Um Series contendo os valores da variável target correspondentes aos dados de validação.

Saída:

A função não retorna nenhum valor explicitamente. Em vez disso, ela imprime na tela duas métricas de desempenho:

- » **Acurácia no conjunto de validação**: Indica a proporção de exemplos do conjunto de validação que foram classificados corretamente pelo modelo.
- » **Acurácia no conjunto de teste**: Indica a proporção de exemplos do conjunto de teste que foram classificados corretamente pelo modelo. Essa métrica fornece uma estimativa mais confiável do desempenho do modelo em dados

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def treina_modelo(X_train, y_train, X_val, y_val):
    # Treinar o modelo no conjunto de treino
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    # Avaliar o modelo no conjunto de validação
    y_val_pred = model.predict(X_val)
    val_accuracy = accuracy_score(y_val, y_val_pred)
    print(f"Acurácia no conjunto de validação: {val_accuracy:.2f}")

    # Avaliar o modelo no conjunto de teste
    y_test_pred = model.predict(X_test)
```

continua

```
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Acurácia no conjunto de teste: {test_accuracy:.2f}")
```

```
treina_modelo(X_train, y_train, X_val, y_val)
Acurácia no conjunto de validação: 0.64
Acurácia no conjunto de teste: 0.61
```

Validação Cruzada: K-Fold Cross-Validation

Validação cruzada é vantajosa porque pode retirar o fator "sorte" e de ter realizado uma divisão vantajosa para o modelo.

No K-fold dividimos os dados de treino em **k** grupos e comparamos entre eles.

Abaixo é a aplicação da validação cruzada k-fold (com k=5) para avaliar o desempenho do modelo de regressão logística.

Entradas:

- » **X_train:** Um DataFrame contendo as features (nesse caso, 'age' e 'fare') do conjunto de treinamento completo.
- » **y_train:** Um Series contendo os valores da variável target (sobreviveu ou não) correspondentes aos dados de treinamento completo.
- » **kf:** Um objeto KFold que define a estratégia de divisão dos dados em 5 folds.

Saída:

A cada iteração do loop, o código:

- » **Divide os dados:** Separa os dados de treinamento em um conjunto de treinamento e um conjunto de validação, de acordo com os índices definidos pelo objeto KFold.
- » **Treina o modelo:** Chama a função treina_modelo com os conjuntos de treinamento e validação atuais, treinando o modelo de regressão logística.
- » **Avalia o modelo:** A função treina_modelo calcula e imprime a acurácia do modelo nos conjuntos de validação e teste.

O objetivo principal da validação cruzada k-fold é obter uma estimativa mais robusta e confiável do desempenho do modelo, reduzindo o viés que poderia ocorrer devido a uma única divisão aleatória dos dados em treinamento e teste. Ao repetir o processo de treinamento e validação 5 vezes, utilizando diferentes combinações dos dados como conjunto de treinamento e validação, obtém-se diferentes métricas de desempenho, o que proporciona uma visão mais precisa do desempenho geral do modelo.

```

# Importa a classe KFold da biblioteca sklearn
from sklearn.model_selection import KFold

# Define o número de 5 folds para a validação cruzada
kf = KFold(n_splits=5)

# Itera sobre cada fold
for train_index, val_index in kf.split(X_train):
    # Divide os dados em conjuntos de treino e validação para o fold atual
    X_train_fold, X_val_fold = X.iloc[train_index], X.iloc[val_index]
    y_train_fold, y_val_fold = y.iloc[train_index], y.iloc[val_index]

    # Aqui chama-se a função de treinamento anteriormente definida
    # verifica-se as métricas a cada fold
    treina_modelo(X_train_fold, y_train_fold, X_val_fold, y_val_fold)

```

```

Acurácia no conjunto de validação: 0.58
Acurácia no conjunto de teste: 0.61
Acurácia no conjunto de validação: 0.64
Acurácia no conjunto de teste: 0.61
Acurácia no conjunto de validação: 0.66
Acurácia no conjunto de teste: 0.58
Acurácia no conjunto de validação: 0.64
Acurácia no conjunto de teste: 0.62
Acurácia no conjunto de validação: 0.56
Acurácia no conjunto de teste: 0.59

```

Exercício 1

Divida o conjunto de dados do titanic em conjuntos de treino (70%), validação (15%) e teste (15%).

OBS. Para questões de comparação, utilize o **random_state** igual a 42

```

# Exercício resolvido

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Carregar um conjunto de dados de exemplo
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv'
df = pd.read_csv(url)

# Visualizar os primeiros dados
df.head()

# Dividir os dados
# O parâmetro random_state=42 garante que a divisão seja sempre a mesma, permitindo
# reproduzibilidade dos resultados.
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_
state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_
state=42)

print(f"Tamanho do conjunto de treino: {X_train.shape}")

```

continua

```
print(f"Tamanho do conjunto de validação: {X_val.shape}")
print(f"Tamanho do conjunto de teste: {X_test.shape}")
```

```
Tamanho do conjunto de treino: (499, 2)
Tamanho do conjunto de validação: (107, 2)
Tamanho do conjunto de teste: (108, 2)
```

Dados balanceados e desbalanceados

Objetivos

- » Compreender a importância de dados balanceados em modelos de machine learning.
- » Identificar técnicas para lidar com dados desbalanceados.
- » Implementar essas técnicas utilizando bibliotecas populares do Python.

1. Introdução

Em muitos problemas de *machine learning*, especialmente aqueles envolvendo classificação, os dados podem estar desbalanceados, significando que algumas classes têm muito mais exemplos do que outras. Isso pode afetar o desempenho do modelo, levando a previsões tendenciosas.

2. Explicação Teórica

Dados Balanceados vs. Desbalanceados

- » **Dados Balanceados:** Quando as classes no conjunto de dados têm aproximadamente o mesmo número de exemplos.
- » **Dados Desbalanceados:** Quando uma ou mais classes têm significativamente mais exemplos do que outras.

3. Exemplos Práticos

A exemplo do que foi utilizado anteriormente, o dataset do Titanic agora será explorado as questões de desbalanceamento.

Teremos algumas etapas a serem seguidas:

- » **Seleção das colunas relevantes para a análise de sobrevivência:** 'pclass' (classe social), 'age' (idade), 'sibsp' (número de irmãos/cônjuges a bordo) e 'fare' (tarifa paga).
- » **Remoção das linhas com valores ausentes (NaN)** para evitar erros no processamento posterior.
- » **Divisão em features e target:** Separa o DataFrame em features (X) contendo

as colunas selecionadas e a target (y) contendo a informação de sobrevivência (0 - não sobreviveu, 1 - sobreviveu).

- » **Divisão treino e teste:** Divide os dados em conjuntos de treinamento (X_train, y_train) e teste (X_test, y_test) utilizando a função train_test_split com uma proporção de 70% para treinamento e 30% para teste. Define um random_state para garantir reproduzibilidade dos resultados.
- » **Visualização da distribuição da classe target:** Imprime a contagem de valores da variável target no conjunto de treinamento, permitindo verificar a distribuição entre as classes "sobreviveu" e "não sobreviveu" (valor idealmente balanceado para um modelo de classificação).

Carregando Bibliotecas e Dados

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# Carregar um conjunto de dados de exemplo
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv'
df = pd.read_csv(url)

# Filtrar colunas relevantes e remover valores ausentes
df = df[['survived', 'pclass', 'age', 'sibsp', 'fare']].dropna()

# Dividir features e target
X = df.drop(columns=['survived'])
y = df['survived']

# Dividir os dados em treino (70%) e teste (30%)
# O parâmetro random_state=42 garante que a divisão seja sempre a mesma, permitindo
# reproduzibilidade dos resultados.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
state=42)

# Visualizar distribuição da classe target
print(y_train.value_counts())

survived
0    298
1    201
Name: count, dtype: int64
```

Treina o modelo

Essa é a mesma função apresentada no item de Conjuntos de Treino, Validação e Teste. Caso tenha dúvida sobre seu funcionamento, volte na célula <https://colab.research.google.com/drive/1s0wK-XDQOdUubutjqhJVQL-6q6jKAfy9#scrollTo=urlttD4B61SW>

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def treina_modelo(X_train, y_train, X_val, y_val):
    # Treinar o modelo no conjunto de treino
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    # Avaliar o modelo no conjunto de validação
    y_val_pred = model.predict(X_val)
    val_accuracy = accuracy_score(y_val, y_val_pred)
    print(f"Acurácia no conjunto de validação: {val_accuracy:.2f}")

    # Avaliar o modelo no conjunto de teste
    y_test_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    print(f"Acurácia no conjunto de teste: {test_accuracy:.2f}")

```

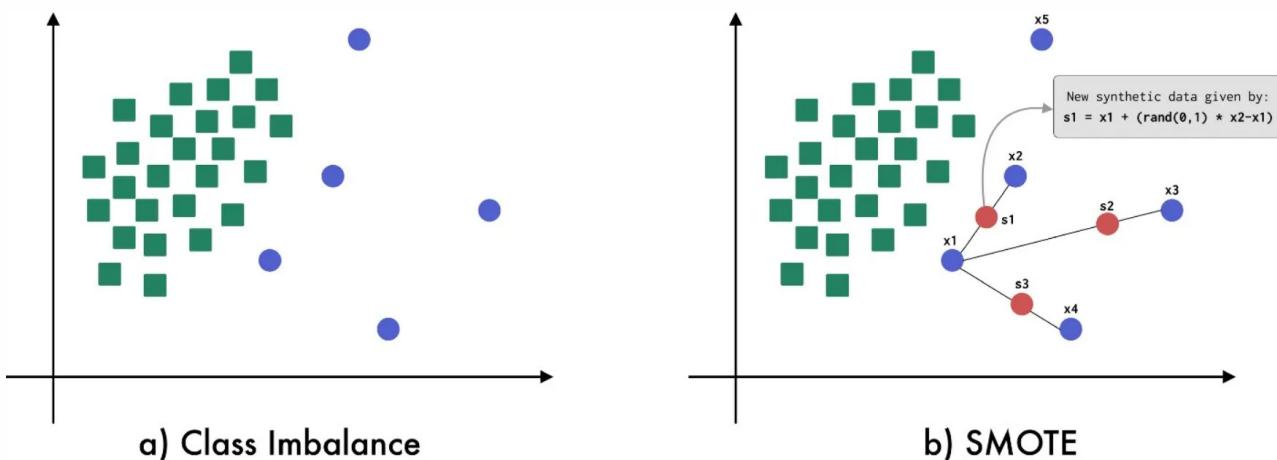
Oversampling com SMOTE

SMOTE cria dados artificiais com base nos **K** vizinhos mais próximos.

Algoritmo simplificado:

1. Seleciona a classe minoritária
2. Utiliza o KNN na classe minoritária com algum valor de **K**
3. Desenha-se linhas entre uma amostra da classe minoritária e seus **K** vizinhos
4. Escolhe um valor aleatório sobre essa linha

Figura 46 - Exemplo da utilização do SMOTE



Fonte: autoria própria.

A seguir é apresentado a implementação do oversampling com o SMOTE. Como entrada tem-se os dados do titanic anteriormente separados. Como saída tem-se ambas classes com a mesma quantidade (da classe com maior número de elementos).

Compare a quantidade de amostras **y_train_smote** apresentado abaixo com o **y_train** apresentado anteriormente. Note que a classe 0 é a que possui mais exemplos (298), enquanto que a classe 1 que possuia 201 antes, agora possui 298.

```

# cria o modelo smote através da classe SMOTE.
# novamente o random_state é utilizado para que os dados sejam replicáveis
smote = SMOTE(random_state=42)
# Cria novos dados com base no X_train e y_train utilizando o SMOTE.
# Esses novos dados são guardados nas variáveis X_train_smote e y_train_smote
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Visualizar distribuição da classe target após SMOTE
print(y_train_smote.value_counts())

survived
0    298
1    298
Name: count, dtype: int64

```

Compare as métricas de acurácia antes e depois do balanceamento.

Nota-se uma melhor acurácia dos dados quando há o balanceamento de classes.

```

# Comparaçao da acurácia com o balancelamento

print("Modelo treinado ignorando o desbalanceamento")
treina_modelo(X_train, y_train, X_test, y_test)

print("\n*****\n")
print("Modelo treinado após o SMOTE")
treina_modelo(X_train_smote, y_train_smote, X_test, y_test)

Modelo treinado ignorando o desbalanceamento
Acurácia no conjunto de validação: 0.71
Acurácia no conjunto de teste: 0.71

*****\n

Modelo treinado após o SMOTE
Acurácia no conjunto de validação: 0.73
Acurácia no conjunto de teste: 0.73

```

Undersampling

Diminui o dataset, retirando aleatoriamente da classe majoritária alguns exemplos. Nessa parte será apresentado o **RandomUnderSampler** que remove aleatoriamente dados da classe majoritária (classe 0).

Apesar de trazer o mesmo efeito de balanceamento entre classes, no undersampling reduz-se a quantidade de amostras. Nesse caso, serão eliminados aleatoriamente dados dos sobreviventes (rotulados com 1 na survived).

```

# cria o modelo undersampler através da classe RandomUnderSampler.
# novamente o random_state é utilizado para que os dados sejam replicáveis
undersampler = RandomUnderSampler(random_state=0)
# Eliminam-se dados com base no X_train e y_train utilizando o RandomUnderSampler.
# Os novos dados são guardados nas variáveis X_train_under e y_train_under
X_train_under, y_train_under = undersampler.fit_resample(X_train, y_train)

# Visualizar distribuição da classe target após undersampling
print(y_train_under.value_counts())

```

continua

```
survived
0    201
1    201
Name: count, dtype: int64
```

Exercício 1: Aplicar *Undersampling* e Treinar Modelo

Utilize a função **treina_modelo** (criada anteriormente) com os dados X_train, y_train, X_test e y_test também criados anteriormente.

Utilize a função **treina_modelo** (criada anteriormente) com os dados X_train_under, y_train_under, X_test e y_test também criados anteriormente.

Compare os resultados de acurácia do modelo treinado original (dados desbalanceados) e com dados balanceados pela técnica de undersampling.

```
# Exercício Resolvido
# Comparaçao da acurácia com o balancelamento

print("Modelo treinado ignorando o desbalancemanento")
treina_modelo(X_train, y_train, X_test, y_test)

print("\n*****\n")
print("Modelo treinado apôs o Undersampling")
treina_modelo(X_train_under, y_train_under, X_test, y_test)
```

```
Modelo treinado ignorando o desbalancemanento
Acurácia no conjunto de validação: 0.71
Acurácia no conjunto de teste: 0.71
```

```
*****
```

```
Modelo treinado apôs o Undersampling
Acurácia no conjunto de validação: 0.73
Acurácia no conjunto de teste: 0.73
```

Underfitting e *Overfitting*

Objetivos

- » Compreender os conceitos de *underfitting* e *overfitting* em modelos de *machine learning*.
- » Aprender a identificar *underfitting* e *overfitting* através de métricas de desempenho e visualizações.
- » Implementar técnicas para prevenir *underfitting* e *overfitting* em modelos utilizando Python.

1. Introdução

Em machine learning, underfitting e overfitting são problemas comuns que afetam a capacidade de generalização dos modelos. Underfitting ocorre quando um modelo é muito simples para capturar a complexidade dos dados, enquanto overfitting ocorre quando um modelo é muito complexo e capture o ruído dos dados de treino. Este notebook abordará esses conceitos e apresentará técnicas para mitigá-los.

Clique duas vezes (ou pressione "Enter") para editar

3. Exemplos Práticos

Exemplo 1

Nesse primeiro exemplo servirá para ilustrar modelos com underfitting, overfitting e um modelo ideal para o mesmo conjunto de dados.

O modelo real é definido pela função `true_function` e é uma função quadrática x^2 .

Utiliza-se essa função para gerar dados sintéticos e aleatórios ao redor desse modelo ideal, representado pelos pontos em vermelho e as variáveis `x` e `y`. `x` e `y` podem ser considerados o conjunto de dados de treino.

Criam-se os dados de teste `x_fit` e `y_true`

Carregando Bibliotecas e Dados

```
# Importação das bibliotecas que serão utilizadas
import matplotlib.pyplot as plt
import numpy as np

# Define o modelo ideal como sendo uma função quadrática
def true_function(x):
    return x**2

# Gerar dados sintéticos
# Define-se um gerador aleatório que seja reproduzível, é semelhante ao parâmetro
random_state visto anteriormente
np.random.seed(22)

# Geram-se 10 valores aleatórios entre -2 e 2
x = np.sort(np.random.rand(10) * 4 - 2) # random values between -2 and 2
# Para cada valor gerado anteriormente, aplica a função verdade `true_function(x)`
para obter o valor de y
# adiciona-se um ruído com `np.random.randn(10)` para que o valor não caia sempre
sobre o próprio modelo
y = true_function(x) + np.random.randn(10)
# Criam-se os dados de teste
# x_fit são 100 valores igualmente espaçados entre -2 e 2
x_fit = np.linspace(-2, 2, 100)
# y_true é o resultado do modelo ideal para os dados de x_fit
y_true = true_function(x_fit)
```

Underfitting

Para exemplificar o underfitting é utilizado o modelo de regressão linear **model_underfit**.

Os dados de treino **x** e **y** servem para treinar o modelo linear.

Com o modelo treinado testa-se seu desempenho passando os valores de teste **x_fit**, obtendo as previsões e guardando em **y_underfit**.

Esses valores de **y_underfit** são utilizados em conjunto com **x_fit** para obter a curva do modelo ajustado da figura a esquerda.

```
from sklearn.linear_model import LinearRegression

# Cria o modelo de regressão linear
model_underfit = LinearRegression()
# Treina o modelo com dados de treino
model_underfit.fit(x.reshape(-1, 1), y)
# Obtem-se os resultados do modelo através do teste dos valores de x_fit
y_underfit = model_underfit.predict(x_fit.reshape(-1, 1))
```

Overfitting

Para exemplificar o overfitting é utilizado o modelo de regressão polinomial de grau 4 definido por **model_overfit**. O modelo de regressão polinomial é um modelo mais complexo que o de regressão linear utilizado no underfitting.

O pipeline consiste em dois passos:

- » Passo 1 (**PolynomialFeatures(degree)**): Primeiro, o pipeline aplica a transformação **PolyomialFeatures** com o grau definido anteriormente (4). Essa transformação gera novas features polinomiais a partir da feature original x.
- » Passo 2 (**LinearRegression()**): Em seguida, o pipeline aplica um modelo de regressão linear para aprender a relação entre as features polinomiais geradas na etapa anterior e a variável target y.

Os dados de treino **x** e **y** servem para treinar o modelo.

Com o modelo treinado testa-se seu desempenho passando os valores de teste **x_fit**, obtendo as previsões e guardando em **y_overfit**.

Esses valores de **y_overfit** são utilizados em conjunto com **x_fit** para obter a curva do modelo ajustado da figura a direita.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Define o grau do modelo polinomial
degree = 4

# Cria o modelo de regressão polinomial
model_overfit = make_pipeline(PolynomialFeatures(degree), LinearRegression())
# Treina o modelo com dados de treino
```

continua

```
model_overfit.fit(x.reshape(-1, 1), y)
# Obtem-se os resultados do modelo através do teste dos valores de x_fit
y_overfit = model_overfit.predict(x_fit.reshape(-1, 1))
```

Comparação

No código abaixo são gerados 2 gráficos de comparação, um para o underfitting e outro para o overfitting.

Em cada gráfico são plotados:

- » Os dados de teste **x** e **y** representados por pontos vermelhos.
- » A curva fruto do modelo real **true_function** fruto do resultado da aplicação do conjunto de teste **x_fit** que obteve **y_true**. A curva é representada pela linha pontilhada.
- » A curva fruto do modelo (underfitting ou overfitting) fruto do resultado da aplicação do conjunto de teste **x_fit** no modelo **model_overfit** ou **model_underfit** que obteveram **y_overfit** e **y_underfit** respectivamente. Essas curvas são representadas pelas linhas cheias.

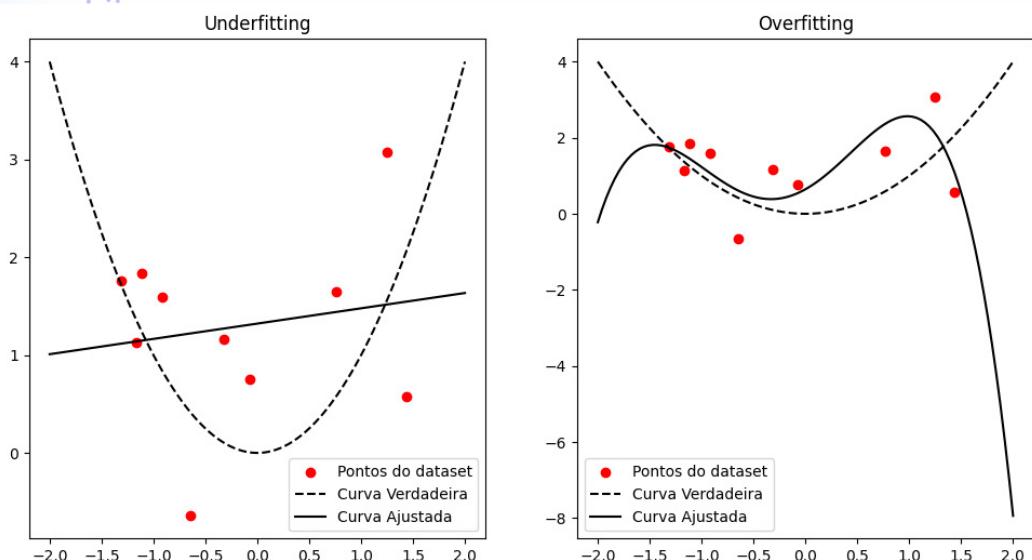
```
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# Underfitting plot
axs[0].scatter(x, y, color='red', label='Pontos do dataset')
axs[0].plot(x_fit, y_true, 'k--', label='Curva Verdadeira')
axs[0].plot(x_fit, y_underfit, 'k-', label='Curva Ajustada')
axs[0].set_title('Underfitting')
axs[0].legend()

# Overfitting plot
axs[1].scatter(x, y, color='red', label='Pontos do dataset')
axs[1].plot(x_fit, y_true, 'k--', label='Curva Verdadeira')
axs[1].plot(x_fit, y_overfit, 'k-', label='Curva Ajustada')
axs[1].set_title('Overfitting')
axs[1].legend()

plt.show()
```

Figura 47 - Underfitting vs Overfitting



Fonte: autoria própria.

Exemplo 2

Mais um exemplo agora para observar o impacto da escolha do parâmetro do modelo (degree) nas métricas de performance.

O código gera dados sintéticos com relação não linear, divide os dados em treino e teste, ordena os dados de teste para plotagem e define duas funções: uma para plotar os resultados do modelo e outra para calcular e imprimir métricas de desempenho (MSE e R²). Este código base pode ser utilizado para treinar diferentes modelos de regressão com features polinomiais e avaliar o seu desempenho.

O código pode ser dividido em algumas partes:

Geração de Dados Sintéticos:

A exemplo do que foi abordado no exemplo anterior, essa fase visa criar alguns dados fictícios.

- » **np.random.seed(0)**: Define uma semente para geração de números aleatórios, garantindo reproduzibilidade dos resultados.
- » **X = np.sort(5 * np.random.rand(80, 1), axis=0)**: Gera um array X com 80 linhas e 1 coluna, contendo valores aleatórios entre 0 e 5, ordenados em ordem crescente.
- » **y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])**: Gera a variável target y baseada no seno dos valores de X com adição de um ruído gaussiano (normal) de média 0 e desvio padrão 0.1. A função ravel() transforma o array y em unidimensional.

Divisão de Dados:

Como visto anteriormente, a divisão dos dados em treino e teste é uma fase importante para a avaliação dos modelos.

- » **train_test_split**: Divide X e y em conjuntos de treinamento (X_train, y_train) e teste (X_test, y_test) com uma proporção de 80% para treinamento e 20% para teste. random_state=0 garante reproduzibilidade.

Ordenação para Plotagem:

Uma fase para auxiliar na visualização dos dados.

- » **z = sorted(zip(X_test,y_test))**: Cria uma lista z ordenando os pares (valor de X, valor de y) do conjunto de teste.
- » **X_test,y_test = zip(*z)**: Desempacota a lista ordenada z separando novamente as listas de X_test e y_test. Isso garante que os pontos de dados no teste sejam plotados na mesma ordem da previsão do modelo.

Função de Plotagem:

Para evitar a repetição de códigos, foi definida uma função específica para a plotagem dos gráficos.

plot_results: Função que recebe os dados de treinamento e teste, o modelo, features polinomiais (se houver) e um título, e gera um gráfico:

- » Plota os dados de treinamento (azul) e teste (vermelho) como pontos dispersos.
- » Plota a linha de previsão do modelo (verde) utilizando os dados de teste e as features polinomiais (se houver).
- » Adiciona título e legenda ao gráfico.
- » Exibe o gráfico utilizando plt.show().

Função de Métricas:

É definida uma função para obtenção das métricas cujo objetivo é análogo à função de plotagem.

print_metrics: Função que recebe os valores verdadeiros (y_true), os valores previstos (y_pred) e o nome do modelo, e calcula e imprime as métricas:

- » **mse**: Calcula o erro médio quadrático (Mean Squared Error) entre os valores verdadeiros e previstos.
- » **r2**: Calcula o R-quadrado (R^2), que indica a proporção da variância explicada pelo modelo.

- » Imprime o nome do modelo seguido das métricas MSE e R² formatadas com 4 casas decimais.

Carregando Bibliotecas e Dados

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score

# Gerar dados sintéticos
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0]) # Adiciona ruído

# Dividir os dados em conjunto de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Ordena os dados para facilitar a plotagem
z = sorted(zip(X_test,y_test))
X_test,y_test = zip(*z)

# Função para plotar os resultados
def plot_results(X_train, y_train, X_test, y_test, model, X_poly, title):
    plt.scatter(X_train, y_train, color='blue', label='Dados de Treinamento')
    plt.scatter(X_test, y_test, color='red', label='Dados de Teste')
    plt.plot(X_test, model.predict(X_poly), color='green', label='Modelo')
    plt.title(title)
    plt.legend()
    plt.show()

# Função para calcular e imprimir as métricas
def print_metrics(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} - MSE: {mse:.4f}, R^2: {r2:.4f}")

```

Underfitting: Modelo Linear Simples

O código apresentado simula um caso de underfitting utilizando um modelo de regressão linear com features polinomiais de grau 1. Vamos analisar passo a passo:

Criando Features Polinomiais para Underfitting:

- » **poly_features_underfit = PolynomialFeatures(degree=1)**: Cria uma instância da classe PolynomialFeatures definindo o grau como 1.
- » **X_poly_train_underfit = poly_features_underfit.fit_transform(X_train)**: Aplica o transformador poly_features_underfit aos dados de treinamento X_train.

A função fit_transform encaixa o transformador nos dados de treinamento (calcula parâmetros internos, se necessário) e então aplica a transformação.

- » **X_poly_test_underfit = poly_features_underfit.transform(X_test):** Aplica o transformador poly_features_underfit (já encaixado nos dados de treinamento) aos dados de teste X_test.

Treinando o Modelo de Underfitting:

- » **poly_reg_underfit = LinearRegression():** Cria um modelo de regressão linear.
- » **poly_reg_underfit.fit(X_poly_train_underfit, y_train):** Treina o modelo poly_reg_underfit utilizando as features polinomiais transformadas do treino (X_poly_train_underfit) e a variável target y_train.

Plotando os Resultados e Imprimindo Métricas:

plot_results:

Função definida anteriormente para plotar os resultados do modelo. Ela recebe os dados de treinamento e teste, o modelo treinado (poly_reg_underfit), as features polinomiais transformadas (X_poly_test_underfit) e um título descritivo ("Underfitting: Modelo Polinomial de Grau 1").

A função irá plotar os dados de treinamento e teste, e a linha de previsão do modelo utilizando as features polinomiais transformadas do teste (X_poly_test_underfit).

O título indica que este é um caso de underfitting devido ao uso de um polinômio de grau 1.

print_metrics:

Função definida anteriormente para calcular e imprimir métricas de desempenho (MSE e R2).

Ela é chamada duas vezes:

1. A primeira vez com os dados de treinamento (y_train) e a previsão do modelo no treinamento (poly_reg_underfit.predict(X_poly_train_underfit)) para avaliar o desempenho no conjunto de treino.
2. A segunda vez com os dados de teste (y_test) e a previsão do modelo no teste (poly_reg_underfit.predict(X_poly_test_underfit)) para avaliar o desempenho no conjunto de teste.

```

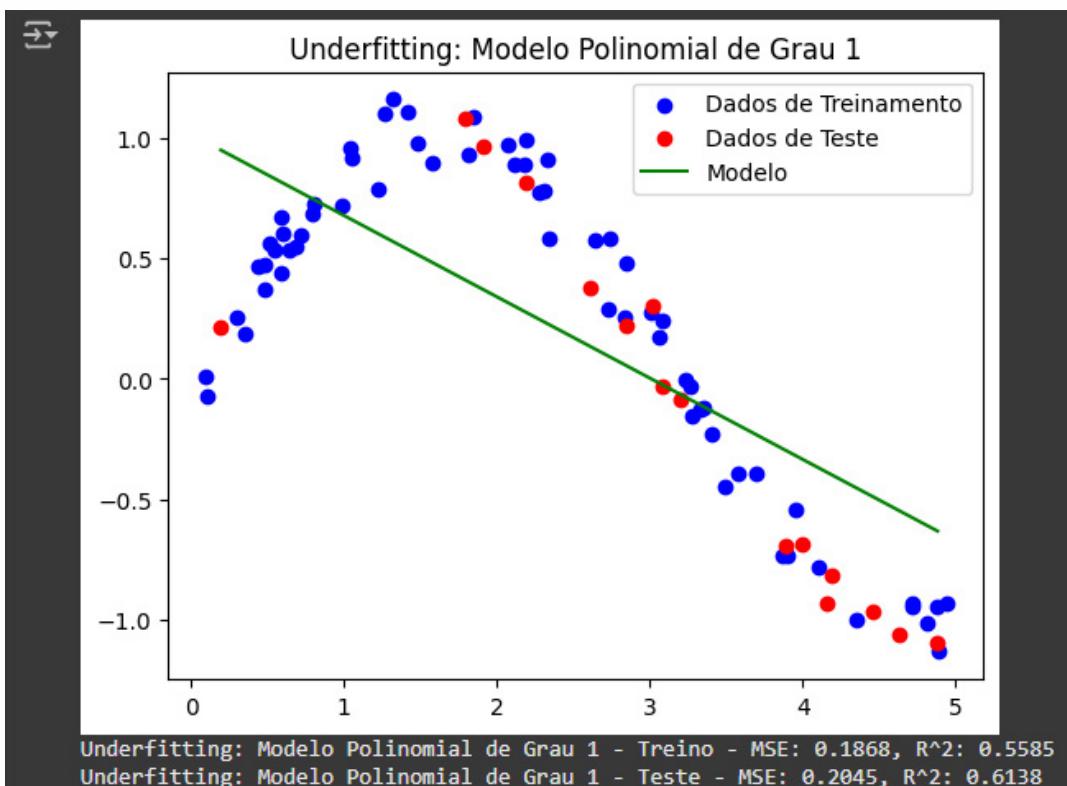
# Criando Features Polinomiais para Underfitting
poly_features_underfit = PolynomialFeatures(degree=1) # Grau 1 para underfitting
X_poly_train_underfit = poly_features_underfit.fit_transform(X_train)
X_poly_test_underfit = poly_features_underfit.transform(X_test)

# Treinando o Modelo de Underfitting
poly_reg_underfit = LinearRegression()
poly_reg_underfit.fit(X_poly_train_underfit, y_train)
plot_results(X_train, y_train, X_test, y_test, poly_reg_underfit, X_poly_test_underfit,
"Underfitting: Modelo Polinomial de Grau 1")

# Plotando os Resultados e Imprimindo Métricas
print_metrics(y_train, poly_reg_underfit.predict(X_poly_train_underfit), "Underfitting:
Modelo Polinomial de Grau 1 - Treino")
print_metrics(y_test, poly_reg_underfit.predict(X_poly_test_underfit), "Underfitting:
Modelo Polinomial de Grau 1 - Teste")

```

Figura 48 - Underfitting: Modelo Polinomial de Grau 1



Fonte: autoria própria.

Overfitting: Modelo Polinomial de Alto Grau

Semelhante ao apresentado anteriormente, com a única diferença que agora eleveu-se o grau do polinômio para 15 (degree=15) para obter mais features e elevar a complexidade do modelo.

```

# Criando Features Polinomiais para overfitting
poly_features_overfit = PolynomialFeatures(degree=15) # Grau alto para overfitting
X_poly_train_overfit = poly_features_overfit.fit_transform(X_train)

```

continua

```

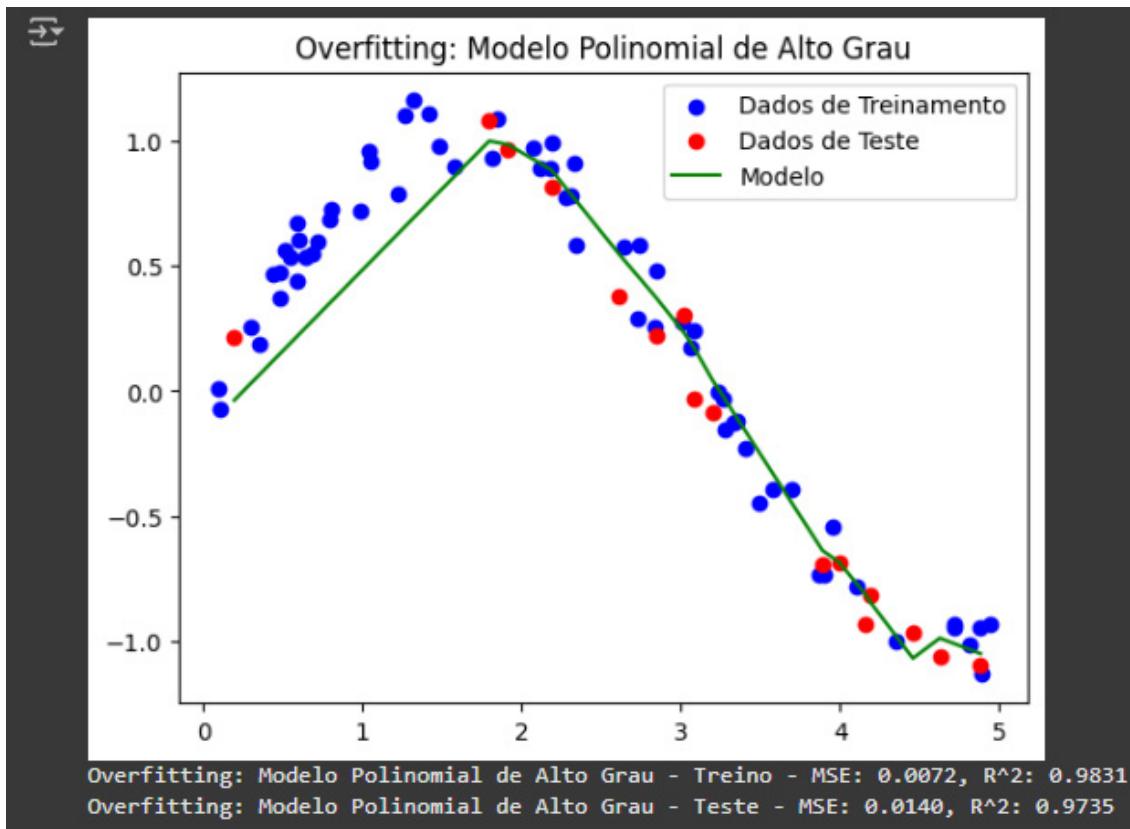
X_poly_test_overfit = poly_features_overfit.transform(X_test)

# Treinando o Modelo de overfitting
poly_reg_overfit = LinearRegression()
poly_reg_overfit.fit(X_poly_train_overfit, y_train)
plot_results(X_train, y_train, X_test, y_test, poly_reg_overfit, X_poly_test_overfit,
"Overfitting: Modelo Polinomial de Alto Grau")

# Plotando os Resultados e Imprimindo Métricas
print_metrics(y_train, poly_reg_overfit.predict(X_poly_train_overfit), "Overfitting:
Modelo Polinomial de Alto Grau - Treino")
print_metrics(y_test, poly_reg_overfit.predict(X_poly_test_overfit), "Overfitting: Modelo
Polinomial de Alto Grau - Teste")

```

Figura 49 - Overfitting: Modelo Polinomial de Alto Grau



Fonte: autoria própria.

Fitting Adequado: Modelo Polinomial de Grau 3

Semelhante ao apresentado anteriormente, com a única diferença que agora utilizou o grau do polinômio 3 (degree=3) para obter mais features e adequar a complexidade do modelo.

```

poly_features_fit = PolynomialFeatures(degree=3) # Grau intermediário para fitting
adequado
X_poly_train_fit = poly_features_fit.fit_transform(X_train)
X_poly_test_fit = poly_features_fit.transform(X_test)

poly_reg_fit = LinearRegression()
poly_reg_fit.fit(X_poly_train_fit, y_train)

```

continua

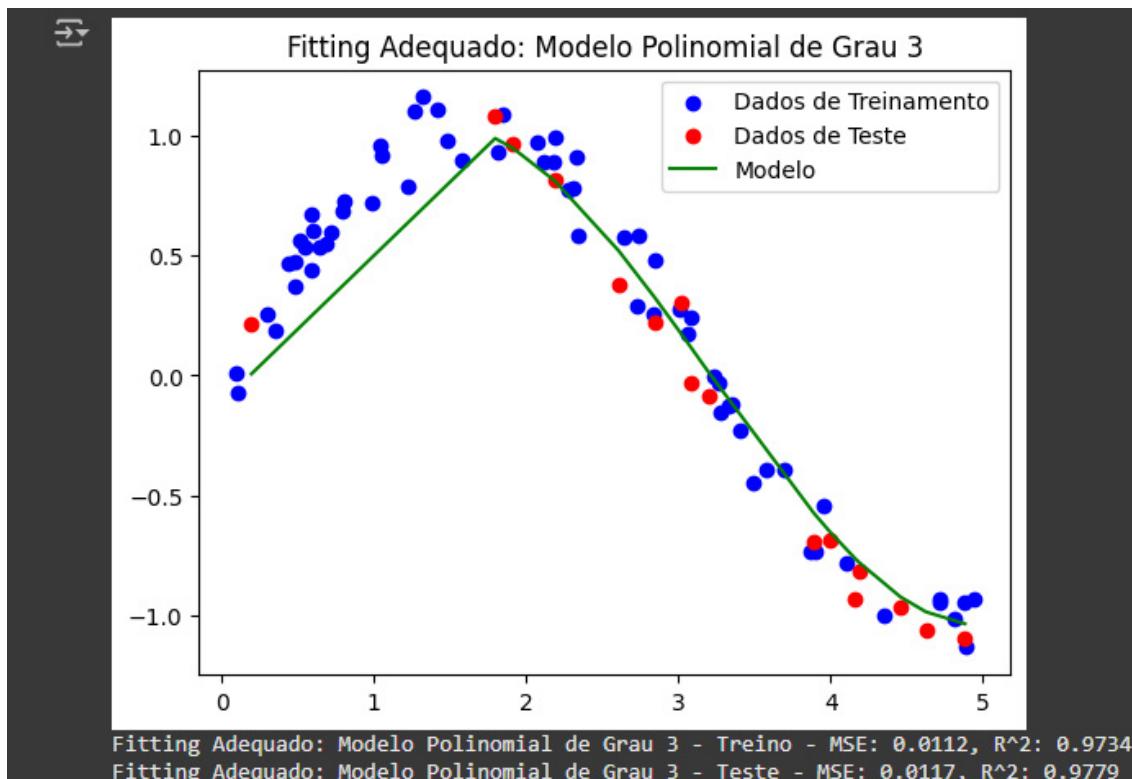
```

plot_results(X_train, y_train, X_test, y_test, poly_reg_fit, X_poly_test_fit, "Fitting Adequado: Modelo Polinomial de Grau 3")

print_metrics(y_train, poly_reg_fit.predict(X_poly_train_fit), "Fitting Adequado: Modelo Polinomial de Grau 3 - Treino")
print_metrics(y_test, poly_reg_fit.predict(X_poly_test_fit), "Fitting Adequado: Modelo Polinomial de Grau 3 - Teste")

```

Figura 50 - Fitting Adequado: Modelo Polinomial de Grau 3



Fonte: autoria própria.

O que observar?

Observe as metrícias MSE e R^2 entre os conjuntos de Treino e Teste em cada modelo.

- » **Underfitting:** valores elevado de MSE quando comparados com os modelos adequado e overfitting. Isso indica que os outros modelos representam melhor. Isso já era esperado uma vez que o modelo linear (grau 1) é o mais simples possível.
- » **Overfitting:** grande diferença no MSE quando comparado o dataset de Treino e Teste. Nesse caso, o modelo acaba se ajustando ao treino e é incapaz de generalizar para os novos dados (Teste).
- » **Adequado:** Não possui o melhor resultado de MSE no treino, mas obteve o melhor MSE no Teste. Também não há grandes diferenças de MSE entre o conjunto de Treino e Teste, indicando que o modelo conseguiu generalizar e manteve o mesmo acerto tanto para os dados que ele conhecia (Treino) como para os que ele nunca viu (Teste).

Referências Bibliográficas

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media.

Seaborn Documentation: <https://seaborn.pydata.org/>

Scikit-learn Documentation: <https://scikit-learn.org/stable/>

Pandas Documentation: <https://pandas.pydata.org/>



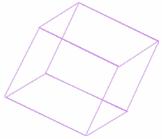
PARA RELEMBRAR...

- » Lembre-se da importância dos dados no ML, um dado de entrada ruim resultará em modelos ruins, por isso é muito importante estar atento aos dados e os efeitos de treinar o modelo sem entender corretamente o que está ocorrendo. Tenha sempre em mente:
- » Qualidade dos dados como base para modelos eficazes
- » Diferenças entre dados estruturados e não estruturados
- » Tipos de conjuntos de dados: supervisionados, não supervisionados e semi-supervisionados
- » Processos de limpeza e pré-processamento de dados
- » Seleção e engenharia de características
- » Divisão dos dados em conjuntos de treino, validação e teste
- » Técnica de *cross-validation*
- » Tratamento de dados desbalanceados
- » Problemas de subajuste (*underfitting*) e sobreajuste (*overfitting*)

SAIBA MAIS...

- » [Conjunto de dados](#)
- » [Subajuste e sobreajuste](#)
- » [Dados desbalanceados](#)
- » [Validação cruzada](#)

Unidade VIII **Régressão**



Unidade VIII - Regressão

A regressão, no contexto do ML, constitui uma técnica estatística fundamental para modelar a relação entre uma variável dependente contínua e uma ou mais variáveis independentes. Variável dependente contínua é aquela que estamos tentando prever ou explicar e que pode assumir qualquer valor numérico dentro de um intervalo. Ela é “dependente” porque seu valor é influenciado pelos valores de outras variáveis. Variáveis independentes são aquelas que utilizamos para prever o valor da variável dependente. Elas são chamadas de “independentes” porque, teoricamente, seus valores não são influenciados por outras variáveis no modelo. Um exemplo prático seria na previsão de consumo de energia em que a variável dependente contínua seria o consumo de energia em kWh (um valor numérico). As variáveis independentes poderiam ser: temperatura externa, número de pessoas na residência, hora do dia, etc. A variável dependente é o “efeito” que queremos explicar. As variáveis independentes são as “causas” que influenciam o efeito.

Essa metodologia, amplamente utilizada em diversas áreas do conhecimento, permite realizar previsões quantitativas com base em padrões identificados nos dados. Ao ajustar um modelo de regressão a um conjunto de dados, os cientistas de dados podem estimar o valor da variável dependente para novos conjuntos de dados, o que é de grande valor para tomada de decisões em diversos setores, como economia, finanças, indústria e marketing.

A escolha do modelo de regressão mais adequado depende das características dos dados e do problema a ser resolvido. Existem diversos tipos de regressão, cada um com suas particularidades e aplicações. A regressão linear, por exemplo, é a mais simples e amplamente utilizada, modelando a relação entre as variáveis através de uma equação linear. Já a regressão polinomial permite modelar relações não lineares, enquanto a regressão logística é utilizada para problemas de classificação, onde a variável dependente é binária.

A importância da regressão no ML reside na sua capacidade de extrair informações valiosas a partir de dados complexos. Ao identificar as relações entre as variáveis, os modelos de regressão podem ser utilizados para entender os fatores que influenciam um determinado fenômeno, otimizar processos, tomar decisões mais precisas e gerar novas hipóteses para pesquisa. Além disso, a regressão é uma técnica fundamental para outras técnicas de ML, como a árvore de decisão e a floresta aleatória.

Enquanto a regressão foca na predição de variáveis contínuas, como preços, temperaturas ou índices, a classificação é empregada para atribuir instâncias a categorias discretas. Por exemplo, determinar se um e-mail é spam ou não é um problema de classifi-

cação binária. Já prever a temperatura máxima para o próximo dia é um problema típico de regressão. A distinção entre esses dois tipos de problemas é crucial, pois os métodos e métricas de avaliação são distintos e a escolha do modelo adequado depende dessa compreensão.

8.1 Métricas de Avaliação

Para avaliar a performance de modelos de regressão, várias métricas são utilizadas:

8.1.1 Coeficiente de Determinação R²

O coeficiente de determinação mede a proporção da variabilidade da variável dependente que é explicada pelas variáveis independentes no modelo. Os valores obtidos dessa métrica variam entre 0 e 1. Um valor de 1 indica que o modelo explica toda a variância dos dados, enquanto um valor de 0 indica que o modelo não explica nenhuma variância. O Coeficiente de Determinação, R², é calculado como:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

\bar{y} é a média dos valores reais,

$\sum_{i=1}^n (y_i - \hat{y}_i)^2$ é a Soma dos Quadrados dos Resíduos (SSR),

$\sum_{i=1}^n (y_i - \bar{y})^2$ é a Soma Total dos Quadrados (SST).

O valor de R² varia entre 0 e 1:

$R^2 = 0$ indica que o modelo não explica nenhuma variação nos dados.

$R^2 = 1$ indica que o modelo explica toda a variação nos dados.

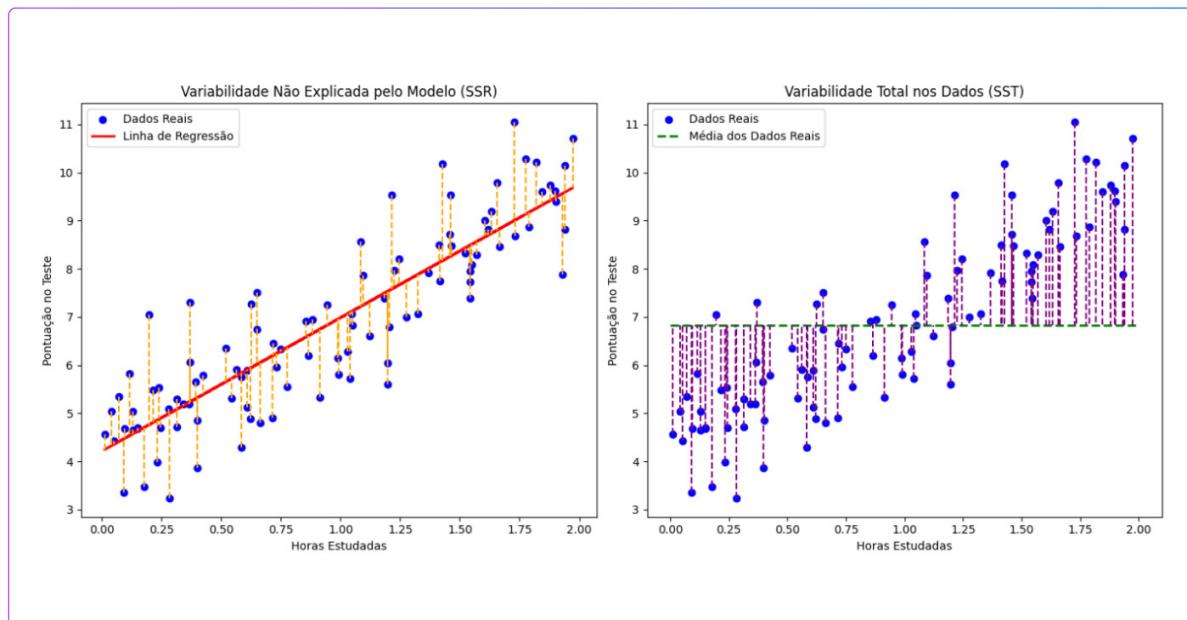
De forma facilitar a compreensão, a figura abaixo ilustra a aplicação das parcelas SSR e SST.

Variabilidade Não Explicada (SSR): As linhas pontilhadas laranja representam os erros do modelo, ou seja, a diferença entre o valor real (ponto em azul) e sua projeção ao modelo (linha vermelha). Quanto menores forem essas linhas, melhor o modelo se ajusta aos dados.

Variabilidade Total (SST): A linha horizontal verde representa a média dos dados. A distância entre os pontos e a linha representa a variabilidade total dos dados. O primeiro gráfico mostra a variabilidade que o modelo não consegue explicar.

Quanto menor a distância entre os pontos e a linha de regressão, melhor o modelo se ajusta aos dados. O segundo gráfico mostra a variabilidade total dos dados. Comparando os dois gráficos, pode-se ter uma ideia de quanto da variabilidade total está sendo explicada pelo modelo.

Figura 51 - Exemplo R²



Fonte: autoria própria e disponível no Google Colab.

8.1.2 Erro Quadrático Médio (MSE)

O Erro Quadrático Médio, *Mean Squared Error* (MSE) é uma métrica utilizada para avaliar a performance de um modelo de regressão. Ele calcula a média dos quadrados das diferenças entre os valores observados (reais) e os valores preditos pelo modelo. O MSE é definido como:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

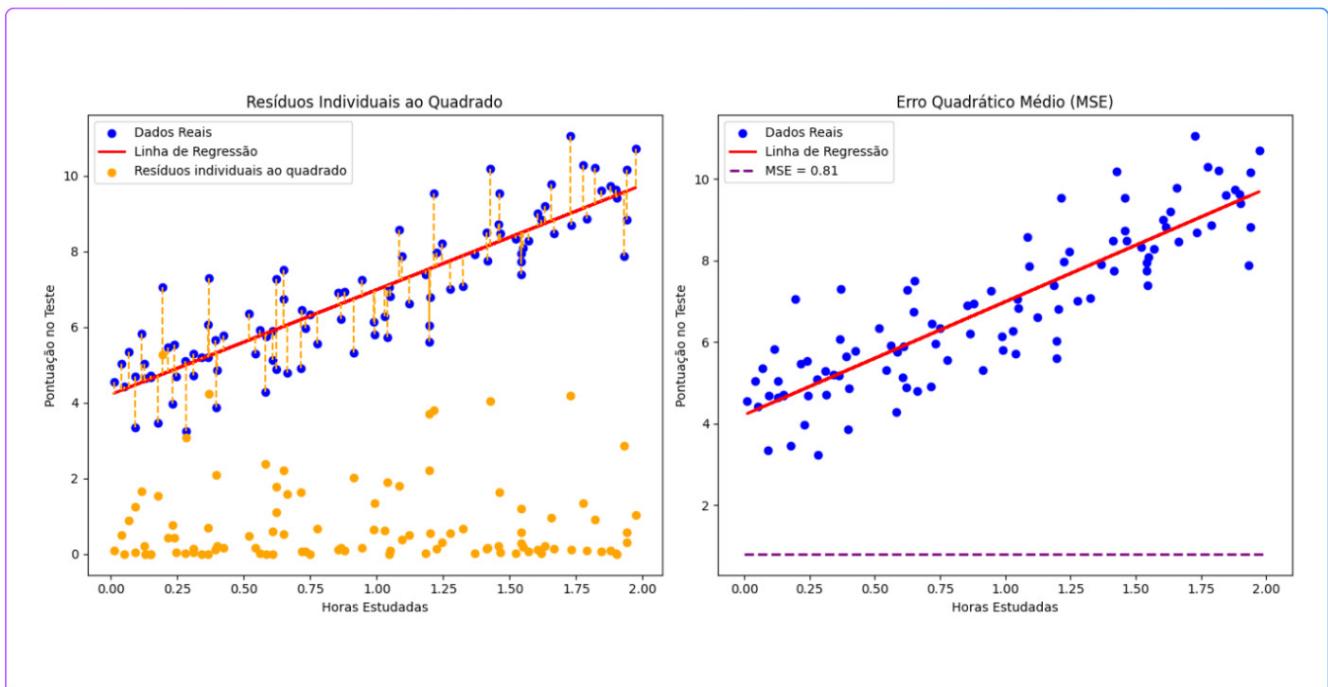
O MSE penaliza erros grandes mais severamente do que erros pequenos devido ao uso do quadrado das diferenças. Um MSE menor indica um modelo com melhor performance, enquanto um MSE maior indica um modelo com pior performance. De forma facilitar a compreensão, a figura abaixo ilustra o MSE. 1. No primeiro gráfico (Resíduos ao Quadrado), tem-se:

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os **erros individuais ao quadrado**, mostrando as diferenças entre os valores observados e preditos.

No segundo gráfico da figura abaixo (MSE), tem-se:

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » A linha tracejada roxa horizontal representa o MSE, que é a média dos resíduos ao quadrado.

Figura 52 - Exemplo de erro quadrático médio (MSE)



Fonte: autoria própria e disponível no Google Colab.

8.1.3 Erro Absoluto Médio (MAE)

O MAE mede a magnitude média dos erros em um conjunto de previsões, sem considerar sua direção. É uma métrica linear, o que significa que todos os erros são ponderados igualmente. Quanto menor o MAE, melhor o modelo. Penaliza de forma linear todos os erros.

O Mean Absolute Error (MAE) é definido como:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

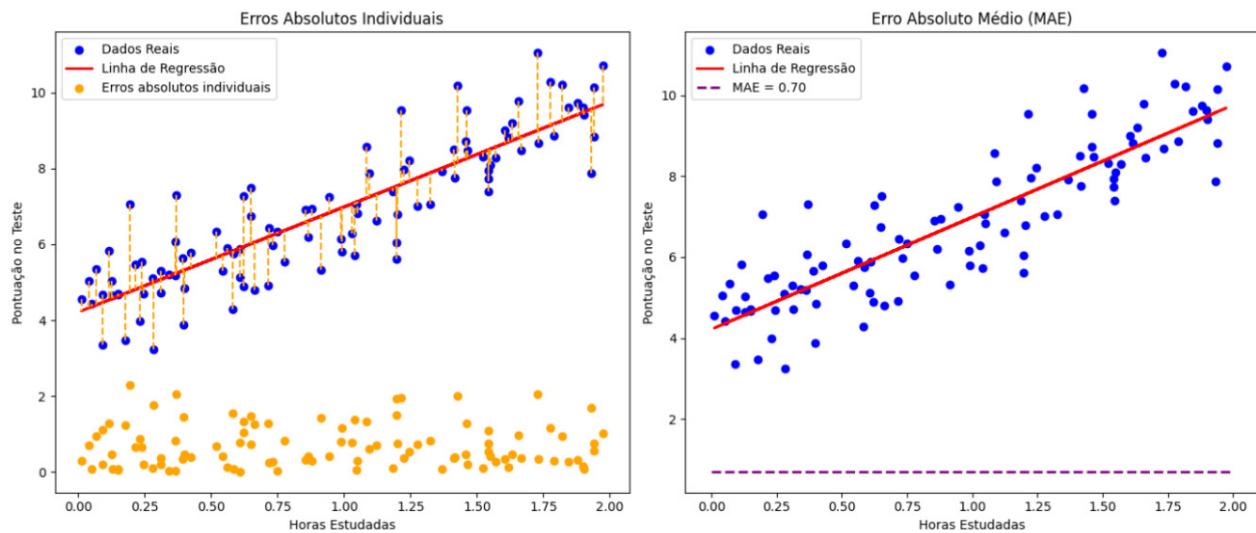
De forma facilitar a compreensão, a figura abaixo ilustra o MAE. No primeiro gráfico (Erros Absolutos Individuais) tem-se:

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os **erros absolutos individuais**, ou seja, as diferenças absolutas entre os valores observados e os valores preditos.

No segundo gráfico (MAE) tem-se:

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » A linha tracejada roxa horizontal representa o valor do MAE, que é a média dos erros absolutos.

Figura 53 - Exemplo de erro absoluto médio (MAE)



Fonte: autoria própria e disponível no Google Colab.

8.1.4 Raiz do Erro Quadrático Médio (RMSE)

A raiz quadrada do MSE, traz a medida de erro para a mesma escala da variável dependente. Quanto menor o RMSE, melhor o modelo. Ela possui a mesma unidade dos dados originais e penaliza mais os erros grandes.

O Root Mean Squared Error (RMSE) é definido como:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

O RMSE penaliza mais severamente grandes erros devido ao uso do quadrado das diferenças, similar ao MSE, mas retorna os erros na mesma unidade dos valores observados.

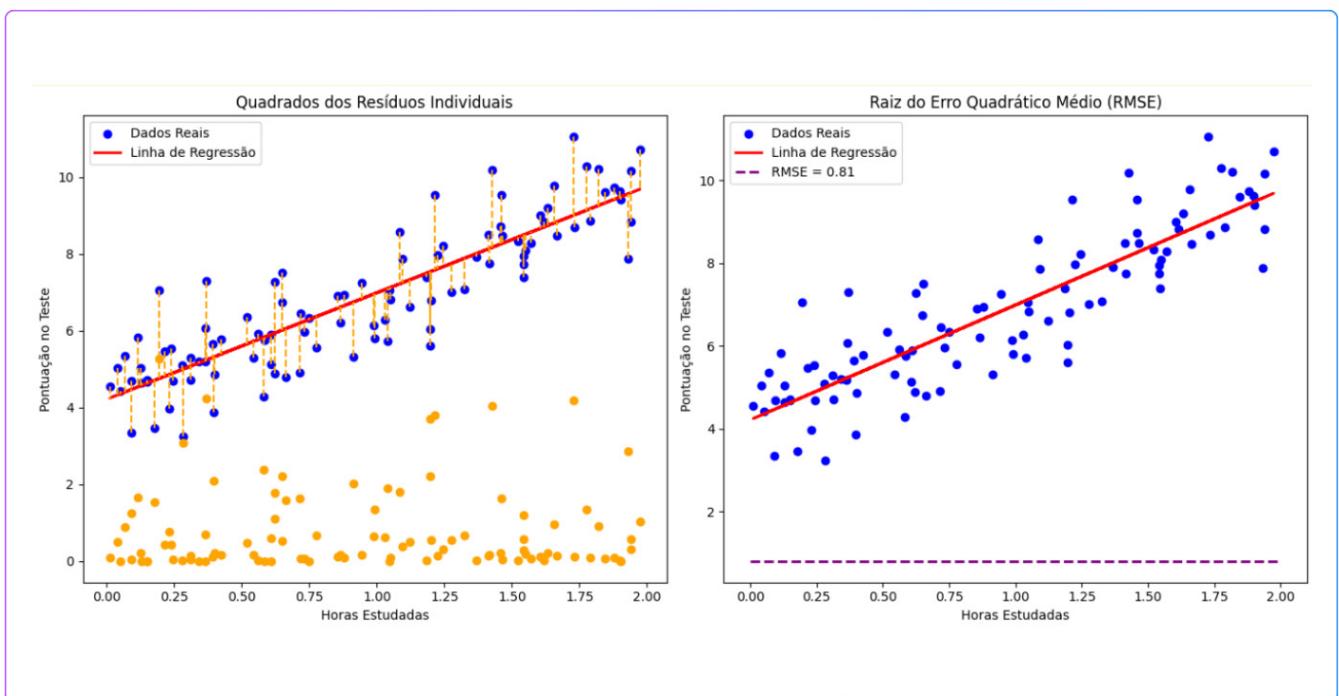
De forma facilitar a compreensão, a figura abaixo ilustra o MAE. No primeiro gráfico (Quadrados dos Resíduos Individuais) tem-se:

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os quadrados dos resíduos individuais, ou seja, as diferenças ao quadrado entre os valores observados e os valores preditos.

No segundo gráfico (RMSE), tem-se: A linha tracejada roxa horizontal representa o valor do RMSE, que é a raiz quadrada da média dos resíduos ao quadrado.

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.

Figura 54 - Exemplo de raiz do erro quadrático médio (RMSE)



Fonte: autoria própria e disponível no Google Colab.

No Quadro 2, é apresentado um resumo das vantagens e desvantagens das métricas acima explicadas. Há também a fórmula para cada uma na qual:

y_i são os valores reais

\hat{y}_i são os valores preditos pelo modelo

n é o número de observações

Quadro 2 - Comparativo entre as métricas abordadas resume essa seção

MÉTRICA	VANTAGENS	DESVANTAGENS	FÓRMULA
R² (Coeficiente de Determinação)	Interpretação intuitiva como a proporção da variância explicada pelo modelo. Útil para comparar modelos diferentes.	Pode ser enganoso em modelos não-lineares ou com outliers. Não indica a qualidade do ajuste em termos absolutos.	$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$
MSE (Mean Squared Error)	Penaliza erros grandes mais severamente devido ao quadrado dos resíduos. Utilizado para otimização em muitos algoritmos de aprendizado de máquina.	Não é intuitivo, pois está na escala do quadrado da variável dependente. Sensível a outliers.	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
MAE (Mean Absolute Error)	Interpretação intuitiva como o erro médio absoluto. Menos sensível a outliers comparado ao MSE.	Menos eficiente do ponto de vista matemático em certos algoritmos de otimização. Não penaliza erros grandes tão severamente quanto o MSE.	$MAE = \frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $
RMSE (Root Mean Squared Error)	Interpretação intuitiva, pois está na mesma escala da variável dependente. Penaliza erros grandes severamente como o MSE.	A penalização de erros grandes pode ser uma desvantagem em alguns contextos. Sensível a outliers.	$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$

Fonte: autoria própria.

8.2 Tipos de Regressão

A análise de regressão é uma técnica estatística fundamental utilizada para modelar a relação entre uma variável dependente e uma ou mais variáveis independentes. Existem vários tipos de modelos de regressão, cada um com suas características e aplicações específicas, algumas serão apresentadas nas próximas subseções.

8.2.1 Regressão Linear

Simples e múltipla, modela a relação linear entre variáveis. É o ponto de partida para muitos estudos de regressão. A regressão linear é um método estatístico para modelar

a relação entre uma variável dependente e uma ou mais variáveis independentes assumindo uma relação linear. A regressão linear simples assume a seguinte fórmula:

$$y = \beta_0 + \beta_1 X + \epsilon$$

onde:

y é a variável dependente.

X é a variável independente.

β_0 é o intercepto (valor de y quando $X = 0$).

β_1 é o coeficiente de inclinação (quanto y muda com uma unidade de mudança em X).

ϵ é o termo de erro (a diferença entre os valores observados e os valores preditos).

8.2.2 Regressão Ridge, Lasso e Elastic Net

A regressão linear é uma ferramenta poderosa para modelar relações entre variáveis. No entanto, quando lidamos com um grande número de preditores ou quando estes estão altamente correlacionados (multicolinearidade), o modelo pode se tornar instável e sobreajustar aos dados de treinamento, generalizando mal para novos dados. Para mitigar esses problemas, foram desenvolvidas técnicas de regularização, como *Ridge*, *Lasso* e *Elastic Net*.

Regressão Ridge (L2):

A Regressão *Ridge* utiliza a regularização L2. Ela adiciona um termo de penalidade baseado na norma L2 dos coeficientes à função de custo da regressão linear. Essa penalização faz com que o novo modelo não obtenha os melhores resultados no conjunto de treino, porém visa-se melhor desempenho no conjunto de teste. Sua fórmula:

$$\min(\|y - X\beta\|^2 + \lambda\|\beta\|^2)$$

Onde:

y é o vetor de variáveis dependentes

X é a matriz de variáveis independentes

β é o vetor de coeficientes

λ é o parâmetro de regularização

ou ainda:

$\|y - X\beta\|^2$: Esta parte é também conhecida como SSR que foi apresentada junto com a métrica R^2 , na qual:

y representa os valores observados;

$X\beta$ representa os valores previstos pelo modelo;

A diferença ($y - X\beta$) representa os resíduos;

Ao elevar ao quadrado ($\| \dots \| ^2$), obtemos a soma dos quadrados desses resíduos.

$\lambda\|\beta\|^2$: Esta é a parte da regularização L2 (*Ridge*), onde:

$\|\beta\|^2$ é a norma L2 ao quadrado dos coeficientes β ;

λ (*lambda*) é o parâmetro de regularização que controla a força da penalidade.

Algumas observações adicionais:

- » O termo RSS ($\|y - X\beta\|^2$) garante que o modelo se ajuste bem aos dados de treinamento.
- » O termo de regularização L2 ($\lambda\|\beta\|^2$) penaliza coeficientes grandes, incentivando o modelo a usar coeficientes menores e mais estáveis.
- » O parâmetro λ controla o equilíbrio entre estes dois objetivos. Um λ maior dá mais peso à regularização, enquanto um λ menor prioriza o ajuste aos dados.

Como principais características da regressão *Ridge* estão a redução do *overfitting* (sobreajuste) ao diminuir a magnitude dos coeficientes, a tendência de distribuir o peso entre todas as variáveis e sua eficácia quando há multicolinearidade entre as variáveis independentes.

Régressão Lasso (L1):

LASSO utiliza a regularização L1. Adiciona um termo de penalidade baseado na norma L1 dos coeficientes.

Fórmula:

$$\min(\|y - X\beta\|^2 + \lambda\|\beta\|_1)$$

Onde:

y é o vetor de variáveis dependentes

X é a matriz de variáveis independentes

β é o vetor de coeficientes

λ é o parâmetro de regularização

Como visto anteriormente, $\|y - X\beta\|^2$ refere-se ao SSR e $\lambda\|\beta\|_1$ é a norma L1. A principal diferença entre a L1 e L2 é que a segunda utiliza a norma ao quadrado. Pode-se entender β como uma inclinação que será adicionado à inclinação já existente do resultado do modelo linear.

No L1 o β está sob módulo, e no L2 ele é elevado o quadrado. Em ambos casos, o objetivo é evitar os valores negativos. A diferença é que para o L2, por menor que seja β , ele nunca poderá ser nulo, uma vez que sempre será elevado ao quadrado. Já para o L1, o módulo de 0 é 0, permitindo assim a seleção de variáveis.

Elastic Net:

O *Elastic Net* combina as regularizações L1 e L2, aproveitando as vantagens de ambas as técnicas.

Fórmula:

$$\min(\|y - X\beta\|^2 + \lambda_1\|\beta\|_1 + \lambda_2\|\beta\|^2)$$

Onde:

$\|y - X\beta\|^2$: Esta parte é também conhecida como SSR

$\lambda_1\|\beta\|_1$: componente da regularização *Lasso* (L1)

$\lambda_2\|\beta\|^2$: componente da regularização *Ridge* (L2)

Combina as propriedades do *Ridge* e do *Lasso*, pode selecionar variáveis (como o *Lasso*) e lidar com multicolinearidade (como o *Ridge*).

8.2.3 Árvores de Regressão

As árvores de regressão constituem uma técnica robusta e versátil no campo do aprendizado de máquina, especialmente adequada para problemas de regressão onde o objetivo é prever uma variável de saída contínua. Esta metodologia se distingue por sua capacidade de capturar relações não-lineares complexas entre variáveis preditoras e a variável alvo, oferecendo uma alternativa valiosa aos modelos lineares tradicionais.

A estrutura fundamental de uma árvore de regressão assemelha-se a um fluxograma invertido, onde cada nó interno representa uma decisão baseada em uma característica específica, e cada nó folha contém uma previsão numérica. O processo de construção da árvore envolve a divisão recursiva do conjunto de dados em subconjuntos cada vez menores, com base em regras de decisão que maximizam a homogeneidade dentro de cada subgrupo resultante. Um exemplo de árvore de regressão é ilustrado abaixo. Uma árvore de regressão é composta por:

- » **Nós de decisão:** Pontos onde o algoritmo faz uma escolha baseada em uma característica.
- » **Ramos:** Caminhos que conectam os nós, representando os resultados das decisões.
- » **Nós folha:** Pontos finais que contêm as previsões numéricas.

O algoritmo constrói a árvore de cima para baixo, seguindo estas etapas:

- » **Seleção da Melhor Característica:** O algoritmo examina todas as características disponíveis. Para cada característica, ele calcula uma métrica de “impureza” (geralmente a variância) para todas as possíveis divisões. A característica que resulta na maior redução da impureza é selecionada.
- » **Determinação do Ponto de Divisão:** Para a característica selecionada, o algoritmo determina o valor ótimo para dividir os dados. Este ponto de divisão minimiza a variância dentro dos subgrupos resultantes.
- » **Divisão do Nó:** O conjunto de dados no nó atual é dividido em dois subconjuntos com base na regra de decisão estabelecida.
- » **Recursão:** o processo é repetido para cada nó filho até que um critério de parada seja atingido.

A construção da árvore para quando:

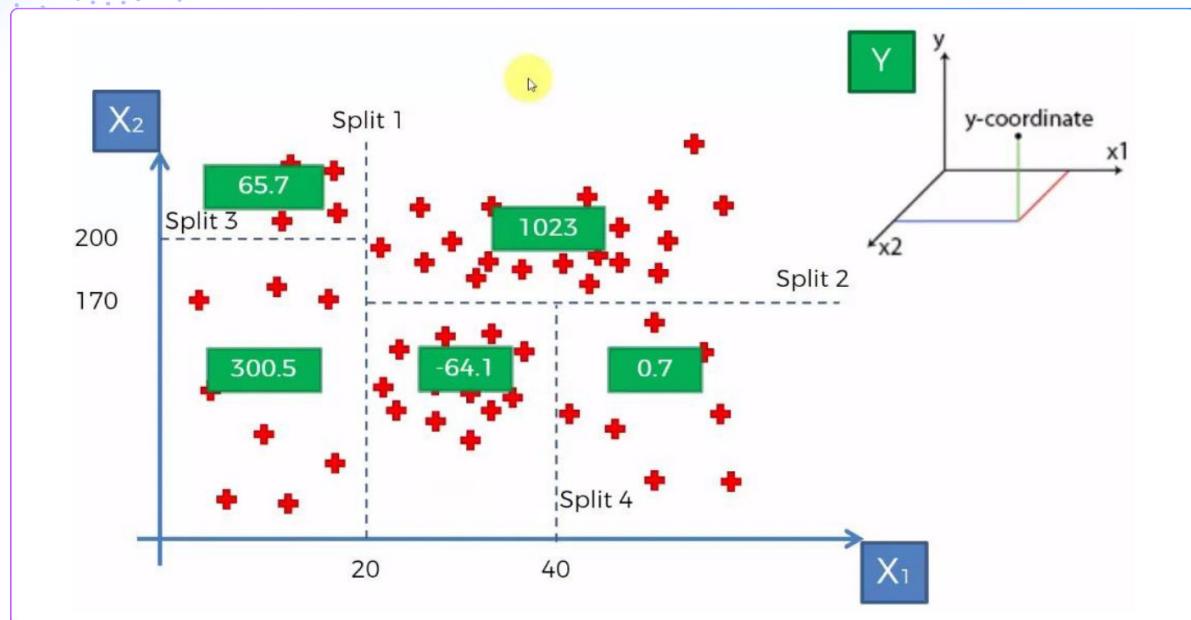
- » Um nó contém um número mínimo pré definido de amostras;
- » A profundidade máxima da árvore é alcançada;
- » A redução na impureza cai abaixo de um limiar especificado.

Para fazer uma previsão, um novo dado é passado pela árvore, seguindo as regras de decisão em cada nó. Quando atinge um nó folha, o valor associado a esse nó é a previsão.

As figuras abaixo apresentam um exemplo da aplicação de uma árvore de regressão. Na Figura 39 tem-se:

- » **Eixos X1 e X2:** Representam as duas variáveis preditoras utilizadas para construir a árvore.
- » **Pontos Vermelhos:** Cada ponto representa uma observação (exemplo) no conjunto de dados. A posição do ponto no gráfico indica os valores das variáveis preditoras para aquela observação.
- » **Linhas Verticais e Horizontais:** Representam os *splits* (divisões) realizados pela árvore de regressão. Cada *split* divide o espaço de características em duas regiões, criando nós filhos.
- » **Caixas Verdes:** Cada caixa representa uma região do espaço de características e contém um valor numérico. Esse valor é a previsão média da variável alvo (Y) para todas as observações que caem nessa região.

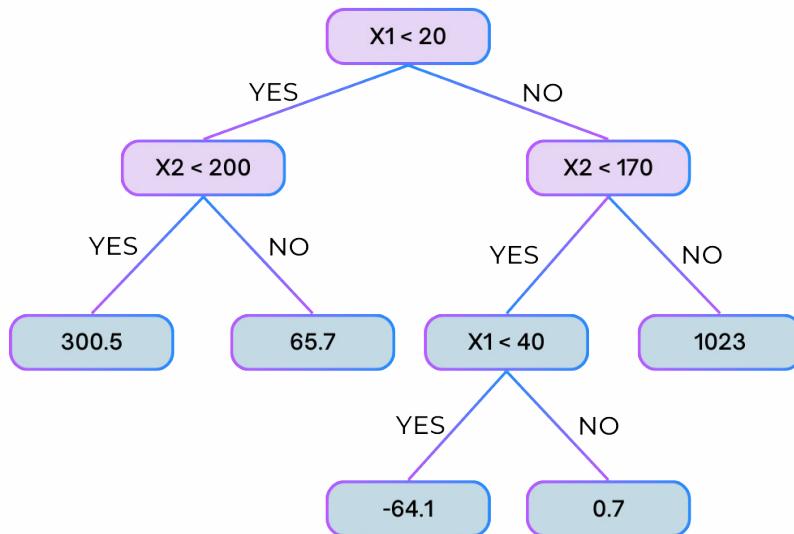
Figura 55 - Exemplo de árvore de regressão - dados



Fonte: [Medium, 2019](#).

Com base nisso, a Figura 40 pode ser interpretada da seguinte maneira. Se X_1 é menor que 20, verifica-se se o valor de X_2 é menor que 200. Se sim, então será retornado 300.5, que é o valor predito para o grupo de dados contidos à esquerda do *split 1* e abaixo do *split 3*. Se o X_2 for maior que 200, então será retornado 65.7, que é o valor predito para o grupo de dados contidos à esquerda do *split 1* e acima do *split 3*.

Figura 56 - Exemplo de árvore de regressão - árvore



Fonte: adaptada de [Medium \(2019\)](#).

As árvores de regressão oferecem diversas vantagens:

- » **Interpretabilidade:** A estrutura hierárquica permite uma fácil visualização e compreensão das decisões do modelo.
- » **Capacidade de lidar com não-linearidades:** Capturam eficientemente relações complexas e interações entre variáveis.
- » **Robustez a outliers:** São menos sensíveis a valores atípicos em comparação com métodos baseados em mínimos quadrados.
- » **Tratamento automático de variáveis categóricas:** Não requerem codificação prévia de variáveis categóricas.

Contudo, também apresentam algumas limitações:

- » **Tendência ao overfitting:** Árvores muito profundas podem se ajustar excessivamente aos dados de treinamento.
- » **Instabilidade:** Pequenas variações nos dados podem resultar em árvores significativamente diferentes.
- » **Dificuldade em capturar relações aditivas:** Podem ser menos eficientes que modelos lineares para relações puramente aditivas.

As árvores de regressão encontram aplicações em diversos domínios, incluindo: previsão de preços imobiliários, estimativa de demanda de produtos, avaliação de risco de crédito, previsão de consumo de energia.

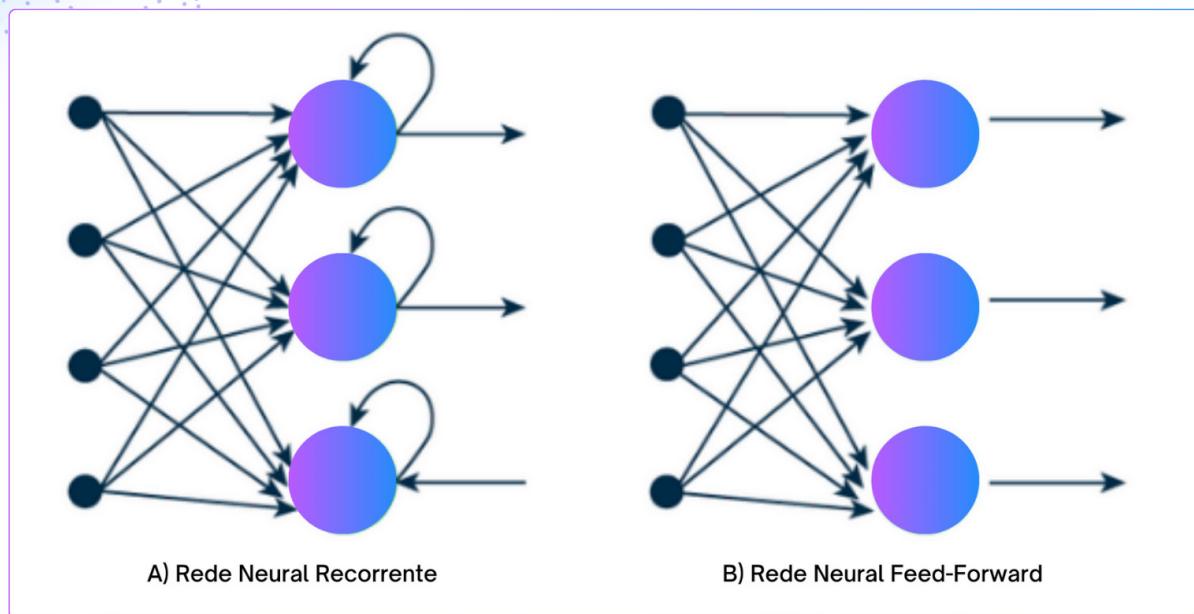
8.2.4 Recurrent Neural Network (RNN) ou Rede Neural Recorrente

As RNNs representam uma classe sofisticada de arquiteturas de RNAs, especificamente concebidas para lidar com dados sequenciais. Esta estrutura inovadora revolucionou diversas áreas da IA, particularmente aquelas que envolvem o processamento de informações temporais ou ordenadas.

A característica distintiva das RNNs reside em sua estrutura cíclica. Diferentemente das redes neurais *feedforward* tradicionais que serão detalhadas na unidade específica, as RNNs incorporam conexões de realimentação, permitindo que a informação persista ao longo do tempo. Esta propriedade confere à rede uma forma de “memória”, capacitando-a a considerar contextos anteriores na geração de saídas atuais.

Na figura abaixo é ilustrado à direita uma rede neural *feedforward*, sem recorrência. À esquerda é ilustrada a RNN, destacando-se pela seta adicional que retorna para a mesma camada de origem.

Figura 57 - Comparação entre redes neurais, RNN e Feed-Forward



Fonte: Adaptada de [GeekForGeeks, 2024](#).

Na rede neural feedforward a informação flui em uma única direção, das camadas de entrada para as camadas de saída. Cada neurônio em uma camada só recebe informações da camada anterior e não possui conexões com neurônios em camadas anteriores. Isso limita a capacidade da rede de processar sequências de dados, onde o contexto anterior é importante.

Na RNN os neurônios em uma camada recebem informações não apenas da camada anterior, mas também de si mesmos em um tempo anterior. Isso cria um “loop” que permite à rede “lembrar” informações passadas. A presença de conexões recorrentes permite que a RNN capture dependências temporais nos dados, tornando-as ideais para processar sequências como texto, séries temporais e fala.

O treinamento de RNNs emprega uma variante do algoritmo de retropropagação, denominada *Backpropagation Through Time* (BPTT). Este método “desdobra” a RNN em uma rede feedforward profunda equivalente, permitindo a aplicação de técnicas de otimização baseadas em gradiente.

As RNNs encontram aplicações em uma vasta gama de domínios incluindo PLN, com tradução automática, geração de texto e análise de sentimentos. No reconhecimento de fala, na conversão de fala para texto e em sistemas de comando por voz. Na análise de Séries Temporais, com a previsão de demanda e análise de mercado financeiro. Na bioinformática com a análise de sequências de ácido desoxirribonucleico (DNA) ou previsão de estruturas proteicas.

Apesar de sua potência, as RNNs apresentam desafios significativos como: desvanecimento e explosão do gradiente. O gradiente é como um sinal que diz à rede neural como ela deve ajustar seus “pesos” (parâmetros internos) para melhorar suas previsões. Este sinal viaja de trás para frente na rede durante o treinamento. Quando trabalha-se com sequências longas em RNNs, esse gradiente precisa percorrer um caminho muito longo, causando o desvanecimento ou explosão do gradiente. Como consequência dis-

so, a rede tem dificuldade em aprender de eventos que ocorreram muito tempo atrás na sequência, ou os ajustes na rede se tornam muito grandes, fazendo com que o treinamento se torne instável. Além disso, o processamento sequencial pode ser computacionalmente intensivo, especialmente para sequências muito longas.

8.3 Notebook Colab

No [link do colab](#) serão abordados exemplos didáticos apresentando o cálculo manual e utilizando as bibliotecas disponíveis, bem serão criadas diversas ilustrações para fixar a aprendizagem. Dentre os conteúdos abordados nesse notebook incluem-se as métricas R^2 , MSE, MAE, RMSE, um exemplo detalhado de uso do modelo regressão linear e a aplicação direta dos modelos de regressão *ridge*, árvores de regressão e RNN.

Definição e Importância da Regressão em Análise de Dados e Machine Learning

A regressão é uma técnica estatística amplamente utilizada para modelar a relação entre uma variável dependente contínua e uma ou mais variáveis independentes. Em análise de dados e *machine learning*, a regressão é fundamental para previsões quantitativas, permitindo que analistas e cientistas de dados façam inferências e estimativas baseadas em padrões observados nos dados. A importância da regressão reside na sua aplicabilidade em diversos campos, desde economia e finanças, onde é usada para prever valores de mercado, até na indústria, onde pode otimizar processos e prever demandas.

Distinção entre Problemas de Regressão e Classificação

Enquanto a regressão foca na predição de variáveis contínuas, como preços, temperaturas ou índices, a classificação é empregada para atribuir instâncias a categorias discretas. Por exemplo, determinar se um e-mail é spam ou não é um problema de classificação binária. Já prever a temperatura máxima para o próximo dia é um problema típico de regressão. A distinção entre esses dois tipos de problemas é crucial, pois os métodos e métricas de avaliação são distintos e a escolha do modelo adequado depende dessa compreensão.

Métricas de Regressão: R^2 , MSE, MAE e RMSE com Python

Objetivos

- » Compreender as métricas de desempenho R^2 , MSE, MAE e RMSE em modelos de regressão.
- » Aprender a calcular e interpretar essas métricas utilizando Python.
- » Implementar exemplos práticos de uso dessas métricas em modelos de regressão.

Introdução

Em modelos de regressão, é fundamental avaliar a performance do modelo para en-

tender quanto bem ele está se ajustando aos dados e para fazer previsões confiáveis. Existem diversas métricas para medir a qualidade de um modelo de regressão.

Para avaliar a performance de modelos de regressão, várias métricas são utilizadas:

- » **R² (Coeficiente de Determinação):** Mede a proporção da variabilidade da variável dependente que é explicada pelas variáveis independentes no modelo.
- » **MSE (Erro Quadrático Médio):** Média dos quadrados dos erros, fornece uma medida da variância dos erros.
- » **MAE (Erro Absoluto Médio):** Média dos erros absolutos, sendo menos sensível a outliers que o MSE.
- » **RMSE (Raiz do Erro Quadrático Médio):** Raiz quadrada do MSE, traz a medida de erro para a mesma escala da variável dependente.

Exemplos Práticos

Carregando Bibliotecas e Dados

Vamos criar um conjunto de dados fictício para ilustrar o cálculo do R². Suponhamos que estamos analisando a relação entre a quantidade de horas estudadas (variável independente) e a pontuação em um teste (variável dependente).

```
# Importação das bibliotecas que serão utilizadas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
# Dependendo da versão do sklearn instalado pode ser que a linha abaixo apresente
problemas
# Essa linha é para uma versão mais recente, porém no colab ela será comentada para
evitar erros
# from sklearn.metrics import root_mean_squared_error

# Gerar dados fictícios
# o random.seed é necessário para manter a reproduzibilidade dos dados.
# ele gera sempre os mesmos números aleatórios
np.random.seed(42)
# Em X são gerados um vetor de 100 dados aleatórios com base no random.seed
X = 2 * np.random.rand(100, 1)
# Em y são gerados outro vetor com 100 dados aleatórios baseados em X e um ruído
y = 4 + 3 * X + np.random.randn(100, 1)

# Plotar os dados para facilitar a visualização
# gera um gráfico de pontos azuis com os valores reais dos pontos
plt.scatter(X, y, color='blue')
# Define-se o título
plt.title('Relação entre Horas Estudadas e Pontuação no Teste')
# Define-se o texto do eixo x
plt.xlabel('Horas Estudadas')
# Define-se o texto do eixo y
```

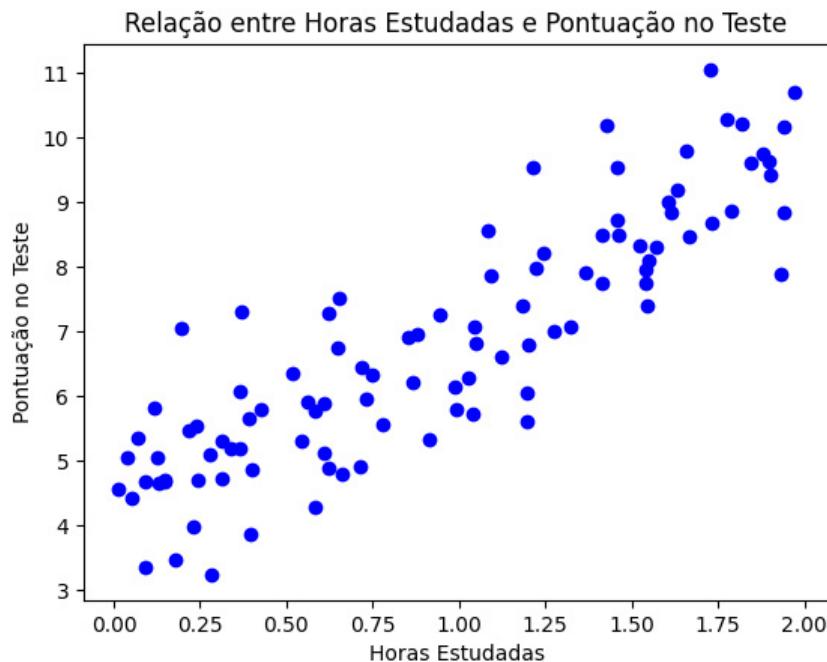
continua

```

plt.ylabel('Pontuação no Teste')
# Mostra a figura
plt.show()

```

Figura 58 - Relação entre Horas Estudadas e Pontuação no Teste



Fonte: autoria própria.

Treinando o Modelo de Regressão Linear

Vamos ajustar um modelo de regressão linear simples aos dados.

Para utilizar as métricas é necessário ter dados reais e preditos, para isso será usado um simples modelo de regressão linear, que fará a predição de valores de **y** com base nos valores de **X**.

Esses valores preditos serão guardados no **y_pred**.

Ao fim, é plotado em vermelho o resultado desse modelo de predição. Em pontos na cor azul estão ilustrados os dados reais.

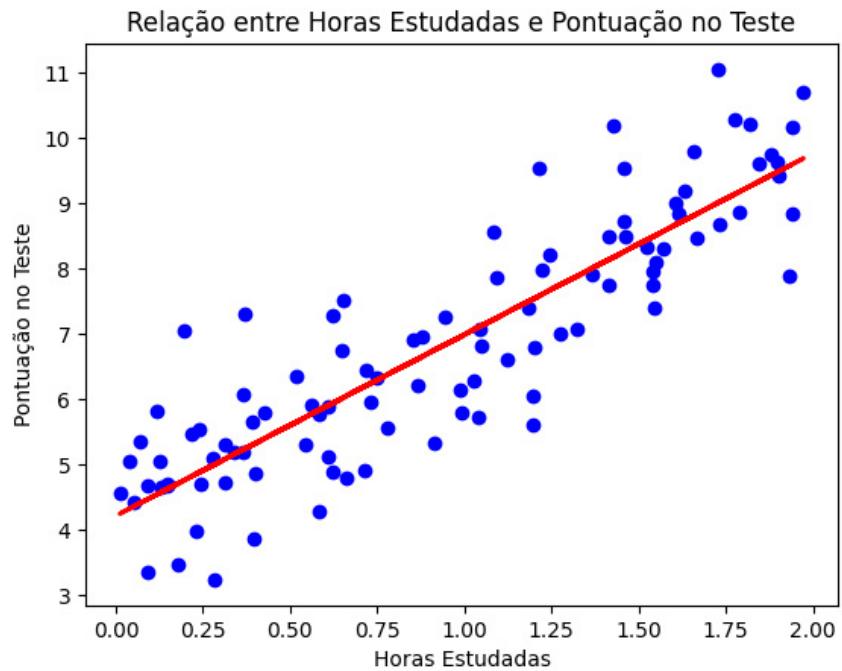
```

# Ajustar o modelo de regressão linear
# Cria um modelo de regressão linear
model = LinearRegression()
# Treina o modelo com base nos valores de X e y
model.fit(X, y)
# Utilizando o modelo já treinado, faz a predição dos valores de X e guarda em y_pred
y_pred = model.predict(X)

# Plotar os dados e a linha de regressão
plt.scatter(X, y, color='blue')
plt.plot(X, y_pred, color='red', linewidth=2)
plt.title('Relação entre Horas Estudadas e Pontuação no Teste')
plt.xlabel('Horas Estudadas')
plt.ylabel('Pontuação no Teste')
plt.show()

```

Figura 59 - Relação entre Horas Estudadas e Pontuação no Teste - Regressão Linear



Fonte: autoria própria.

Explicações Teóricas com Exemplos práticos

R^2 (Coeficiente de Determinação)

- » **Definição:** Mede a proporção da variância dos dados dependentes que é explicada pelo modelo de regressão.
- » **Interpretação:** Varia de 0 a 1. Um valor de 1 indica que o modelo explica toda a variância dos dados, enquanto um valor de 0 indica que o modelo não explica nenhuma variância.
- » **Função:** pode ser utilizado o pacote do sklearn `r2 = r2_score(y, y_pred)`

Explicação Detalhada do R^2

O coeficiente de Determinação R^2 é calculado como:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

\bar{y} é a média dos valores reais,

$\sum_{i=1}^n (y_i - \hat{y}_i)^2$ é a Soma dos Quadrados dos Resíduos (SSR),

$\sum_{i=1}^n (y_i - \bar{y})^2$ é a Soma Total dos Quadrados (SST).

O valor de R^2 varia entre 0 e 1:

$R^2 = 0$ indica que o modelo não explica nenhuma variação nos dados.

$R^2 = 1$ indica que o modelo explica toda a variação nos dados.

Abaixo serão apresentados duas maneiras de cálculo do R^2 , uma manualmente utilizando a formulação matemática acima e outra mais direta, utilizando o pacote do sklearn.

Como é de se esperar, em ambos casos os valores encontrados são os mesmos.

```
# Calculando manualmente
# Média dos valores reais
y_mean = np.array([np.mean(y)])

# Calcular os componentes da fórmula do R²
ssr = np.sum((y - y_pred) ** 2)
sst = np.sum((y - y_mean) ** 2)

# Cálculo do R² utilizando a fórmula
r2_manual = 1 - (ssr/sst)

# Impressão do resultado
print(f"Coefficient of Determination (R²): {r2_manual:.2f}")

Coefficient of Determination (R²): 0.77
```



```
# Calcular o R²
# Passa-se os valores reais (y), os valores obtidos do modelo (y_pred)
# o valor do erro é guardado em r2
r2 = r2_score(y, y_pred)
print(f'Coefficient of Determination (R²): {r2:.2f}')

Coefficient of Determination (R²): 0.77
```

Visualização da Variabilidade Explicada

Este código permite visualizar graficamente como um modelo de regressão se ajusta aos dados e qual a proporção da variabilidade dos dados que o modelo consegue explicar. Apresentam-se a variabilidade não explicada pelo modelo (SSR) e a variabilidade Total (SST). Esses conceitos ajudam a entender melhor a fórmula do R^2 .

Variabilidade Não Explicada (SSR): As linhas pontilhadas laranja representam os erros do modelo, ou seja, a diferença entre o valor real (ponto em azul) e sua projeção ao

modelo (linha vermelha). Quanto menores forem essas linhas, melhor o modelo se ajusta aos dados.

Variabilidade Total (SST): A linha horizontal verde representa a média dos dados. A distância entre os pontos e a linha representa a variabilidade total dos dados.

O primeiro gráfico mostra a variabilidade que o modelo não consegue explicar. Quanto menor a distância entre os pontos e a linha de regressão, melhor o modelo se ajusta aos dados. O segundo gráfico mostra a variabilidade total dos dados. Comparando os dois gráficos, podemos ter uma ideia de quanto da variabilidade total está sendo explicada pelo modelo.

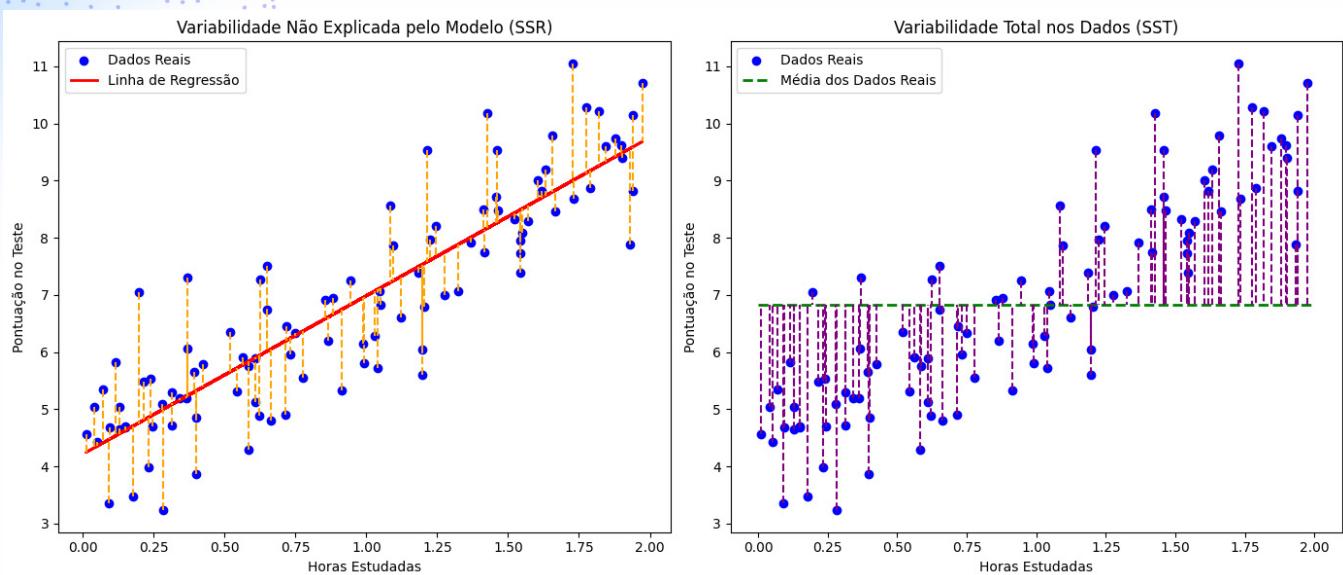
```
# Criar as subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plotar a variabilidade não explicada (SSR)
axs[0].scatter(X, y, color='blue', label='Dados Reais')
axs[0].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
for i in range(len(X)):
    axs[0].plot([X[i], X[i]], [y[i], y_pred[i]], color='orange', linestyle='dashed')
axs[0].set_title('Variabilidade Não Explicada pelo Modelo (SSR)')
axs[0].set_xlabel('Horas Estudadas')
axs[0].set_ylabel('Pontuação no Teste')
axs[0].legend()

# Plotar a variabilidade total (SST)
axs[1].scatter(X, y, color='blue', label='Dados Reais')
axs[1].hlines(y=y_mean, xmin=0, xmax=2, color='green', linestyle='dashed',
               linewidth=2, label='Média dos Dados Reais')
for i in range(len(X)):
    axs[1].plot([X[i], X[i]], [y[i], y_mean], color='purple', linestyle='dashed')
axs[1].set_title('Variabilidade Total nos Dados (SST)')
axs[1].set_xlabel('Horas Estudadas')
axs[1].set_ylabel('Pontuação no Teste')
axs[1].legend()

# Mostrar os gráficos
plt.tight_layout()
plt.show()
```

Figura 60 - Gráfico SSR / SST



Fonte: autoria própria.

MSE (Mean Squared Error)

- » **Definição:** É a média dos quadrados dos erros entre os valores previstos e os valores reais.
- » **Interpretação:** Quanto menor o MSE, melhor o modelo. Penaliza mais os erros grandes do que os erros pequenos.
- » **Função:** pode ser utilizado o pacote do sklearn `mse = mean_squared_error(y, y_pred)`

Explicação Detalhada do MSE

O Mean Squared Error (MSE) é uma métrica utilizada para avaliar a performance de um modelo de regressão. Ele calcula a média dos quadrados das diferenças entre os valores observados (reais) e os valores preditos pelo modelo. A fórmula do MSE é:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

O MSE penaliza erros grandes mais severamente do que erros pequenos devido ao uso do quadrado das diferenças. Um MSE menor indica um modelo com melhor performance, enquanto um MSE maior indica um modelo com pior performance.

Abaixo serão apresentados duas maneiras de cálculo do MSE, uma manualmente utilizando a formulação matemática acima e outra mais direta, utilizando o pacote do sklearn.

Como é de se esperar, em ambos casos os valores encontrados são os mesmos.

```
# Calculando manualmente

# Calcular a soma dos quadrados dos resíduos (SSR)
ssr = np.sum((y - y_pred) ** 2)
# Médias dos valores
mse_manual = ssr/len(y)

print(f"Mean Squared Error (MSE): {mse_manual:.2f}")
Mean Squared Error (MSE): 0.81

# Calcular o MSE
# Passa-se os valores reais (y), os valores obtidos do modelo (y_pred)
# o valor do erro é guardado em mse
mse = mean_squared_error(y, y_pred)
print(f'Mean Squared Error (MSE): {mse:.2f}')
Mean Squared Error (MSE): 0.81
```

Visualização dos Resíduos ao quadrado e MSE

Para ilustrar melhor, podemos plotar os resíduos ao quadrado e o MSE obtido pelo modelo.

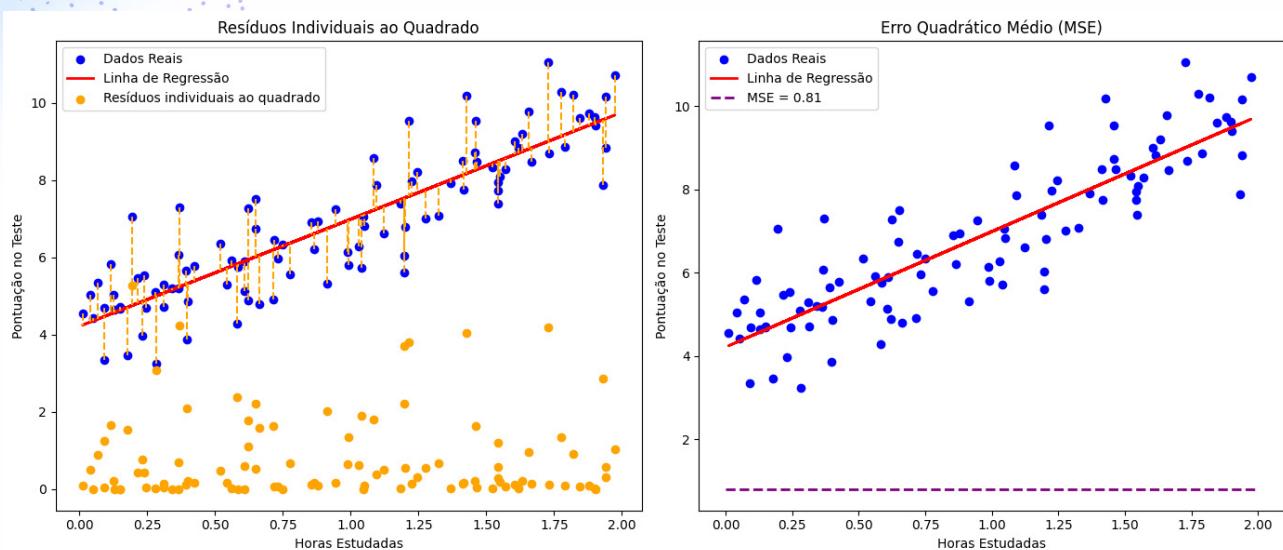
```
# Criar as subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plotar os resíduos individuais ao quadrado
axs[0].scatter(X, y, color='blue', label='Dados Reais')
axs[0].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
for i in range(len(X)):
    axs[0].plot([X[i], X[i]], [y[i], y_pred[i]], color='orange', linestyle='dashed')
axs[0].scatter(X, (y - y_pred)**2, color='orange', label='Resíduos individuais ao
quadrado')
axs[0].set_title('Resíduos Individuais ao Quadrado')
axs[0].set_xlabel('Horas Estudadas')
axs[0].set_ylabel('Pontuação no Teste')
axs[0].legend()

# Plotar o MSE
axs[1].scatter(X, y, color='blue', label='Dados Reais')
axs[1].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
axs[1].hlines(y=mse, xmin=0, xmax=2, color='purple', linestyle='dashed', linewidth=2,
label=f'MSE = {mse:.2f}')
axs[1].set_title('Erro Quadrático Médio (MSE)')
axs[1].set_xlabel('Horas Estudadas')
axs[1].set_ylabel('Pontuação no Teste')
axs[1].legend()

# Mostrar os gráficos
plt.tight_layout()
plt.show()
```

Figura 61 - Gráfico dos Resíduos ao quadrado e MSE



Fonte: autoria própria.

Explicação dos Gráficos

1. Primeiro Gráfico (Resíduos ao Quadrado):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os erros individuais ao quadrado, mostrando as diferenças entre os valores observados e preditos.

2. Segundo Gráfico (MSE):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » A linha tracejada roxa horizontal representa o Erro Quadrático Médio (MSE), que é a média dos resíduos ao quadrado.

Mean Absolute Error (MAE)

- » **Definição:** É a média dos valores absolutos dos erros entre os valores previstos e os valores reais.
- » **Interpretação:** Quanto menor o MAE, melhor o modelo. Penaliza de forma linear todos os erros.
- » **Função:** pode ser utilizado o pacote do sklearn `mae = mean_absolute_error(y, y_pred)`

O MAE mede a magnitude média dos erros em um conjunto de previsões, sem considerar sua direção. É uma métrica linear, o que significa que todos os erros são ponderados igualmente.

Explicação Detalhada do MAE

O Mean Absolute Error (MAE) é definido como:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

Abaixo serão apresentados duas maneiras de cálculo do MAE, uma manualmente utilizando a formulação matemática acima e outra mais direta, utilizando o pacote do sklearn.

Como é de se esperar, em ambos casos os valores encontrados são os mesmos.

```
# Cálculo manual do MAE
err_abs = abs(y - y_pred)
# Somam-se os erros
sum_erro = np.sum(err_abs)
# Calcula o MAE
mae_manual = sum_erro/len(y)
# Impressão do resultado obtido
print(f'Mean Absolute Error (MAE): {mae_manual:.2f}')
```

```
Mean Absolute Error (MAE): 0.70
```

```
# Calcular o MAE com o uso do sklearn
# Passa-se os valores reais (y), os valores obtidos do modelo (y_pred)
# o valor do erro é guardado em mae
mae = mean_absolute_error(y, y_pred)
print(f'Mean Absolute Error (MAE): {mae:.2f}')
```

```
Mean Absolute Error (MAE): 0.70
```

Visualização dos Erros Absolutos e do MAE

Para ilustrar melhor, podemos plotar os resíduos absolutos e o MAE obtido pelo modelo.

```
# Criar as subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plotar os erros absolutos individuais
axs[0].scatter(X, y, color='blue', label='Dados Reais')
axs[0].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
for i in range(len(X)):
```

continua

```

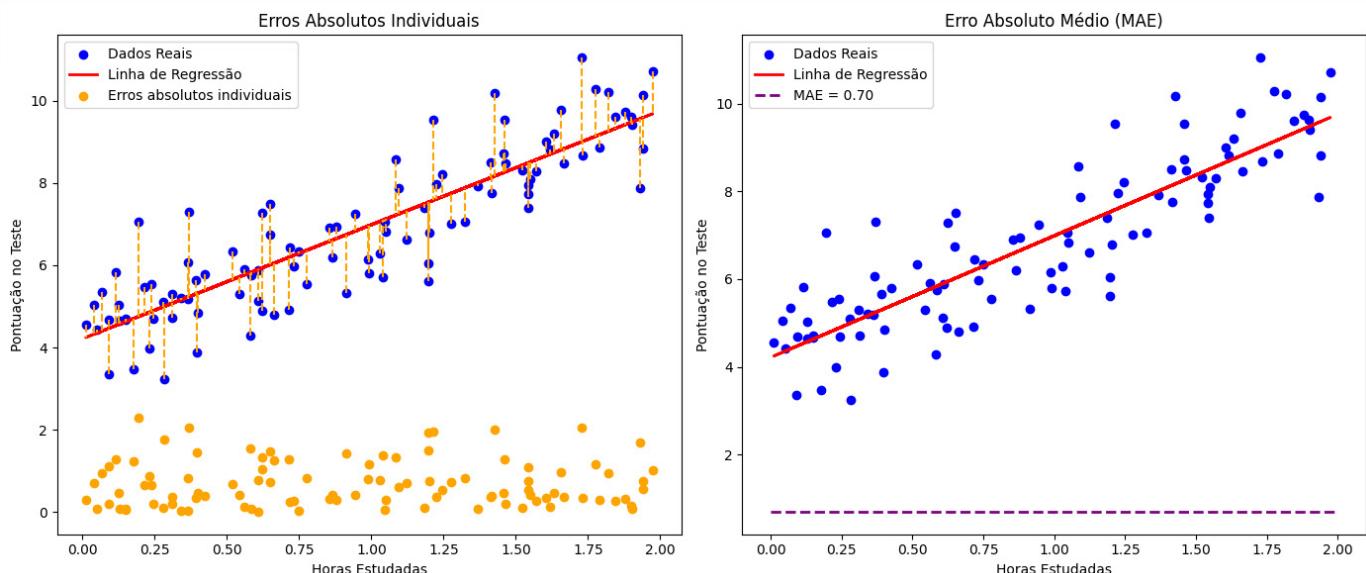
    axs[0].plot([X[i], X[i]], [y[i], y_pred[i]], color='orange', linestyle='dashed')
    axs[0].scatter(X, abs(y - y_pred), color='orange', label='Erros absolutos individuais')
    axs[0].set_title('Erros Absolutos Individuais')
    axs[0].set_xlabel('Horas Estudadas')
    axs[0].set_ylabel('Pontuação no Teste')
    axs[0].legend()

# Plotar o MAE
axs[1].scatter(X, y, color='blue', label='Dados Reais')
axs[1].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
axs[1].hlines(y=mae, xmin=0, xmax=2, color='purple', linestyle='dashed', linewidth=2,
label=f'MAE = {mae:.2f}')
axs[1].set_title('Erro Absoluto Médio (MAE)')
axs[1].set_xlabel('Horas Estudadas')
axs[1].set_ylabel('Pontuação no Teste')
axs[1].legend()

# Mostrar os gráficos
plt.tight_layout()
plt.show()

```

Figura 62 - Gráfico dos Erros Absolutos e do MAE



Fonte: autoria própria.

Explicação dos Gráficos

1. Primeiro Gráfico (Erros Absolutos Individuais):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os erros absolutos individuais, ou seja, as diferenças absolutas entre os valores observados e os valores preditos.

2. Segundo Gráfico (MAE):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » A linha tracejada roxa horizontal representa o valor do MAE, que é a média dos erros absolutos.

Para pensar:

Pergunta: Se o MAE utiliza os erros absolutos e o MSE os mesmos erros porém ao quadrado, por que o valor do MAE foi superior ao do MSE?

Resposta: Compare os pontos em laranja dos dois gráficos (MAE e MSE), note como ao elevar o quadrado (gráfico MSE), erros grandes ficam muito maiores ($10^2=100$), e erros pequenos ficam muito menores ($0.1^2=0.01$). Isso causa essa diferença nos valores.

RMSE (Root Mean Squared Error)

- » **Definição:** É a raiz quadrada da média dos quadrados dos erros.
- » **Interpretação:** Quanto menor o RMSE, melhor o modelo. Tem a mesma unidade dos dados originais e penaliza mais os erros grandes.

Explicação Detalhada do RMSE

O Root Mean Squared Error (RMSE) é definido como:

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

onde:

y_i são os valores reais,

\hat{y}_i são os valores preditos pelo modelo,

n é o número de observações.

O RMSE penaliza mais severamente grandes erros devido ao uso do quadrado das diferenças, similar ao MSE, mas retorna os erros na mesma unidade dos valores observados.

Abaixo serão apresentados duas maneiras de cálculo do MSE, uma manualmente utilizando a formulação matemática acima e outra mais direta, utilizando o pacote do sklearn.

Como é de se esperar, em ambos casos os valores encontrados são os mesmos.

```
# Calcular o RMSE manualmente

# Calcular a soma dos quadrados dos resíduos (SSR)
ssr = np.sum((y - y_pred) ** 2)
# Médias dos valores
mse_manual = ssr/len(y)
# A raiz quadrada da média
rmse_manual = np.sqrt(mse_manual)

print(f'Root Mean Squared Error (RMSE): {rmse_manual:.2f}')
```

```
Root Mean Squared Error (RMSE): 0.90
```

```
#Cálculo utilizando o pacote do sklearn
# Passa-se os valores reais (y), os valores obtidos do modelo (y_pred)
# nesse caso o parametro `squared` é necessário para que seja calculado a raiz
# quadrada
# o valor do erro é guardado em rmse
rmse = mean_squared_error(y, y_pred, squared=False)

# Impressão dos resultados obtidos
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
```

```
Root Mean Squared Error (RMSE): 0.90
```

Visualização dos Quadrado dos Resíduos e do RMSE

Para ilustrar melhor, podemos plotar o quadrado dos resíduos e o RMSE obtido pelo modelo.

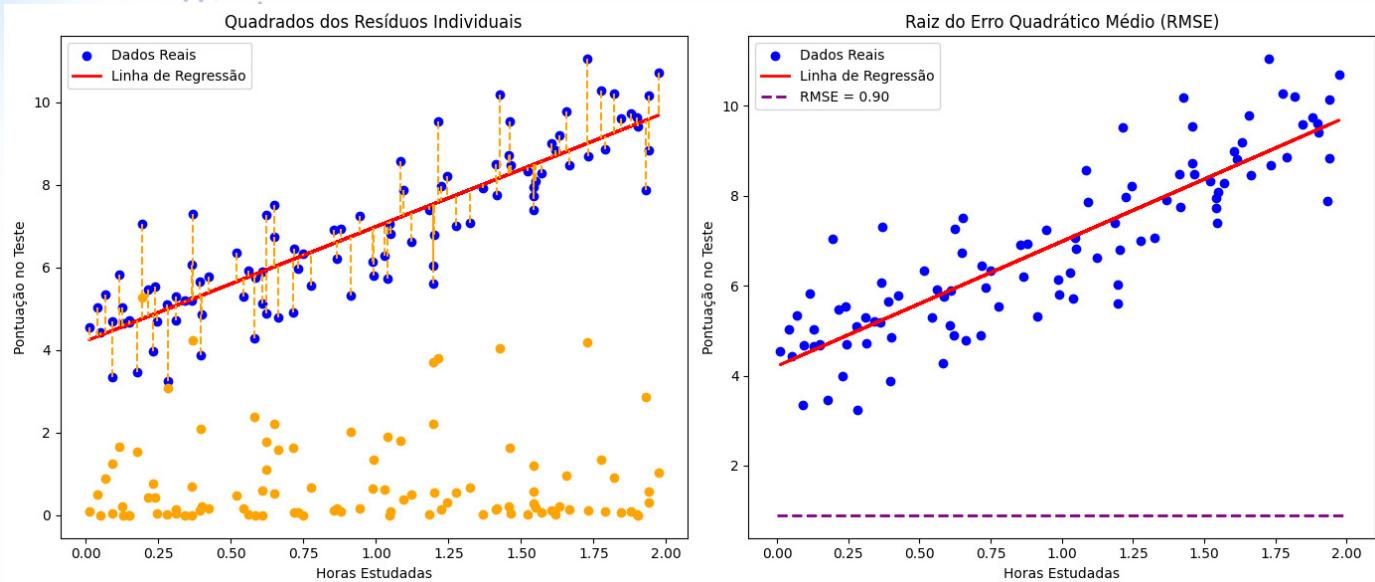
```
# Criar as subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plotar os resíduos ao quadrado individuais
axs[0].scatter(X, y, color='blue', label='Dados Reais')
axs[0].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
for i in range(len(X)):
    axs[0].plot([X[i], X[i]], [y[i], y_pred[i]], color='orange', linestyle='dashed')
    axs[0].scatter(X[i], (y[i] - y_pred[i])**2, color='orange')
axs[0].set_title('Quadrados dos Resíduos Individuais')
axs[0].set_xlabel('Horas Estudadas')
axs[0].set_ylabel('Pontuação no Teste')
axs[0].legend()

# Plotar o RMSE
axs[1].scatter(X, y, color='blue', label='Dados Reais')
axs[1].plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
axs[1].hlines(y=rmse, xmin=0, xmax=2, color='purple', linestyle='dashed', linewidth=2,
label=f'RMSE = {rmse:.2f}')
axs[1].set_title('Raiz do Erro Quadrático Médio (RMSE)')
axs[1].set_xlabel('Horas Estudadas')
axs[1].set_ylabel('Pontuação no Teste')
axs[1].legend()

# Mostrar os gráficos
plt.tight_layout()
plt.show()
```

Figura 63 - Gráfico do Quadrado dos Resíduos e do RMSE



Fonte: autoria própria.

Explicação dos Gráficos

1. Primeiro Gráfico (Quadrados dos Resíduos Individuais):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » As linhas tracejadas laranja e os pontos laranja representam os quadrados dos resíduos individuais, ou seja, as diferenças ao quadrado entre os valores observados e os valores preditos.

2. Segundo Gráfico (RMSE):

- » Os pontos azuis representam os dados reais.
- » A linha vermelha representa a linha de regressão ajustada pelo modelo.
- » A linha tracejada roxa horizontal representa o valor do RMSE, que é a raiz quadrada da média dos resíduos ao quadrado.

Para pensar:

Pergunta: Se o MSE calcula a média dos erros ao quadrado e o RMSE a raiz quadrada dessa média dos erros ao quadrado, por que o valor do RMSE foi superior ao do MSE?

Resposta: O valor de $\sqrt{0.81}$ (valor encontrado do MSE) é 0.90, a raiz quadrada de um numero menor que 1 sempre vai ser um valor maior que o próprio valor.

Pergunta: O valor do RMSE sempre vai ser maior que o do MSE?

Resposta: Não, porque o valor do MSE não está limitado apenas a valores menores que 1. Se o MSE encontrado for de 4, seu RMSE será $\sqrt{4}=2$.

Para pensar:

Se possuo dois conjuntos de dados diferentes, sobre assuntos diferentes, por exemplo:

- » diferença de velocidade entre atletas de uma maratona
- » diferença de velocidade entre atletas de uma corrida de automóveis

Ao utilizar um modelo de regressão é possível estimar os valores preditos.

Pergunta: É possível comparar os erros e afirmar que o mesmo modelo é bom para predição em ambas atividades?

Resposta: Não, como os dados são de magnitudes diferentes, ou seja velocidade de atletas da ordem de dezenas de km/h e velocidade entre corridas de automóveis da ordem de centenas de km/h, os erros podem possuir escalas diferentes. Porém, se os erros estiverem na mesma escala, é possível fazer a comparação.

Pergunta: Uma diferença de 10% entre a velocidade real e predita em ambos datasets, resultariam sempre em erros equivalentes?

Resposta: Não necessariamente, o código abaixo ilustra melhor. Dependendo da métrica o fator escala pode fazer grande diferença.

```
# Conjunto de dados de uma maratona
y_foot = np.array([10.5, 12.1, 15.3, 17.2, 18.5, 23.2])
y_foot_pred = y_foot * 1.1

# Conjunto de dados de uma corrida de automóveis
y_car = np.array([150.4, 220.2, 157.8, 149.9, 153.0])
y_car_pred = y_car * 1.1

# Cálculo do R2 para ambos datasets
r2_foot = r2_score(y_foot, y_foot_pred)
r2_car = r2_score(y_car, y_car_pred)
print(f'R2 foot: {r2_foot:.2f}')
print(f'R2 car: {r2_car:.2f}')
print('*****')

# Cálculo do MAE para ambos datasets
mae_foot = mean_absolute_error(y_foot, y_foot_pred)
mae_car = mean_absolute_error(y_car, y_car_pred)
print(f'MAE foot: {mae_foot:.2f}')
print(f'MAE car: {mae_car:.2f}')
print('*****')
```

continua

```

# Cálculo do MSE para ambos datasets
mse_foot = mean_squared_error(y_foot, y_foot_pred)
mse_car = mean_squared_error(y_car, y_car_pred)
print(f'MSE foot: {mse_foot:.2f}')
print(f'MSE car: {mse_car:.2f}')
print('*****')

# Cálculo do RMSE para ambos datasets
rmse_foot = mean_squared_error(y_foot, y_foot_pred, squared=False)
rmse_car = mean_squared_error(y_car, y_car_pred, squared=False)
print(f'RMSE foot: {rmse_foot:.2f}')
print(f'RMSE car: {rmse_car:.2f}')
print('*****')

R2 foot: 0.84
R2 car: 0.61
*****
MAE foot: 1.61
MAE car: 16.63
*****
MSE foot: 2.78
MSE car: 283.78
*****
RMSE foot: 1.67
RMSE car: 16.85
*****

```

Exercícios

Com base nos dados gerados anteriormente:

1. Plote em um gráfico com os pontos de velocidade real e velocidade predita para cada dataset (maratona, corrida de veículos)
2. Plote em um gráfico de barras os valores comparativos das metricas entre os datasets

Exercício resolvido 1: Plote em um gráfico com os pontos de velocidade real e velocidade predita para cada dataset (maratona, corrida de veículos)

```

# Criação de um DataFrame com os valores das métricas para o caso da maratona
df_maratona = pd.DataFrame({'y': y_foot,
                            'y_pred': y_foot_pred
                           })

```

df_maratona

	y	y_pred
0	10.5	11.55
1	12.1	13.31
2	15.3	16.83
3	17.2	18.92
4	18.5	20.35
5	23.2	25.52

```
# Criação de um DataFrame com os valroes das métricas para o caso da corrida de
automóveis
```

```

df_carros = pd.DataFrame({'y': y_car,
                           'y_pred': y_car_pred
                          })
df_carros



|   | y     | y_pred |
|---|-------|--------|
| 0 | 150.4 | 165.44 |
| 1 | 220.2 | 242.22 |
| 2 | 157.8 | 173.58 |
| 3 | 149.9 | 164.89 |
| 4 | 153.0 | 168.30 |



# Criar as subplots
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

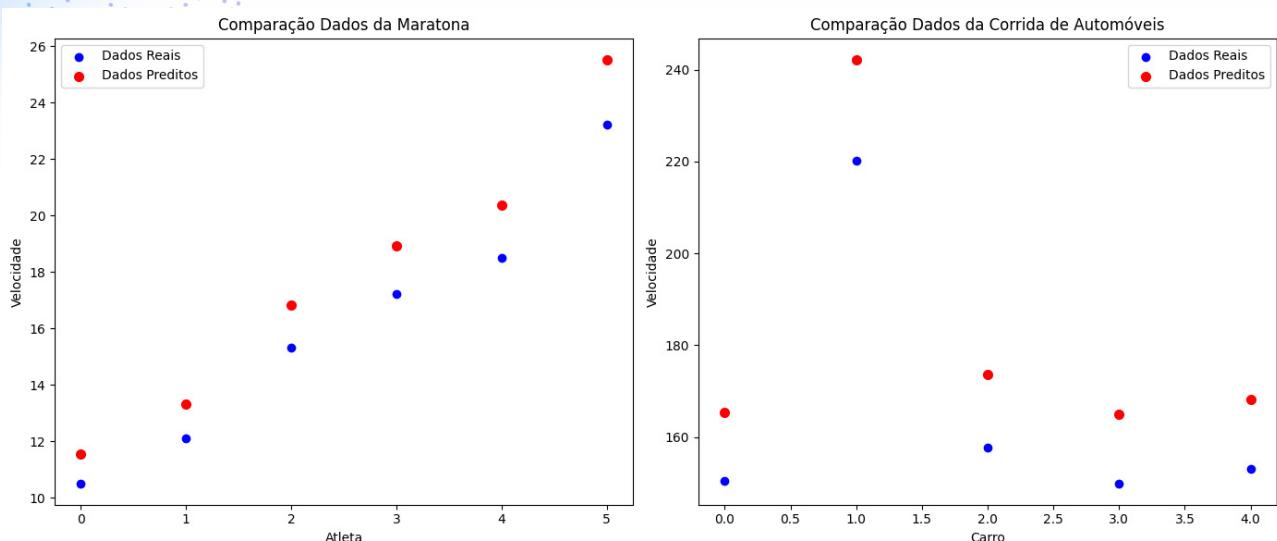
# Plotar os resíduos ao quadrado individuais
axs[0].scatter(df_maratona.index, df_maratona['y'], color='blue', label='Dados Reais')
axs[0].scatter(df_maratona.index, df_maratona['y_pred'], color='red', linewidth=2,
label='Dados Preditos')
axs[0].set_xlabel('Atleta')
axs[0].set_ylabel('Velocidade')
axs[0].set_title('Comparação Dados da Maratona')
axs[0].legend()

# Plotar os resíduos ao quadrado individuais
axs[1].scatter(df_carros.index, df_carros['y'], color='blue', label='Dados Reais')
axs[1].scatter(df_carros.index, df_carros['y_pred'], color='red', linewidth=2,
label='Dados Preditos')
axs[1].set_xlabel('Carro')
axs[1].set_ylabel('Velocidade')
axs[1].set_title('Comparação Dados da Corrida de Automóveis')
axs[1].legend()

# Mostrar os gráficos
plt.tight_layout()
plt.show()

```

Figura 64 - Comparação Dados da Corrida de Automóveis



Fonte: autoria própria.

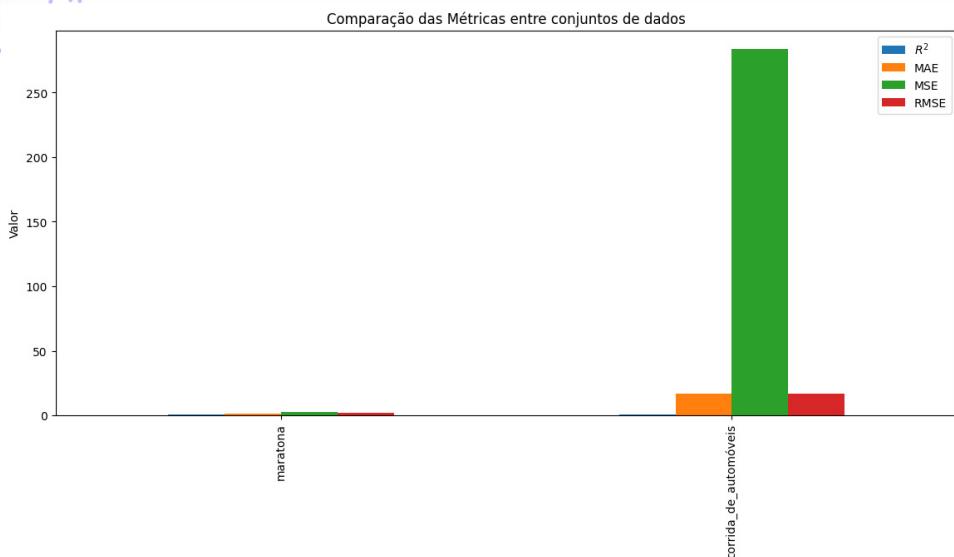
Exercício resolvido 2: Plote em um gráfico de barras os valores comparativos das métricas entre os datasets

```
# Criação de um DataFrame com os valroes das métricas
df_metrics = pd.DataFrame({'$R^2$': [r2_foot, r2_car],
                           'MAE': [mae_foot, mae_car],
                           'MSE': [mse_foot, mse_car],
                           'RMSE': [rmse_foot, rmse_car],
                           },
                           index=['maratona', 'corrida_de_automóveis'],
                           )
df_metrics
```

	\$R^2\$	MAE	MSE	RMSE
maratona	0.841793	1.613333	2.778467	1.666873
corrida_de_automóveis	0.614027	16.626000	283.776100	16.845655

```
df_metrics.plot(kind='bar', figsize=(14, 6))
plt.title('Comparação das Métricas entre conjuntos de dados')
plt.ylabel('Valor')
plt.show()
```

Figura 65 - Comparação das Métricas entre conjuntos de dados



Fonte: autoria própria.

Tipos de Regressão

Existem vários tipos de modelos de regressão, cada um com suas características e aplicações específicas:

- » **Regressão Linear:** Simples e múltipla, modela a relação linear entre variáveis. É o ponto de partida para muitos estudos de regressão.
- » **Regressão Ridge, Lasso e Elastic Net:** Incorporam penalizações para lidar com multicolinearidade e reduzir a complexidade do modelo, prevenindo sobreajuste.
- » **Árvores de Regressão:** As árvores de regressão são uma técnica de aprendizado de máquina que utiliza uma estrutura em forma de árvore para modelar a relação entre uma variável dependente contínua e uma ou mais variáveis independentes.
- » **RNN (Recurrent Neural Network) ou Rede Neural Recorrente:** é um tipo de arquitetura de rede neural projetada para processar sequências de dados. Suas principais características são: a existência de loops internos que permitem que informações persistam; sua capacidade de lidar com entradas e saídas de comprimento variável; o compartilhamento de parâmetros ao longo do tempo, reduzindo o número total de parâmetros a serem aprendidos. Especialmente útil para tarefas que envolvem dados sequenciais, como processamento de linguagem natural, reconhecimento de fala e previsão de séries temporais.

Objetivos

- » Compreender os fundamentos da regressão linear
- » Aprender a implementar e avaliar esses modelos utilizando Python.

- » Aplicar esses modelos em problemas de regressão com conjuntos de dados reais.

Introdução

A análise de regressão é uma técnica estatística fundamental utilizada para modelar a relação entre uma variável dependente e uma ou mais variáveis independentes. Este notebook abordará a Regressão Linear, porém cada método tem suas próprias características e aplicações específicas, que serão discutidas e implementadas em Python.

Regressão Linear

- » **Definição:** A regressão linear é um método estatístico para modelar a relação entre uma variável dependente e uma ou mais variáveis independentes assumindo uma relação linear.
- » **Fórmula:** $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$
- » **Interpretação:** Utilizada quando se espera que a variável dependente varie linearmente com as variáveis independentes.

Explicação Detalhada da Regressão Linear

A regressão linear simples assume a seguinte fórmula:

$$y = \beta_0 + \beta_1 X + \epsilon$$

onde:

y é a variável dependente.

X é a variável independente.

β_0 é o intercepto (valor de y quando $X = 0$).

β_1 é o coeficiente de inclinação (quanto y muda com uma unidade de mudança em X).

ϵ é o termo de erro (a diferença entre os valores observados e os valores preditos).

Passo 1: Carregando Biblioteca e dados

Vamos gerar alguns dados para exemplificar o processo de regressão linear.

Nesse exemplo é proposta a criação de um modelo linear para ser possível prever qual seria o valor esperado de produção de uma máquina conforme fosse o ajuste de sua velocidade de produção.

- » Em X temos a velocidade de produção de uma máquina
- » Em y temos a quantidade de produtos produzidos

A máquina real possui uma série de variáveis que a influenciam, por isso só observar só 2 pontos e concluir algo é muito perigoso.

Os dados gerados nesse exemplo são aleatórios, com o uso do `random.seed` e seguem a mesma técnica utilizada ao longo desse notebook.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Gerar dados aleatórios
np.random.seed(42)
X = 2 * np.random.rand(10, 1)
# Ordena os dados de X
X = sorted(X)
# Adequa o formato
X = np.array([arr.tolist() for arr in X])

# Cria os valores de y observados
y = 4 + 3 * X + np.random.randn(10, 1)
```

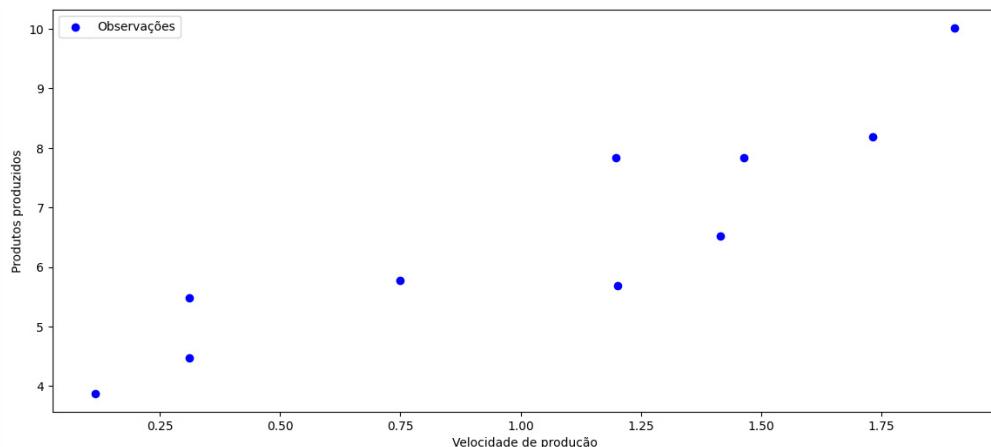
Passo 2: Plotagem dos dados

Plotar os dados em um gráfico ajuda a entender todo o processo. É possível ver no gráfico, que quanto maior a velocidade de produção, maior a produção. Mas, com base no que se tem até agora, só é possível saber a produção para o conjunto testado.

O modelo de regressão visa resolver a questão: Qual seria o valor esperado de produção para um tempo que ainda não teste?

```
fig, axs = plt.subplots(1, 1, figsize=(14, 6))
axs.scatter(X,y, color='blue', label='Observações')
# Adiciona a legenda
plt.legend()
# Adiciona rótulos dos eixos
plt.xlabel("Velocidade de produção")
plt.ylabel("Produtos produzidos")
plt.show()
```

Figura 66 - Gráfico para exemplificar o processo de Regressão Linear



Fonte: autoria própria.

Passo 3: Testa-se um modelo

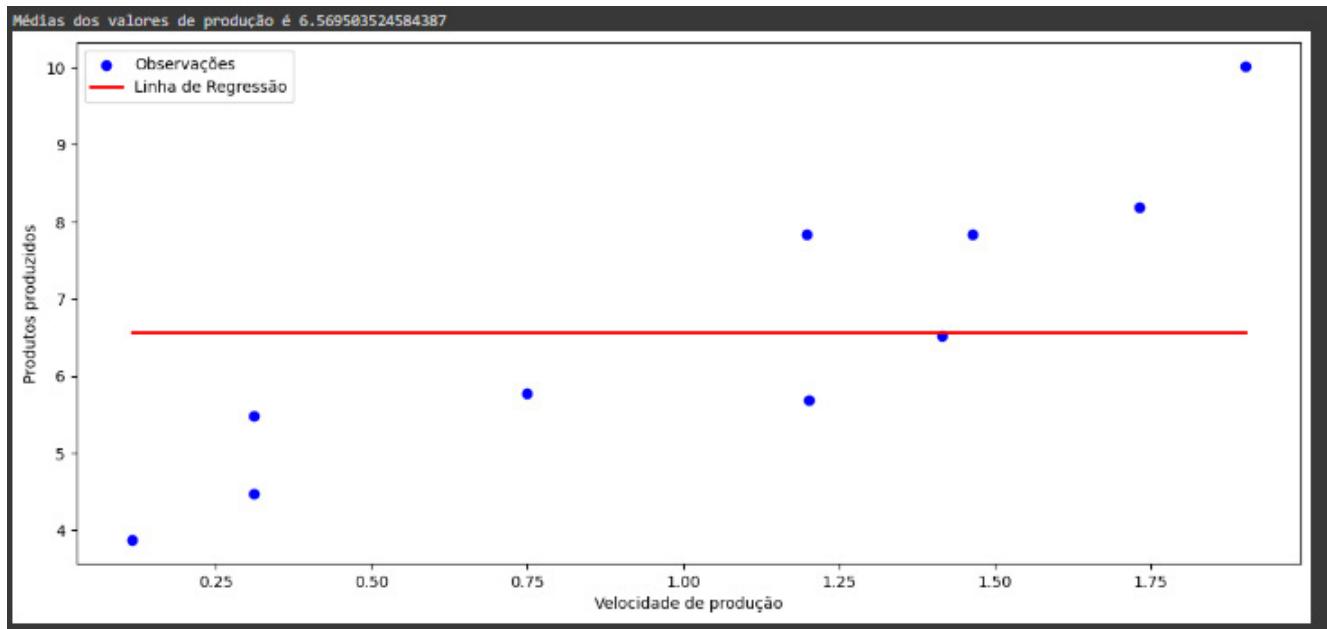
Um primeiro modelo mais intuitivo é encontrar a média dos valores e adotar esse como se fosse o modelo.

```
# Primeiro modelo: calcula-se a média dos valores de y
y_mean = np.mean(y)
print(f"Medias dos valores de produção é {y_mean}")

# Repetindo esse valor para cada valor de X, temos um vetor com a média
# y_pred é o valor predito para os valores X, utilizando um modelo que retorna a média
# dos valores
y_pred = np.array([y_mean for _ in range(len(X))])

# Plotando no gráfico
fig, axs = plt.subplots(1, 1, figsize=(14, 6))
# Plota os pontos
axs.scatter(X,y, color='blue', label='Observações')
# Plota o modelo
axs.plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')
# Adiciona a legenda
plt.legend()
# Adiciona rótulos dos eixos
plt.xlabel("Velocidade de produção")
plt.ylabel("Produtos produzidos")
plt.show()
```

Figura 67 - Gráfico de Plotagem dos Dados



Fonte: autoria própria.

Passo 4: Encontrando a soma do quadrado dos resíduos

1. Para cada ponto (azul) é verificado então sua distância ao modelo (vermelho).
2. Eleva-se ao quadrado cada uma dessas distâncias (diferenças)
3. Soma-se esses quadrado das diferenças

```
# Distância ponto ao modelo
residuais = y_pred - y

# Residuais ao quadrado
residuais_sq = np.power(residuais, 2)

# Soma do quadrado dos resíduos
soma_residuos_sq = np.sum(residuais_sq)

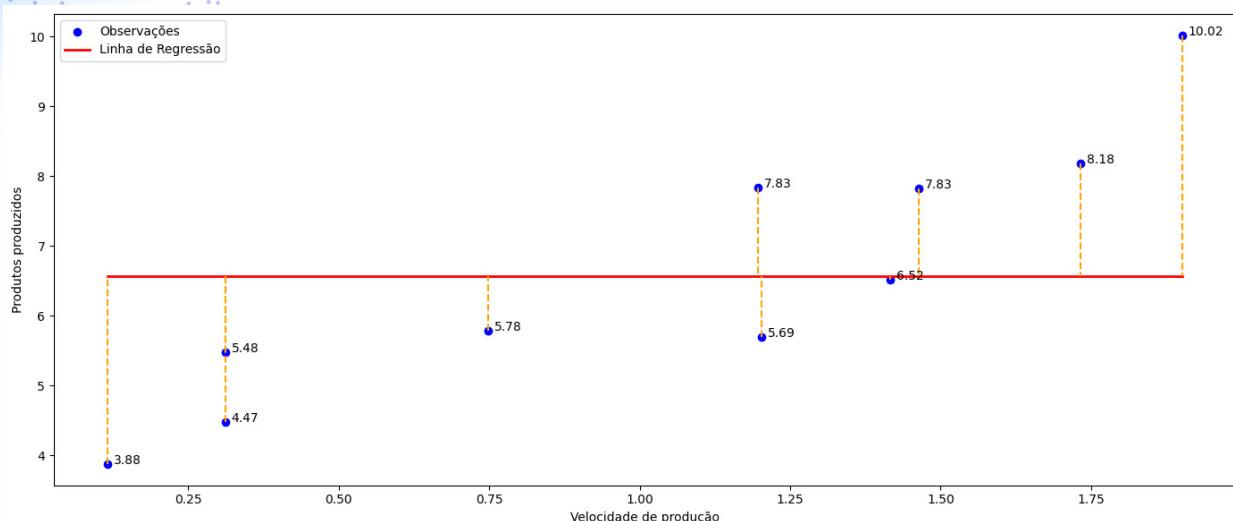
print(f"Soma do quadrado dos resíduos (SSR): {soma_residuos_sq.item():.2f}")
Soma do quadrado dos resíduos (SSR): 319.06
```

Para ilustrar o andamento, plotam-se as linhas de diferença (laranja) entre o valor real (pontos em azul) e o valor que seria no modelo (linha em vermelho).

```
# Plotando no gráfico
fig, axs = plt.subplots(1, 1, figsize=(14, 6))
# Plota os pontos
axs.scatter(X,y, color='blue', label='Observações')
# Plota o modelo
axs.plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')

# Plota as distâncias entre pontos e modelo
for i in range(len(X)):
    # Plota as linhas das distâncias em laranja
    axs.plot([X[i], X[i]], [y[i], np.array([y_pred[i]])], color='orange',
    linestyle='dashed')
    # Adiciona o rótulo do valor observado
    axs.text(x=X[i]+.01, y=y[i], s=str(np.round([y[i]][0],2)))
# Adiciona a legenda
plt.legend()
# Adiciona rótulos dos eixos
plt.xlabel("Velocidade de produção")
plt.ylabel("Produtos produzidos")
plt.tight_layout()
plt.show()
```

Figura 68 - Gráfico da Soma do Quadrado dos Resíduos



Fonte: autoria própria.

Passo 5: Testa-se um novo modelo (gira a reta vermelha) e verifica-se o SSR (soma do quadrado dos resíduos).

Um novo modelo poderia ser:

$$y = 4 + 1.5 \cdot X$$

Então aplica-se o Passo 4 novamente.

```
# Novo modelo
y_pred = 4 + 1.5 * X
```

Repetindo o passo 4

```
# Distância ponto ao modelo
residuais = y_pred - y

# Residuais ao quadrado
residuais_sq = np.power(residuais, 2)

# Soma do quadrado dos resíduos
soma_residuos_sq = sum(residuais_sq)

print(f"Soma do quadrado dos resíduos (SSR): {soma_residuos_sq.item():.2f}")
```

Soma do quadrado dos resíduos (SSR): 21.08

```
# Plotando no gráfico
fig, axs = plt.subplots(1, 1, figsize=(14, 6))
# Plota os pontos
axs.scatter(X, y, color='blue', label='Observações')
# Plota o modelo
axs.plot(X, y_pred, color='red', linewidth=2, label='Linha de Regressão')

# Plota as distâncias entre pontos e modelo
for i in range(len(X)):
```

continua

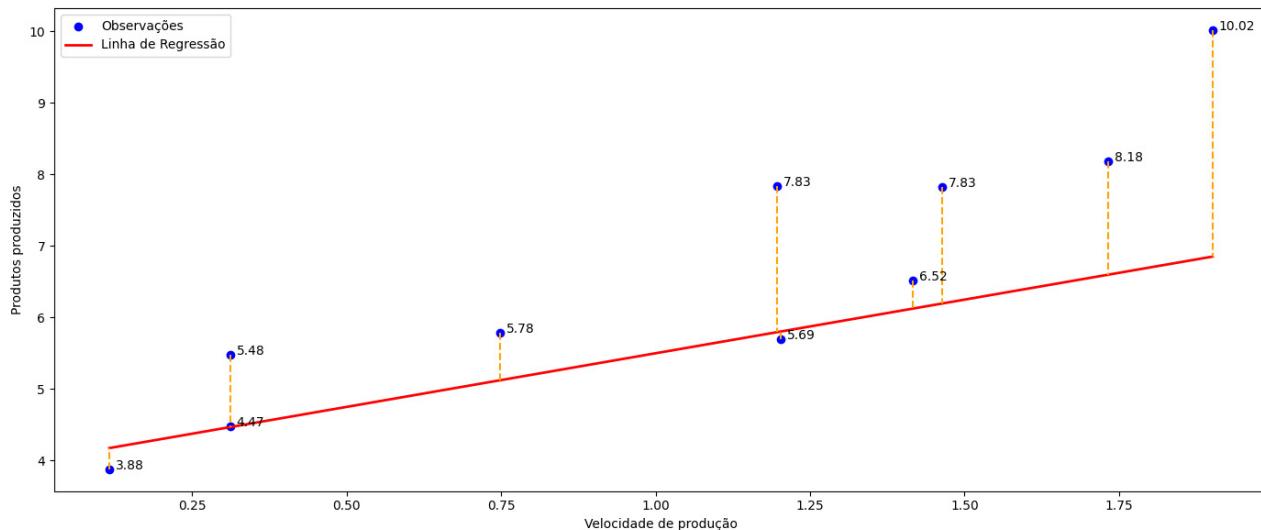
```

# Plota as linhas das distâncias
axs.plot([X[i], X[i]], [y[i], y_pred[i]], color='orange', linestyle='dashed')
# Adiciona o rótulo do valor observado
axs.text(x= X[i]+.01, y= y[i], s=str(np.round([*y[i]][0],2)))

# Adiciona a legenda
plt.legend()
# Adiciona rótulos dos eixos
plt.xlabel("Velocidade de produção")
plt.ylabel("Produtos produzidos")
plt.tight_layout()
plt.show()

```

Figura 69 - Gráfico do SSR do Novo Modelo



Fonte: autoria própria.

Na primeira iteração foi obtido SSR de 31.91.

Na segunda iteração foi obsido valor de SSR de 21.08, portanto o segundo modelo é melhor que o primeiro.

Esse processo de teste de modelos se repete até que se encontre o valor mínimo do SSR. E esse processo otimização que identifica o menor SSR chama-se **Método dos mínimos quadrados**

Regressão com uso do sklearn

Aqui será abordado o emprego direto de modelos de regressão utilizando a biblioteca sklearn.

Esses modelos são de simples implementação, porém é necessario o entendimento de seu funcionamento, o ajuste de parâmetros, portanto a documentação deve ser sempre consultada.

Serão abordados os algoritmos: regressão linear, regressão ridge, árvores de decisão e redes neurais recorrentes.

Criação do dataset para treino dos modelos

O conjunto de dados Diabetes disponível através do `load_diabetes()` do Scikit-learn é um conjunto de dados padrão, frequentemente utilizado para fins de aprendizado e demonstração em tarefas de machine learning. Ele contém informações sobre pacientes com diabetes, incluindo diversas características médicas como idade, sexo, índice de massa corporal (IMC), pressão arterial, níveis de glicose, etc.

Suas principais características são:

- » **Dados reais:** Os dados são provenientes de um estudo real sobre diabetes, o que o torna um conjunto de dados realista para testar algoritmos de aprendizado de máquina.
- » **Variedade de features:** O dataset possui um número considerável de features, o que permite explorar diferentes técnicas de seleção e engenharia de features.
- » **Target:** O objetivo (target) é uma medida quantitativa da progressão da doença, o que o torna um problema de regressão.

```
# Importação da biblioteca
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Carregar um conjunto de dados de exemplo
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
# Em X ficam os dados de características que serão os preditores do progresso da doença
X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
# Em y, ficam os valores de progresso da doença
y = diabetes.target

# Explicação do dataset
print(diabetes.DESCR)

.. _diabetes_dataset:

Diabetes dataset
-----
Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
```

[continua](#)

quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- age age in years
- sex
- bmi body mass index
- bp average blood pressure
- s1 tc, total serum cholesterol
- s2 ldl, low-density lipoproteins
- s3 hdl, high-density lipoproteins
- s4 tch, total cholesterol / HDL
- s5 ltg, possibly log of serum triglycerides level
- s6 glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_samples` (i.e. the sum of squares of each column totals 1).

Source URL:

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>

For more information see:

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004)
"Least Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

```
# Para ilustrar melhor cada modelo, será utilizado apenas a característica de IMC  
(bmi)  
# Nesse caso, estará sendo utilizado os dados de IMC para treinar um modelo que,  
# com base apenas no valor do IMC consiga predizer o valor da progressão da doença  
col = 'bmi'  
  
# Dividir os dados em treino (80%) e teste (20%)  
# utilização do random_state para manter a reprodutibilidade  
X_train, X_test, y_train, y_test = train_test_split(X[col].to_numpy(), y, test_size=0.2, random_state=42)  
  
# Ordena os dados para facilitar a plotagem  
# Coloca X_test e y_test em uma tupla e ordena pela X_test  
z = sorted(zip(X_test,y_test))  
# "Descompacta" a lista ordenada em X_test e y_test  
X_test,y_test = zip(*z)  
# Transforma a tupla de valores X-test e y_tes em um array.  
X_test = np.array(X_test)  
y_test = np.array(y_test)
```

Função para relatório

Para evitar repetição foi definida uma função para imprimir um relatório de performance do modelo.

O relatório consta de 2 partes: Gráfico e Métricas.

Gráfico:

- » No eixo x tem-se os valores da característica, no caso proposto bmi ou IMC.
- » No eixo y, tem-se os valores da progressão da doença.
- » Os pontos pretos correspondem os dados de Teste
- » A linha colorida representa o modelo ajustado aos dados de Teste

Métricas: Faz a comparação das métricas R^2 , MSE e MAE de performance do modelo entre os conjuntos de Treino e Teste.

```
def relatorio(X_test, y_test, y_train, y_pred_model, y_train_model, modelo, col, color_model):  
    # Plot outputs  
    plt.scatter(X_test, y_test, color="black")  
    plt.plot(X_test, y_pred_model, color=color_model, linewidth=3)  
    plt.xticks(())  
    plt.yticks(())  
    plt.title(f"{modelo} treinado com dados apenas de {col}")  
    plt.xlabel(col)  
    plt.ylabel("Progressão da doença")  
    plt.show()  
  
    # Avaliação das Métricas  
    print(f"{modelo} - R² - Treino {r2_score(y_train, y_train_model):.4f} - Teste {r2_ score(y_test, y_pred_model):.4f}")  
    print(f"{modelo} - MSE - Treino {mean_squared_error(y_train, y_train_model):.4f} - Teste {mean_squared_error(y_test, y_pred_model):.4f}")  
    print(f"{modelo} - MAE - Treino {mean_absolute_error(y_train, y_train_model):.4f} - Teste {mean_absolute_error(y_test, y_pred_model):.4f}")
```

Regressão Linear

Utilizando a biblioteca do sklearn, abaixo é treinado um modelo de regressão linear com base no valores de treino do IMC **X_train** e de progressão da doença **y_train**.

- » São obtidos os valores preditos para o **X_test** e guardado em **y_pred_lin**.
- » São obtidos os valores preditos para o **X_train** e guardado em **y_train_lin**.
- » São verificadas as métricas de R^2 , MSE e MAE para os conjuntos de Treino e Teste.

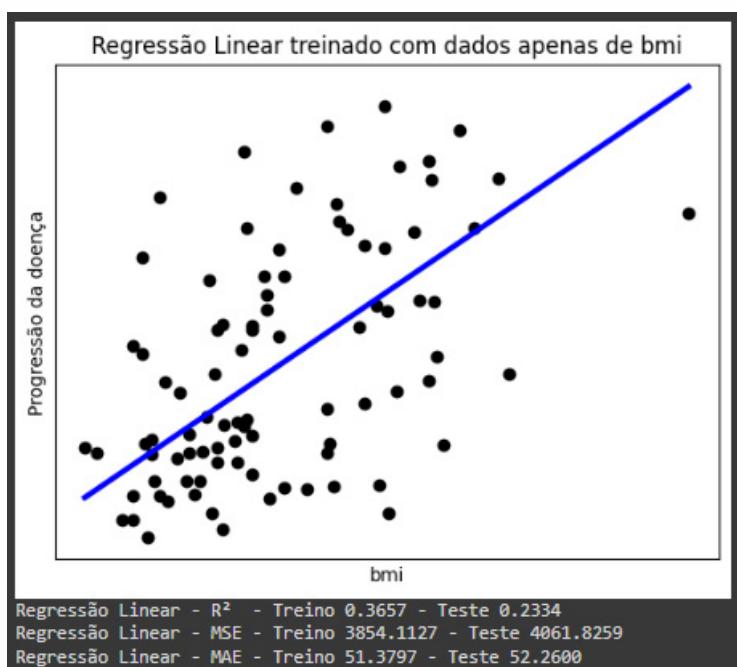
```

# Modelo de Regressão Linear
lin_reg = LinearRegression()
# Treino do modelo
lin_reg.fit(X_train.reshape(-1, 1), y_train)
# Valores da predição com base nos dados de Teste
y_pred_lin = lin_reg.predict(X_test.reshape(-1, 1))
# Valores da predição com base nos dados de Treino
y_train_lin = lin_reg.predict(X_train.reshape(-1, 1))

# Imprime o relatório do modelo
modelo = 'Regressão Linear'
color_model = 'blue'
relatorio(X_test, y_test, y_train, y_pred_lin, y_train_lin, modelo, col, color_model)

```

Figura 70 - Gráfico de Regressão Linear treinado com dados apenas de bmi



Fonte: autoria própria.

Regressão Ridge

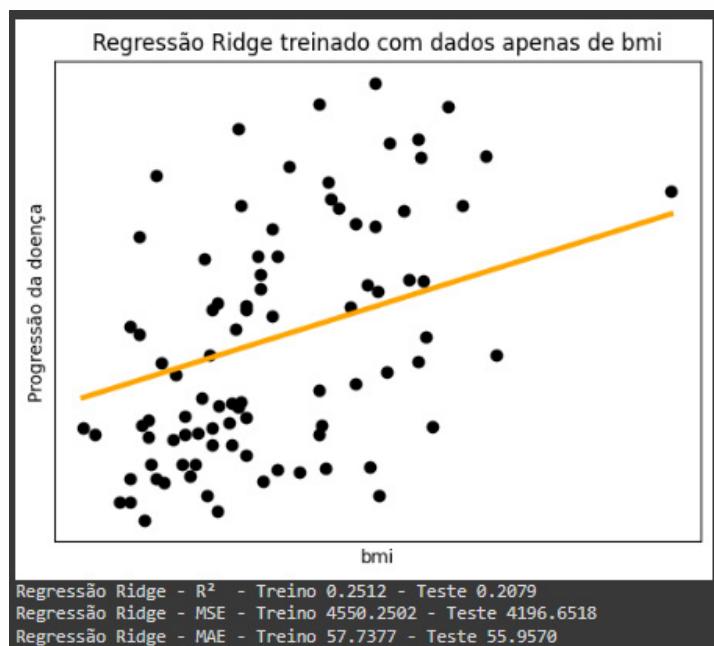
Utilizando a biblioteca do sklearn, abaixo é treinado um modelo de regressão ridge com base no valores de treino do IMC **X_train** e de progressão da doença **y_train**.

- » São obtidos os valores preditos para o **X_test** e guardado em **y_pred_ridge**.
- » São obtidos os valores preditos para o **X_train** e guardado em **y_train_ridge**.
- » São verificadas as métricas de R^2 , MSE e MAE para os conjuntos de Treino e Teste.

```
# Modelo de Regressão Ridge
ridge_reg = Ridge(alpha=1.0)
# Treino do modelo
ridge_reg.fit(X_train.reshape(-1, 1), y_train)
# Valores da predição com base nos dados de Teste
y_pred_ridge = ridge_reg.predict(X_test.reshape(-1, 1))
# Valores da predição com base nos dados de Treino
y_train_ridge = ridge_reg.predict(X_train.reshape(-1, 1))
```

```
# Imprime o relatório do modelo
modelo = 'Regressão Ridge'
color_model = 'orange'
relatorio(X_test, y_test, y_train, y_pred_ridge, y_train_ridge, modelo, col, color_
model)
```

Figura 71 - Gráfico de Regressão Ridge treinado com dados apenas de bmi



Fonte: autoria própria.

Árvore de Regressão

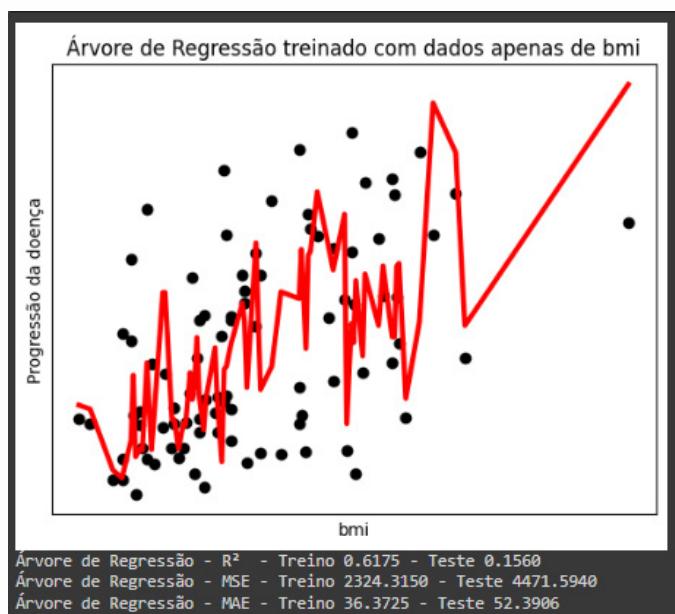
Utilizando a biblioteca do sklearn, abaixo é treinado um modelo de regressão ridge com base no valores de treino do IMC **X_train** e de progressão da doença **y_train**.

- » São obtidos os valores preditos para o **X_test** e guardado em **y_pred_tree**.
- » São obtidos os valores preditos para o **X_train** e guardado em **y_train_tree**.
- » São verificadas as métricas de R^2 , MSE e MAE para os conjuntos de Treino e Teste.

```
# Modelo de Árvore de Regressão
tree_reg = DecisionTreeRegressor(random_state=42)
# Treino do modelo
tree_reg.fit(X_train.reshape(-1, 1), y_train)
# Valores da predição com base nos dados de Teste
y_pred_tree = tree_reg.predict(X_test.reshape(-1, 1))
# Valores da predição com base nos dados de Treino
y_train_tree = tree_reg.predict(X_train.reshape(-1, 1))
```

```
# Imprime o relatório do modelo
modelo = 'Árvore de Regressão'
color_model = 'red'
relatorio(X_test, y_test, y_train, y_pred_tree, y_train_tree, modelo, col, color_model)
```

Figura 72 - Gráfico da Árvore de Regressão treinado com dados apenas de bmi



Fonte: autoria própria.

Redes Neurais Recorrentes (RNN)

Utilizando a biblioteca do sklearn, abaixo é treinado um modelo de regressão ridge com base no valores de treino do IMC **X_train** e de progressão da doença **y_train**.

A RNN necessita de uma formatação adicional nos vetores, portanto é realizado essa adaptação transformando **X_train** de dimensão (353,) em **X_train_rnn** de dimensão (353,1,1). De forma análoga é transformado **X_test** em **X_test_rnn**.

- » São obtidos os valores preditos para o **X_test_rnn** e guardado em **y_pred_rnn**.
- » São obtidos os valores preditos para o **X_train_rnn** e guardado em **y_train_rnn**.

```

# Preparação dos dados para RNN
X_train_rnn = X_train.reshape((X_train.shape[0], 1, 1))
X_test_rnn = X_test.reshape((X_test.shape[0], 1, 1))

# Cria o Modelo de RNN empilhando diversas camadas
# Esta arquitetura cria uma rede neural recorrente com duas camadas RNN,
# seguida por duas camadas densas para processamento não linear
# e uma camada de saída para a predição.
model = Sequential([
    SimpleRNN(64, activation='tanh', return_sequences=True, input_shape=(X_train_rnn.
shape[1], X_train_rnn.shape[2])),
    SimpleRNN(64, activation='tanh'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

# Treinamento
model.fit(X_train_rnn, y_train, epochs=500, verbose=0)

# Previsões
y_pred_rnn = model.predict(X_test_rnn)
y_train_rnn = model.predict(X_train_rnn)

/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
    super().__init__(**kwargs)
3/3 ━━━━━━━━━━ 1s 141ms/step
12/12 ━━━━━━━━ 0s 3ms/step

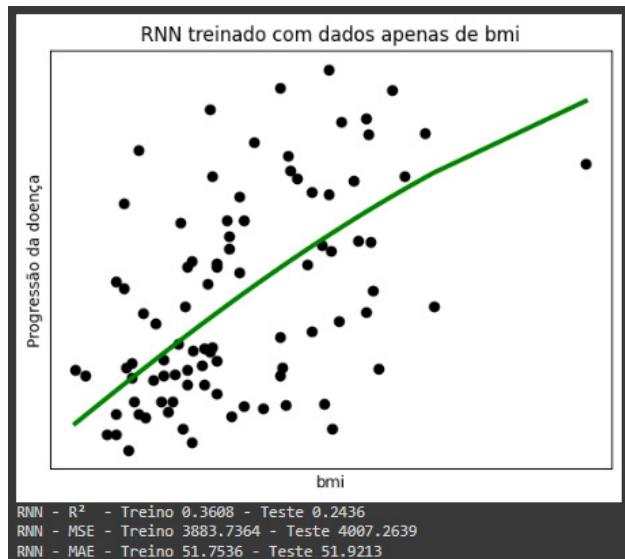
```

```

# Imprime o relatório do modelo
modelo = 'RNN'
color_model = 'green'
relatorio(X_test, y_test, y_train, y_pred_rnn, y_train_rnn, modelo, col, color_model)

```

Figura 73 - Gráfico RNN treinado com dados apenas de bmi



Fonte: autoria própria.

Conclusões

Foram testados 4 modelos distintos, sendo: Regressão Linear, Regressão Ridge, Árvore de Regressão e uma RNN.

Para cada modelo foram verificados as métricas para os conjuntos de Teste e Treino, bem como foi apresentado um gráfico do modelo ajustado ao conjunto de Teste.

Análise Crítica dos Resultados de Desempenho

A análise dos resultados de desempenho dos modelos de regressão linear, regressão Ridge, árvore de regressão e redes neurais recorrentes (RNN) revela importantes insights sobre o comportamento de cada modelo e sua adequação ao conjunto de dados. Abaixo, são discutidos aspectos relacionados ao desempenho, possíveis problemas de overfitting e a escolha do melhor modelo a ser aplicado.

- » A **regressão linear** apresenta um desempenho modesto, com uma redução significativa do R^2 no conjunto de teste, indicando que o modelo tem uma capacidade limitada de generalização. A proximidade entre os erros absolutos médios (MAE) de treino e teste sugere que o modelo não está superajustado (overfitted), mas também não é suficientemente robusto para capturar toda a variabilidade dos dados.
- » A **regressão Ridge**, uma variação regularizada da regressão linear, tem um desempenho ainda mais modesto, com valores de R^2 mais baixos tanto para treino quanto para teste. O maior valor de MSE no treino, comparado ao MSE do teste, sugere que a regularização pode estar penalizando excessivamente os coeficientes, levando a um underfitting.
- » A **árvore de regressão** apresenta um alto R^2 no conjunto de treino, mas um desempenho significativamente pior no teste, com uma queda abrupta no R^2 e um aumento no MSE. Isso é um forte indicador de overfitting, onde o modelo se ajusta muito bem aos dados de treino, mas não generaliza bem para novos dados.
- » A **RNN** apresenta valores de R^2 e MSE próximos tanto no treino quanto no teste. Isso é um indicador de que não há overfitting. Comparando graficamente a árvore de decisão e RNN, fica claro a questão do overfitting na árvore de decisão. A RNN mostra um desempenho muito semelhante ao da regressão linear no treino porém com um custo computacional muito mais elevado. Isso decorre principalmente porque no caso apresentado, o problema com apenas uma característica (bmi), não apresenta grandes dificuldades para a Regressão Linear, a RNN provavelmente obteria melhores resultados se todas as características do dataset fossem utilizadas.

Referências Bibliográficas

- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.
- McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics.



SAIBA MAIS...

- » Regressão Linear - <https://youtu.be/7ArmBVF2dCs?si=XV-Ax7DZz0sN2Wx1>
- » Regressão Linear - <https://www.youtube.com/watch?v=PaFPbb66DxQ>
- » Regressão Ridge - <https://youtu.be/Q81RR3yKn30?si=K-TkT-W69SAM60uK>
- » Regressão Lasso - <https://youtu.be/NGf0voTMlcs?si=T8z8AlNTh8zl6iZF>
- » Comparação entre Ridge e Lasso
 - » https://youtu.be/Xm2C_gTAI8c?si=Ti2D0PTfnJAl1e4B
- » Árvores de regressão
 - » <https://youtu.be/g9c66TUylZ4?si=-fP85nMbFRgT2nw7>
 - » R_2 - <https://youtu.be/bMccdk8EdGo?si=zRO2xrswQPq6xSzs>
 - » R_2 - <https://youtu.be/aP8YP1aX2zc?si=WDFJh6kkSNPjdtgO>
 - » Regressão Linear - <https://youtu.be/bMccdk8EdGo?si=zRO2xrswQPq6xSzs>
 - » Regressão Linear - https://youtu.be/Wo9Vt7Sc_E0?si=hd7X7u5QX-2m57AI

PARA RELEMBRAR...

Regressão: Técnica estatística para modelar relações entre variáveis dependentes contínuas e variáveis independentes.

Importância: Fundamental para previsões quantitativas em diversos campos como economia, finanças e indústria.

Regressão vs. Classificação:

- » Regressão: Predição de variáveis contínuas (ex.: preços, temperaturas)
- » Classificação: Atribuição de instâncias a categorias discretas (ex.: spam ou não spam)

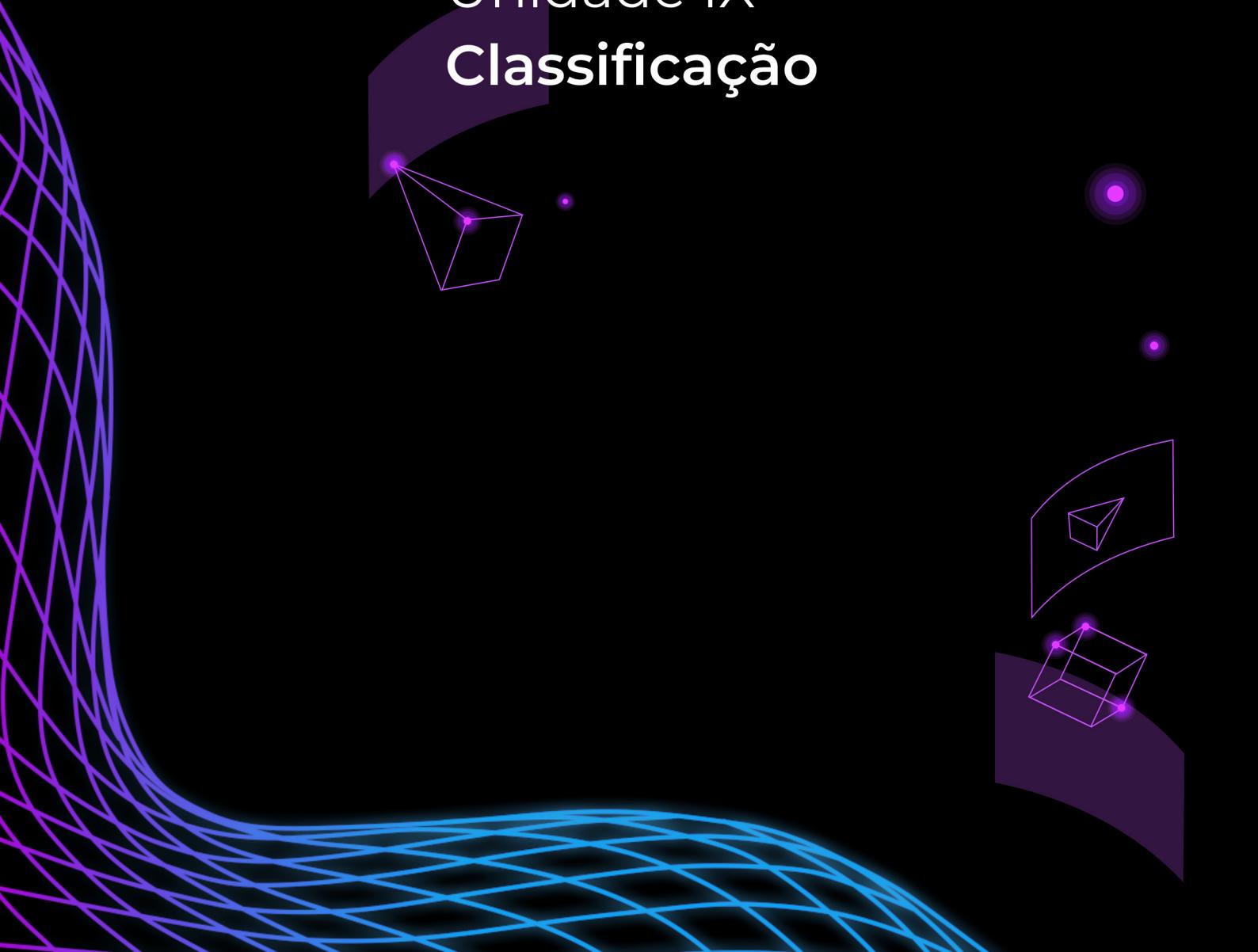
Métricas de Avaliação:

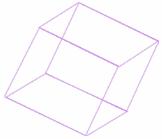
- » R^2 : Coeficiente de Determinação
- » MSE: Erro Quadrático Médio
- » MAE: Erro Absoluto Médio
- » RMSE: Raiz do Erro Quadrático Médio

Tipos de Regressão:

- » Linear: Simples e múltipla
- » *Ridge, Lasso* e *Elastic Net*: Incluem penalizações
- » Árvores de Regressão: Estrutura em forma de árvore
- » RNN: Para processar sequências de dados

Unidade IX Classificação





Unidade IX - Classificação

Como já foi comentado nas unidades anteriores deste eBook, um dos problemas comuns no Aprendizado de Máquina é a tarefa de identificar as classes associadas a um determinado objeto. A partir de suas características, também chamadas de variáveis, utilizamos métodos computacionais inteligentes para tentar determinar a qual classe esse objeto pertence. Podemos usar como exemplo a simples tarefa de identificar se um determinado e-mail é spam ou não. Um e-mail pode conter texto, imagens e arquivos diversos, que são suas variáveis, e pode ser interpretado como spam ou não spam. O que determina se um e-mail é spam ou não é uma interpretação dessas variáveis de acordo com o interesse do dono da caixa de e-mail. Para uma pessoa, um e-mail de propaganda de um pet shop, por exemplo, pode ser irrelevante, logo, seria classificado como spam; mas para outra pessoa, pode ser de interesse, logo, não seria considerado spam. Dessa forma, classificar diferentes mensagens de e-mail entre as duas classes (spam e não spam) é uma tarefa que exige a interpretação das variáveis a cada nova mensagem, compondo assim uma tarefa de classificação de mensagens.

As tarefas de classificação são comuns por serem amplamente utilizadas na automação de processos que envolvem a interpretação de dados de diversos tipos. Se dividirmos por tipo de dado de entrada, podemos listar alguns exemplos de aplicações.

Figura 74 - Exemplos de aplicações de classificação

IMAGENS:
RECONHECIMENTO DE OBJETOS
Identificar objetos em imagens, como distinguir entre gatos, cães, carros, pessoas, etc.

IMAGENS:
RECONHECIMENTO FACIAL
Identificar e classificar rostos em imagens em sistemas de segurança.



IMAGENS:

DETECÇÃO DE DOENÇAS

Classificar imagens médicas (como radiografias, tomografias ou ressonâncias magnéticas) para identificar a presença ou ausência de doenças, como câncer, pneumonia ou outras condições.



LINGUAGEM NATURAL:

ANÁLISE DE SENTIMENTO

Classificar textos (como comentários em redes sociais ou avaliações de produtos) como positivos, negativos ou neutros.



LINGUAGEM NATURAL:

CLASSIFICAÇÃO DE TÓPICOS

Classificar textos em categorias específicas, como política, esportes, tecnologia, etc.



ÁUDIO:

DETECÇÃO DE ATIVIDADE

Classificar trechos de áudio entre voz humana e outros sons.



DADOS TABULARES:
DETECÇÃO DE FRAUDE
Classificar transações financeiras como legítimas ou fraudulentas.

DADOS TABULARES:
AVALIAÇÃO DE RISCO DE CRÉDITO
Classificar candidatos a empréstimos como de alto ou baixo risco de inadimplência.

DADOS TABULARES:
PREVISÃO DE CHURN
Classificar clientes com maior probabilidade de cancelar um serviço ou assinatura.

Fonte: autoria própria.

9.1 Classificadores

O elemento principal ao tratar um problema de classificação é o classificador. Classificadores são métodos de Aprendizado de Máquina que, a partir das variáveis extraídas de um objeto, tentam determinar a qual classe ele pertence. Esses métodos são desenvolvidos com o objetivo de compreender as possíveis variações dentro de uma classe. Por exemplo, em uma tarefa de reconhecimento de gatos a partir de uma imagem, podemos ter diversas variações, como posição, raça, cor, tamanho e formato, conforme ilustrado na Figura 45. Todas essas variações geram variáveis com algumas diferenças, mas que ainda assim representam exemplos da mesma classe. O objetivo dos classificadores é identificar os padrões comuns a cada classe (dois olhos, bigodes, orelhas, quatro patas, etc.) enquanto tentam generalizar esses padrões, considerando as possíveis variações. Assim, um bom classificador de gatos deve ser capaz de generalizar a definição de gato e reconhecer todos os diferentes bichanos.

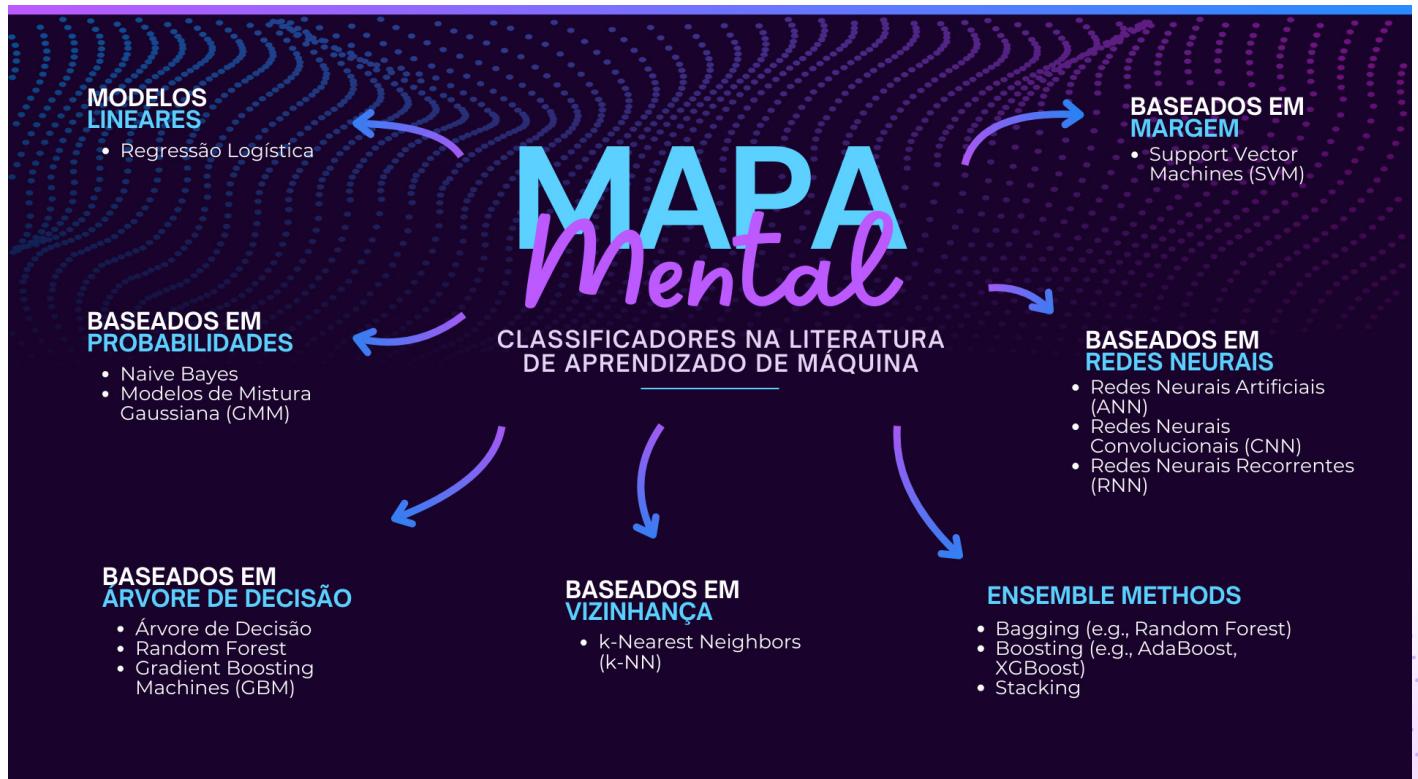
Figura 75 - Exemplos de diferentes amostras de uma classe “gato”



Fonte: autoria própria.

Na literatura de Aprendizado de Máquina temos diversos classificadores. Podemos categorizar estes métodos a partir do princípio de funcionamento de cada um como mostra o mapa mental a seguir.

Figura 76 - Mapa mental classificadores na literatura de aprendizado de máquina



Fonte: autoria própria.

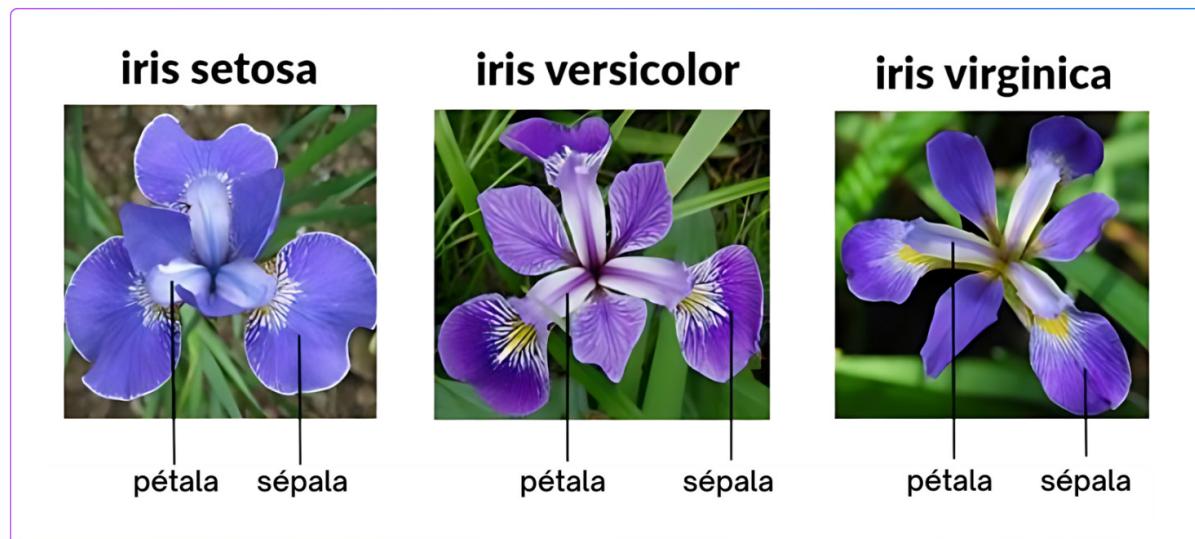
Para entendermos melhor vamos estudar alguns destes métodos de forma mais detalhada começando pelo KNN, ou k-Vizinhos Mais Próximos em português, que realiza a classificação com base na vizinhança de cada amostra de dados.

9.1.1 KNN: *k*-Nearest Neighbors

O KNN é um classificador clássico do Aprendizado de Máquina e apesar de possuir funcionamento simples e intuitivo é muito utilizado. O conceito principal do KNN é que os dados utilizados no treinamento do classificador delimitam a região do espaço de variáveis que compõem as classes com base na proximidade. Para melhor visualizar esta propriedade vamos usar como exemplo o dataset Íris .

O dataset Íris é composto por 50 amostras de três diferentes espécies de flores de plantas Íris, onde cada flor teve sua sépala e pétala medida. As partes das flores e diferentes espécies estão na imagem abaixo. O objetivo aqui é diferenciar as espécies de acordo com as características das suas flores.

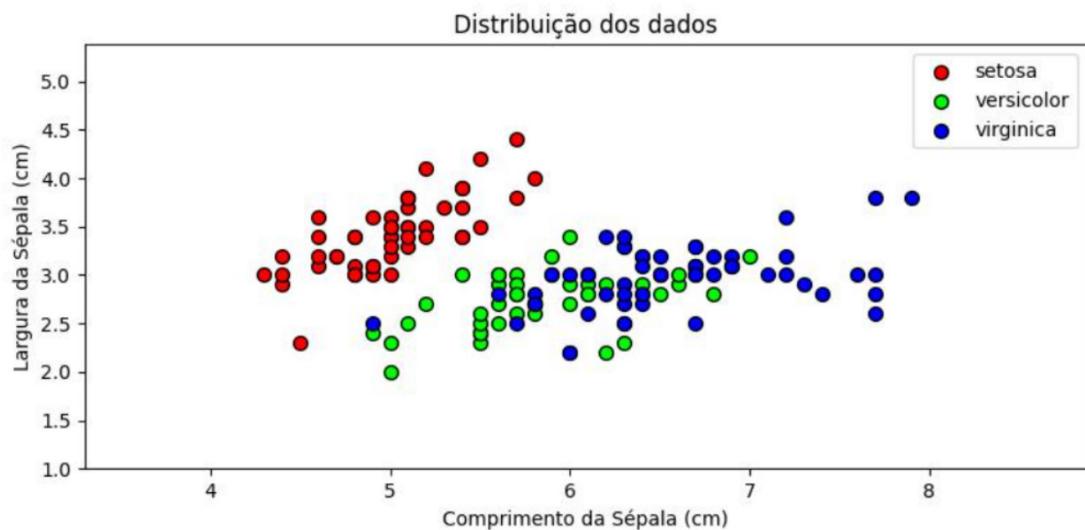
Figura 77 - Exemplos das três espécies de flores Iris



Fonte: adaptada de [Leitura de um dataset, 2020](#).

Se visualizarmos a distribuição dos dados em torno de duas das variáveis (comprimento e largura da pétala) podemos observar que as diferentes espécies formam regiões no espaço das variáveis. A espécie setosa apresenta uma sobreposição quando olhamos apenas o comprimento (eixo x) ou largura (eixo y), mas usando ambas as variáveis fica claro o surgimento de uma região própria da classe. O KNN atua justamente nesse conceito de vizinhança classificando um novo dado a partir dos k vizinhos mais próximos. Vizinhos são os dados já existentes na base de treinamento e que estão rotulados (já sabemos a sua resposta), a partir destes são encontrados os k mais próximos e feita uma votação para decidir qual classe atribuir a um novo dado. O valor de k é atribuído no processo de parametrização do algoritmo e deve ser otimizado para obter o melhor resultado.

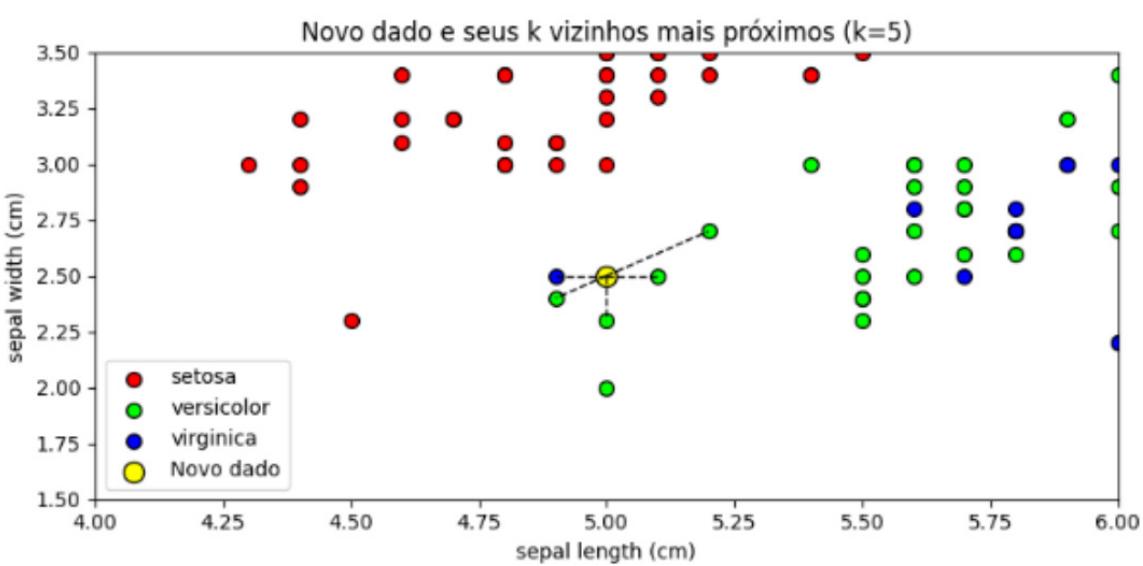
Figura 78 - Distribuição das amostras de flores com base no comprimento e largura da sépala



Fonte: autoria própria.

Dessa forma, se imaginarmos uma nova flor cuja pétala foi medida em termos de largura e comprimento, ao utilizarmos o KNN com valor de k igual a 5, iremos buscar os 5 exemplos de pétalas já conhecidos mais próximos para definir qual classe deve ser atribuída a esse novo exemplo. Na figura a seguir podemos ver um exemplo deste caso, onde o novo ponto será atribuído a versicolor por possuir 4 vizinhos mais próximos que faz parte desta classe.

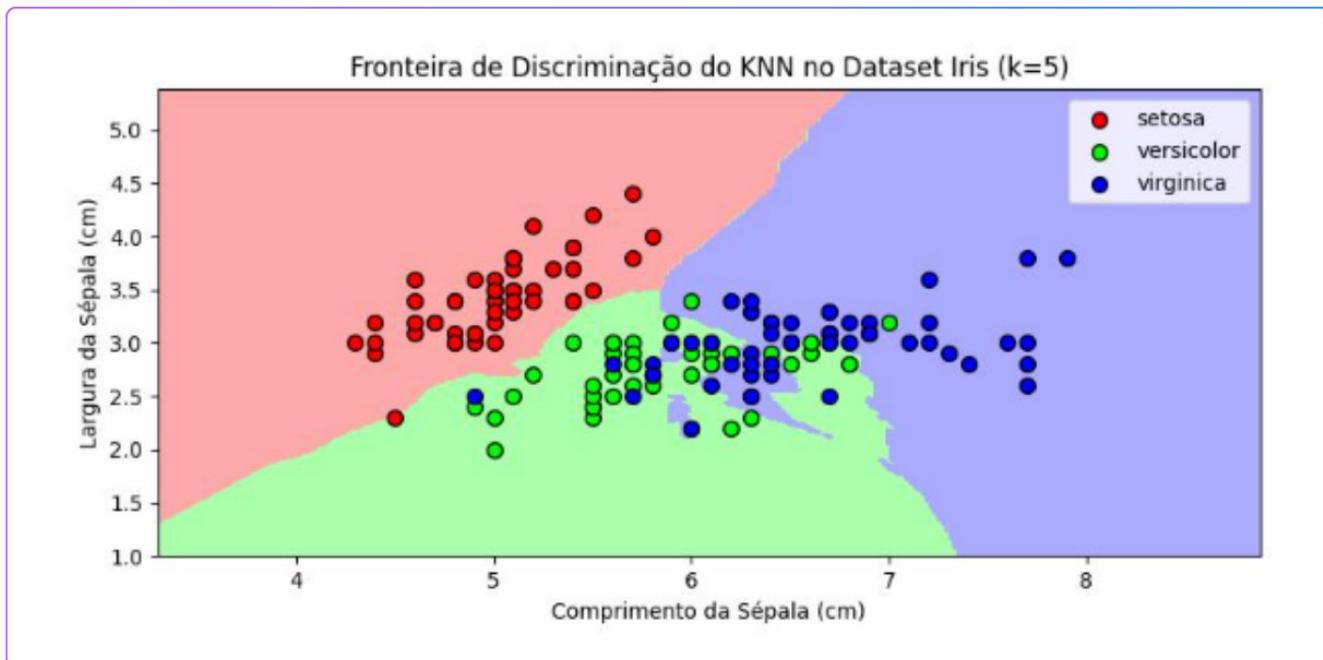
Figura 79 - Exemplo de aplicação do KNN em um novo dado



Fonte: autoria própria.

A partir deste processo podemos definir as regiões do espaço das variáveis conforme a proximidade dos 5 vizinhos mais próximos, criando assim uma fronteira de discriminação baseada na vizinhança dos dados. Na figura a seguir podemos visualizar a região de cada classe. Neste exemplo podemos ver que a classe setosa está quase perfeitamente discriminada das outras duas classes, porém as classes versicolor e virginica estão sobrepostas de uma forma que não é possível separá-las perfeitamente com o KNN. Dessa forma podemos afirmar que o KNN com apenas estas duas variáveis não é capaz de obter uma boa generalização. Uma alternativa para se obter um melhor resultado pode ser com uso das técnicas discutidas nas unidades anteriores, como: adicionar outras variáveis, normalizar ou transformar as variáveis utilizadas ou talvez remover outliers do conjunto de treino. De qualquer maneira, para mensurarmos a qualidade desta modelagem em classificar flores e sua capacidade de generalização é necessária uma metodologia de avaliação e métricas objetivas.

Figura 80 - Fronteira de discriminação de um KNN



Fonte: autoria própria.

9.2 Avaliação de Classificadores

A avaliação de modelos de Aprendizado de Máquina varia conforme o problema a ser tratado. No caso de um problema de classificação, há alguns critérios específicos que precisam ser observados.

O primeiro ponto a ser tratado é a divisão dos dados em conjuntos de treino e teste, ou em treino, validação e teste. A divisão dos dados deve ser coerente com o problema, de forma que cada amostra de treino seja independente das amostras de teste. Por exemplo, em um problema de imagens médicas, caso existam várias imagens de um mesmo paciente, elas devem ser alocadas ou no conjunto de treino ou no de teste, mas

sempre juntas. Dessa forma, garantimos que os conjuntos são Independentes e Iden-ticamente Distribuídos (IID). Esse conceito vem da estatística e define que as amostras devem ser:

- » **Independentes:** Se os conjuntos de treino e teste não forem independentes, pode haver uma correlação entre os dados nos dois conjuntos. Isso pode levar o modelo a “lembrar” informações específicas do conjunto de treino e usar essas informações para obter um desempenho artificialmente alto no conjunto de teste, em vez de aprender padrões generalizáveis. Isso resulta em uma avaliação otimista e irrealista do modelo.
- » **Identicamente Distribuídos:** Se os conjuntos de treino e teste não forem identicamente distribuídos, o modelo treinado em um conjunto de dados pode não se comportar bem no conjunto de teste, pois os padrões que ele aprendeu podem não ser aplicáveis ao conjunto de teste. Isso levaria a uma avaliação incorreta do modelo, onde o desempenho no conjunto de teste não reflete o desempenho esperado em novos dados do mundo real.

Esse cuidado deve ser considerado para que a avaliação reflita de forma mais fide-digna como o modelo se comportará no futuro ao lidar com novos dados, conseguindo assim, mensurar a capacidade de generalização e identificando ao mesmo tempo um possível *overfitting* (sobreajuste), que no caso seria equivalente a “decorar” o conjunto de treino, que leva o classificador a um alto erro nos dados de teste.

Por fim, após treinar o modelo e executar o conjunto de teste, mensuramos a qua-lidade das previsões a partir dos rótulos já conhecidos. Para isso temos métricas espe-cíficas para problemas de classificação.

9.2.1 Métricas para Classificação

Dentre as métricas a serem usadas, alguns pontos de atenção são importantes a serem considerados. Primeiro, o objetivo da aplicação. Cada métrica tem como função avaliar alguma qualidade ou defeito específico dos modelos, não existem métricas uni-versais. Assim, é necessário avaliar com base no objetivo da aplicação que está sendo de-senvolvida qual é o resultado esperado. Como por exemplo em um sistema médico vol-tado a classificação de risco de pacientes que chegam ao pronto atendimento em risco baixo e alto. Se o objetivo for fazer uma triagem automatizada espera-se que o modelo acerte as classificações em geral, e também é preciso verificar o desempenho por clas-se, para garantir que os pacientes de alto risco estão sendo classificados corretamente, pois são os casos mais críticos. Caso o objetivo seja algo parcialmente automatizado, de forma que um profissional de saúde irá receber uma probabilidade de risco entre 0 e 1 e interpretar tal informação, já é preciso avaliar não só se a classificação é correta, mas o quão precisa ela é, pois se em dois casos de altíssimo risco um tiver probabilidade de 0,6 e o outro de 0,9, isso pode gerar interpretações diferentes.

Para interpretarmos as métricas, é necessário entender o contexto de classificação como a tentativa de classificar corretamente cada classe individualmente. Nesse contexto, usamos os termos “positivos” e “negativos” para se referir a pertencente ou não pertencente à classe. Ao realizar a classificação, temos os casos positivos classificados corretamente, chamados de Verdadeiros Positivos, e os classificados erroneamente como negativos, chamados de Falsos Negativos. Da mesma forma, temos os negativos classificados corretamente, chamados de Verdadeiros Negativos, e os erroneamente classificados como positivos, chamados de Falsos Positivos. Esses quatro valores formam a chamada matriz de confusão, a partir da qual extraímos as métricas.

Figura 81 - Matriz de confusão e seus componentes

		Valor Preditivo	
		Sim	Não
Real	Sim	Verdadeiro Positivo (TP)	Falso Negativo (FN)
	Não	Falso Positivo (FP)	Verdadeiro Negativo (TN)

Fonte: autoria própria.

A partir das combinações dos quatro valores temos as principais métricas:

9.2.1.1 Acurácia

Equação:

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

Definição: Acurácia é a proporção de acertos com relação ao número de respostas, sendo assim a métrica mais intuitiva pois nos entrega uma medida do quanto o modelo acerta/errra.

Pontos positivos:

- » Fácil compreensão

Pontos negativos:

- » Pode ser enganosa em conjuntos de dados desbalanceados, pois um modelo que prevê a classe majoritária para todos os exemplos ainda pode ter alta acurácia.

9.2.1.2 Precisão

Equação:

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

Definição: A precisão mede a proporção de instâncias corretamente identificadas como positivas em relação ao total de instâncias previstas como positivas. Ela indica o quanto as previsões positivas do modelo são confiáveis.

Pontos positivos:

- » Útil quando o custo de um falso positivo é alto.
- » Boa métrica quando se deseja minimizar alarmes falsos.

Pontos negativos:

- » Não considera os falsos negativos, o que pode ser problemático em casos onde a identificação de todos os positivos é crucial.

9.2.1.3 Recall

Equação:

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

Definição: O recall (ou sensibilidade) mede a proporção de instâncias positivas corretamente identificadas pelo modelo em relação ao total de instâncias positivas reais. Ele indica a capacidade do modelo de detectar todos os positivos.

Pontos positivos:

- » Útil quando o custo de um falso negativo é alto.
- » Boa métrica quando se deseja capturar o máximo de positivos possível.

Pontos negativos:

- » Pode levar a muitos falsos positivos se não for balanceado com precisão.

9.2.1.4 F1-score

Equação:

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

Definição: O F1-Score é a média harmônica da precisão e do *recall*, oferecendo um equilíbrio entre as duas métricas. Ele é especialmente útil em situações onde é necessário um equilíbrio entre precisão e *recall*.

Pontos positivos:

- » Útil em cenários de classes desbalanceadas.
- » Balanceia os erros de falsos positivos e falsos negativos.

Pontos negativos:

- » Pode ser difícil de interpretar isoladamente.
- » Não diferencia a importância relativa de precisão e *recall*.

9.2.1.5 Matriz de confusão

Definição: A própria matriz de confusão pode ser usada como métrica, pois mostra o desempenho do modelo de classificação ao exibir o número de instâncias verdadeiras e falsas para cada classe. Ela ajuda a entender onde o modelo está cometendo erros.

Pontos positivos:

- » Proporciona uma visão detalhada do desempenho do modelo.

Pontos negativos:

- » Pode ser difícil de interpretar quando o número de classes é muito grande.
- » Não oferece uma métrica única de desempenho, exigindo cálculo de outras métricas para análise quantitativa.

Como podemos ver, nenhuma métrica é perfeita. As diferentes formas de interpretar acertos e erros nos permitem avaliar a performance de um classificador sob diferentes ângulos. Agora, vamos refletir sobre como essas métricas podem ser aplicadas em diferentes problemas.

9.2.1.6 Acurácia em Detecção de Fraude

Problema: Um banco quer identificar transações fraudulentas em tempo real.

Uso da Acurácia: A acurácia pode ser usada como uma métrica geral para avaliar o modelo se as classes de transações fraudulentas e não fraudulentas estiverem平衡adas.

Dificuldade: Se 99% das transações são legítimas, um modelo com 99% de acurácia pode parecer bom, mas pode estar ignorando completamente as fraudes (1% dos casos). Nesse cenário, a acurácia sozinha pode ser enganosa.

Alternativas:

- » **Recall:** Para garantir que o modelo detecte a maioria das fraudes.
- » **F1-Score:** Para equilibrar a precisão (evitar falsos positivos) e *recall* (detectar fraudes).

9.2.1.7 Precisão em Diagnóstico Médico (Detecção de Câncer)

Problema: Um sistema de apoio à decisão médica é usado para detectar câncer em imagens de exames.

Uso da Precisão: A precisão mede a proporção de diagnósticos corretos entre os casos identificados como positivos. Alta precisão significa menos falsos positivos.

Dificuldade: A precisão sozinha não garante que todos os casos de câncer sejam detectados.

Alternativas:

- » **Recall:** Para garantir que a maioria dos casos de câncer seja identificada.
- » **F1-Score:** Para equilibrar precisão e *recall*, maximizando a detecção de cânceres enquanto minimiza falsos positivos.

9.2.1.8 F1-Score em Detecção de Spam

Problema: Um sistema de e-mail precisa classificar mensagens como spam ou não-spam.

Uso do F1-Score: O F1-score equilibra precisão e *recall*, sendo útil para lidar com o trade-off entre capturar a maioria dos spams e evitar falsos positivos (e-mails legítimos marcados como spam).

Dificuldade: F1-Score pode ser difícil de interpretar isoladamente e não considera a distribuição das classes.

Alternativas:

- » **Matriz de Confusão:** Para entender a distribuição dos erros (falsos positivos e falsos negativos) e ajustar o modelo de acordo.

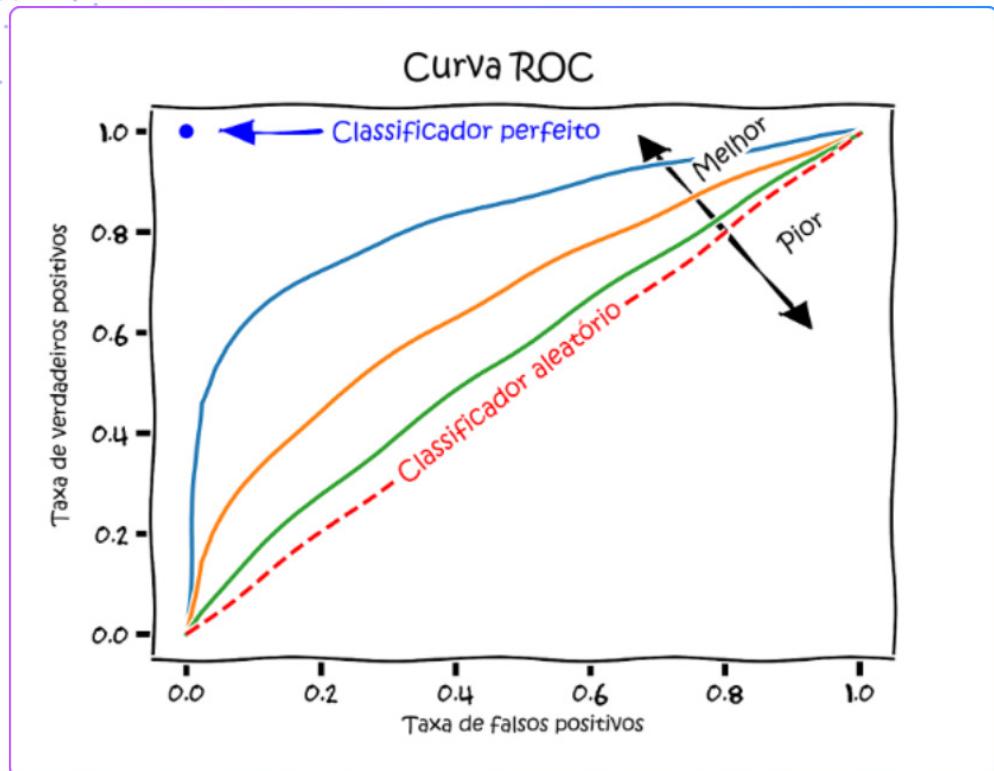
Alguns problemas exigem interpretações mais complexas para a avaliação de classificadores, indo além de apenas interpretar acertos e erros, mas também avaliando a qualidade da classificação. Uma das alternativas mais comuns para isso é o uso da Curva Receiver Operating Characteristic (ROC).

9.2.1.9 Curva ROC

A maioria dos classificadores realiza as classificações utilizando valores reais, de forma que um novo objeto pode ser classificado, por exemplo, como sendo 0,7 “gato”. Essas respostas são posteriormente convertidas para classes a fim de calcular métricas clássicas como Acurácia e Precisão, resultando em acertos e erros. No entanto, ao analisarmos mais a fundo a qualidade da classificação, podemos considerar que, se o objeto é um gato, ele deveria ser classificado como 0,9999... “gato” e não apenas 0,7 “gato”. Esse tipo de interpretação é comum em problemas chamados binários, onde há apenas duas classes, normalmente encaradas como positivo e negativo, ou presença e ausência. Alguns problemas já discutidos na Unidade são binários, como diagnóstico médico, identificação de spam e detecção de fraude. Por outro lado, o problema das flores Íris é um problema multiclasse, onde temos três classes diferentes, e uma métrica como a acurácia, por exemplo, precisa ser calculada por classe para, então, termos uma média geral entre todas as classes.

Voltando aos problemas binários, podemos interpretar a saída numérica dos classificadores e medir a qualidade a partir da Curva ROC. A Curva ROC mensura a taxa de Verdadeiros Positivos (*Recall*) junto da taxa de Falsos Positivos ($1 - \text{Especificidade}$, outra métrica calculada a partir da matriz de confusão) a partir de diferentes limiares de interpretação das classes. É como se considerássemos que, acima de 0,1, é “gato”, depois acima de 0,2, é “gato”, acima de 0,3, é “gato”, e assim por diante. Calculando as taxas em diferentes limiares, obtemos diferentes interpretações do classificador e, por fim, desenhamos uma curva com esses valores. Ao observar a curva, podemos visualizar a qualidade geral do classificador, tendo como referência um classificador aleatório (que acerta 50% das vezes) e um classificador perfeito, que formaria uma curva cobrindo toda a área do gráfico. Na figura a seguir, podemos visualizar três exemplos de classificadores e suas Curvas ROC. Quanto mais alta a curva, maior a área e, portanto, melhor o classificador.

Figura 82 - Exemplos de Curva ROC de diferentes classificadores



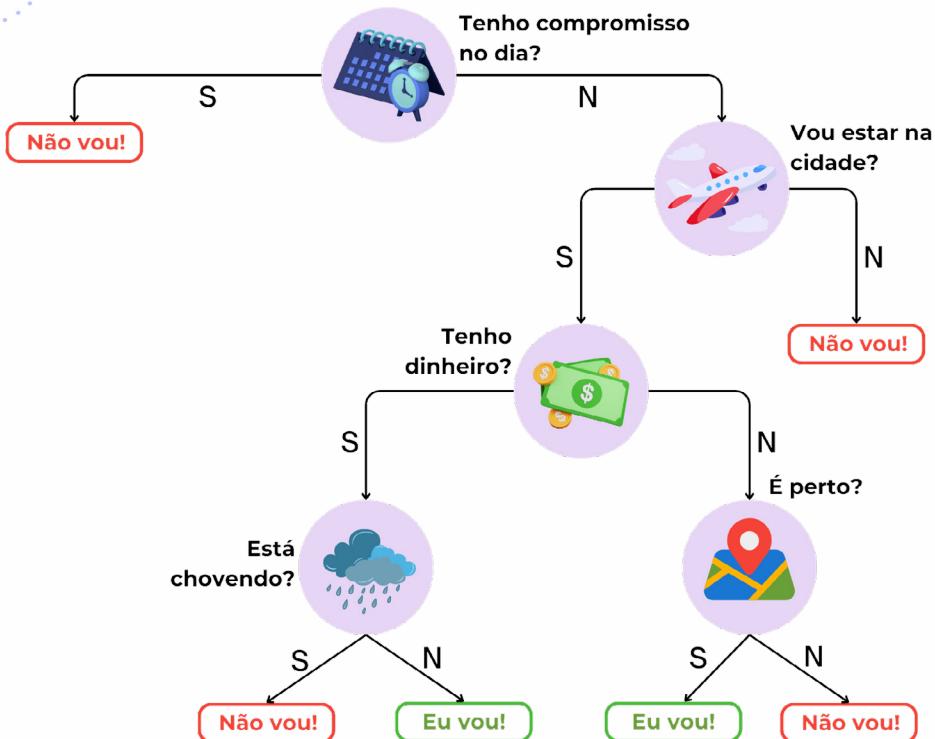
Fonte: Wikipedia, 2023.

Como a Curva ROC pode ser complexa de interpretar, uma alternativa é calcular a Área Sob a Curva (AUC - Area Under the Curve). Dessa forma, podemos resumir toda a curva em um único valor entre 0 e 1 que sintetiza a qualidade do classificador.

9.3 Classificadores Modernos

Com o tempo muitos classificadores, como os listados no início da unidade, foram propostos a fim de maximizar as métricas que discutimos anteriormente. Dentro destes os que mais obtiveram sucesso em problemas complexos foram os classificadores baseados em *Ensembles* que usam de Árvores de Decisão. O conceito de *Ensemble* refere-se à combinação de vários classificadores individuais para formar um modelo mais robusto e preciso. A ideia principal por trás dos *ensembles* é que a combinação de vários modelos “fracos” pode resultar em um modelo “forte”. Os métodos de *ensemble*, como *Bagging* e *Boosting*, têm sido amplamente utilizados para melhorar o desempenho em tarefas de classificação. Enquanto isso, uma Árvore de Decisão realiza a classificação por meio de uma estrutura hierárquica, onde cada “nó” da árvore representa uma condição sobre as *features* (características) dos dados, e cada “ramo” representa o resultado de um teste sobre essa condição, conduzindo a decisões finais ou previsões nos “nós folha”. Como na figura a seguir onde podemos ver um exemplo de Árvore de Decisão para decidir entre ir em um compromisso ou não.

Figura 83 - Exemplo de uma árvore de decisão sobre o problema de decidir sair de casa ou não



Fonte: autoria própria.

Dentre os classificadores que combinam *Ensemble* e Árvores de Decisão podemos destacar o *Extreme Gradient Boosting* (XGBoost) e *Light Gradient Boosting Machine* (LightGBM).

9.3.1 XGBoost

- » **Arquitetura:** XGBoost é um algoritmo baseado em *Gradient Boosting* que implementa técnicas como regularização para evitar *overfitting* (sobreajuste), além de outras otimizações como paralelização e uso de histogramas para acelerar sua execução.
- » **Desempenho:** XGBoost é conhecido por seu alto desempenho em termos de precisão e velocidade, sendo capaz de lidar com grandes conjuntos de dados e modelos complexos.
- » **Aplicações:** É amplamente utilizado em diversas áreas, desde previsão de risco de crédito até diagnóstico médico, onde a maximização de métricas como F1-score e AUC-ROC é crucial.

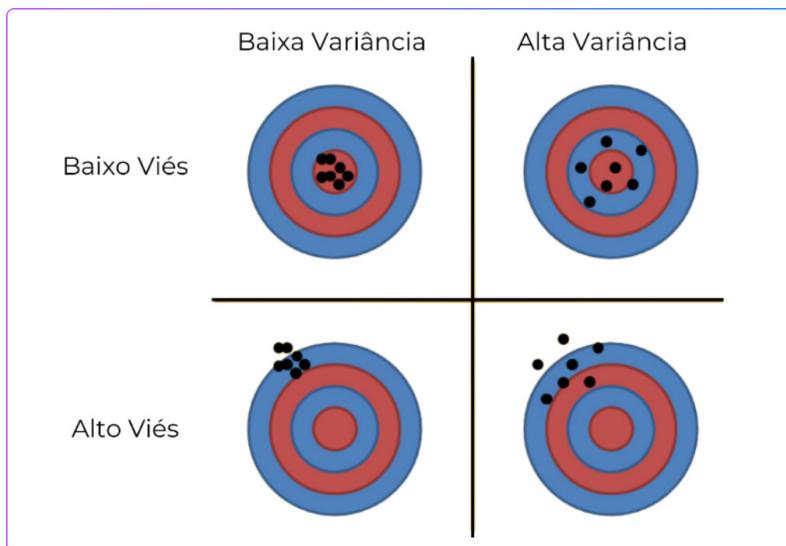
9.3.2 LightGBM

- » **Arquitetura:** LightGBM é outro algoritmo de *Gradient Boosting*, mas com foco em maior eficiência. Ele utiliza técnicas como o algoritmo de aprendizado baseado em folhas (leaf-wise), que cresce as árvores de decisão de maneira mais eficiente em termos de memória e tempo de computação.
- » **Desempenho:** LightGBM é particularmente rápido em treinar modelos em grandes volumes de dados e pode lidar melhor com características de alta dimensão e dados esparsos.
- » **Aplicações:** LightGBM é usado em muitos problemas industriais, incluindo classificação de texto, recomendação de produtos e detecção de fraudes, onde a precisão e a velocidade são críticas.

XGBoost e LightGBM são algoritmos altamente eficazes em evitar o *overfitting* (sobreajuste), graças ao uso de técnicas de regularização e otimização que melhoram a capacidade de generalização dos modelos. Eles são otimizados para alta performance, utilizando paralelização e métodos de processamento eficientes, o que permite o treinamento em grandes conjuntos de dados de maneira relativamente rápida. Além disso, esses algoritmos são extremamente flexíveis e escaláveis, suportando diferentes tipos de dados e permitindo ajustes finos nos parâmetros do modelo para alcançar o melhor desempenho possível em métricas como acurácia, F1-score e AUC-ROC. Outra vantagem significativa é a capacidade de lidar com dados desbalanceados, graças a opções internas para ajustar pesos de classe e métricas específicas, permitindo que XGBoost e LightGBM encontrem o equilíbrio ideal entre precisão e *recall*.

Outro ponto relevante de ambos os classificadores é a capacidade de equilibrar viés e variância, abordando de forma eficiente esse dilema clássico do Aprendizado de Máquina. O desafio de encontrar o equilíbrio ideal entre viés e variância é fundamental para o desenvolvimento de modelos eficazes e robustos. Esses dois conceitos estão diretamente ligados ao desempenho de um modelo em termos de sua capacidade de generalização para dados novos e não vistos. Na figura a seguir temos uma ilustração do efeito de ambos os conceitos.

Figura 84 - Ilustração do efeito do viés e variância



Fonte: autoria própria.

Viés: Refere-se à diferença entre as previsões médias de um modelo e os valores reais que ele está tentando prever. Um modelo com alto viés tende a fazer suposições fortes e simplistas sobre os dados, o que pode levar a subajuste (*underfitting*), onde o modelo é incapaz de capturar a complexidade dos dados de treinamento, resultando em baixa performance tanto nos dados de treinamento quanto nos dados de teste.

Variância: Refere-se à sensibilidade do modelo às variações nos dados de treinamento. Um modelo com alta variância tende a se ajustar muito perto aos dados de treinamento, capturando até mesmo o ruído presente, o que leva a um superajuste (*overfitting*). Embora o modelo apresente alta performance nos dados de treinamento, ele falha ao generalizar para novos dados, resultando em desempenho inconsistente nos dados de teste.

9.4 Notebook Colab

No [notebook](#) a seguir iremos praticar os conceitos estudados na unidade ao estudar e solucionar um problema de classificação de vinhos através das suas características químicas.

Objetivos de Aprendizagem

- » Compreender os fundamentos do algoritmo k-Nearest Neighbors (k-NN) e sua aplicação em problemas de classificação;
- » Desenvolver habilidades práticas na manipulação e análise de dados utilizando bibliotecas Python, como Scikit-learn e Pandas;
- » Avaliar o desempenho de modelos de Machine Learning através de técnicas de validação e visualização.

Classificação usando o k-NN

Neste notebook, vamos explorar o dataset de vinhos disponível na biblioteca **sklearn.datasets**. Utilizaremos técnicas de classificação e validação para construir e avaliar um modelo de Machine Learning utilizando o algoritmo k-Nearest Neighbors (k-NN).

O que é a biblioteca Scikit-learn?

Scikit-learn é uma biblioteca de Machine Learning em Python que fornece ferramentas simples e eficientes para análise e modelagem de dados. Ela suporta tanto aprendizado supervisionado quanto não supervisionado e é amplamente utilizada devido à sua facilidade de uso e vasta gama de funcionalidades, incluindo algoritmos de classificação, regressão, clustering, e redução de dimensionalidade.

Datasets de exemplos disponíveis no Scikit-learn

A biblioteca **sklearn.datasets** inclui vários datasets de exemplo que são amplamente usados para aprendizado e benchmarking em Machine Learning. Alguns dos mais populares incluem:

- » Iris: Um conjunto de dados clássico de classificação com informações sobre 3 tipos de flores de íris.
- » Digits: Um dataset de classificação que contém imagens de dígitos escritos à mão (0 a 9).
- » Wine: O dataset que vamos utilizar neste notebook, que contém informações sobre a composição química de diferentes tipos de vinhos.
- » Breast Cancer: Dados sobre características de tumores para diagnóstico de câncer de mama.

O dataset Wine

O dataset Wine consiste em 178 amostras de vinho com 13 características químicas cada, como o teor alcoólico, a quantidade de ácido málico, magnésio, fenóis totais, e outros. Estas amostras são divididas em três classes diferentes, representando três cultivares de uva diferentes. O objetivo em muitos estudos usando este dataset é classificar corretamente o tipo de vinho com base nas suas características químicas.

```
from sklearn import datasets # Biblioteca de datasets
vinho = datasets.load_wine() # Escolha do dataset de vinhos
print(vinho.DESCR)
.. _wine_dataset:

Wine recognition dataset
-----
**Data Set Characteristics:**

:Number of Instances: 178
:Number of Attributes: 13 numeric, predictive attributes and the class
```

continua

```
:Attribute Information:
  - Alcohol
  - Malic acid
  - Ash
  - Alcalinity of ash
  - Magnesium
  - Total phenols
  - Flavanoids
  - Nonflavanoid phenols
  - Proanthocyanins
  - Color intensity
  - Hue
  - OD280/OD315 of diluted wines
  - Proline

  - class:
    - class_0
    - class_1
    - class_2
```

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None
 :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
 :Creator: R.A. Fisher
 :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
 :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
<https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -
 An Extendible Package for Data Exploration, Classification and Correlation.

continua

Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[<https://archive.ics.uci.edu/ml>]. Irvine, CA: University of California,
School of Information and Computer Science.

```
|details-start|
**References**
|details-split|
```

(1) S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various
classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.

(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

```
|details-end|
```

O código acima importa a biblioteca de datasets do **sklearn** e carrega o dataset de vinhos usando **datasets.load_wine()**. O comando **print(vinho.DESCR)** exibe uma descrição detalhada do dataset, incluindo informações sobre as características e classes, que é útil para entender melhor a estrutura dos dados.

```
print (vinho.feature_names) # Mostra os rótulos das características
print (vinho.target_names) # Mostra os rótulos das classes
print (vinho.data[0:5]) # Mostra as 5 primeiras amostras do dataset
print(vinho.target) # Mostra os identificadores das classes
print(vinho.data.shape) # Mostra o tamanho total do dataset

['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
['class_0' 'class_1' 'class_2']
[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
 2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
 2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
 3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
 2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
```

continua

Aqui, estamos explorando o dataset mais a fundo. Usamos

- » `vinho.feature_names` para listar os nomes das características (atributos) dos dados.
 - » `vinho.target_names` para listar os nomes das classes (tipos de vinho).
 - » `vinho.data[0:5]` para visualizar as cinco primeiras amostras do dataset.
 - » `vinho.target` para ver os identificadores das classes correspondentes às amostras.
 - » `vinho.data.shape` para obter a dimensão do dataset, mostrando o número total de amostras e características.

Agrupamento dos identificadores das classes com List Comprehension

Para entender a distribuição das classes no dataset, utilizamos a função `groupby` da biblioteca `itertools`. Esta função nos permite agrupar as amostras de acordo com suas classes e contar quantas amostras pertencem a cada classe. A técnica de list comprehension utilizada aqui é um método conciso e eficiente para criar listas em Python.

```
from itertools import groupby  
[len(list(group)) for _, group in groupby(vinho.target)]  
  
[59, 71, 48]
```

Este código executa os seguintes passos:

1. **groupby(vinho.target)**: Agrupa os identificadores das classes (**vinho.target**) sequencialmente.
 2. **for _, group in groupby(vinho.target)**: Itera sobre cada grupo criado pelo groupby. O primeiro valor (chave de agrupamento) é ignorado (**_**), e o segundo valor (**group**) é a lista de itens pertencentes a esse grupo.
 3. **len(list(group))**: Para cada grupo, converte-o em uma lista e calcula o tamanho dessa lista, que representa o número de amostras naquela classe.
 4. **[len(list(group)) for _, group in groupby(vinho.target)]**: Gera uma lista com o número de amostras para cada classe.

Divisão do dataset em amostras treinamento e de teste

Uma prática comum em Machine Learning é dividir o dataset em conjuntos de treinamento e teste. O conjunto de treinamento é usado para treinar o modelo, enquanto o conjunto de teste é usado para avaliar seu desempenho em dados que não foram vistos durante o treinamento. Aqui, utilizamos a função `train_test_split` do `sklearn` para realizar essa divisão.

Escolhendo a proporção dos dados

A proporção entre os conjuntos de treinamento e teste é geralmente escolhida com base no tamanho do dataset e na necessidade de obter um bom balanceamento entre o treinamento do modelo e a validação de sua performance. Uma divisão comum é 70% dos dados para treinamento e 30% para teste. Isso proporciona um conjunto de treinamento suficientemente grande para modelar bem os dados, enquanto mantém um conjunto de teste significativo para avaliação.

```
from sklearn.model_selection import train_test_split
X_treino, X_teste, y_treino, y_teste = train_test_split(vinho.data, vinho.target,
test_size=0.3)
print(y_treino) # Amostras selecionadas para treinamento
print(y_teste) # Amostras selecionadas para teste
```

```
[0 1 0 2 2 1 1 1 0 1 0 0 2 1 0 1 0 1 0 2 1 0 2 0 2 2 1 2 1 1 1 2 0 2 1 1 0
2 1 1 0 2 2 1 0 0 0 1 1 1 1 1 1 0 0 2 2 2 2 1 1 0 1 2 2 0 0 1 2 2 0
2 0 2 2 0 0 1 1 1 1 2 1 0 0 1 2 1 2 1 2 0 1 1 1 0 1 1 1 1 0 2 1 1 1 0 1 2
2 0 0 1 2 1 2 0 0 1 0 1 2]
[0 1 2 2 1 1 1 1 2 0 0 1 0 0 2 1 1 0 0 1 0 2 0 1 2 1 2 2 1 0 0 0 0 0 1 2
2 1 2 0 1 0 0 1 0 0 2 1 2 0 1 0 0]
```

O código divide o dataset em dois subconjuntos: `X_treino` e `y_treino` (dados e classes de treinamento), e `X_teste` e `y_teste` (dados e classes de teste). O parâmetro `test_size=0.3` indica que 30% dos dados serão usados para teste. Após a divisão, o código exibe as classes selecionadas para os conjuntos de treinamento e teste. Em machine learning é usual utilizarmos `X` (maiúsculo) para denotar uma matriz que representa os objetos nas linhas e as características nas colunas. Já o `y` é utilizando para representar a resposta dos objetos, ou seja, suas respectivas classes.

Treinamento e avaliação do modelo k-NN

Agora que os dados estão preparados, podemos treinar um modelo de classificação. Vamos usar o algoritmo k-Nearest Neighbors (k-NN), que é um dos mais simples e intuitivos em Machine Learning.

Mas como funciona o k-NN?

O k-NN é um algoritmo de classificação baseado em instâncias. Ele opera sob o princípio de que uma amostra deve ser classificada da mesma forma que as amostras mais próximas a ela em um espaço de características. Para determinar a classe de uma nova amostra, o k-NN analisa os `k` exemplos mais próximos no conjunto de treinamento e atribui ao objeto desconhecido a classe que é mais comum entre esses vizinhos. A esco-

lha de **k** é crítica e pode afetar significativamente o desempenho do modelo. Um valor muito pequeno de **k** pode tornar o modelo sensível ao ruído, enquanto um valor muito grande pode diluir a influência dos vizinhos mais próximos além de tender para classes majoritárias em problemas com conjunto de dados desbalanceados.

Treinamento do modelo com o método **fit**

O método **fit** do k-NN é usado para treinar o modelo. Ele simplesmente armazena as amostras de treinamento e suas classes correspondentes para posterior consulta durante a fase de predição. As entradas para o método **fit** são os dados de treinamento (**X_treino**) e as respectivas classes (**y_treino**). O método **fit** não possui saídas; ele apenas prepara o modelo para realizar predições.

```
from sklearn.neighbors import KNeighborsClassifier # Importa o classificador
knn = KNeighborsClassifier(n_neighbors=5) # Cria o modelo com k = 5
knn.fit(X_treino, y_treino) # Treinamento
```

```
▼ KNeighborsClassifier
  KNeighborsClassifier()
```

Predição com o método **predict**

Após o treinamento, podemos usar o método **predict** para prever as classes das amostras no conjunto de teste. O método **predict** recebe como entrada os dados de teste (**X_teste**) e retorna as classes previstas para essas amostras. Este é o processo onde o modelo compara as novas amostras com as armazenadas durante o treinamento e decide a classe de cada nova amostra com base nos 'k' vizinhos mais próximos.

```
y_predito = knn.predict(X_teste) # Predição
print(y_predito) # Mostra o resultado da predição
[0 1 1 1 1 2 1 1 2 2 0 0 1 0 1 2 2 1 0 1 1 0 1 0 1 1 1 2 2 1 0 0 0 0 0 1 1
 2 1 1 0 2 0 0 1 0 2 2 1 1 0 1 0 1]
```

Avaliação do modelo com o método **score**

O método **score** é utilizado para avaliar a performance do modelo após a predição. Ele calcula a acurácia, que é a proporção de amostras corretamente classificadas em relação ao total de amostras testadas. A entrada para o método **score** são os dados de teste (**X_teste**) e as classes reais (**y_teste**). A saída é a acurácia, um valor entre 0 e 1 que indica a precisão do modelo.

```
acuracia = knn.score(X_teste, y_teste) # Calcula a acurácia da predição
print(acuracia) # Mostra a acurácia da predição
0.7222222222222222
```

Validação cruzada

Para obter uma estimativa mais confiável do desempenho do modelo, utilizamos a validação cruzada. Na validação cruzada, o dataset é dividido em várias partes, ou 'folds',

e o modelo é treinado e testado múltiplas vezes, cada vez utilizando uma parte diferente como conjunto de teste e as demais como conjunto de treinamento. Existem vários tipos de validação cruzada, cada um com suas vantagens e desvantagens.

Tipos de validação cruzada

- » **k-Fold Cross Validation:** O dataset é dividido em 'k' partes, e o modelo é treinado 'k' vezes, cada vez utilizando uma parte diferente como conjunto de teste. A vantagem é que todos os dados são eventualmente utilizados tanto para treinamento quanto para teste. A desvantagem é que pode ser computacionalmente caro.
- » **Leave-One-Out Cross Validation (LOOCV):** Cada amostra do dataset é utilizada uma vez como conjunto de teste, enquanto as outras amostras formam o conjunto de treinamento. Embora seja mais preciso, é ainda mais caro computacionalmente do que o k-Fold.
- » **Stratified k-Fold:** Semelhante ao k-Fold, mas garante que a proporção de classes em cada *fold* seja representativa do dataset original, o que é especialmente útil em datasets desbalanceados.

Validação cruzada com `cross_val_score`

A função `cross_val_score` facilita a realização da validação cruzada. Ela recebe como entradas o modelo (neste caso, um classificador k-NN), os dados (`vinho.data`), as classes (`vinho.target`) e o número de folds (`cv=5`). A função executa a validação cruzada e retorna uma lista com as acurárias obtidas em cada iteração. A média dessas acurárias pode ser utilizada como uma estimativa da performance geral do modelo.

```
from sklearn.model_selection import cross_val_score
import numpy as np
knn = KNeighborsClassifier(n_neighbors=3)
acuracias_vc = cross_val_score(knn, vinho.data, vinho.target, cv=5)
print(acuracias_vc)
print('Média das acurárias: {}'.format(np.mean(acuracias_vc)))
```

[0.63888889 0.69444444 0.66666667 0.65714286 0.85714286]
Média das acurárias: 0.7028571428571428

No código acima, realizamos uma validação cruzada 5-folds. O modelo k-NN com `k=3` é avaliado 5 vezes, cada vez com uma parte diferente dos dados sendo usada para teste. O resultado mostra as acurárias de cada uma das 5 iterações, e a média dessas acurárias é calculada e exibida.

Avaliando diversos valores de `k`

No código abaixo, geramos um gráfico de linha representando os valores das acurárias do modelo k-NN com `k` variando entre 10 e 35, de 5 em 5, aplicando a validação cruzada.

```

import seaborn as sns
import matplotlib.pyplot as plt

# Lista dos valores de 'k' para o k-NN
ks = [10, 15, 20, 25, 30, 35]

# Lista vazia para armazenar as acuráncias médias
acuracias = []

# Loop para calcular a acurácia média para cada valor de 'k'
for k in ks:
    # Cria o classificador k-NN com o valor de 'k' atual
    knn = KNeighborsClassifier(n_neighbors=k)

    # Executa a validação cruzada 5-fold e armazena as acuráncias obtidas
    acuracias_vc = cross_val_score(knn, vinhos.data, vinhos.target, cv=5)

    # Calcula a acurácia média das 5 dobras e adiciona à lista de acuráncias
    acuracias.append(np.mean(acuracias_vc))

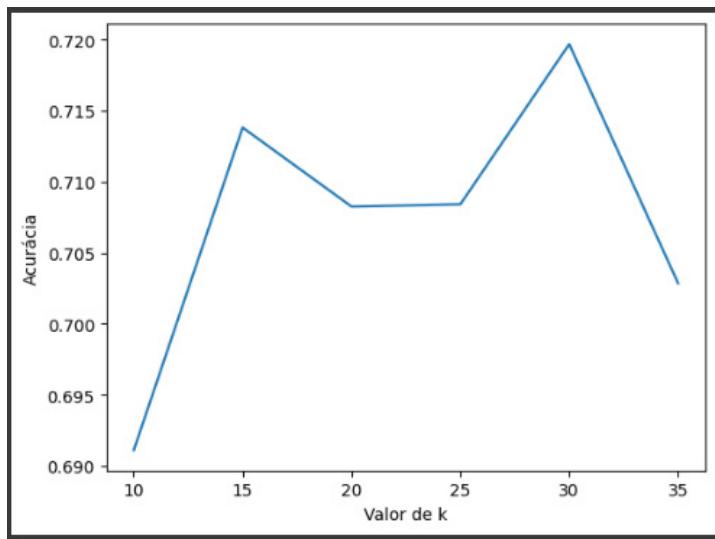
# Criar o gráfico de linha com os valores de 'k' no eixo x e as acuráncias médias no
# eixo y
sns.lineplot(x=ks, y=acuracias)

# Adicionar as legendas dos eixos
plt.ylabel('Acurácia')
plt.xlabel('Valor de k')

# Mostrar o gráfico na tela
plt.show()

```

Figura 85 - Gráfico de linha representando os valores das acuráncias do modelo k-NN



Fonte: autoria própria.

O código começa importando as bibliotecas necessárias: **Seaborn** para a criação de gráficos e **Matplotlib** para manipulação de gráficos. Em seguida, uma lista de valores de **k** é definida: **[10, 15, 20, 25, 30, 35]**. Esses valores serão usados para configurar o número de vizinhos que o algoritmo k-NN considera ao fazer previsões. Para cada valor de **k**, o código realiza o seguinte:

- » Cria um modelo k-NN com o valor atual de **k**.
- » Executa uma validação cruzada 5-fold usando o modelo e o dataset de vinhos.
- » Calcula a acurácia média dos 5 folds da validação cruzada.
- » Armazena a acurácia média em uma lista para posterior plotagem no gráfico.

Após calcular as acuráncias médias para todos os valores de **k**, o código utiliza **Seaborn** para criar um gráfico de linha. O eixo **x** do gráfico representa os valores de **k**, enquanto o eixo **y** mostra as acuráncias médias correspondentes. Por fim, o gráfico gerado é exibido, permitindo uma visualização clara de como o desempenho do modelo k-NN varia com diferentes valores de **k**. Isso ajuda a identificar qual valor de **k** proporciona a melhor acurácia para o dataset utilizado.



SAIBA MAIS...

- » k-nn - <https://youtu.be/HVXime0nQeI?si=zRbX-4XwNUouNy9h>
- » Árvores de decisão - https://www.youtube.com/watch?v=_L39rN6gz7Y
- » Florestas aleatórias - https://www.youtube.com/watch?v=J4Wdy0Wc_xQ
- » Regressão Logística - <https://www.youtube.com/watch?v=yIYKR4sgzI8>
- » SVM - <https://www.youtube.com/watch?v=efR1C6CvhmE&feature=youtu.be>
- » ROC e AUC - <https://youtu.be/4jRBRDbJemM?si=HFvY9Ish7trA8Ww1>
- » Matriz de Confusão - <https://www.youtube.com/watch?v=Kdsp6soqA7o>
- » Métricas - <https://towardsdatascience.com/understanding-confusion-matrix-a-9ad42dcfd62>

PARA RELEMBRAR...

Classificação:

- » Tarefa fundamental em aprendizagem de máquina
- » Atribui categorias ou rótulos a dados de entrada
- » Ampla aplicabilidade em medicina, finanças, marketing e segurança

Aplicações:

- » Diagnóstico de doenças
- » Detecção de fraudes bancárias
- » Análise de sentimentos
- » Reconhecimento de imagens

Métricas de avaliação:

- » Precisão: proporção de previsões positivas corretas
- » *Recall* (sensibilidade): proporção de instâncias positivas identificadas corretamente
- » *F1-score*: média harmônica entre precisão e *recall*
- » Matriz de confusão: visão detalhada do desempenho do modelo

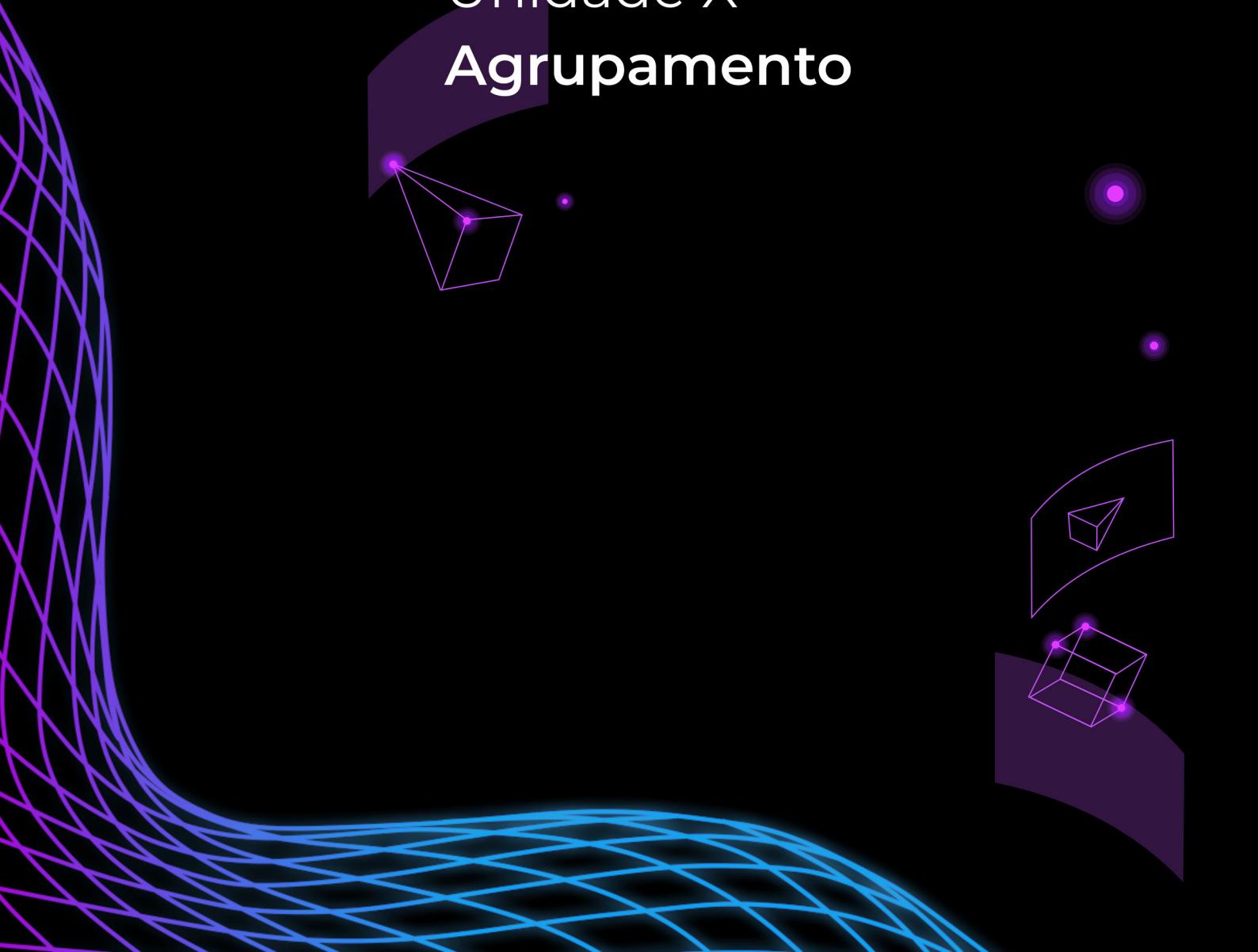
Algoritmos básicos:

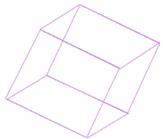
- » k-NN (Vizinhos mais próximos)
- » Árvores de Decisão
- » Florestas Aleatórias
- » Regressão Logística
- » SVM

Curva ROC e AUC:

- » ROC: representa o desempenho do modelo em diferentes limiares
- » AUC: área sob a curva ROC, medida agregada de desempenho
- » AUC de 1,0 indica classificador perfeito, 0,5 indica classificador aleatório
- » Úteis para comparar modelos e selecionar limiares ótimos, especialmente com classes desbalanceadas

Unidade X **Agrupamento**





Unidade X - Agrupamento

Dentre as técnicas de aprendizado não supervisionado, a mais comum é o agrupamento, também conhecido como clusterização (do inglês *clustering*). O objetivo do agrupamento é encontrar padrões em dados não rotulados. Diferente da classificação, que identifica o padrão nas variáveis que representam uma classe, no agrupamento buscamos padrões apenas observando as variáveis em busca de grupos, ou clusters, que são subconjuntos dos dados com comportamentos em comum, mas que apresentam diferenças entre si.

Um exemplo de agrupamento é a interpretação de padrões de consumo de clientes. Imaginando uma loja *online* que vende calçados, podemos ter diversos tipos de clientes. Alguns podem comprar mais calçados esportivos, outros preferem calçados casuais, e alguns apenas opções mais formais. Os clientes não possuem rótulos explícitos para determinarmos classes, mas, ao analisarmos o padrão de consumo, podemos identificar perfis comuns. Ao observar o histórico de compras combinado com dados como idade, gênero e profissão, é possível agrupar os clientes em agrupamentos que reúnem perfis de compra semelhantes, ao mesmo tempo que separam grupos com perfis diferentes.

Os diferentes algoritmos de agrupamento buscam identificar similaridades entre os dados enquanto tentam compreender a dissimilaridade entre os grupos. Baseado nas diferentes abordagens para tratar o problema, podemos listar alguns desses métodos no mapa mental a seguir.

Figura 86 - Mapa mental com os algoritmos de agrupamento

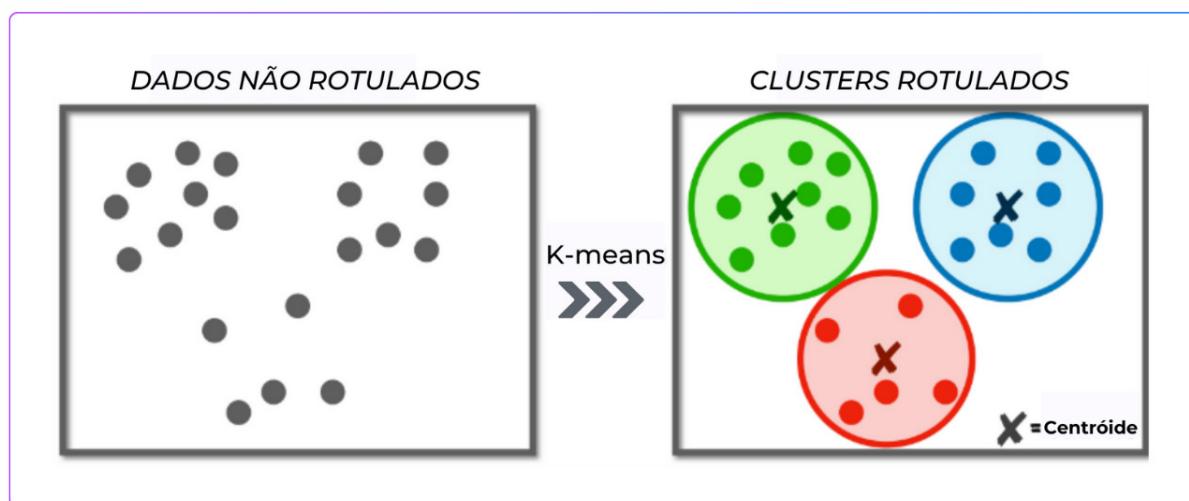


Fonte: autoria própria.

10.1 K-Means

O algoritmo de agrupamento mais comum é o *K-Means*. Ele pertence à categoria dos algoritmos de clusterização baseados em partição, cujo objetivo é dividir um conjunto de dados em K agrupamentos distintos, onde cada ponto de dados pertence ao agrupamento mais próximo do seu centroide. Um centroide é o ponto central do agrupamento e é usado para representar o “centro” do mesmo. Ele é calculado como a média aritmética das coordenadas de todos os pontos de dados que pertencem a esse agrupamento. O número de agrupamentos, K , é definido pelo usuário e ao fim da execução o esperado é ter K diferentes agrupamentos como demonstrado na Figura a seguir.

Figura 87 - Ilustração de um *K-Means* aplicado com três agrupamentos



Fonte: adaptada de [Murilo \(2023\)](#).

O algoritmo *K-Means* segue um processo iterativo que pode ser resumido nos seguintes passos:

1. **Inicialização:**
 - » O usuário define o número de agrupamentos K que deseja encontrar.
 - » Inicialmente, os centroides são escolhidos aleatoriamente. Esses centroides podem ser selecionados diretamente dos pontos de dados ou gerados aleatoriamente dentro do espaço de variáveis.
2. **Atribuição de Agrupamentos :**
 - » Cada ponto de dados é atribuído ao agrupamento cujo centroide é o mais próximo. A proximidade é geralmente medida pela distância euclidiana.
3. **Atualização dos Centroides:**
 - » Depois que todos os pontos foram atribuídos a seus respectivos agrupamentos, os centroides de cada agrupamento são recalculados.

O novo centroide é a média aritmética de todos os pontos dentro do agrupamento.

4. Convergência:

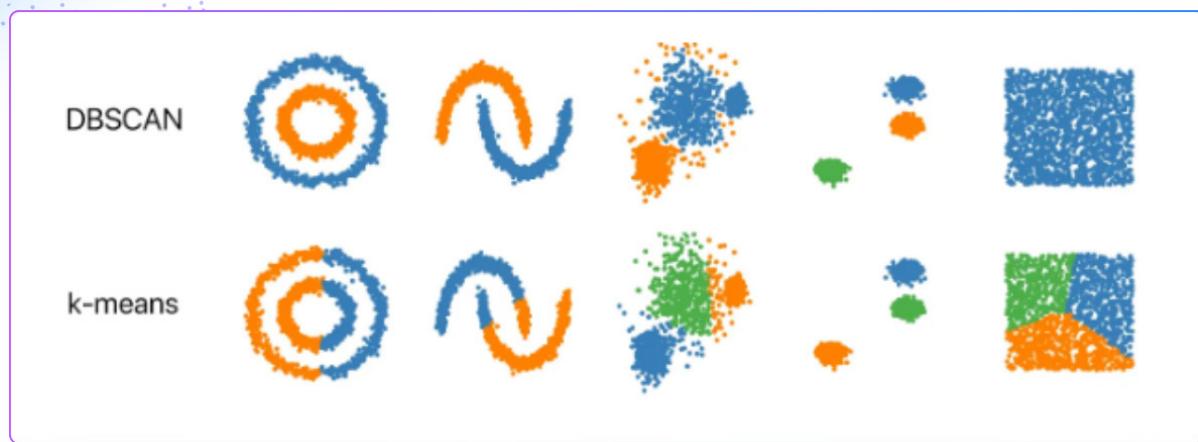
- » Os passos de atribuição de agrupamentos e atualização dos centroides são repetidos iterativamente até que os centroides não mudem mais significativamente ou até que um número máximo de iterações seja atingido. A convergência é geralmente alcançada quando a atribuição dos pontos aos agrupamentos não muda mais, ou seja, os centroides permanecem estáveis. Para calcular a estabilidade dos centroides é usado o cálculo da inércia que refere-se à soma das distâncias quadradas entre cada ponto de dados e o centroide.

O algoritmo *K-Means* é eficiente para uso em grandes bases de dados e possui uma convergência relativamente rápida. Por outro lado, o parâmetro K pode ser um desafio em sua aplicação, pois é necessário definir previamente a quantidade de agrupamentos a serem encontrados. Outro problema comum é a sensibilidade a *outliers*, que podem alterar significativamente a posição dos centroides. Algumas variações do *K-Means* propõem o uso de outras formas de calcular os centroides, como o *K-Medoids*, que utiliza um dado real do agrupamento como referência central, ou o *K-Medians*, que usa a mediana do agrupamento.

10.2 DBSCAN

Uma alternativa mais moderna ao *K-means* é o DBSCAN. O algoritmo possui duas principais vantagens, a capacidade de detectar *outliers* e excluir tais dados dos agrupamentos os identificando como ruído, e a definição automática do número de agrupamentos. Para alcançar tais vantagens o método funciona baseado em densidade considerando regiões de alta densidade como possíveis agrupamentos. Após definir agrupamentos iniciais ele considera as regiões próximas a partir do raio entre os pontos já pertencentes ao agrupamento, isso faz com que o método encontre agrupamentos de quaisquer formatos, o que é diferente do *K-means*, que por considerar apenas a distância para a condroide ele assuma que os agrupamentos sejam esféricos. Após encontrar as regiões de alta densidade únicas estas são consideradas agrupamentos únicos, possibilitando assim identificar quando agrupamentos existiram no conjunto de dados.

Figura 88 - Exemplos de aplicação do *K-means* e DBSCAN



Fonte: [Medium \(2020\)](#).

10.3 Avaliação de Agrupamentos

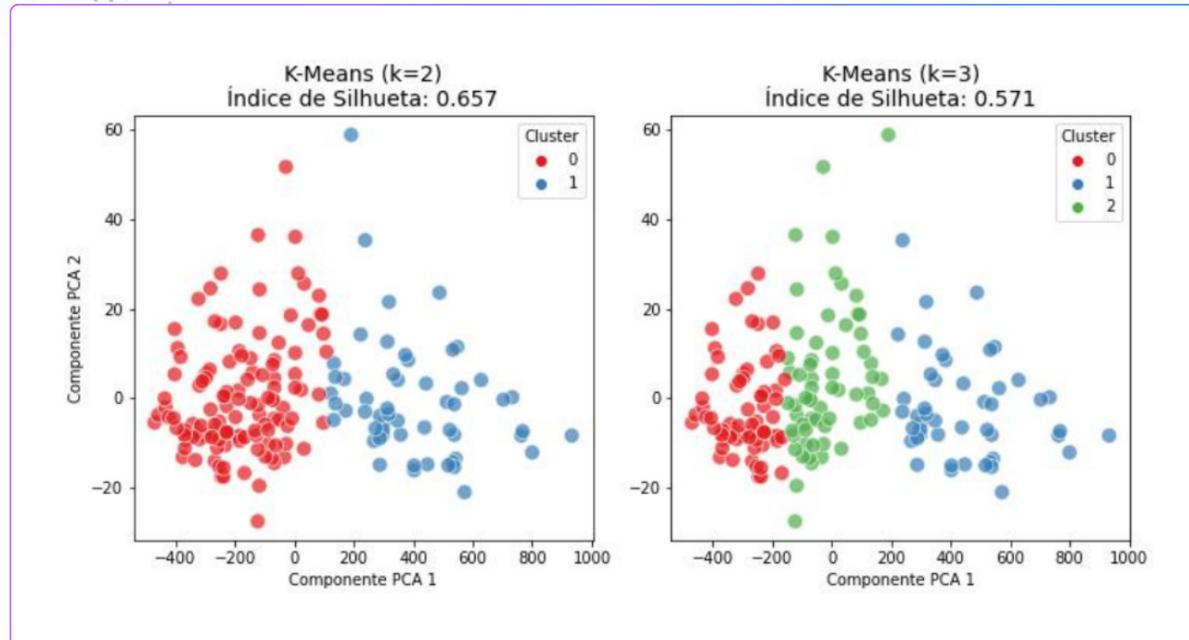
A avaliação de agrupamentos é um aspecto crucial na análise de dados, pois ajuda a determinar a qualidade e a utilidade dos clusters identificados por algoritmos de agrupamento. Diferente de tarefas de aprendizado supervisionado, onde as métricas de avaliação como acurácia, precisão e *recall* podem ser facilmente calculadas com base em rótulos conhecidos, o agrupamento geralmente é um problema de aprendizado não supervisionado, onde não se tem rótulos. Isso torna a avaliação dos agrupamentos mais desafiadora e subjetiva. Existem duas principais abordagens para avaliação de agrupamentos: avaliação interna e avaliação externa.

As métricas de avaliação interna avaliam a qualidade dos agrupamentos com base nas propriedades intrínsecas dos dados e dos agrupamentos formados, sem usar informações externas ou rótulos verdadeiros.

A métrica de avaliação interna mais comum é o Índice de Silhueta (*Silhouette Score*) que mede a coesão e a separação dos agrupamentos. Para cada ponto, a silhueta é calculada como a diferença entre a distância média ao próprio agrupamento (coesão) e a distância média ao agrupamento mais próximo (separação), normalizada pela maior dessas distâncias. O índice de silhueta varia de -1 a 1, onde valores próximos de 1 indicam que os pontos estão bem agrupados em seus próprios clusters, valores próximos de 0 indicam sobreposição de clusters, e valores negativos sugerem que os pontos podem estar mal agrupados.

Na figura a seguir podemos ver a avaliação do Índice de Silhueta do agrupamento do dataset *Wine* usando duas variáveis obtidas com PCA e o método *K-Means* com *K* igual a 2 e 3. Neste exemplo o resultado com dois agrupamentos possui maior índice e isso pode ser visualizado pela separação dos agrupamentos. Com três, o agrupamento vermelho e verde estão muito próximos, o que pode indicar que sejam apenas um e não dois agrupamentos separados.

Figura 89 - Exemplos de agrupamentos e índice de silhueta com uso do KNN com diferentes valores de k



Fonte: autoria própria.

Quando se tem ao menos alguns rótulos ou a possibilidade de rotular dados o agrupamento pode ser validado a partir destes rótulos, realizando assim a avaliação externa. A técnica mais comum para tal avaliação é o *Rand Index*, ou Índice de Rand, que calcula a porcentagem de agrupamentos corretos. O índice de Rand varia de -1 a 1, onde 1 indica um agrupamento perfeito em relação aos rótulos verdadeiros, 0 indica agrupamento aleatório e valores negativos indicam desacordo com os rótulos de referência.

10.4 Notebook Colab

No [notebook](#) a seguir realizaremos passo a passo o agrupamento do dataset *Wine* citado anteriormente com uso do algoritmo *K-Means*.

Objetivos de Aprendizagem

- » Compreender os conceitos fundamentais de clusterização e o funcionamento do algoritmo k-means;
- » Aplicar técnicas de redução de dimensionalidade e visualização de dados para interpretar resultados de clusterização;
- » Desenvolver competências na análise comparativa de resultados de clusterização com dados rotulados e não rotulados.

Clusterização com o K-Means

Neste notebook, vamos explorar o algoritmo de clusterização k-means aplicado ao

famoso dataset Iris. Este conjunto de dados contém informações sobre 150 amostras de flores de íris, divididas em três espécies diferentes. Embora o dataset seja rotulado, vamos utilizá-lo para aplicar o algoritmo k-means, que é um método não supervisionado, para comparar os resultados obtidos com as classes originais.

O que vamos fazer?

O objetivo aqui é aplicar o k-means para identificar possíveis agrupamentos nas amostras do dataset Iris, mesmo sabendo que os dados já possuem rótulos. Inicialmente, usaremos os rótulos para exibir um gráfico de dispersão dos dados originais. Em seguida, retiraremos a coluna que identifica as classes e aplicaremos o k-means, gerando um gráfico para comparar visualmente os resultados.

Montagem do Google Drive

Primeiro, precisamos acessar os arquivos armazenados no Google Drive. Usaremos o comando `drive.mount` para montar o Google Drive no ambiente Colab, permitindo acessar os arquivos diretamente.

```
from google.colab import drive
drive.mount('/content/drive') # Monta o drive do Google Drive
Mounted at /content/drive
```

O código acima monta o Google Drive no diretório `/content/drive`, permitindo que você accesse seus arquivos diretamente no ambiente Colab.

Introdução à biblioteca Pandas

Pandas é uma poderosa biblioteca de Python para análise e manipulação de dados. Ela fornece estruturas de dados flexíveis, como DataFrames e Series, que facilitam o tratamento de dados tabulares (similar a tabelas em bancos de dados ou planilhas). Pandas é particularmente útil para carregar, limpar, transformar e analisar dados. Ele pode manipular vários tipos de dados, incluindo arquivos CSV, planilhas Excel, tabelas SQL, e muito mais.

Abaixo você encontra uma explicação detalhada cada uma delas:

DataFrame

- » O que é: Um DataFrame é uma estrutura de dados bidimensional (tabular), similar a uma tabela em uma base de dados ou a uma planilha em Excel. Ele é composto por linhas e colunas, onde cada coluna pode conter diferentes tipos de dados (números, strings, booleanos, etc.).
- » Características:
 - » Etiquetado: As linhas e colunas de um DataFrame são etiquetadas, o que

significa que você pode acessar os dados através de rótulos (ou índices) em vez de apenas por posição.

- » Flexível: Pode armazenar diferentes tipos de dados em colunas diferentes, por exemplo, inteiros em uma coluna, strings em outra, e assim por diante.
- » Manipulação e análise: É amplamente utilizado para operações de filtragem, agregação, pivotamento, e outras formas de manipulação de dados.
- » Exemplo: Imagine um DataFrame com as colunas "Nome", "Idade", e "Cidade". Cada linha desse DataFrame representa uma pessoa diferente, com os valores dessas três características.

Series

- » O que é: Uma Series é uma estrutura de dados unidimensional, similar a uma coluna de uma tabela ou a uma lista. É essencialmente uma lista de elementos com rótulos (índices), o que a torna uma versão mais poderosa de um array do NumPy.
- » Características:
 - » Etiquetado: Cada elemento de uma Series tem um rótulo de índice associado, o que facilita o acesso a dados específicos. Homogênea: Todos os elementos de uma Series são do mesmo tipo de dados (inteiros, strings, etc.).
 - » Parte de um DataFrame: Uma Series pode ser vista como uma coluna individual em um DataFrame, mas também pode existir independentemente.
- » Exemplo: A coluna "Nome" desse DataFrame pode ser extraída como uma Series, contendo apenas os nomes das pessoas, etiquetados pelos índices das linhas.

Antes de executar o código a seguir, [clique aqui](#) para baixar o arquivo **iris-dataset.csv** e salve-o na pasta raiz (fora que qualquer diretório) do Google Drive vinculado à conta que você está utilizando para acessar esse notebook.

```
import pandas as pd
filename = '/content/drive/My Drive/Colab Notebooks/iris-dataset.csv'
data = pd.read_csv(filename) # Lê o arquivo do dataset iris em um dataframe
```

O comando **pd.read_csv(filename)** lê o arquivo CSV e cria um DataFrame **data** contendo todas as amostras e características do dataset Iris. Esse DataFrame é essencial para manipulação e análise dos dados.

Explorando o dataset

Vamos explorar rapidamente o dataset visualizando os primeiros registros e listando as colunas do DataFrame. Isso nos ajuda a entender a estrutura dos dados que estamos manipulando.

```
data.head() # Mostra os primeiros registros do dataframe
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
list(data.columns) # Mostra os nomes das colunas do dataframe
```

```
['sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety']
```

Os primeiros registros do DataFrame e a lista de colunas nos permitem verificar se os dados foram carregados corretamente. As colunas do dataset Iris geralmente incluem medidas como comprimento e largura das sépalas e pétalas, além da espécie da flor.

Preparação dos dados

Agora, vamos separar as características (atributos) que serão usadas para a clusterização. Neste exemplo, vamos utilizar as quatro primeiras colunas do DataFrame, que representam as medidas das flores.

```
X = data.iloc[:, 0:4].values # Separa somente as colunas das características
```

O código acima utiliza `iloc` para selecionar as quatro primeiras colunas do DataFrame `data` e armazena essas colunas na matriz `X`. Essa matriz será usada como entrada para o algoritmo de clusterização.

Redução de dimensionalidade com t-SNE

Antes de aplicar o algoritmo k-means, vamos reduzir a dimensionalidade dos dados usando t-SNE (t-Distributed Stochastic Neighbor Embedding). O t-SNE é uma técnica de redução de dimensionalidade particularmente útil para visualizar dados de alta dimensão. Ele transforma os dados de um espaço de características de alta dimensão para um espaço de 2 ou 3 dimensões, mantendo as relações de proximidade entre as amostras, o que é ideal para visualizar agrupamentos em um gráfico.

```
from sklearn.manifold import TSNE # Classe para redução da dimensionalidade
X_tsne = TSNE(n_components = 2).fit_transform(X) # Reduz de 4 para 2 características
```

set Iris para duas. A redução de dimensionalidade é especialmente útil quando queremos visualizar dados em um espaço 2D ou 3D sem perder muita informação sobre as relações entre as amostras.

Preparação para visualização

Agora que os dados foram reduzidos para duas dimensões, vamos organizá-los em um novo DataFrame, juntamente com as classes originais (espécies de íris) para facilitar a visualização.

```
data_tsne = pd.DataFrame(X_tsne) # Transforma as amostras obtidas pelo TSNE em  
dataframe  
classes = data.iloc[:, 4] # Extrai a coluna dos rótulos das classes  
data_tsne.insert(2, 'classe', classes) # Reúne as amostras e os rótulos das classes  
data_tsne = data_tsne.rename(columns = {0: "carac_tsne_0", 1: "carac_tsne_1"}) #Altera  
os nomes das colunas
```

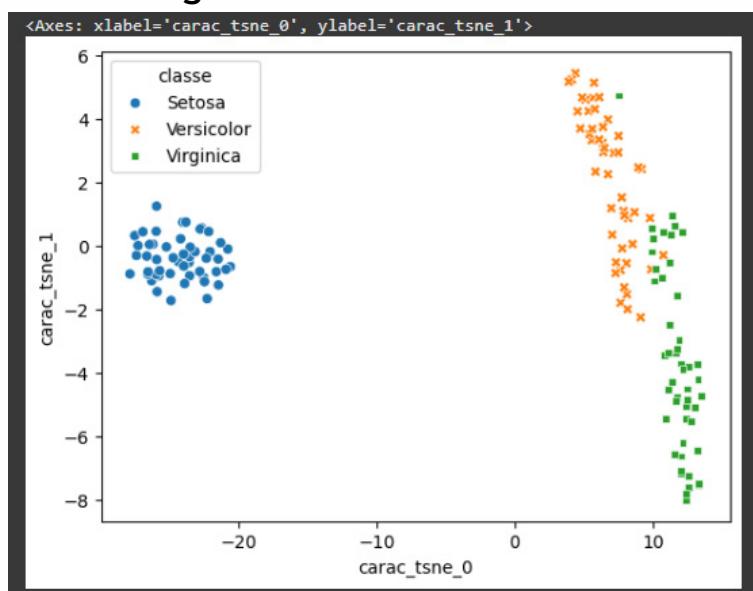
tantes do t-SNE e as classes das amostras. As colunas são renomeadas para refletir as novas características derivadas do t-SNE, facilitando a interpretação. Resta plotar esses dados em um gráfico de dispersão.

Introdução à biblioteca Seaborn

Seaborn é uma biblioteca Python baseada no Matplotlib que facilita a criação de gráficos estatísticos. Ela fornece uma interface de alto nível para desenhar gráficos atraentes e informativos. Seaborn integra bem com Pandas DataFrames, tornando muito simples a criação de gráficos a partir de dados tabulares. Seus recursos incluem gráficos de dispersão, gráficos de linha, gráficos de distribuição, gráficos de boxplot, entre outros.

```
import seaborn as sns  
sns.scatterplot(x = data_tsne['carac_tsne_0'], y = data_tsne['carac_tsne_1'], hue =  
data_tsne.classe, style = data_tsne.classe)
```

Figura 90 - Gráfico - Seaborn



Fonte: autoria própria.

Utilizamos a função **scatterplot** para gerar o gráficos com os seguinte parâmetros:

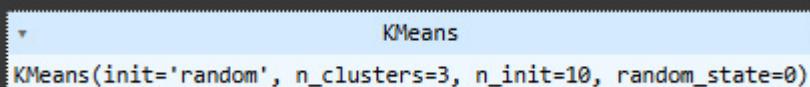
- » **x = data_tsne['carac_tsne_0']**: Define os valores para o eixo x do gráfico, que correspondem à primeira característica reduzida pelo t-SNE.
- » **y = data_tsne['carac_tsne_1']**: Define os valores para o eixo y do gráfico, que correspondem à segunda característica reduzida pelo t-SNE.
- » **hue = data_tsne.classe**: Determina a coloração dos pontos no gráfico com base na classe de cada amostra. Isso permite distinguir visualmente as diferentes classes (ou clusters) no gráfico.
- » **style = data_tsne.classe**: Diferencia as classes usando diferentes estilos de marcadores (como círculos, quadrados, etc.), proporcionando uma camada adicional de distinção visual entre as classes.

O gráfico de dispersão mostra como as amostras são distribuídas em um espaço bidimensional após a redução de dimensionalidade. As cores e os estilos indicam as diferentes classes, facilitando a visualização de possíveis agrupamentos naturais nos dados.

Aplicando o algoritmo K-Means

Agora que temos uma visualização inicial dos dados rotulados, vamos aplicar o algoritmo de clusterização k-means sem os rótulos das classes, ou seja, um aprendizado não supervisionado e verificar o resultado. Assim podemos comparar os 2 agrupamentos e, visualmente, checar a acurácia o k-means. K-means tenta dividir os dados em 'k' clusters, onde cada amostra pertence ao cluster mais próximo de seu centro (centroide). Após a aplicação do k-means, vamos visualizar os clusters obtidos e compará-los com as classes originais.

```
from sklearn.cluster import KMeans # Importação do algoritmo k - means
km = KMeans(n_clusters = 3, init = 'random', n_init = 10, max_iter = 300, tol = 1e-04, random_state = 0) # Parametrização do modelo
km.fit(X_tsne) # Computa os centroides dos clusters
```



No código acima, criamos um modelo k-means com os seguintes parâmetros:

- » **n_clusters**: Define o número de clusters a serem formados. Como sabemos que o dataset Iris tem três classes, usamos **n_clusters=3**. EM uma aplicação no mundo real esse número poderia ser escolhido com base no objetivo do agrupamento ou no deliamento experimental realizado por um especialista.
- » **init**: Método para inicialização dos centroides. Usamos **random** para escolher centroides iniciais aleatoriamente.
- » **n_init**: Número de vezes que o algoritmo k-means será executado com

diferentes centroides iniciais. O melhor resultado (com menor inércia) é retornado.

- » **max_iter**: Número máximo de iterações para uma única execução do k-means.
- » **tol**: Tolerância para declarar convergência, ou seja, o critério de parada baseado em pequenas variações na inércia.
- » **random_state**: Define a semente do gerador de números aleatórios para garantir reproduzibilidade.

Predição e atribuição aos Clusters

Após ajustar o modelo aos dados com **fit**, podemos prever a qual cluster cada amostra pertence usando **predict**. Essa etapa associa cada amostra ao cluster mais próximo, com base na distância ao centroide do cluster.

```
predicao = km.predict(X_tsne) # Associa os objetos aos índices da classe
```

O método **predict** retorna uma lista de rótulos de cluster, onde cada elemento indica o cluster ao qual a amostra correspondente pertence. Essa lista é usada para visualizar os clusters no espaço bidimensional.

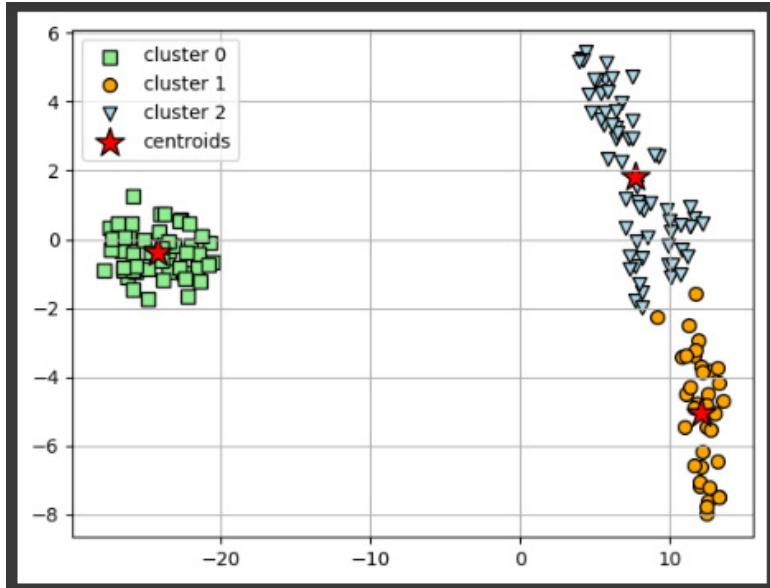
Visualizando os clusters

Vamos agora visualizar os clusters encontrados pelo k-means, colorindo cada um com uma cor diferente. Também vamos destacar os centroides dos clusters, que são os pontos centrais ao redor dos quais os clusters foram formados.

```
from matplotlib import pyplot as plt
plt.scatter(X_tsne[predicao == 0, 0], X_tsne[predicao == 0, 1], s = 50, c = 'lightgreen', marker = 's', edgecolor = 'black', label = 'cluster 0')
plt.scatter(X_tsne[predicao == 1, 0], X_tsne[predicao == 1, 1], s = 50, c = 'orange', marker = 'o', edgecolor = 'black', label = 'cluster 1')
plt.scatter(X_tsne[predicao == 2, 0], X_tsne[predicao == 2, 1], s = 50, c = 'lightblue', marker = 'v', edgecolor = 'black', label = 'cluster 2')

# Plota os centroides de cada cluster
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s = 250, marker = '*', c = 'red', edgecolor = 'black', label = 'centroids')
plt.legend(scatterpoints = 1)
plt.grid()
```

Figura 91 - Gráfico - Visualização de Cluster



Fonte: autoria própria.

No código acima, usamos a função `scatter` nativa do `matplotlib` para gerar o gráfico. Ao utilizar diretamente o `matplotlib`, apesar de serem necessários mais parâmetros, temos um controle maior da renderização. Nessa função:

- » `X_tsne[predicao == 0, 0]`: Combina a matriz `X_tsne` com a máscara booleana para selecionar todas as amostras (linhas) que pertencem ao cluster 0. O `0, 0` no final do índice indica que estamos interessados apenas na primeira coluna (a primeira dimensão) dessas amostras. Já `X_tsne[predicao == 0, 1]` fornece os valores das características da segunda coluna.
- » `s = 50`: Define o tamanho dos marcadores (pontos) no gráfico. Aqui, `s = 50` significa que cada ponto terá um tamanho moderado.
- » `c = 'lightgreen'`: Define a cor dos marcadores como verde claro (lightgreen), o que facilita a visualização e a distinção dos pontos pertencentes ao cluster 0.
- » `marker = 's'`: Define o formato dos marcadores como quadrados (`s`), diferenciando visualmente os pontos deste cluster dos de outros clusters que podem usar formatos diferentes.
- » `edgecolor = 'black'`: Define a cor da borda dos marcadores como preta, o que pode ajudar a destacar os marcadores no gráfico, especialmente se a cor de preenchimento for clara.
- » `label = 'cluster 0'`: Define o rótulo (label) deste conjunto de pontos como "cluster 0". Esse rótulo será utilizado na legenda do gráfico, permitindo identificar visualmente quais pontos pertencem ao cluster 0.

O gráfico resultante mostra os três clusters encontrados pelo k-means em diferentes cores, com os centroides destacados em vermelho. Essa visualização nos permite avaliar a separação dos clusters e a proximidade das amostras aos seus centroides.



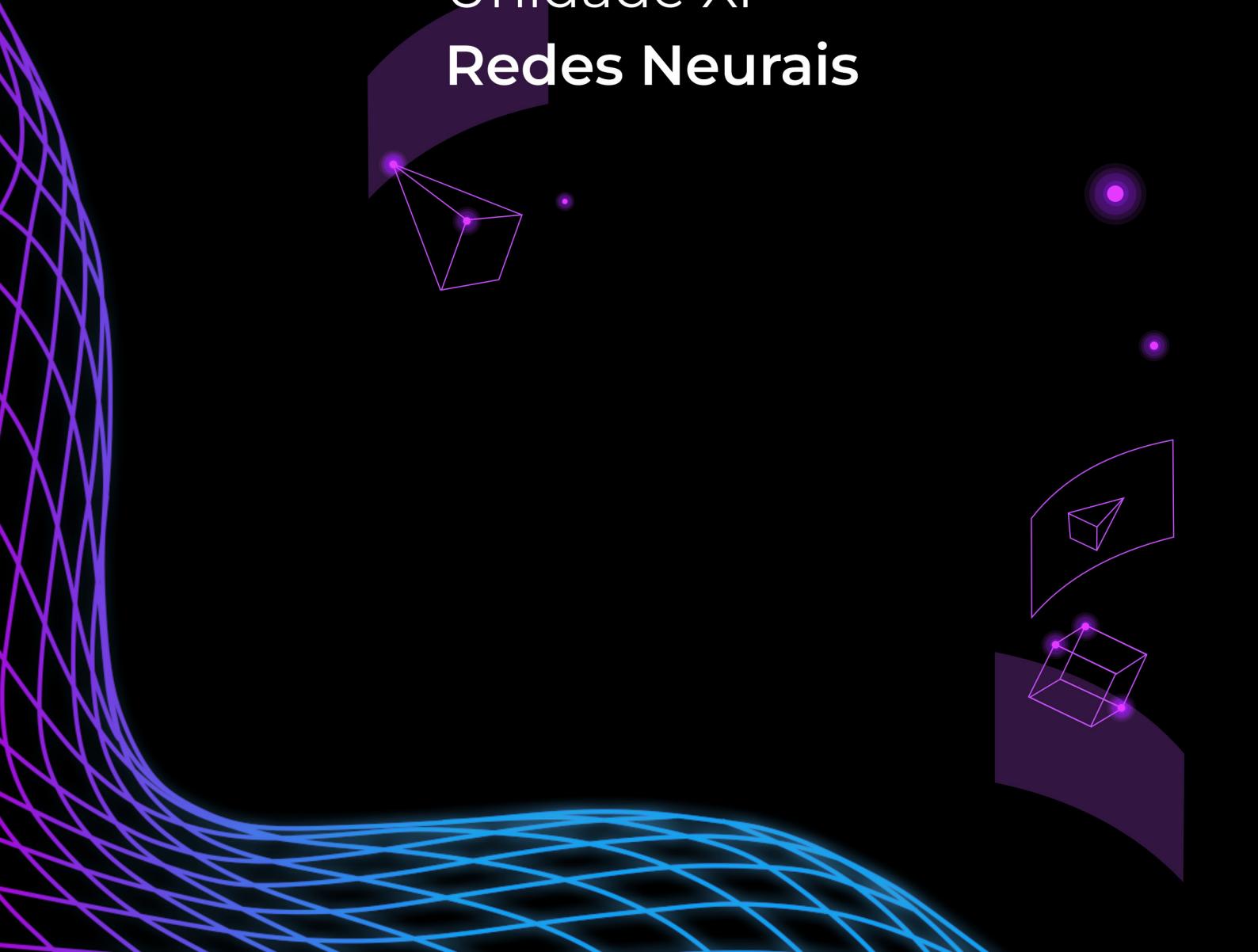
PARA RELEMBRAR...

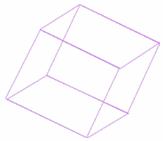
- » Apesar de semelhantes, classificação e agrupamento são técnicas distintas. Classificação temos **dados rotulados**, já sabemos de antemão qual é a resposta correta. Com métodos de agrupamento, os dados estão **sem rótulos** e buscamos identificar possíveis formas de agrupamento desses dados criando algumas classes.

SAIBA MAIS...

- » *k-means* - <https://www.youtube.com/watch?v=4b5d3muPQmA>
- » DBSCAN - <https://www.youtube.com/watch?v=RDZUDRSDOok>
- » Agrupamento hierárquico - <https://www.youtube.com/watch?v=7xHsRkOdVwo>
- » Coeficiente de silhueta - https://www.youtube.com/watch?v=a2Kg2_13L8M
- » Método Cotovelo - https://www.youtube.com/watch?v=4E_DFMt60rc
- » Como analisar agrupamentos - <https://www.youtube.com/watch?v=MHnoWs-BJpeM>

**Unidade XI
Redes Neurais**





Unidade XI - Redes Neurais



As RNAs constituem um ramo do Aprendizado de Máquina, o qual, por sua vez, é um subcampo da IA. Inspiradas pela estrutura e função das redes neurais biológicas, as RNAs são sistemas de processamento de informação compostos por múltiplos elementos de processamento interconectados, chamados neurônios artificiais. Esses neurônios são organizados em camadas que transformam entradas em saídas, permitindo que a rede aprenda e faça inferências a partir de dados.

Dentro do espectro do ML, as RNAs são particularmente notáveis por sua capacidade de modelar relações complexas e não lineares. Elas são uma ferramenta poderosa para tarefas como classificação, regressão, e até mesmo para geração de conteúdo, como em Redes Generativas Adversárias (GANs). As RNAs se destacam principalmente quando se trata de aprender representações úteis de dados, o que é fundamental em tarefas de visão computacional, PLN e muitas outras áreas.

A história das RNAs é marcada por avanços e retrocessos, com períodos de entusiasmo seguidos por períodos de ceticismo. A ideia de simular a atividade cerebral surgiu na década de 1940, com trabalhos pioneiros como o de Warren McCulloch e Walter Pitts, que propuseram um modelo simples de neurônio artificial.

O desenvolvimento do Perceptron, na década de 1950 por Frank Rosenblatt, foi um marco importante, mas logo se mostrou limitado devido à sua incapacidade de resolver problemas não lineares. Na década de 1980, o algoritmo de retropropagação (*backpropagation*) foi popularizado, o que permitiu o treinamento eficiente de redes neurais multicamadas.

A virada do século XXI trouxe avanços significativos, impulsionados por melhorias em hardware, algoritmos e disponibilidade de dados. O surgimento das DL revolucionou campos como o reconhecimento de voz e a visão computacional. A partir daí, o interesse pelas RNAs cresceu exponencialmente, levando a aplicações práticas em diversos setores e contribuindo para o desenvolvimento de tecnologias inovadoras.



11.1 Neurônio Artificial

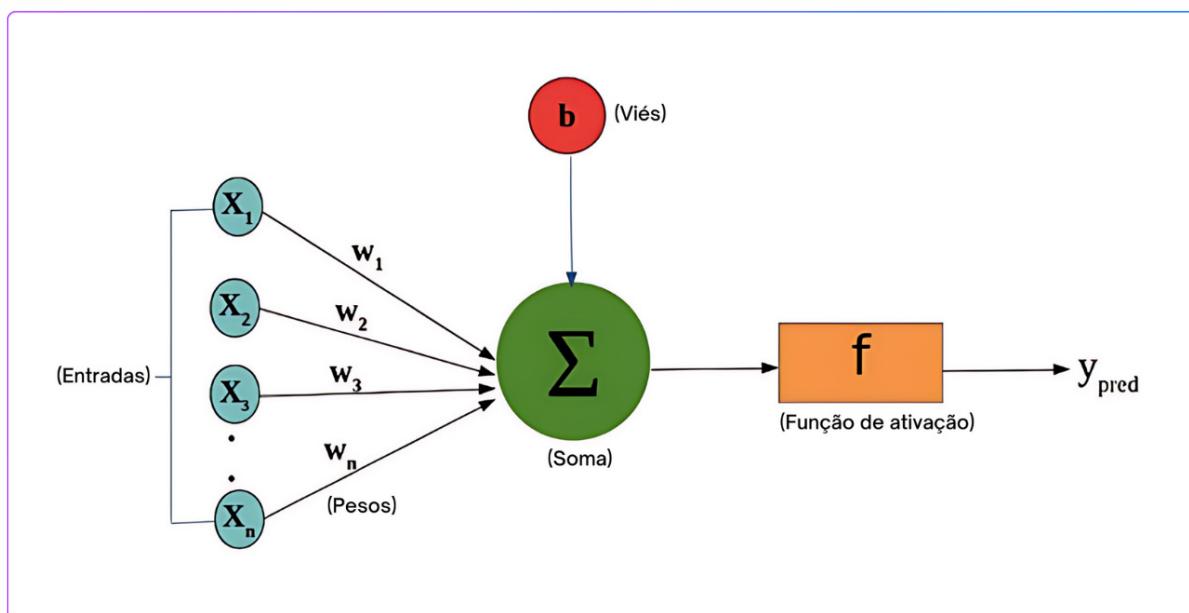
O neurônio biológico é a unidade básica de processamento no cérebro humano, caracterizado por sua capacidade de receber sinais elétricos, processá-los e transmiti-los a outros neurônios. Um neurônio artificial, por sua vez, é uma abstração simplificada desse modelo biológico. Ele recebe entradas (sinais), pondera cada uma delas com um

peso específico e as combina, geralmente adicionando um viés. A soma ponderada é então passada por uma função de ativação, que determina se e como o “neurônio” deve responder — ou seja, se deve “disparar”.

Embora a inspiração seja clara, é importante notar que a analogia não é perfeita. As RNAs são construções simplificadas que capturam apenas aspectos gerais da função neural biológica, sendo projetadas para otimizar a eficiência computacional e a capacidade de aprendizado de padrões, e não para replicar o cérebro humano com precisão.

O neurônio artificial, ilustrado na figura abaixo, recebe múltiplas entradas, que podem ser sinais de outros neurônios ou dados de entrada direta (X), cada uma com um peso (W) associado. Esses pesos representam a força das conexões entre os neurônios, similar às sinapses no cérebro humano.

Figura 92 - Esquema de um neurônio artificial - Perceptron



Fonte: adaptada de [Medium \(2024\)](#).

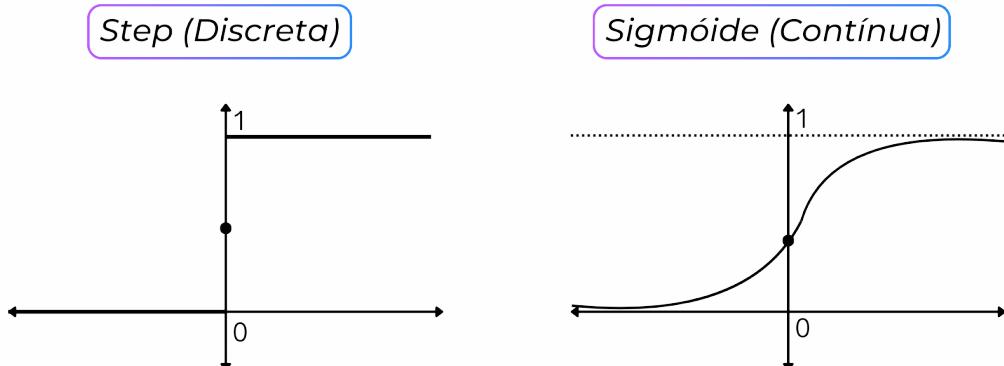
As entradas ponderadas são somadas, e um viés (b) pode ser adicionado a esta soma para ajustar o ponto de ativação do neurônio. O resultado é então passado através de uma função de ativação, que introduz não linearidade ao modelo. A função de ativação (f) determina a saída do neurônio, que pode ser enviada para outros neurônios em camadas subsequentes ou ser parte da saída final da rede.

Os pesos (W) em uma rede neural artificial são parâmetros ajustáveis que determinam a influência relativa das entradas na saída do neurônio. Durante o treinamento da rede, os pesos são modificados através do processo de aprendizado, permitindo que a rede ajuste suas respostas para melhor se adequar aos dados.

O viés, também conhecido como *bias*, é um parâmetro adicional que permite ajustar a resposta do neurônio de forma independente das entradas, funcionando como uma tentativa de representar o viés presente no problema.

A função de ativação tem a função de transformar o resultado do somatório em uma saída coerente. Inicialmente, foi utilizada uma função *step*, que respondia com 0 quando o somatório resultava em um valor negativo e com 1 quando o valor era positivo. Posteriormente, começaram a ser usadas funções diferentes, como a sigmoide, que mapeia a saída em valores contínuos entre 0 e 1, como ilustrado na figura a seguir.

Figura 93 - Funções de ativação step e sigmoide



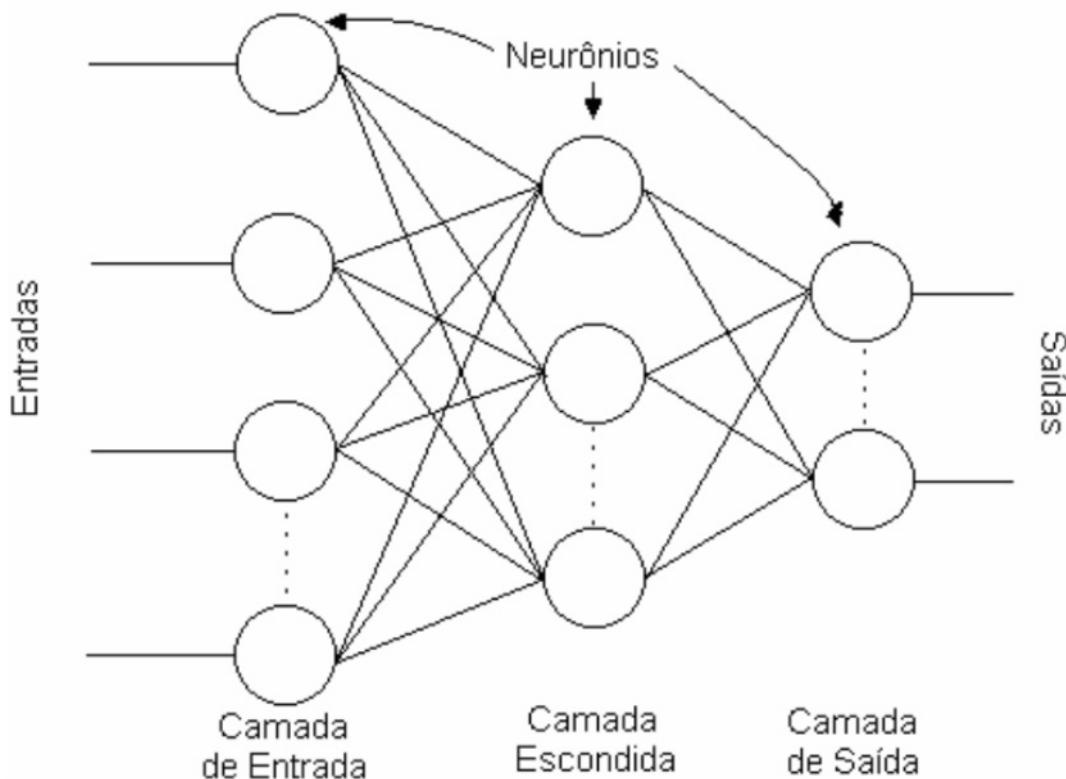
Fonte: autoria própria.

Este modelo de neurônio foi inicialmente proposto como uma nova abordagem de Aprendizado de Máquina, capaz de resolver múltiplas tarefas. No entanto, devido à sua simplicidade, não conseguiu obter resultados significativos. O avanço das RNAs ocorreu com a introdução do algoritmo de *backpropagation*, que possibilitou a criação de redes com múltiplas camadas, conhecidas como *Multi-Layer Perceptron* (MLP).

11.2 Multi-Layer Perceptron

O cérebro humano é composto por uma complexa rede de neurônios biológicos interligados. De maneira similar, às pesquisas em RNAs buscaram formas de combinar vários neurônios artificiais para formar uma rede mais complexa. A solução surgiu com o algoritmo de *backpropagation*, que deu origem às redes chamadas MLPs. Como o nome sugere, essas redes organizam os perceptrons, o neurônio artificial proposto anteriormente, em forma de camadas, como ilustrado na figura a seguir.

Figura 94 - Esquema básico de um Multi-Layer Perceptron



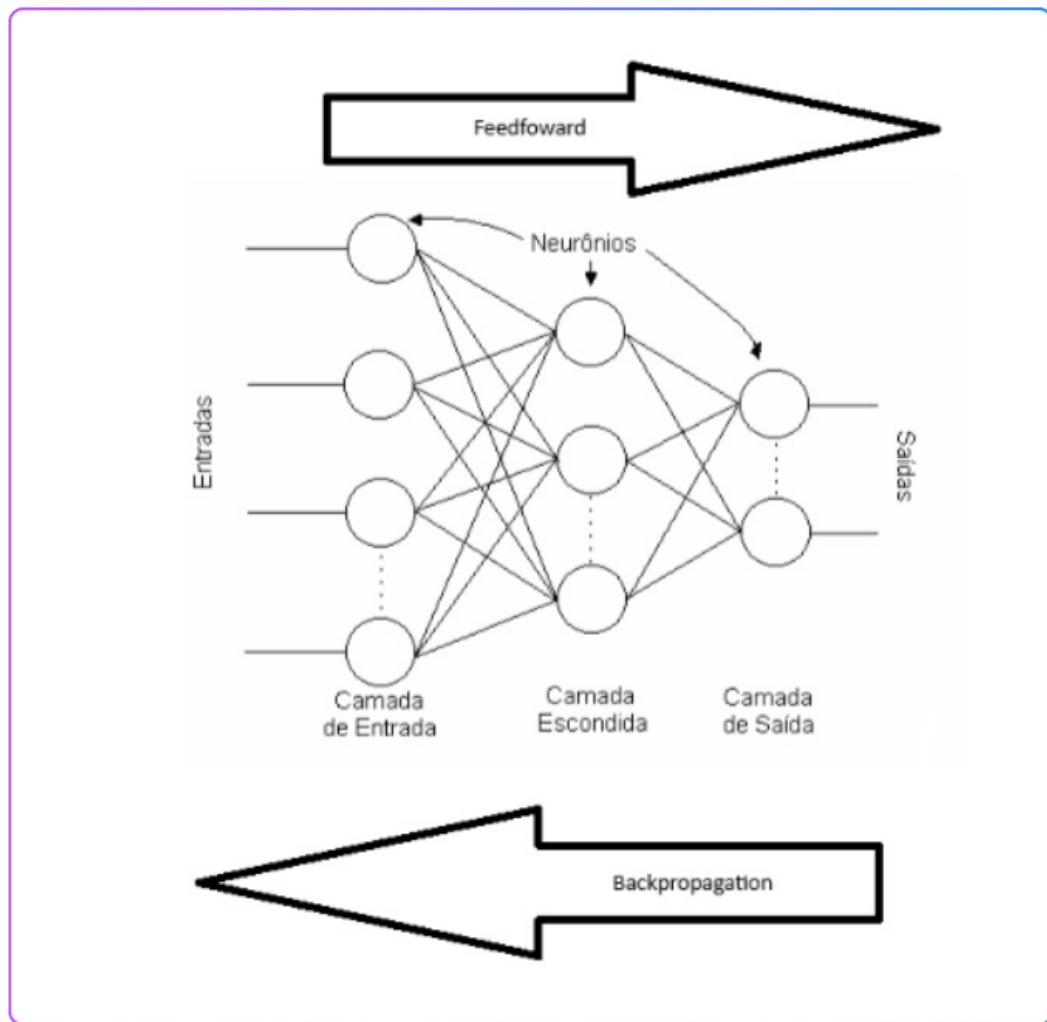
Fonte: adaptada de [Xiang \(2024\)](#).

Em um MLP temos três principais estruturas:

- » **Camada de Entrada:** A camada de entrada recebe as variáveis do problema a ser tratado. Os neurônios desta camada multiplicam seus pesos pelas variáveis, com cada variável tendo um peso independente. Por conta disso, o número de neurônios é dado pelo número de entradas.
- » **Camada Escondida:** A camada escondida recebe as saídas da camada anterior e gera uma nova saída. O conceito por trás da camada escondida é proporcionar uma interpretação mais complexa dos dados por meio da combinação de camadas. Um MLP básico possui ao menos uma camada escondida, mas o mesmo processo de treinamento pode ser aplicado a várias camadas, o que pode proporcionar uma compreensão mais profunda do problema. O número de neurônios na camada escondida é arbitrário e quanto mais, maior a capacidade da rede.
- » **Camada de Saída:** A camada de saída é responsável por fornecer a resposta ao problema, seja uma classificação ou regressão. Ela transforma a abstração gerada pela camada escondida em saídas específicas para o problema tratado. Em um problema de classificação com duas classes, por exemplo, é comum termos dois neurônios de saída, cada um representando a probabilidade da entrada pertencer a uma das classes.

O *backpropagation* é o algoritmo responsável por treinar o MLP. Em um treinamento de uma RNA temos dois estágios: o *feedforward* e logo após o *backpropagation*, como na figura a seguir. No *feedforward* os diversos exemplos de um *dataset* são passados pela rede em direção a saída, isso faz com que tenhamos respostas para um dos exemplos e com base nessas respostas é calculado o erro do MLP.

Figura 95 - Esquema básico de um *Multi-Layer Perceptron* com indicação de fluxo de dados no *feedforward* e *backpropagation*



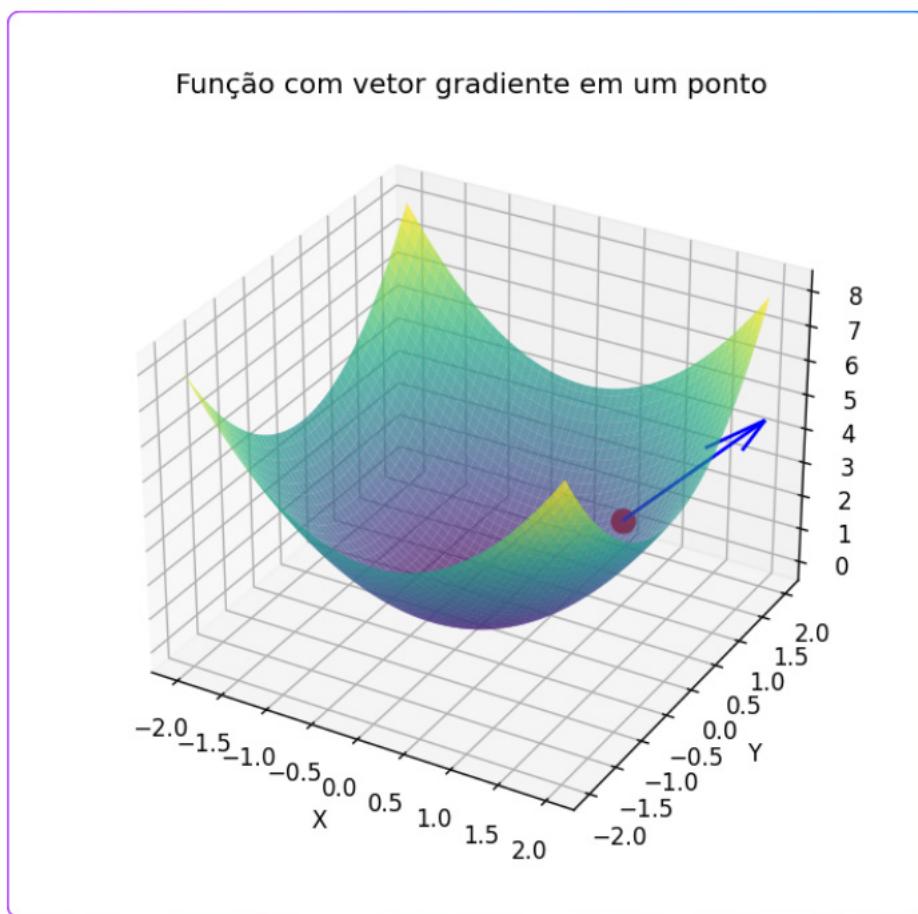
Fonte: adaptada de [Xiang, 2024](#).

O erro é mensurado pela chamada função de custo, também chamada de função de perda ou *loss function* em inglês. A função de custo é responsável por guiar o treinamento para que o MLP reduza o erro a cada tentativa. Uma função de custo comum e que foi usada inicialmente para treinar MLPs é o MSE. A função consiste em calcular a média do erro quadrático de cada amostra como mostra a equação:

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

Na equação o n se refere as n amostras do conjunto de treino, o y é o rótulo, resposta esperada, e \hat{y} a resposta calculada pela rede. Existem outras funções de perda como a Entropia Cruzada, Divergência KL e Erro Absoluto Médio, cada uma possui suas características e são usadas conforme o tipo de saída da rede neural. A partir do erro é calculado o vetor gradiente em cada neurônio da camada de saída e logo em seguida ajustados os pesos e o viés. O vetor gradiente é um operador no cálculo vetorial que indica a direção na qual uma função cresce mais rapidamente. Na figura a seguir, podemos ver um exemplo com a superfície de uma função e o vetor gradiente calculado em um ponto específico.

Figura 96 - Exemplo de vetor gradiente em uma função com duas variáveis



Fonte: autoria própria.

Como a função de perda mensura o erro do modelo, ao movimentarmos os pesos na direção oposta ao gradiente, obteremos novos pesos que reduzirão o erro calculado. Esse processo de otimização é chamado de Gradiente Descendente, e a repetição iterativa deste processo realiza o treinamento do MLP, reduzindo progressivamente o erro em relação às amostras de treinamento.

Para corrigir as camadas escondidas e de entrada de um MLP o cálculo do gradiente é realizado novamente a cada camada, gerando assim uma série de cálculos que propagam o erro, daí o nome *backpropagation*. A retropropagação do erro garante que a rede

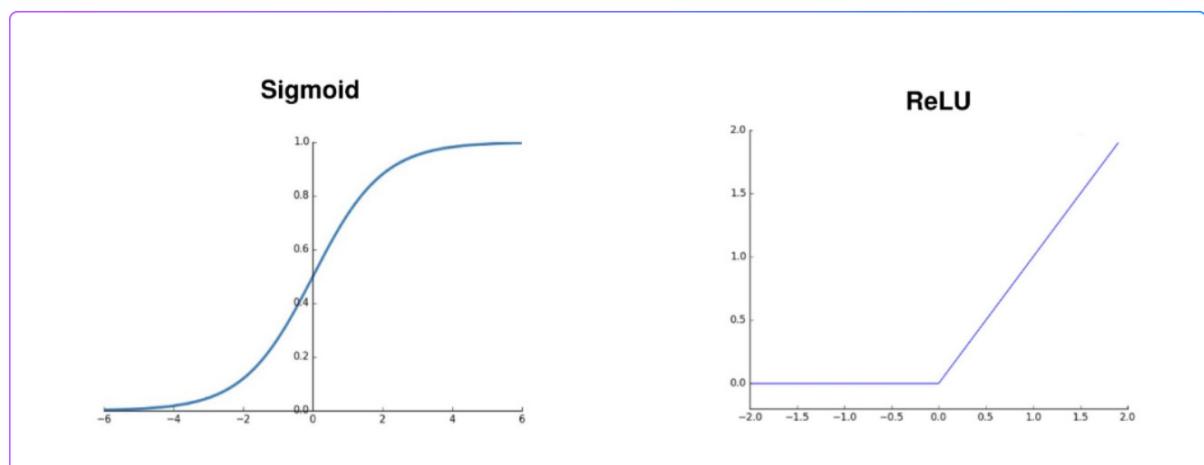
seja ajustada de forma que as camadas trabalhem em conjunto para minimizar o erro e assim gerar respostas corretas.

O MLP oferece uma capacidade muito maior do que um neurônio artificial individual, pois, com várias camadas, é capaz de construir funções discriminantes complexas. No entanto, com a introdução de várias camadas, surge o problema do *vanishing gradient*, ou “sumiço do gradiente” em português. Ao retropropagar o gradiente para camadas anteriores, temos um produto de múltiplos gradientes, o que se torna problemático devido à função de ativação sigmoide, que gera gradientes pequenos e sempre menores que 1. Como resultado, quanto mais camadas escondidas, menor o gradiente nas primeiras camadas, tornando-as mais difíceis de serem treinadas. Este problema conteve o avanço das redes neurais por algumas décadas e a solução para este problema deu origem às chamadas DL.

11.3 Deep Learning

O estudo das redes neurais avançou significativamente após a criação do *backpropagation*, resultando em diversos aprimoramentos em relação ao MLP clássico. Um dos primeiros avanços foi a introdução da função *Rectified Linear Unit* (ReLU), que substituiu a função sigmoide e ajudou a mitigar o problema do *vanishing gradient*. Como mostrado na figura a seguir, a ReLU funciona de maneira simples, permitindo a passagem de sinais positivos e anulando valores negativos, substituindo-os por 0. Esse funcionamento é suficiente para permitir que os neurônios mapeiem suas saídas, ao mesmo tempo em que gera gradientes melhores, pois a derivada (inclinação) da função ReLU é 1 e constante. Com isso, multiplicar seus gradientes através de várias camadas não apresenta o mesmo problema que ocorre com a sigmoide. Com uso da ReLU surgiram as primeiras redes neurais com muitas camadas, caminhando assim para as redes profundas.

Figura 97 - Funções de ativação sigmoide e ReLU



Fonte: adaptada de [Locke \(2023\)](#).

Outros pontos avançaram para o surgimento das redes profundas, sendo os principais:

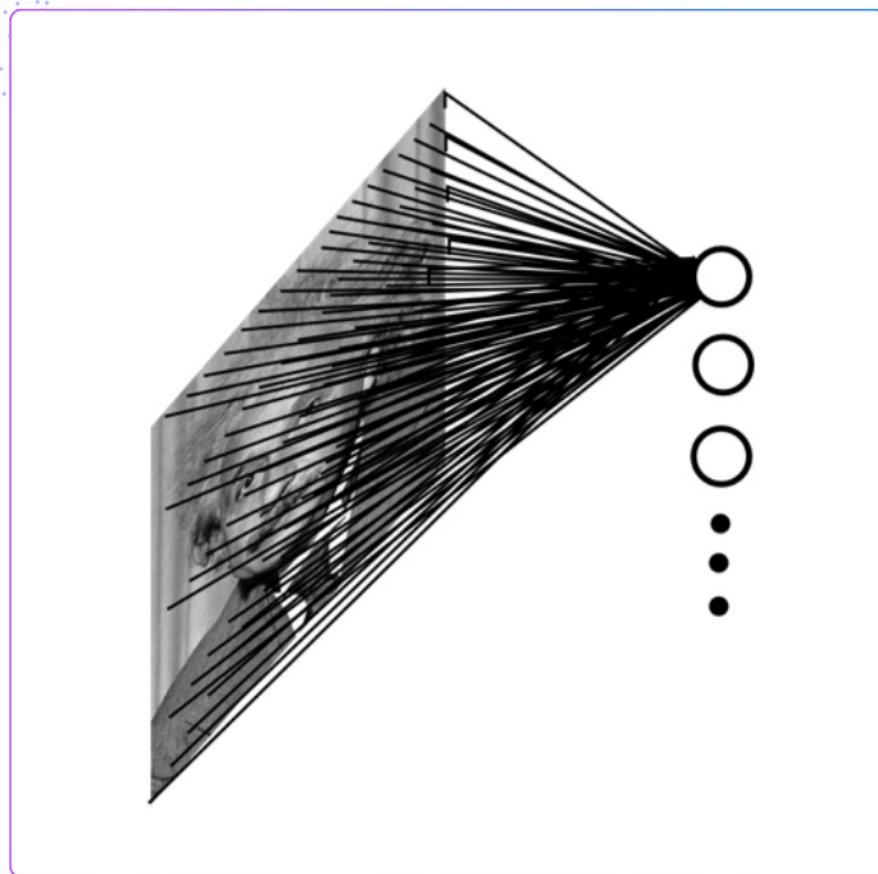
- » **Otimizadores:** Novos algoritmos para a aplicação do gradiente nos pesos, também conhecidos como otimizadores, foram propostos para otimizar as redes de maneira mais rápida e eficiente.
- » **Inicialização dos pesos:** Toda rede neural precisa iniciar com alguns pesos aleatórios para dar início ao processo de *feedforward*. Estudos descobriram formas melhores de definir esses pesos iniciais, melhorando o desempenho do treinamento.
- » **Regularização:** O sobreajuste dos dados sempre foi um problema nas RNAs, pois com muitas camadas e neurônios, o modelo passa a ter alta capacidade, o que facilita o sobreajuste aos dados de treinamento. Para evitar esse problema, foram propostas técnicas como *Dropout* e *Batch Normalization*.
- » **Infraestrutura computacional:** O surgimento de hardware especializado contribuiu enormemente para a evolução das RNAs. Unidades de Processamento Gráfico (GPUs) e Unidades de Processamento de Tensor (TPUs) viabilizaram o treinamento de redes muito maiores do que se tinha anteriormente.

Além dos pontos apontados acima, tivemos também o desenvolvimento de arquiteturas alternativas ao MLP. Sendo a principal a CNN.

11.3.1 CNNs: Redes Neurais Convolucionais

As CNNs propostas nos anos 90 substituem o neurônio artificial, como o perceptron, por uma operação de convolução que pode ser treinada. A convolução é uma operação entre sinais que tem como objetivo buscar por um padrão. A ideia de utilizar uma convolução vem da dificuldade do MLP em lidar com imagens. Uma imagem digital é um mapa de pixel e para usar uma imagem como entrada em um MLP seria necessário um peso para cada pixel. Esse arranjo gera uma entrada com altíssima dimensão, com uma imagem de 200x200 seriam 40 mil entradas, e os pesos não possuem compreensão de que pixels podem ser vizinhos.

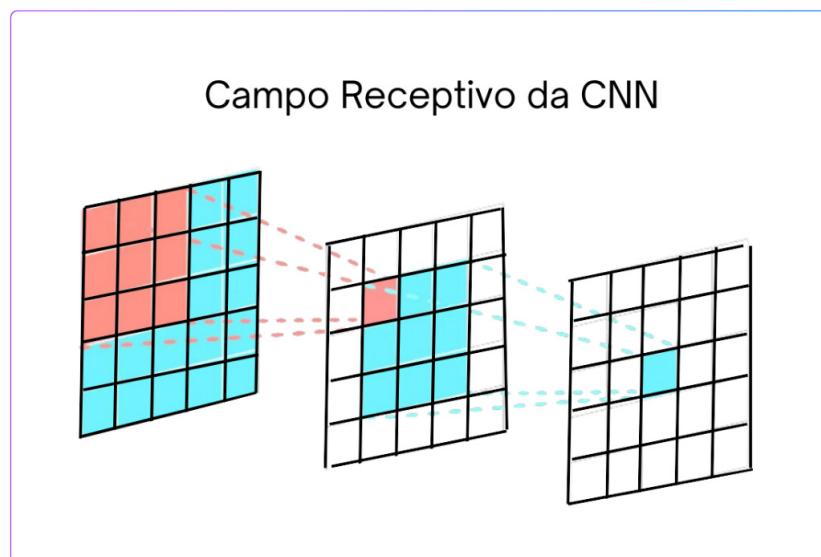
Figura 98 - Esquema básico de um *Multi-Layer Perceptron* ao receber uma imagem



Fonte: [Anderson \(2024\)](#).

A convolução, por outro lado, oferece uma operação que permite compreender a vizinhança dos pixels e reduz a necessidade de tantos pesos, pois funciona como um filtro que captura uma região da imagem, gerando uma interpretação dessa região como saída. A figura a seguir mostra de forma simplificada uma sequência de duas convoluções sendo aplicadas.

Figura 99 - Esquema de um campo receptivo de duas camadas convolucionais

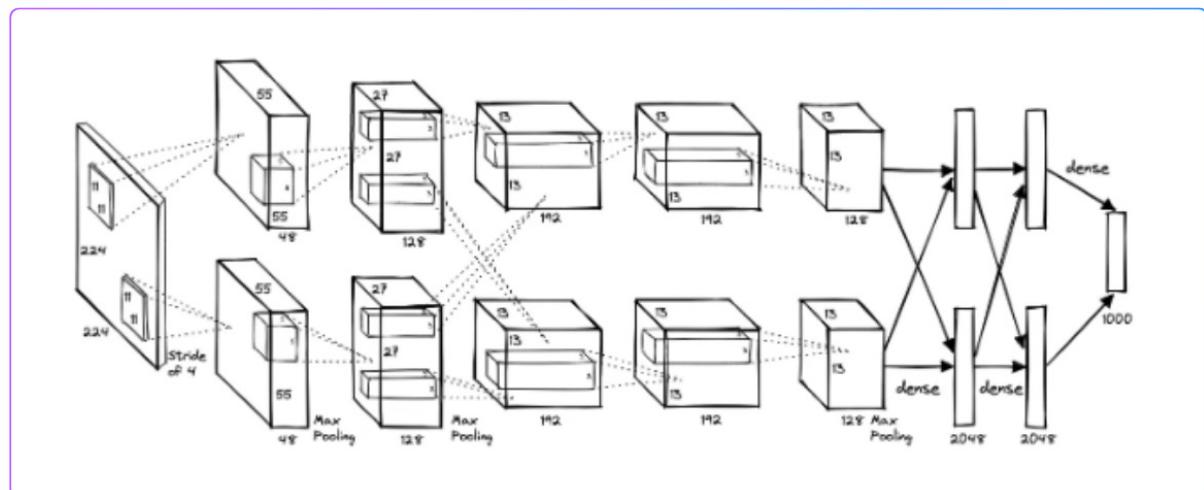


Fonte: Adaptada de [Deepia \(2024\)](#).

A convolução pode ser aplicada em toda a imagem, reutilizando o mesmo filtro, tornando-se assim muito mais eficiente, em termos de número de parâmetros, do que um MLP. A inspiração para as CNNs vem dos estudos sobre campos receptivos. A descoberta do funcionamento da percepção visual revelou que os sinais gerados pelos olhos são processados em várias camadas de neurônios, criando uma abstração do que está sendo observado. Assim, as CNNs são construídas com uso das convoluções seguidas de um MLP, que realiza a classificação a partir da abstração criada.

Apesar das CNNs terem sido introduzidas nos anos 90 foi só com o avanço das RNAs citados anteriormente que passamos a ter CNNs com grande número de camadas. Em 2012 o surgimento da arquitetura chamada AlexNet deu início ao que hoje chamamos de DL. Uma grande arquitetura, treinada em GPU, que combina várias técnicas de otimização e é treinada em um grande volume de dados.

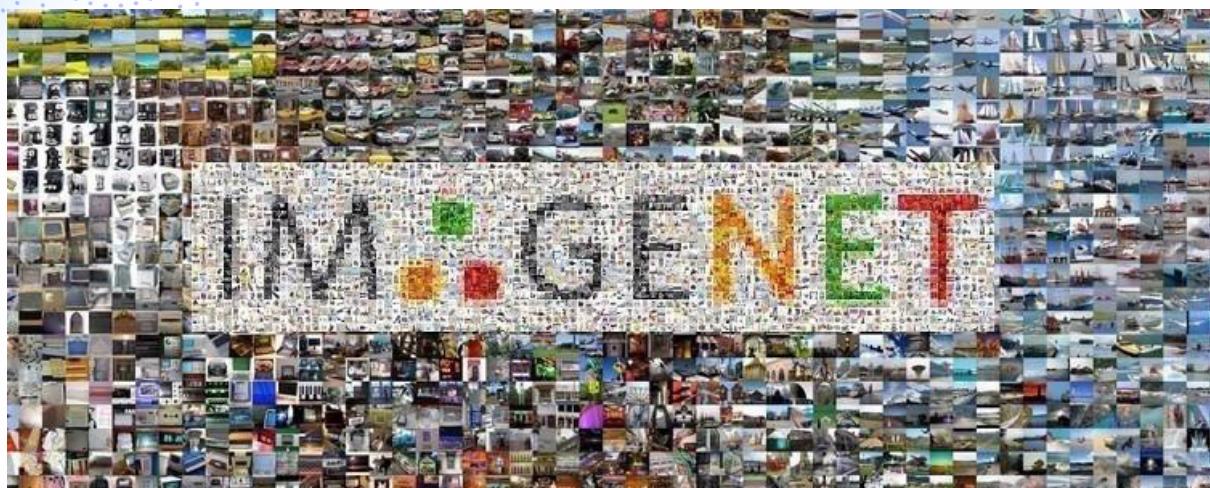
Figura 100 - Esquema da AlexNet



Fonte: [Pinecone \(2012\)](#).

A AlexNet contém cinco camadas convolucionais e um MLP com três camadas em sua saída, conforme mostrado na figura anterior, e foi treinada com o conjunto de imagens chamado ImageNet, que contém cerca de 1,2 milhão de imagens obtidas na internet, anotadas em 1000 categorias. O desempenho reportado por essa arquitetura foi muito superior aos métodos tradicionais de visão computacional da época.

Figura 101 - Exemplo de imagens do ImageNet

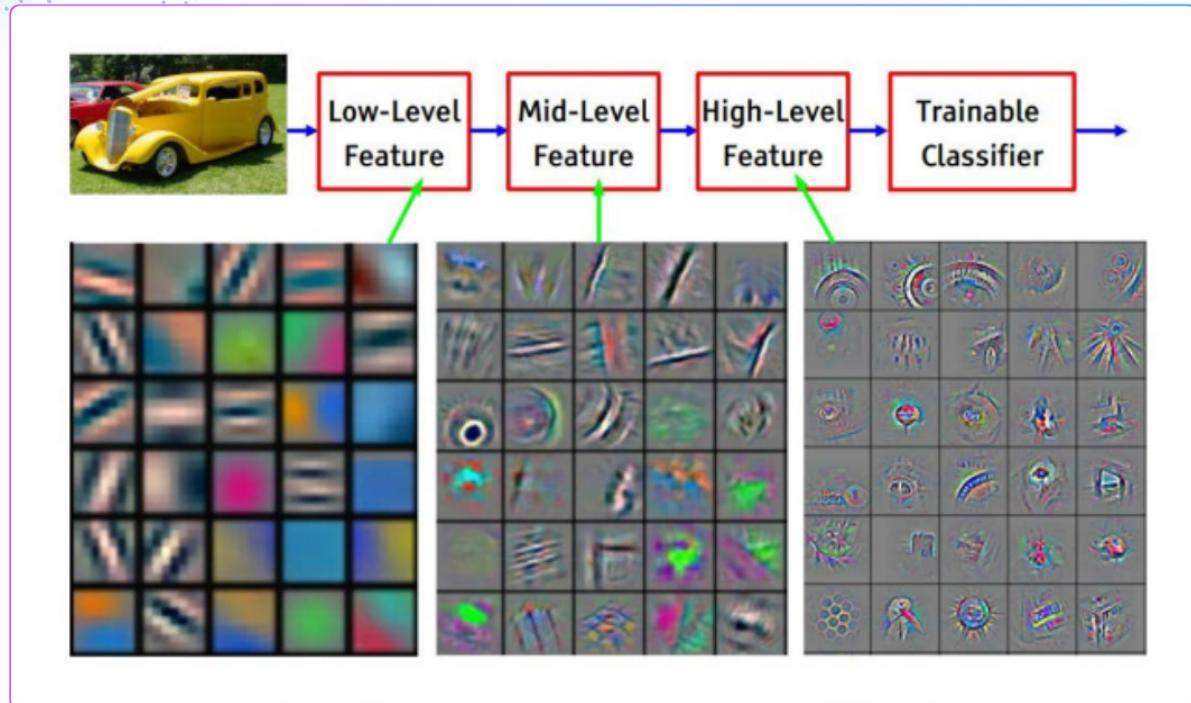


Fonte: [Gluon \(2012\)](#).

O resultado da AlexNet trouxe grande atenção às redes neurais a partir de 2012, levando ao surgimento de várias outras arquiteturas e avanços nas RNAs. Na visão computacional, surgiram soluções revolucionárias para tarefas como segmentação de imagens, detecção de objetos e descrição de imagens. Grande parte desse avanço se deve à introdução do conceito de Transferência de Aprendizado, conhecido como *transfer learning* em inglês.

A transferência de aprendizado ocorre ao treinar uma CNN em um grande conjunto de dados, como o ImageNet, em uma tarefa mais simples — no caso do ImageNet, a classificação de imagens — e, em seguida, usar os pesos resultantes para iniciar outro treinamento em uma tarefa semelhante, mas com um menor volume de dados disponível. Assim, o primeiro treino passou a ser chamado de pré-treino, enquanto o segundo treinamento é conhecido como ajuste fino, ou *fine tuning*. Ao retreinar a rede, ela é capaz de trazer consigo o entendimento básico dos objetos do dataset do seu treino inicial, como exemplificado na figura a seguir, e adaptar esse conhecimento prévio para uma nova tarefa.

Figura 102 - Visualização das convoluções obtidas ao treinar uma CNN



Fonte: adaptada de [Zeiler, Fergus, 2013](#).

Essa abordagem permitiu o uso de DL em diversos problemas que anteriormente não eram abordados com RNAs devido à falta de volume de dados. No entanto, as CNNs tratavam majoritariamente de problemas que envolviam imagens como entradas. Em outros campos, como Linguagem Natural e Processamento de Áudio, as CNNs e o *transfer learning* ainda enfrentavam dificuldades para obter bons resultados. A mudança nessas áreas ocorreu com o surgimento da arquitetura Transformer, que introduziu uma operação diferente tanto do MLP quanto da convolução, permitindo o *transfer learning* com textos e áudios.

11.3.2 Transformer

Em 2017 foi introduzida pela equipe da Google a arquitetura chamada Transformer. Inicialmente proposta para tradução de textos, a arquitetura usa de uma operação chamada “mecanismo de atenção” para processar suas entradas. O mecanismo de atenção é uma técnica que permite que o modelo foque em partes específicas da entrada enquanto realiza uma tarefa. Ele atribui diferentes pesos a diferentes partes da entrada, destacando informações mais relevantes, como no exemplo da figura a seguir, melhorando assim a performance em tarefas sequenciais e de PLN.

Figura 103 - Visualização da atenção em um problema de tradução



Fonte: autoria própria.

Com o uso dos *Transformers* (transformadores), foi possível realizar um pré-treino eficiente tanto em Linguagem Natural quanto em Áudio. Utilizando a arquitetura pré-treinada na tarefa de modelagem de linguagem, que consiste em prever a próxima palavra de um texto, surgiram modelos como BERT, GPT e Llama, que revolucionaram o PLN ao resolverem tarefas anteriormente consideradas complexas. O impacto foi semelhante ao das primeiras CNNs treinadas com o ImageNet, que, no caso do PLN, introduziu o DL em várias tarefas anteriormente tratadas com métodos clássicos de PLN. No contexto do processamento de áudio, modelos baseados em *Transformers* como Whisper, Wav2Vec, HuBERT e Tacotron viabilizaram tarefas complexas, como reconhecimento, transcrição e sintetização de fala.

11.3.3 Rede Neurais Generativas

RNAs atualmente passaram a dominar a maior parte do campo da IA devido ao ótimo desempenho em tarefas complexas. A possibilidade de explorar os grandes conjuntos de dados que se formaram a partir da digitalização de processo e do avanço da internet age como combustível em soluções baseadas em RNAs. Com o desenvolvimento das redes neurais geradoras e de modelos, o campo testemunhou uma explosão na criação de conteúdo sintético, desde imagens hiper-realistas até música e vídeo. Essas inovações não só melhoraram o desempenho em tarefas tradicionais, mas também abriram novas aplicações e indústrias, como arte digital, realidade aumentada e IA criativa. O contexto atual das RNAs é, portanto, marcado por um ciclo virtuoso de inovação, onde grandes dados alimentam redes mais poderosas, que, por sua vez, geram novos dados e tendências, continuando a impulsionar o progresso tecnológico.

11.3.4 Frameworks para DL

Com o avanço das RNAs e o surgimento do DL, foram criados *frameworks* para viabilizar a implementação de novas arquiteturas neurais. Inicialmente, as implementações eram feitas usando Nvidia *Compute Unified Device Architecture* (CUDA), que permite a programação de GPUs de forma geral, e não apenas para a geração de gráficos. No entanto, devido à complexidade da programação com CUDA, foram propostos *frameworks* que servem como interface para o uso das GPUs no treinamento de redes neurais.

Entre os vários *frameworks* que surgiram, os principais são o TensorFlow e o PyTorch. O TensorFlow foi inicialmente desenvolvido pela Google para uso interno e, em 2015, foi disponibilizado como uma biblioteca de código aberto. O PyTorch foi lançado pela Meta AI (antigo Facebook) em 2016 e foi desenvolvido a partir do framework Torch. Ambos os *frameworks* são baseados principalmente em Python e atuam como uma interface para operações implementadas em CUDA. Essas bibliotecas foram fundamentais para popularizar a implementação e a criação de novos modelos de DL, facilitando o processo de codificação do treinamento e da inferência.



SAIBA MAIS...

- » Perceptron - <https://youtu.be/6yYUc6nU3Cw?si=LZGZbtV-Ph0d4IJo>
- » Redes Neurais - <https://www.youtube.com/watch?v=CqOfi41LfDw>
- » Redes Neurais - <https://youtu.be/XxZ0BibMTjw?si=vtO6zjtguK1a95nX>
- » Gradiente Descendente -
 - » https://youtu.be/31w-xQX0Z_8?si=dkeqa39mdwSTYzhB
- » Propagação reversa - <https://www.youtube.com/watch?v=IN2XmBhILt4>
- » Propagação reversa - <https://youtu.be/DGNbd2FGw2s?si=HJMNSUrcfyBsQcG4>
- » ReLU - <https://www.youtube.com/watch?v=68BZ5f7P94E>
- » RNN - <https://www.youtube.com/watch?v=AsNTP8Kwu80>
- » LSTM - <https://www.youtube.com/watch?v=YCzL96nL7j0>
- » Transformers - <https://www.youtube.com/watch?v=zxQyTK8quyY>
- » CNN - https://youtu.be/n4rmrZg1_58?si=NxSTMjKr9Ax0lxaH
- » AlexNet and ImageNet: The Birth of Deep Learning: -
 - » <https://www.pinecone.io/learn/series/image-search/imagenet/>
- » Tutorial de Pytorch: <https://pytorch.org/tutorials/beginner/basics/intro.html>
- » Tutorial de TensorFlow: <https://www.tensorflow.org/learn?hl=pt-br>

PARA RELEMBRAR...

Estrutura das Redes Neurais:

- » Camada de Entrada: recebe dados brutos
- » Camadas Ocultas: processam e capturam complexidade dos dados
- » Camada de Saída: produz a decisão final ou inferência

Fluxo de Dados e Treinamento:

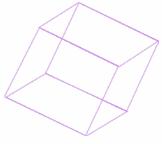
- » *Feedforward*: propagação direta dos dados da entrada para a saída
- » *Backpropagation*: ajuste de pesos baseado no erro entre saída desejada e produzida

Gradient Descent: algoritmo para minimizar a função de custo

Tipos de Redes Neurais:

- » *Perceptrons*: neurônio artificial simples
- » MLP: múltiplas camadas de *Perceptrons*
- » CNNs: para processamento de imagens
- » RNNs: para dados sequenciais (LSTM, GRU)
- » *Autoencoders*: para compressão e geração de dados
- » GANs: para geração de dados realistas

**Unidade XII
Encerramento**



Unidade XII - Encerramento



12.1 Considerações Finais

A jornada pelas técnicas de ML e redes neurais destaca a complexidade e a interconexão das diversas etapas envolvidas no desenvolvimento de modelos eficazes. Desde a definição clara e a aplicação prática dos conceitos até a implementação de algoritmos avançados, cada passo é crucial para alcançar resultados precisos e úteis. A importância de uma boa prática de pré-processamento, incluindo a extração, seleção e normalização de características, não pode ser subestimada, pois essas etapas formam a base sobre a qual modelos robustos são construídos. Além disso, a compreensão dos desafios como a maldição da dimensionalidade e o *overfitting* (sobreajuste) é vital para criar modelos que não apenas aprendem bem com os dados de treinamento, mas também se generalizem bem para novos dados.

Ao longo dessa unidade, discutimos uma série de tópicos cruciais no campo do ML, onde exploramos sua definição, suas aplicações, e as diferenças entre IA, ML e DL. Entendemos que ML é a prática de treinar algoritmos para aprender com dados e fazer previsões ou decisões com base nesses dados. Exploramos como diferentes tipos de ML, como aprendizado supervisionado e não supervisionado, são aplicados em áreas diversas, desde reconhecimento de imagem até previsão de mercado.

Voltamos nossa atenção para conceitos como detecção de padrões e anomalias, e as tarefas de detecção, reconhecimento e previsão. Falamos sobre como a detecção de padrões e anomalias pode ser usada para identificar comportamentos esperados e inesperados nos dados, enquanto a detecção, reconhecimento e previsão são fundamentais para a construção de sistemas inteligentes capazes de interpretar dados e antecipar eventos futuros.

Foram discutidos os diferentes tipos de aprendizagem: supervisionada, não supervisionada, semi-supervisionada e por reforço. Cada tipo foi explicado tecnicamente, complementado com metáforas para facilitar a compreensão e exemplos práticos para ilustrar sua aplicação. Discutimos a importância de entender os diferentes métodos de aprendizagem de máquina para escolher a abordagem mais adequada para cada situação.

Abordamos o processamento e transformação de características, discutindo a extração e seleção de características, a relação entre características e dimensionalidade, a maldição da dimensionalidade, a normalização e transformação do espaço de caracte-

rísticas, e a redução da dimensionalidade. Vimos como técnicas bem aplicadas podem ajudar a simplificar os dados e melhorar a eficiência computacional, e a importância de normalizar dados para equilibrar a contribuição de diferentes características.

Exploramos algumas tarefas clássicas como Regressão, Classificação e Agrupamento, passando por diferentes abordagens e modelos para cada problema. Por fim, estudamos as RNAs que tanto estão em destaque atualmente e o processo de surgimento do DL.

Este curso tem como objetivo apresentar os conceitos e ferramentas que fundamentam o ML. Ao passar por todos esses tópicos, esperamos que você tenha construído uma base sólida de conhecimento para navegar pelo universo dos problemas orientados a dados e criar novas soluções.

Referências

- AEBERHARD, Stefan; COOMANS, Danny; DE VEL, Olivier. Comparative analysis of statistical pattern recognition methods in high dimensional settings. **Pattern Recognition**, v. 27, n. 8, p. 1065-1077, 1994.
- BELLMAN R.E. Adaptive Control Processes. **Princeton: Princeton University Press**, 1961.
- BISHOP, C. M. Pattern Recognition and *Machine Learning*. New York: **Springer**, 2006.
- CHAPELLE, O.; SCHOLKOPF, B.; ZIEN, A. *Semi-Supervised Learning*. MIT Press, 2006.
- CHEN, Tianqi et al. Package 'xgboost'. **R version**, v. 90, n. 1-66, p. 40, 2019.
- ESTER, Martin et al. **A density-based algorithm for discovering clusters in large spatial databases with noise**. In: kdd. 1996. p. 226-231.
- FISHER, Ronald A. The use of multiple measurements in taxonomic problems. **Annals of eugenics**, v. 7, n. 2, p. 179-188, 1936.
- FIX, Evelyn. Discriminatory analysis: nonparametric discrimination, consistency properties. **USAF school of Aviation Medicine**, 1985.
- GÉRON, A. Hands-On *Machine Learning* with *Scikit-Learn*, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. **O'Reilly Media**, 2019.
- GUYON, I.; ELISSEFF, A. An Introduction to Variable and Feature Selection. **Journal of Machine Learning Research**, 2003.
- HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. **Springer Series in Statistics**, 2009.
- HYNDMAN, R.J.; ATHANASOPOULOS, G. Forecasting: principles and practice. Melbourne, Austrália: **O Texts**, 2021.
- KE, Guolin et al. Lightgbm: A highly efficient gradient boosting decision tree. **Advances in neural information processing systems**, v. 30, 2017.
- MCKINNEY, W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. **O'Reilly Media**, 2017.
- MITCHELL, T. M. *Machine Learning*. **McGraw-Hill**, 1997.
- NORVIG, Peter; RUSSELL, Stuart. Inteligência artificial. Rio de Janeiro: **Grupo GEN**, 2013.
- STARMER, J. The StatQuest Illustrated Guide to *Machine Learning!!! Packt Publishing*, 2022.
- THEODORIDIS, S.; KOUTROUMBAS, K. Nonlinear classifiers. **Academic Press**, p. 151–260, 2009.
- VANDERPLAS, J. **Python Data Science Handbook: Essential Tools for Working with Data**. O'Reilly Media, 2016.



AKCIT

CENTRO DE COMPETÊNCIA EMBRAPII
EM TECNOLOGIAS IMERSIVAS



CEIA
CENTRO DE EXCELENCIA EM
INTELIGENCIA ARTIFICIAL



GOIÁS
GOVERNO DO ESTADO DE GOIÁS
O ESTADO QUE DÁ CERTO



EMBRAPYI



INF
INSTITUTO DE
INFORMÁTICA



PRPI
PRÓ-REITORIA DE
PESQUISA E INovação



UFG
UNIVERSIDADE
FEDERAL DE GOIÁS

SOBRE O E-BOOK

Tipografia: Montserrat

Publicação: Cegraf UFG

Câmpus Samambaia, Goiânia -
Goiás. Brasil. CEP 74690-900

Fone: (62) 3521-1358

<https://cegraf.ufg.br>