IPOG

Machine Learning

EDITORA IPOG

Todos os direitos quanto ao conteúdo desse material didático são reservados ao(s) autor(es). A reprodução total ou parcial dessa publicação por quaisquer meios, seja eletrônico, mecânico, fotocópia, de gravação ou outros, somente será permitida com prévia autorização do IPOG.

IP5p Instituto de Pós-Graduação e Graduação – IPOG

Machine Learning. / Autor: Thiago Santana Lemes.

240f. :il.

ISBN:

1. Machine Learning 2. Ciência de Dados 3. Estatística 4. Matemática 5. Análise de Dados.

CDU: 005

Bibliotecário responsável: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

IPOG **Sede**

Instituto de Pós Graduação e Graduação

http://www.ipog.edu.br

Av. T-1 esquina com Av. T-55 N. 2.390 - Setor Bueno - Goiânia-GO. Telefone (0xx62) 3945-5050

SUMÁRIO

APRESENTAÇÃO

Seja bem-vindo ou bem-vinda à disciplina de Machine Learning.

Esse curso introdutório de Aprendizado de Máquina (*Machine Learning* ou ML) tem por objetivo apresentar ao aluno uma parte do maravilhoso mundo dos Dados. O desenvolvimento de todos os exemplos será feito em linguagem de programação Python. Diversos termos são citados a todo tempo e vários conceitos acabam por se sobrepor, por isso, antes de iniciar a jornada, vale a pena destacar alguns pontos a fim de proporcionar um melhor esclarecimento. A Inteligência Artificial (AI) se tornou o termo/palavra da moda atualmente, mas esse conceito é mais geral e engloba outros. De forma mais simplificada, pode-se entender a AI como um conjunto de técnicas (matemáticas, estatísticas e computacionais) inspiradas no comportamento humano e de animais que tem por objetivo resolver problemas. Uma visão bastante aceita na comunidade e facilmente vista na internet é a que trata a AI como uma grande área, dentro dela está o ML, dentro de ML se encontra o Aprendizado Profundo (*Deep Learning* ou DL) e finalmente, dentro de DL está a Inteligência Artificial Generativa (Generative AI). De posse dessas ideias iniciais vale ressaltar que modelos de ML no geral, fazem parte de um conceito mais amplo de IA discriminativa, ou seja, são treinados em conjuntos de dados rotulados para executar tarefas como predição e classificação. De forma mais simplificada, baseiam-se em dados existentes para tomar algum tipo de decisão. Por outro lado, Generative AI cria dados/informações, sendo, portanto, uma maneira distinta de pensar modelos de AI.

ML possui tantas aplicações que é impossível citar nessa breve introdução, porém vale destacar alguns casos. Para se ter uma ideia, diversas empresas em áreas diferentes como petróleo e gás, mineração, financeira, tecnologia, logística e saúde aplicam técnicas de ML em seus inúmeros processos. Seja para prever resultados, comportamentos, demandas, seja para classificar exames, imagens e dados em geral. Além disso, uma combinação muito poderosa é a de ML com otimização matemática que produz soluções

extremamente sofisticadas e inteligentes em ambientes industriais por exemplo, os quais são constituídos por diversos equipamentos e a redução de custo e aumento na eficiência é um fator crucial para o sucesso do negócio.

Dados vem sendo considerados como o novo petróleo a algum tempo e devido à essa importância é imprescindível ter conhecimento dessa área atualmente. Aprender ML, mesmo que não seja para se tornar um especialista, mas, ao menos entender do que se trata é um diferencial.

Boa leitura e estudos!

Prof. Dr. Thiago Santana Lemes

OBJETIVOS

OBJETIVO GERAL

Proporcionar uma compreensão dos princípios, algoritmos e aplicações do aprendizado de máquina, capacitando os estudantes a aplicar esses conhecimentos na solução de problemas reais, na análise de dados e no desenvolvimento de sistemas inteligentes.

OBJETIVOS ESPECÍFICOS

- Conhecer os conceitos fundamentais do aprendizado de máquina incluindo aprendizado supervisionado e não supervisionado;
- Desenvolver habilidades práticas em modelagem, seleção de algoritmos, treinamento de modelos e avaliação de desempenho;
- Entender os fundamentos da análise exploratória de dados;
- Implementar modelos de machine learning;
- Compreender questões éticas e o impacto social do aprendizado de máquina, incluindo viés algorítmico, privacidade de dados, e responsabilidade na tomada de decisões.

Conheça como esse conteúdo foi organizado!

Unidade 1: Conceitos iniciais de machine learning

Unidade 2: Pré-processamento de Dados

Unidade 3: Aprendizado supervisionado – Regressão

Unidade 4: Aprendizado supervisionado – Classificação parte 1

Unidade 5: Aprendizado supervisionado – Classificação parte 2

UNIDADE 1 CONCEITOS INICIAIS DE MACHINE LEARNING

É com grande prazer que damos início a essa jornada de aprendizado e descobertas. Nesta unidade, teremos a oportunidade de explorar os conceitos iniciais do fascinante universo de *machine learning*.

OBJETIVOS DA UNIDADE 1

Ao final dos estudos, você deverá ser capaz de:

- Definição de conceitos básicos da estatística.
- Configuração do ambiente.

1.1 AMBIENTE DE DESENVOLVIMENTO NO GOOGLE COLAB

O Google fornece uma plataforma de desenvolvimento para projetos de ciência de dados e inteligência artificial chamada Colab. Esse ambiente é gratuito para utilização, podendo fornecer recursos computacionais extras por meio de pagamento. A Figura 1.1 abaixo mostra a página inicial que pode ser acessada por meio do link https://colab.research.google.com.

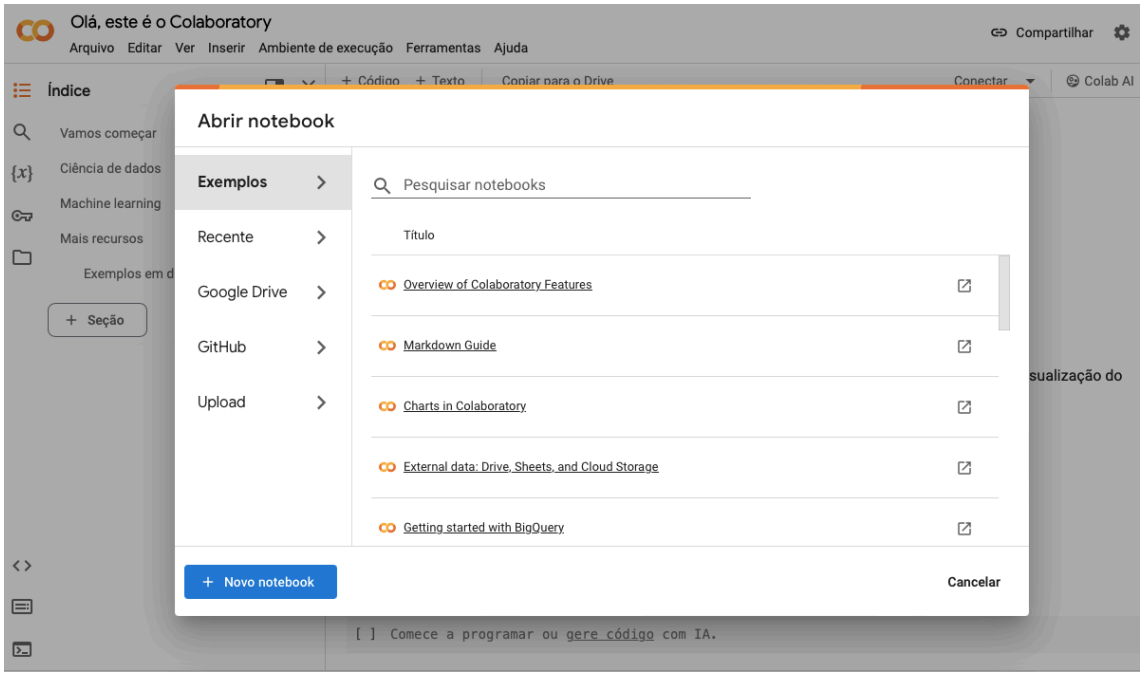


Figura 1.1: Página inicial do Google Colab

A plataforma possui diversos exemplos e tutoriais e todos estão acessíveis já na aba Exemplos como mostrado na Figura 1.1. Para iniciar um projeto basta clicar no botão azul Novo notebook. Todos os códigos que serão apresentados nesse texto podem ser desenvolvidos no ambiente do Google Colab. A vantagem é que a maioria das bibliotecas padrão no universo de *machine learning* já estão instaladas facilitando muito o início do desenvolvimento de um projeto. Obviamente é possível instalar bibliotecas caso elas não estejam disponíveis de início no ambiente.

1.2. ANÁLISE EXPLORATÓRIA DE DADOS

Sempre que se inicia um projeto de Aprendizado de Máquina (*Machine Learning* ou ML) uma análise exploratória de dados é necessária. As tomadas de decisão, escolha de algoritmos e escolha de apresentação de resultados passa pelo completo entendimento do conjunto de dados que está sendo utilizado.

Como primeiros passos, é preciso visualizar todas as variáveis disponíveis, entender os seus tipos (*float, int, str*), no caso dos dados numéricos, como estão distribuídos. Variáveis numéricas podem ser provenientes de medições ou registros como altura e peso de indivíduos ou mesmo indicações de que algo estava ligado ou desligado (funcionando ou não funcionando, presente ou ausente, ...) ou se determinada ação foi tomada ou não. Variáveis que indicam dois estados são conhecidas como binárias e podem ser representadas por palavras (masculino/feminino - *string*) ou por booleanos (Verdadeiro/Falso – *True/False*). Além disso, essa lógica de representar dois estados pode ser estendida para diversos outros. A Figura 1.2 a seguir ilustra alguns tipos de variáveis.

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Superior	Sim	1	1.67	78	True
1	F	Médio	Sim	1	1.80	89	False
2	F	Médio	Não	1	1.88	101	False
3	M	Superior	Sim	1	1.90	77	False
4	M	Médio	Não	1	1.73	65	True
5	M	Pós-graduação	Não	0	1.72	90	False
6	F	Médio	Não	1	1.80	97	False
7	F	Médio	Sim	0	1.77	98	True

Figura 1.2: Representação de um *dataframe* contendo alguns tipos diferentes de variáveis

Na Figura 1.2, as variáveis “Sexo”, “Escolaridade”, “Possui casa própria”, “Possui trabalho CLT” e “Já realizou algum procedimento cirúrgico” são categóricas pois possuem um número finito de categorias (opções) representadas. Altura é uma variável de ponto flutuante (*float*) uma vez que representa uma informação com maior precisão. Peso é uma variável do tipo inteiro (*int*). A Figura 1.2 foi gerada utilizando a biblioteca Pandas (The pandas development team, 2024) e o trecho de código abaixo (Figura 1.3) ilustra isso, sendo a primeira linha utilizada para importar a biblioteca.

```
import pandas as pd
df = pd.DataFrame({'Sexo':['M', 'F', 'F', 'M', 'M', 'M', 'F', 'F'],
                   'Escolaridade':['Superior', 'Médio', 'Médio', 'Superior', 'Médio', 'Pós-graduação', 'Médio', 'Médio'],
                   'Possui casa própria':['Sim', 'Sim', 'Não', 'Sim', 'Não', 'Não', 'Não', 'Sim'],
                   'Possui trabalho CLT':[1, 1, 1, 1, 1, 0, 1, 0],
                   'Altura':[1.67, 1.80, 1.88, 1.90, 1.73, 1.72, 1.80, 1.77],
                   'Peso':[78, 89, 101, 77, 65, 90, 97, 98],
                   'Já realizou algum procedimento cirúrgico':[True, False, False, False, True, False, False, True]})
```

Figura 1.3: Geração de dados fictícios utilizando a biblioteca Pandas

A biblioteca Pandas fornece uma maneira simples de obter os tipos de todas as variáveis contidas no *dataframe*, bastando utilizar o comando .info() (Figura 1.4). As variáveis do tipo inteiro aparecem na coluna *Dtype* como *int64*, variáveis do tipo *float* aparecem como *float64*, variáveis do tipo *string* aparecem como *object* e variáveis do tipo booleano aparecem como *bool*.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 7 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Sexo                                  8 non-null      object
 1   Escolaridade                          8 non-null      object
 2   Possui casa própria                   8 non-null      object
 3   Possui trabalho CLT                   8 non-null      int64
 4   Altura                                8 non-null      float64
 5   Peso                                  8 non-null      int64
 6   Já realizou algum procedimento cirúrgico  8 non-null  bool
dtypes: bool(1), float64(1), int64(2), object(3)
memory usage: 520.0+ bytes
```

Figura 1.4: Inspeção do tipo de dados utilizando o Pandas

Após essa inspeção inicial nos dados é de muita importância verificar as medidas de tendência central, dispersão dos dados, correlações entre variáveis e finalmente uma representação visual. A Figura 1.5 abaixo mostra a utilização do comando *describe()* do Pandas que retorna de uma só vez a média, o desvio padrão, valores mínimos e máximos e os percentis dos dados, sendo um comando extremamente útil.

```
df[['Altura', 'Peso']].describe()
```

	Altura	Peso
count	8.000000	8.000000
mean	1.783750	86.875000
std	0.078729	12.506427
min	1.670000	65.000000
25%	1.727500	77.750000
50%	1.785000	89.500000
75%	1.820000	97.250000
max	1.900000	101.000000

Figura 1.5: Descrição dos dados utilizando Pandas e *describe*

Para verificar a correlação entre as variáveis pode-se utilizar o comando *corr()* do Pandas como mostrado na Figura 1.6 abaixo.

```
df[['Altura', 'Peso']].corr()
```

	Altura	Peso
Altura	1.000000	0.325544
Peso	0.325544	1.000000

Figura 1.6: Correlação entre as variáveis

A Figura 1.7 ilustra uma análise em relação à Escolaridade. É fácil verificar que pessoas com Pós-graduação são a minoria e que a maioria das pessoas possuem ensino Médio.

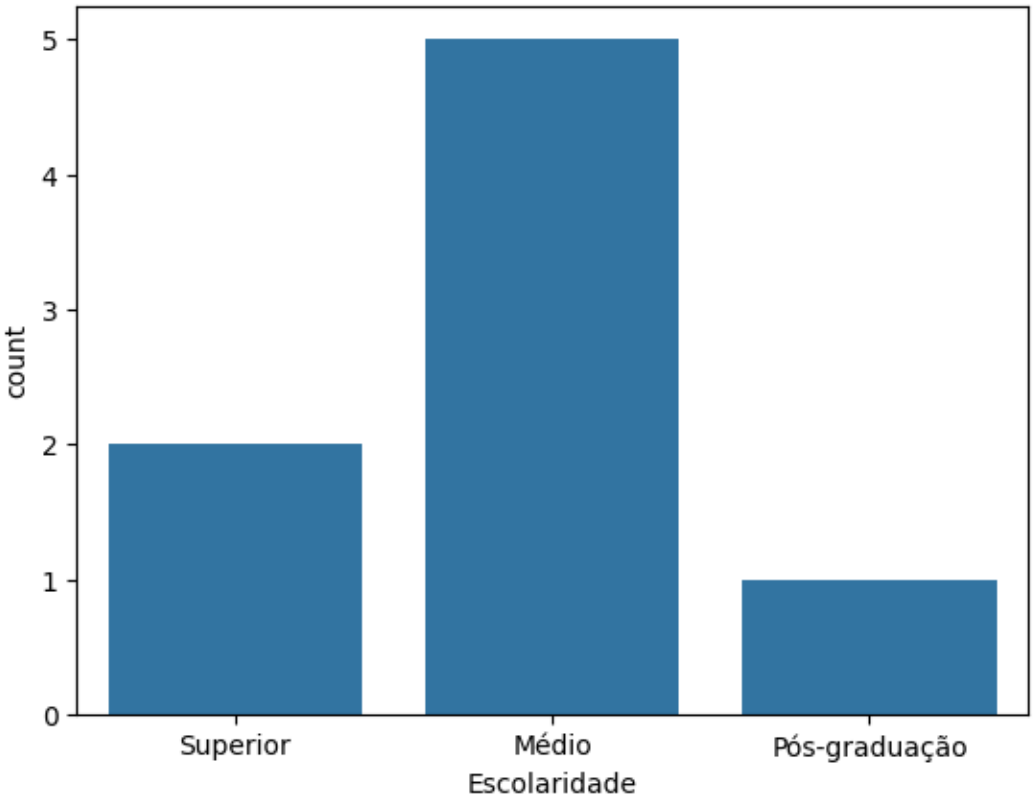


Figura 1.7: Contagem do nível de escolaridade dos indivíduos

Vale ressaltar que para as medidas de tendência central e a correlação somente as variáveis numéricas foram utilizadas. Variáveis categóricas são melhor entendidas por meio de contagens e distribuição como na Figura 1.7 acima. Para entender um pouco melhor os dados é importante gerar visualizações. Aqui serão apresentados alguns histogramas (Figura 1.8) que dão início ao processo de entendimento dos dados, mas ao longo das próximas unidades, diversos outros gráficos serão apresentados. O comando *.hist()* do Pandas facilita a construção desses gráficos.

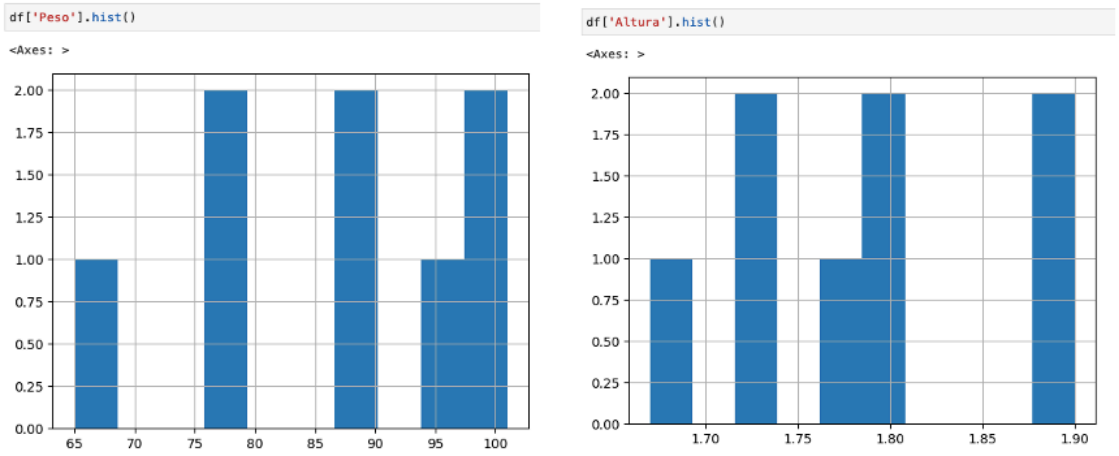


Figura 1.8: Histogramas para as variáveis Peso e Altura

UNIDADE 2 PRÉ-PROCESSAMENTO DE DADOS

Tivemos uma ideia básica a respeito de estatística além de algumas ideias a respeito de visualização de dados. Foi possível perceber que existem diversos conceitos envolvidos e que eles são essenciais para o bom andamento de um projeto que tem por base os dados.

Nessa nova unidade iremos caminhar no sentido de realizar manipulações nos conjuntos de dados e deixar eles mais limpos/validados para serem utilizados nos modelos de *machine learning*.

OBJETIVOS DA UNIDADE 2

Ao final dos estudos, você deverá ser capaz de:

- Realizar o pré-processamento dos dados
- Validar um conjunto de dados

2.1. PRÉ-PROCESSAMENTO DE DADOS

Na maioria dos projetos reais que envolvem análise de dados existem problemas relacionados à qualidade dos dados. Esses problemas podem surgir por diversos motivos, como por exemplo um conjunto de dados que é formado pela aquisição de informações de sensores e que em alguns momentos falham, deixando registros em branco, ou mesmo dados provenientes de pesquisas, os quais podem ter registros faltantes. Para lidar com esses dados incompletos existem diversas abordagens, entre elas, preencher os elementos faltantes com a média ou a mediana dos registros ou então remover os registros da análise.

A remoção tem a desvantagem da perda de informações, mas em alguns casos não há solução, por outro lado, o preenchimento com a média insere valores que não são exatos e para cada projeto é necessário validar qual a melhor estratégia. A Figura 1.4 abaixo ilustra um *dataset* com alguns registros faltantes (NaN – *Not a Number*, -)

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	M	Superior	NaN	1	1.67	78.0	True
1	NaN	Médio	Sim	1	1.8	NaN	False
2	F	Médio	Não	1	1.88	101.0	NaN
3	M	-	Sim	1	-	77.0	False
4	M	Médio	Não	1	1.73	65.0	True
5	M	Pós-graduação	Não	0	1.72	90.0	False
6	F	Médio	Não	1	1.8	97.0	False
7	F	Médio	Sim	0	1.77	NaN	NaN

Figura 1.4: *Dataset* com valores faltantes

Outra questão importante em projetos de *Machine Learning* é a conversão de dados simbólicos em numéricos. Isso se deve ao fato de que a maioria dos modelos só conseguem processar dados numéricos. Para clarificar esse conceito vamos tomar como exemplo o *dataset* da Figura 1 no qual temos a variável (coluna) Sexo com valores “M” ou “F”. Nesse caso, “M” pode ser representado por 1 e “F” por 0. Veja a Figura 1.5 abaixo.

	Sexo	Escolaridade	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	1	Superior	Sim	1	1.67	78	True
1	0	Médio	Sim	1	1.80	89	False
2	0	Médio	Não	1	1.88	101	False
3	1	Superior	Sim	1	1.90	77	False
4	1	Médio	Não	1	1.73	65	True
5	1	Pós-graduação	Não	0	1.72	90	False
6	0	Médio	Não	1	1.80	97	False
7	0	Médio	Sim	0	1.77	98	True

Figura 1.5: *Dataset* com variável Sexo convertida

A variável Escolaridade apresenta uma situação interessante pois possui três valores distintos (Superior, Médio e Pós-graduação). A mesma lógica empregada na variável Sexo pode ser aplicada aqui, mas pela quantidade maior de opções irá gerar mais colunas.

	Médio	Pós-graduação	Superior	Sexo	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	0	0	1	M	Sim	1	1.67	78	True
1	1	0	0	F	Sim	1	1.80	89	False
2	1	0	0	F	Não	1	1.88	101	False
3	0	0	1	M	Sim	1	1.90	77	False
4	1	0	0	M	Não	1	1.73	65	True
5	0	1	0	M	Não	0	1.72	90	False
6	1	0	0	F	Não	1	1.80	97	False
7	1	0	0	F	Sim	0	1.77	98	True

Figura 1.6: *Dataset* com variáveis categóricas convertidas

Esse tipo de abordagem na qual variáveis categóricas são convertidas em numéricas é também conhecido como *One-Hot-Encoding*. As variáveis binárias criadas para representar as variáveis categóricas são conhecidas como *dummy*.

Outro ponto crucial na etapa de pré-processamento dos dados é o de padronização, o qual leva todos os valores registrados para um mesmo patamar de comparação. O *dataset* da Figura 1.2 nos ajuda a entender essa situação ao tomar como exemplo as variáveis “Peso” e “Altura”. De forma geral, a altura de uma pessoa é um número maior do que zero e menor do que 2,5, considerando a unidade de medida metros (m). O peso de uma pessoa também é um número maior do que zero e normalmente abaixo de 150, considerando a unidade de medida quilogramas (kg). Vale ressaltar que obviamente podem existir pessoas

com medidas superiores às comentadas, mas aqui nesse texto, iremos nos ater a poucos valores para fins pedagógicos. Considerando essa explicação, vemos que os valores de peso e altura estão em patamares bem diferentes, e isso pode levar os modelos de *Machine Learning* a não compreenderem bem a relação entre os dados, ou mesmo atribuir maior importância a um conjunto por possuir valores muito maiores do que o outro conjunto. Para mitigar esse problema, aplica-se um procedimento de transformação aos dados que padroniza todos baseado em uma lógica. A Figura 1.7 mostra o mesmo *dataset* da Figura 1.2, mas agora com as variáveis “Peso” e “Altura” transformadas.

	Médio	Pós-graduação	Superior	Sexo	Possui casa própria	Possui trabalho CLT	Altura	Peso	Já realizou algum procedimento cirúrgico
0	0	0	1	M	Sim	1	0.000000	0.361111	True
1	1	0	0	F	Sim	1	0.565217	0.666667	False
2	1	0	0	F	Não	1	0.913043	1.000000	False
3	0	0	1	M	Sim	1	1.000000	0.333333	False
4	1	0	0	M	Não	1	0.260870	0.000000	True
5	0	1	0	M	Não	0	0.217391	0.694444	False
6	1	0	0	F	Não	1	0.565217	0.888889	False
7	1	0	0	F	Sim	0	0.434783	0.916667	True

Figura 1.7: *Dataset* com variáveis “Altura” e “Peso” transformadas

Vale destacar que existem vários tipos de métodos para padronizar/transformar os dados, dentre eles se destacam o mínimo e máximo (*MinMax*) e a estandardização (*Standardization*). Em muitos casos, uma certa transformação pode ajudar os algoritmos a performarem melhor e por isso, cabe ao profissional de Ciência de Dados juntamente com a equipe do projeto definir qual abordagem é a mais adequada. A Figura 6 mostra o resultado da padronização após a utilização do método *MinMax* que tem por base pegar um valor mínimo e um valor máximo no conjunto observado. A coluna “Peso” possuía o valor máximo de 101 na terceira linha e após o *MinMax* se transformou em 1, enquanto o valor mínimo era de 77 e agora é 0. Dessa forma, a lógica aqui é transformar todos os valores que estão entre 77 e 101 para uma nova faixa de valores correspondentes entre 0 e 1.

Um fator muito impactante em projetos que utilizam ML é o uso de dados não balanceados, ou seja, dados que possuem determinada característica ou

são provenientes de determinado grupo e aparecem mais do que outros. A Figura 6 possui uma coluna com a escolaridade de um grupo de pessoas e é fácil perceber que a maioria possui ensino médio, enquanto apenas uma possui pós-graduação. Se um algoritmo de ML for treinado nessa base de dados para atingir qualquer objetivo que seja, provavelmente, a falta de exemplos de pessoas com pós-graduação fará falta para que o modelo possa compreender os padrões inerentes dessa categoria. São inúmeras as técnicas para atacar o problema de balanceamento dos dados, mas só para citar algumas, é possível aplicar estratégias de reamostragem nos dados com o intuito de apresentar de forma justa a mesma quantidade de exemplos de diferentes classes/grupos. Pode-se simplesmente descartar uma parte dos dados no conjunto com maior número de exemplos a fim de parear as quantidades de amostras. Existem também bibliotecas especialistas na criação de dados sintéticos que podem ser utilizadas para completar os dados com menor representatividade a fim de tornar o treinamento dos algoritmos de ML mais balanceados. O fato é que não existe técnica perfeita e um entendimento claro do problema se faz necessário antes da tomada de decisão.

UNIDADE 3 APRENDIZADO SUPERVISIONADO - REGRESSÃO

Nesse momento já somos capazes de aplicar operações e transformações aos dados. Toda a parte de pré-processamento torna os dados disponíveis para que os modelos de *machine learning* possam tirar o máximo de proveito.

OBJETIVOS DA UNIDADE 3

Ao final dos estudos, você deverá ser capaz de:

- Implementar um modelo de regressão
- Analisar as relações existentes nos dados

3.1. REGRESSÃO LINEAR SIMPLES

A regressão linear é uma técnica estatística que tem por objetivo ajustar uma reta a um conjunto de dados. Para isso, parte-se do pressuposto que temos uma variável dependente que explica uma variável resposta. Nesse caso, tem-se uma equação da forma

$$y = ax + b \qquad \text{Equação 3.1}$$

na qual *a* representa o coeficiente angular, *b* o coeficiente linear, *x* é a variável independente e *y* é a resposta esperada. O intuito é estimar os parâmetros *a* e *b* da equação de maneira que o resultado seja o mais próximo do real. Para ilustrar melhor a situação, imagine um exemplo em que a demanda de passagens aéreas depende do preço dessas passagens. Com certeza existem diversos fatores que influenciam nessa demanda, mas para fins didáticos nesse texto, consideraremos apenas a variável preço. A Tabela 3.1 apresenta valores das variáveis para esse problema.

Demanda	**Preço da passagem**
121	110
108	112
97	130
90	125
83	170

66	167
60	200
55	240
42	231
33	239
32	220
34	236
28	230
27	255
20	268

Tabela 3.1: Valores das variáveis para o problema de demanda por passagens aéreas

Olhando a Tabela 3.1 percebe-se uma tendência de diminuição na demanda por passagens na medida em que o preço das passagens aumenta. Estamos interessados em descobrir qual é a equação da reta que se ajusta da melhor maneira a esse conjunto de dados e descreve essa relação.

Os trechos de código abaixo (Figura 3.1 e Figura 3.2), escritos em linguagem Python, utilizando as bibliotecas *Pandas*, *Matplotlib* e *Seaborn*

(Hunter, 2007; The pandas development team, 2024; Waskom, 2021), exibem a implementação do problema, primeiro definindo o conjunto de dados e em seguida montando o gráfico. Vale ressaltar que são implementações simples com objetivo pedagógico e que pode haver diversos níveis de personalização e aprofundamento nas análises. A Figura 3.3 exibe uma perspectiva gráfica desses dados exibindo a dispersão dos pontos e juntamente uma reta que possui o melhor ajuste aos dados.

```
import pandas as pd
dados = pd.DataFrame({'Demanda':[121, 108, 97, 90, 83, 66, 60, 55, 42, 33, 32, 34, 28, 27, 20],
                      'Preço da passagem':[110, 112, 130, 125, 170, 167, 200, 240, 231, 239, 220, 236, 230, 255, 268]})
dados
```

	Demanda	Preço da passagem
0	121	110
1	108	112
2	97	130
3	90	125
4	83	170
5	66	167
6	60	200
7	55	240
8	42	231
9	33	239
10	32	220
11	34	236
12	28	230
13	27	255
14	20	268

Figura 3.1: Definição dos dados para o problema de demanda por passagens aéreas

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,6))
sns.regplot(data=dados, x='Preço da passagem', y='Demanda')
plt.xlabel('Preço da passagem', fontsize=14, fontweight='bold')
plt.ylabel('Demanda', fontsize=14, fontweight='bold')
plt.title('Demanda de passagens aéreas em função do preço', fontsize=14, fontweight='bold')
```

Figura 3.2: Código que constrói o gráfico dos dados para o problema de demanda por passagens aéreas

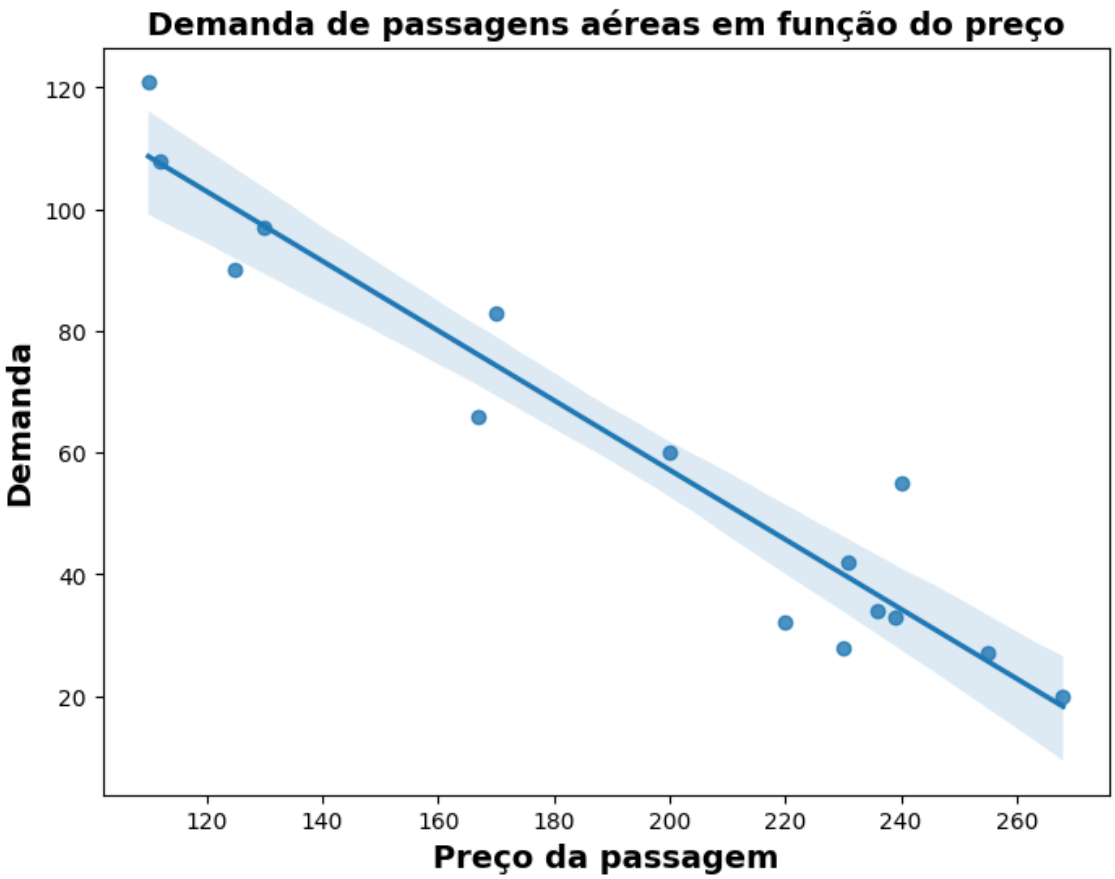


Figura 3.3: Relação entre o preço da passagem aérea e a demanda por passagens

Para ajustar uma regressão linear será utilizada a biblioteca *scikit-learn* (Pedregosa et al., 2011) que já possui a implementação matemática e disponibiliza de forma facilitada métodos para avaliar a qualidade dos resultados. Os trechos de código abaixo ilustram a solução.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data_transformed = scaler.fit_transform(dados[['Preço da passagem']])
data_transformed = pd.DataFrame(data_transformed, columns=['Preço da passagem'])
data_transformed
```

	Preço da passagem
0	0.000000
1	0.012658
2	0.126582
3	0.094937
4	0.379747
5	0.360759
6	0.569620
7	0.822785
8	0.765823
9	0.816456
10	0.696203
11	0.797468
12	0.759494
13	0.917722
14	1.000000

Figura 3.4: Transformação dos dados utilizando *MinMax*

A Figura 3.4 mostra os dados já padronizados pelo método *MinMax*. Esse processo é realizado antes de aplicar os dados ao modelo de regressão constituindo um *scaler* ajustado que pode ser utilizado posteriormente para transformar novos dados. Como dito anteriormente, a padronização dos dados antes de aplicá-los a um modelo estatístico ou de ML é benéfica uma vez que remove as diferenças naturais de grandezas distintas que foram mensuradas. A primeira linha dessa célula de código importa o método que irá padronizar os dados, a segunda linha instancia o objeto (esse texto não irá tratar do paradigma de programação orientada a objetos) *MinMaxScaler*, na terceira linha o ajuste necessário para a transformação é feito, na quarta linha é criada uma estrutura

de *dataframe* proveniente da biblioteca Pandas e finalmente os novos dados são exibidos.

Figura 3.5: Implementação de um modelo de regressão linear simples

A Figura 3.5 mostra a implementação de uma regressão linear simples utilizando os métodos da biblioteca *Scikit-learn*. A primeira célula faz a importação do método que ajusta o modelo aos dados, instancia o objeto de regressão linear e, na sequência, ajusta aos dados. As duas células logo abaixo da definição do modelo mostram o coeficiente angular e o coeficiente linear da reta de regressão. Dessa forma, a equação ajustada para o problema de demanda por passagens aéreas é (apresentando apenas quatro casas decimais):

$$y = -90.4402x + 108.6931 \qquad \text{Equação 3.2}$$

A partir de agora tem-se um modelo de regressão linear simples ajustado, que na prática é a Equação 3.2. De posse disso, podemos fazer previsões com

novos dados, que não foram utilizados no ajuste do modelo, para prever os valores de demanda por passagens aéreas. Imagine a situação em que uma companhia aérea tenha ajustado esse modelo para que sua gestão possa ter maior previsibilidade da demanda com base nos preços de passagens adotados. De agora em diante eles podem analisar a demanda esperada para diferentes preços. Veja abaixo um exemplo no qual estamos interessados em saber a previsão de demanda por passagens aéreas para um preço de passagem igual a R$ 227,00.

```
import numpy as np
novo_preco_transformado = scaler.transform(np.array([[227]]))
novo_preco_transformado
```

```
array([[0.74050633]])
```

```
novo_valor_predito = lr.predict(novo_preco_transformado)
```

```
print(np.round(novo_valor_predito[0],2))
```

```
41.72
```

Figura 3.6: Previsão de demanda por passagens aéreas

No trecho de código acima, na primeira linha da primeira célula de código a biblioteca *Numpy* (Harris et al., 2020) é importada para realizar alguns ajustes nos dados e em seguida utiliza-se o *scaler MinMax* previamente ajustado para transformar o preço de 227 para um novo valor entre 0 e 1. Esse novo preço transformado é então aplicado ao modelo de regressão linear já ajustado e finalmente obtém-se a resposta 41.72 na terceira célula. De forma simples, esse bloco acima está mostrando que se o preço de uma passagem aérea for R$ 227,00 espera-se que a demanda por passagens seja de aproximadamente 41. Vale reforçar que esse resultado é uma previsão e não necessariamente irá representar a realidade, mas um valor próximo com base nas informações disponíveis. Além disso, esse modelo pode prever diversos valores ao mesmo

tempo e não apenas um como no exemplo, basta passar um vetor com um conjunto de valores. Na prática, ao se ajustar um modelo de regressão, inserimos variáveis que possivelmente explicam o problema e obtemos uma previsão dos valores. Isso é utilizado pela gestão para tomar uma decisão.

Como observação final para esse tópico, destaca-se que nesse texto não aprofundaremos na metodologia para ajuste dos parâmetros *a* e *b* da regressão linear que podem ser obtidos pelo método dos mínimos quadrados por exemplo.

3.2. REGRESSÃO LINEAR MÚLTIPLA

A regressão linear múltipla segue exatamente o mesmo racional apresentado pela regressão linear simples, porém nesse caso, existe mais do que uma variável preditora, ou seja, a ideia aqui é de que existem pelo menos duas variáveis que explicam uma determinada variável resposta. Veja o exemplo a seguir extraído do livro (Morettin & Singer, 2023):

Os dados da Tabela 2 foram extraídos de um estudo em que um dos objetivos era avaliar o efeito do índice de massa corpórea (IMC) e da carga aplicada numa esteira ergométrica no consumo de oxigênio (VO2) numa determinada fase do exercício.

índice	V02	IMC	carga	índice	V02	IMC	carga
1	14,10	24,32	71,00	15	22,00	22,45	142,00
2	16,30	27,68	91,00	16	13,20	30,86	62,00
3	9,90	23,93	37,00	17	16,20	25,79	86,00
4	9,50	17,50	32,00	18	13,40	33,56	86,00
5	16,80	24,46	95,00	19	11,30	22,79	40,00
6	20,40	26,41	115,00	20	18,70	25,65	105,00
7	11,80	24,04	56,00	21	20,10	24,24	105,00

8	29,00	20,95	104,00	22	24,60	21,36	123,00
9	20,30	19,03	115,00	23	20,50	24,48	136,00
10	14,30	27,12	110,00	24	29,40	23,67	189,00
11	18,00	22,71	105,00	25	22,90	21,60	135,00
12	18,70	20,33	113,00	26	26,30	25,80	189,00
13	9,50	25,34	69,00	27	20,30	23,92	95,00
14	17.5	29,93	145,00	28	31,00	24,24	151,00

Tabela 3.2: Dados do experimento de consumo de oxigênio

A partir desse exemplo serão incluídos alguns conceitos a mais em relação à análise de resultados. Primeiramente, é necessário avaliar se o ajuste do modelo de regressão é bom o suficiente para ser utilizado em previsões futuras, e por isso vamos separar uma parte dos dados da tabela acima para testar o modelo. A ideia é escolher um percentual dos dados para ajustar/treinar e a outra parte para avaliar o resultado. Dessa forma, o modelo não tem conhecimento a respeito dos dados separados para o teste. Em especial, para modelos de regressão, são necessárias métricas que indiquem o quão perto a resposta está do resultado real. Métricas como o coeficiente de determinação R^2 (R2) e o erro absoluto percentual médio (MAPE) conseguem estimar essa aproximação. Para o R2, quanto mais próximo de 1 (1 corresponde à 100%) melhor e para o MAPE quanto mais próximo de zero melhor. O R2 mede a porcentagem da variação total dos valores da variável resposta (y) em relação à sua média explicada pelo modelo de regressão (Morettin & Singer, 2023)

enquanto o MAPE mede o erro percentual absoluto entre esses valores. Vamos apresentar agora alguns trechos de código para a solução desse problema.

```
plt.figure(figsize=(14,6))

plt.suptitle('Dispersão das variáveis IMC e carga em comparação ao consumo de oxigênio (V02)', fontsize=14, fontweight='bold')

plt.subplot(1, 2, 1)
sns.regplot(data=df, x='IMC', y='V02')
plt.xlabel('IMC', fontsize=14, fontweight='bold')
plt.ylabel('V02', fontsize=14, fontweight='bold')

plt.subplot(1, 2, 2)
sns.regplot(data=df, x='carga', y='V02')
plt.xlabel('carga', fontsize=14, fontweight='bold')
plt.ylabel('V02', fontsize=14, fontweight='bold')
```

Figura 3.7: Trecho de código para fazer um gráfico ilustrando a dispersão das variáveis IMC, carga em relação ao consumo de oxigênio (VO2)

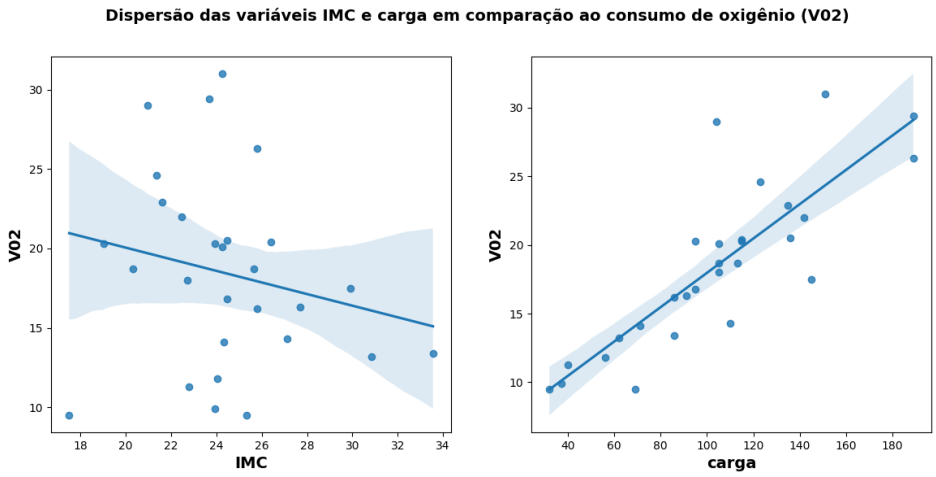


Figura 3.8: Gráfico ilustrando a dispersão das variáveis

O primeiro quadro da Figura 3.8 mostra a relação entre IMC e VO2 e aparentemente existe uma relação inversamente proporcional, ou seja, quanto maior o IMC menor o VO2, mas não é uma relação tão forte. No segundo quadro entretanto, parece haver uma relação clara de aumento das duas variáveis, quanto maior a carga, maior o VO2.

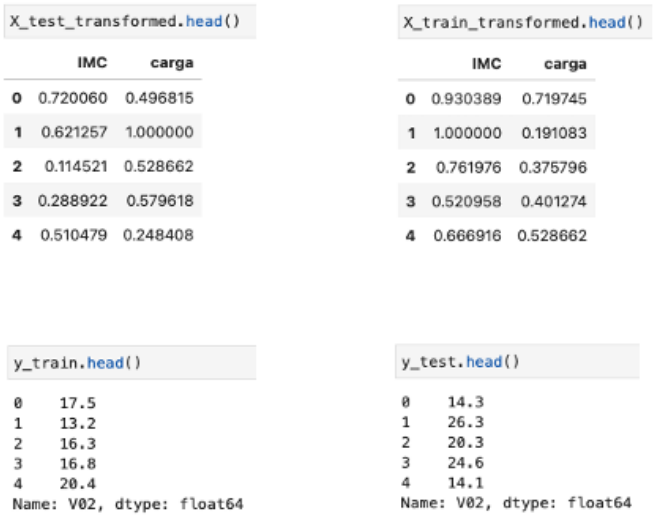
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop('V02', axis=1), df['V02'], test_size=0.3, random_state=42)
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

Figura 3.9: Separação dos dados em treino e teste

Na Figura 3.9, utilizando o método *train_test_split* da biblioteca *scikit-learn* dividimos os dados disponíveis na Tabela 3.2 em treino e teste, sendo que o parâmetro *test_size* indica que 30% dos dados serão utilizados no teste do modelo de regressão, o que implica diretamente que 70% dos dados serão utilizados no ajuste/treino do modelo. Essa proporção 70/30 é uma prática comum entre os cientistas de dados, mas não é obrigatória, podendo ser 80/20, 90/10 ou qualquer outro valor que faça sentido no projeto.

Os dados são então padronizados como já feito anteriormente (*MinMax*). Aqui vale uma observação conceitual: O *scaler* utilizado para padronizar os dados é inicialmente ajustado somente no conjunto de treinamento (70%) e então aplicado aos dados restantes de teste (30%) para fazer a transformação. Na prática, 70% dos dados entram no *scaler* que faz o cálculo para transformar tudo e definir novos valores entre 0 e 1, e a partir daí, com o cálculo já feito, os outros 30% dos dados são transformados na sequência. Isso se deve ao fato de que quando padronizamos os dados, o *scaler* leva em consideração o conjunto completo aplicado a ele, ou seja, ele “tem conhecimento” de 100% dos dados na hora de padronizar/transformar e, portanto, os dados de teste carregam consigo informações a respeito dos dados de treinamento. Esse problema é conhecido como vazamento de dados e para evitá-lo é necessário padronizar os dados somente após a divisão do conjunto em treino e teste. A padronização dos dados após a separação do conjunto garante o isolamento das amostras, de forma que o ajuste será feito de maneira realmente justa, sem que o modelo tenha qualquer informação a respeito dos dados de teste. Isso garantirá que o desempenho medido no conjunto de testes é realmente robusto e não foi afetado por alguma “dica” prévia.

Toda essa discussão relativa à avaliação da performance do modelo, bem como a separação dos dados de treino e teste se aplica a qualquer modelo supervisionado de ML e por isso, esses procedimentos serão repetidos em exemplos à frente.



```
X_test_transformed.head()
```

	IMC	carga
0	0.720060	0.496815
1	0.621257	1.000000
2	0.114521	0.528662
3	0.288922	0.579618
4	0.510479	0.248408

```
X_train_transformed.head()
```

	IMC	carga
0	0.930389	0.719745
1	1.000000	0.191083
2	0.761976	0.375796
3	0.520958	0.401274
4	0.666916	0.528662

```
y_train.head()
```

```
0    17.5
1    13.2
2    16.3
3    16.8
4    20.4
Name: V02, dtype: float64
```

```
y_test.head()
```

```
0    14.3
1    26.3
2    20.3
3    24.6
4    14.1
Name: V02, dtype: float64
```

Figura 3.10: Primeiros cinco registros dos dados de treinamento e teste

A Figura 3.10 apresenta os dados de treinamento e teste que foram utilizados no modelo de regressão linear múltipla. Somente os cinco primeiros registros são exibidos. Na parte superior da figura estão os dados de treino e teste já padronizados (*MinMax*), lembrando que o modelo possui duas variáveis preditoras: IMC e carga. Na parte inferior está a variável resposta (VO2) também dividida em treino e teste. Não há necessidade de padronizar a variável resposta uma vez que o modelo é capaz de ajustar seus parâmetros para qualquer faixa de valores.

Finalmente na Figura 3.11 vemos o trecho de código correspondente ao ajuste do modelo de regressão utilizando os dados de treinamento e na

sequência os coeficientes para cada variável (IMC e carga) e o coeficiente linear (constante).

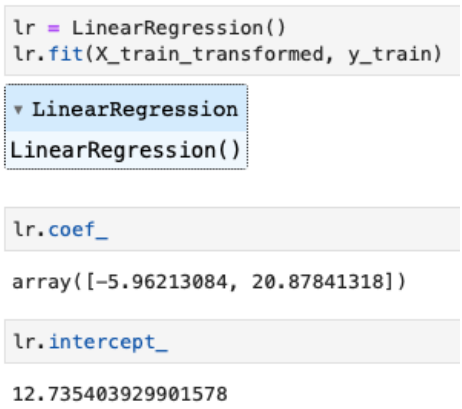


Figura 3.11: Trecho de código com ajuste do modelo de regressão linear múltipla

Podemos agora exibir a equação do modelo ajustado:

$$y = -5.9621x_1 + 20.8784x_2 + 12.7354 \qquad \text{Equação 3.3}$$

Nessa equação x_1 representa a variável IMC e x_2, variável carga e y é o consumo de oxigênio (VO2).

Para finalizar o exemplo é necessário avaliar a qualidade do modelo utilizando as métricas (R2 e MAPE) comentadas anteriormente. Com o modelo ajustado utiliza-se o método *predict* para obter previsões do modelo em cima dos dados de teste (não vistos pelo modelo anteriormente). Veja a Figura 3.12 abaixo:

```
y_pred = lr.predict(X_test_transformed)
np.round(y_pred, 2)
```

```
array([18.82, 29.91, 23.09, 23.11, 14.88, 14.16, 12.75, 23.45, 22.24])
```

```
np.array(y_test)
```

```
array([14.3, 26.3, 20.3, 24.6, 14.1,  9.5, 13.4, 20.5, 18.7])
```

Figura 3.12: Comparação entre os valores preditos pelo modelo de regressão linear múltipla e valores reais

Na célula superior da Figura 3.12 é possível ver os valores preditos pelo modelo, ou seja, no caso em que aplicamos informações de IMC e carga à regressão, essa é a sua resposta para cada par de dados. Na célula inferior da Figura 3.12 estão os dados reais, deixados à parte (não vistos pelo modelo) para testar a performance do modelo. É possível perceber que ao comparar os dois conjuntos de dados tem-se respostas de certa forma próximas, o que é bom e indica que a regressão linear está com um ajuste pelo menos razoável. Finalmente a Figura 3.13 ilustra os resultados de R2 e MAPE que são produzidos comparando as respostas da regressão linear múltipla e os dados reais de teste por meio dos métodos *r2_score* e *mean_absolute_percentage_error* da biblioteca *Scikit-learn*. Como explicado anteriormente, o R2 mais próximo de 1 é melhor e nesse caso tem-se 0,6459 (64,59%) que é bom, mas não excepcional. Além disso, o MAPE é 0,1754 (17,54%) que é um valor aceitável, mas não excelente. A avaliação de modelos estatísticos e de ML deve ser conduzida sempre com cautela e atenção visto que para cada projeto existem um nível de erro tolerável e que faz sentido para o negócio. Dessa forma, recomenda-se atenção redobrada e discussão com a equipe.

```
from sklearn.metrics import r2_score, mean_absolute_percentage_error
r2_test = np.round(r2_score(y_test, y_pred),4)
mape_test = np.round(mean_absolute_percentage_error(y_test, y_pred),4)
```

```
print(f'O valor de R2 é: {r2_test}')
```

```
O valor de R2 é: 0.6459
```

```
print(f'O valor de R2 é: {mape_test}')
```

```
O valor de R2 é: 0.1754
```

Figura 3.13: Resultados de R2 e MAPE para a regressão linear múltipla

Uma análise bastante relevante ao se utilizar modelos de regressão linear é a importância das variáveis que estão sendo utilizadas no modelo e como essas variáveis afetam o resultado. No problema do consumo de oxigênio estudado, temos as variáveis carga e IMC. Olhando a Equação 3.3 percebe-se que o coeficiente da variável IMC (x_1) é -5.9621, que indica uma relação inversamente proporcional à variável resposta y (consumo de oxigênio), ou seja, quanto maior o IMC menor será y. Além disso, o coeficiente da variável carga é 20,8784, indicando uma relação diretamente proporcional com a variável resposta y, e, portanto, quanto maior a carga, maior será o consumo de oxigênio. Essa relação é percebida também na Figura 14. Além disso, se olharmos o valor dos coeficientes em módulo 20,8784 é bem maior do que 5,9621, o que indica que a variável carga é mais importante, ou seja, tem mais impacto no resultado do modelo do que a variável IMC. Analisar os coeficientes do modelo de regressão trás informações importantes e direciona o entendimento do resultado.

Finalmente, é necessário falar dos gráficos de resíduos da regressão, que são ferramentas diagnósticas interessantes capazes de mostrar tendências nos dados. Os resíduos são calculados fazendo a diferença entre os dados reais e os dados preditos pelo modelo de regressão linear ($residuos = y_{real} - y_{pred}$). A Figura 3.14 mostra os resíduos gerados para as variáveis IMC e carga e a Figura 3.15 mostra o gráfico de resíduos comparando com os valores preditos pelo modelo.

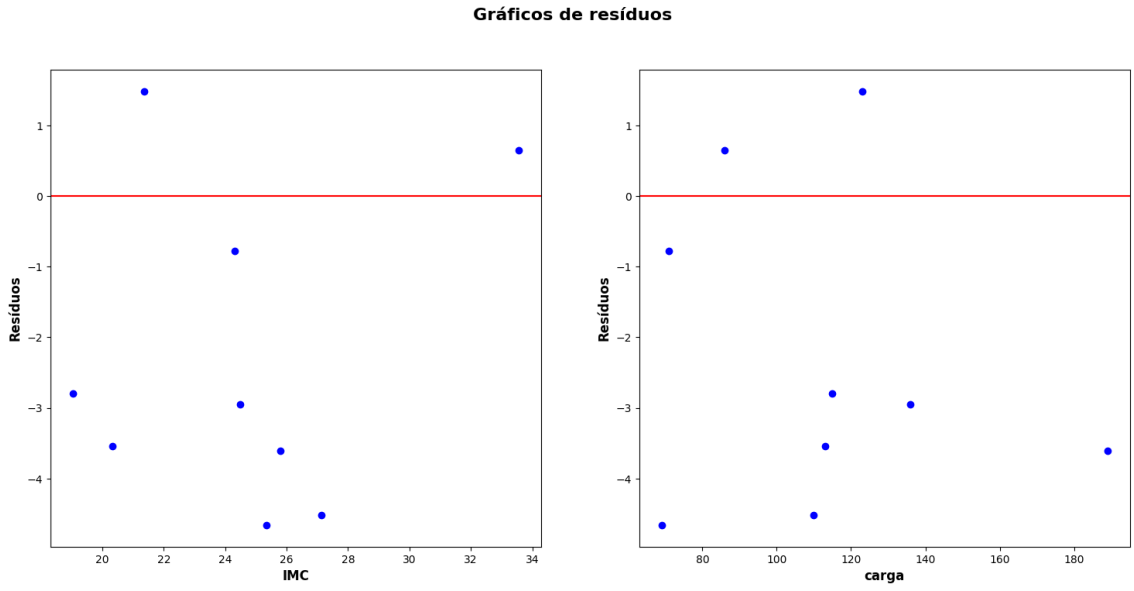


Figura 3.14: Resíduos do modelo de regressão linear múltipla para as variáveis IMC e carga

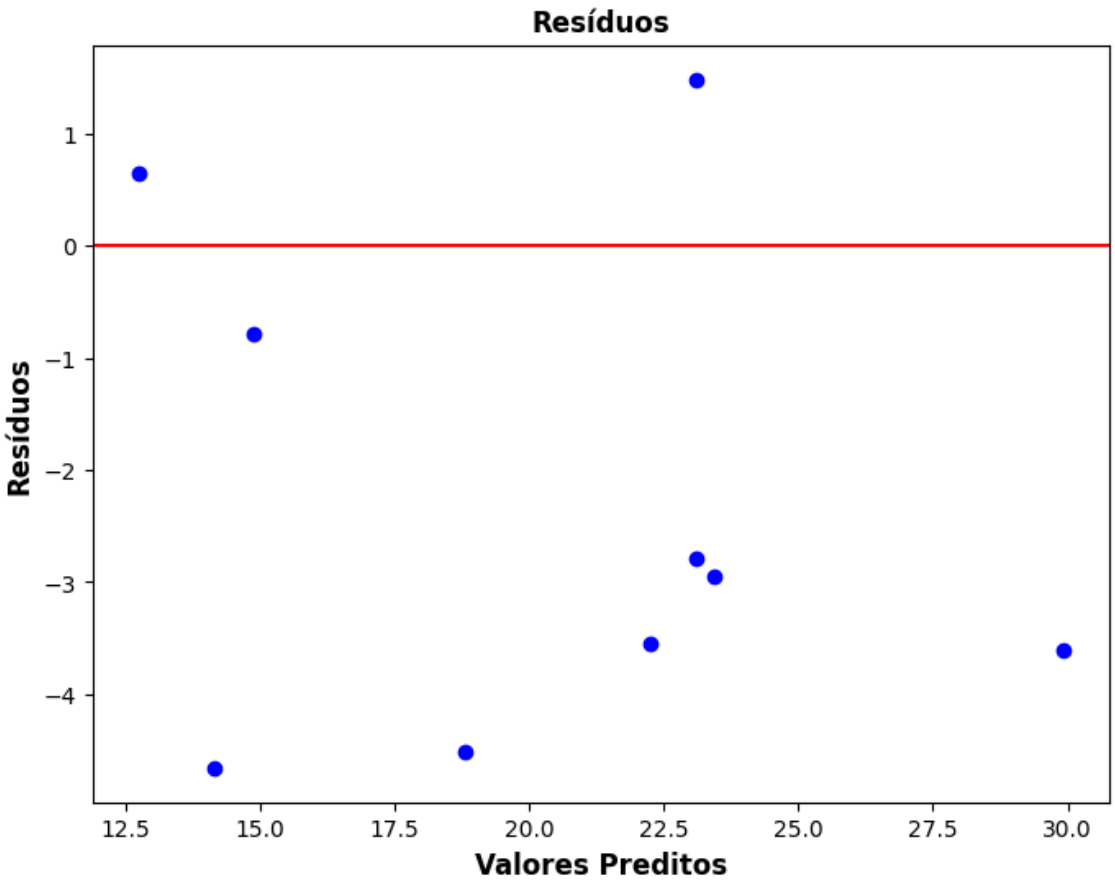


Figura 3.15: Resíduos do modelo de regressão linear múltipla

Em todos os três gráficos não é visto nenhum tipo de tendência ou padrão nos pontos, pelo contrário, eles estão dispersos de forma aleatória. Alguns pontos estão um pouco mais afastados, o que pode indicar presença de *outliers* nos dados. Certamente o modelo implementado nesse exemplo precisa de mais análise além da necessidade de explorar mais estratégias de pré-processamento e ajuste de dados, entretanto, o exercício feito aqui, ilustra o ponto de partida para a implementação de modelos de regressão linear.

3.3. MODELO DE REGRESSÃO BASEADO EM ÁRVORE

Modelos baseados em árvores são constituídos por nós conectados que tem a função de tomar decisões relativas ao fluxo de dados, ou seja, os dados são inseridos no modelo e de acordo com a regras estabelecidas em cada nó, os dados seguirão um “caminho” até a saída do resultado. A Figura 3.16 possui um diagrama simples ilustrando a ideia de um modelo baseado em árvore.

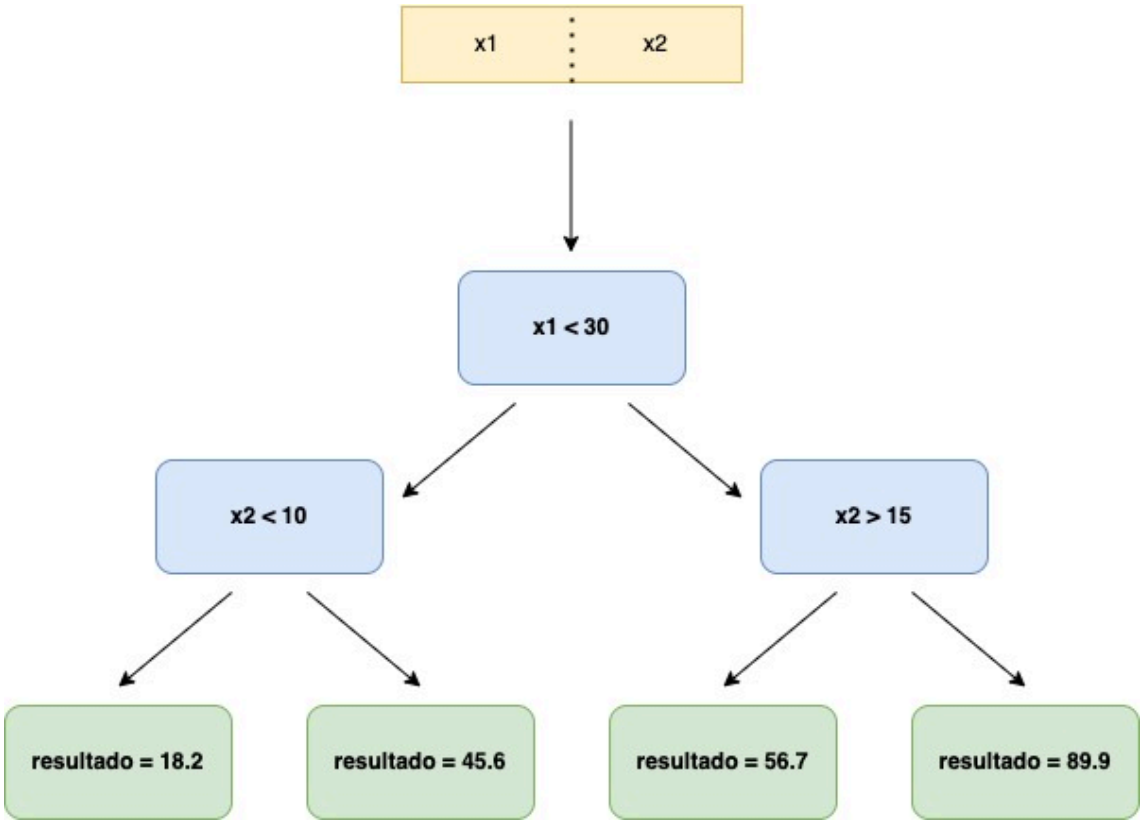


Figura 3.16: Diagrama de representação de um modelo baseado em árvore

Os retângulos verdes são denominados nó folha, pois são a parte final em cada caminho e possuem a respectiva resposta. O primeiro nó (x1 < 30) é denominado raiz e os demais são nós de divisão visto que contém testes para as tomadas de decisões. Em resumo, modelos baseados em árvores trabalham

de forma a dividir problemas complexos em problemas menores. O exemplo acima apresenta apenas duas variáveis x1 e x2, porém podem existir diversas outras variáveis.

Para formalizar o funcionamento via código Python, veja o exemplo a seguir no qual tem-se um conjunto de dados contendo diversas características a respeito de casas no estado da Califórnia (disponível em: link). O objetivo é conseguir predizer o preço mediano de uma casa com base nas características fornecidas, ou seja, implementar um modelo de regressão que seja capaz capturar as relações entre as variáveis de maneira a explicar o preço das casas.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200

Figura 3.17: *Dataset* relativo ao problema dos preços das casas no estado da Califórnia

A Figura 3.17 ilustra o conjunto de dados relativo ao problema. Nele podemos ver informações relativas à localização da casa (latitude e longitude), idade da casa, total de salas, total de quartos, população do quarteirão, a quantidade de famílias no quarteirão, média salarial das famílias no quarteirão, proximidade com o mar e finalmente o valor mediano da casa que é a variável a ser predita.

A lógica de desenvolvimento desse problema é muito semelhante ao exemplo anterior e para iniciar será necessário realizar um pré-processamento nos dados que envolve, como já explicado, limpeza, conversões e transformações nos dados. A Figura 3.18 a seguir mostra dois conjuntos de dados: *X_train_transformed* e *X_test_transformed*. Aqui o *dataset* da Figura 3.18 foi dividido em duas partes (treino e teste) e então padronizado utilizando

estandardização e a variável *ocean_proximity* foi convertida utilizando *one-hot-encoding* em INLAND, ISLAND, NEAR BAY e NEAR OCEAN.

Lembrando que os dados de treino serão utilizados para que o modelo aprenda os padrões necessários para predizer o alvo e na sequência a qualidade das predições serão mensuradas utilizando os dados de teste.

```
X_train_transformed.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	-0.111314	0.513964	1.856210	-1.153589	-1.210431	-1.071870	-1.199496	-1.000304	1	0	0	0
1	-1.317384	0.999935	1.141712	-0.740637	-0.498771	-0.512516	-0.580834	-0.849386	0	0	1	0
2	0.760844	-0.822456	-0.922393	-0.128737	-0.337462	-0.033194	-0.353731	1.286205	0	0	0	0
3	0.641234	-0.733673	-0.922393	-0.055729	1.560297	1.414342	1.306475	-1.135200	0	0	0	0
4	-0.121281	0.513964	1.697432	-0.852431	-0.816646	-0.712596	-0.867977	-0.870432	1	0	0	0

```
X_test_transformed.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	1.254236	-1.425246	-0.525450	-0.246007	-0.330345	0.109472	-0.335458	0.150777	0	0	0	1
1	0.795730	-0.799092	0.268437	-0.009187	-0.332718	-0.112356	-0.184056	1.011549	0	0	0	0
2	-1.132985	1.411141	0.030271	0.240410	0.023112	0.112082	0.090035	0.254896	1	0	0	0
3	1.976881	-1.135533	0.665380	-0.881177	-0.833252	-0.900497	-0.933236	-0.748545	1	0	0	0
4	-1.427027	1.009280	-0.287284	-0.457730	-0.342206	-0.675189	-0.314575	0.601892	0	0	1	0

Figura 3.18 Conjunto de dados pré-processados e divididos em treino e teste

De posse dos dados pré-processados, eles já podem ser utilizados para treinar e testar os modelos. A Figura 3.19 mostra quatro células com as implementações necessárias para o modelo de árvore de regressão.

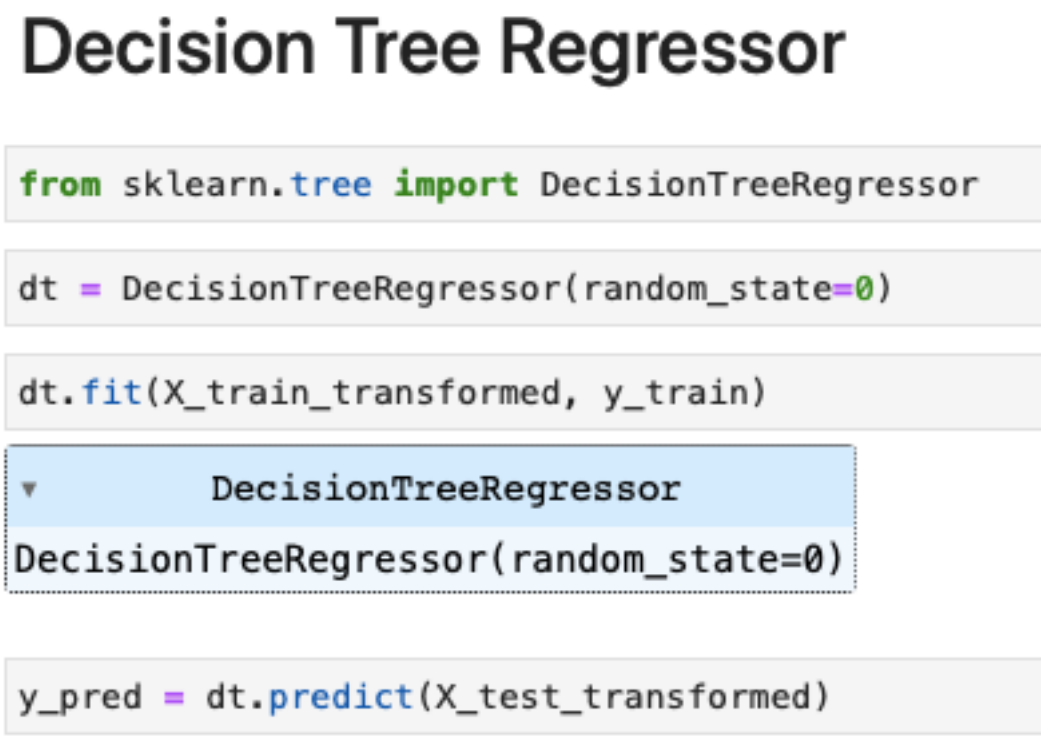


Figura 3.19: Implementação do modelo de árvore de regressão

A primeira célula exibida na Figura 3.19 importa o modelo de árvore de regressão da biblioteca *scikit-learn*, na sequência o modelo é instanciado e na terceira linha o modelo é ajustado (treinado) por meio do método *fit*. Finalmente, a última célula gera a predições da árvore de regressão com base nos dados de teste, ou seja, o modelo irá gerar respostas com base no seu treinamento. As predições do modelo precisam ser avaliadas quanto à sua qualidade e isso é feito pelas métricas R2 e MAPE que já foram explicadas anteriormente. A Figura 3.20 mostra essas métricas calculadas.

```
r2_test = np.round(r2_score(y_test, y_pred),4)
mape_test = np.round(mean_absolute_percentage_error(y_test, y_pred),4)
```

```
r2_test
```

```
0.6354
```

```
mape_test
```

```
0.2417
```

Figura 3.20: Métricas de qualidade do modelo de árvore de regressão

O R2 é aproximadamente 63% que é um bom resultado, enquanto o MAPE está em torno de 24%, sendo considerado um pouco mais alto. Dizer se o modelo é bom de verdade é um pouco mais complicado, e isso vai depender da tolerância à erro do projeto, qualidade dos dados e expectativas da equipe. Como já dito anteriormente, essa validação de qualidade deve ser feita sempre com cuidado.

Como já foi dito na parte de regressão linear, é extremamente relevante em modelos de ML compreender a importância das variáveis. Especificamente para o modelo de árvore de regressão, uma das técnicas utilizadas para compreender a importância das variáveis é o índice Gini. A biblioteca *scikit-learn* fornece de maneira simples esses valores já calculados. A Figura 3.21 exibe o trecho de código de obtém os valores do índice Gini para cada variável do modelo.

```
dt.feature_names_in_
```

```
array(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'], dtype=object)
```

```
dt.feature_importances_
```

```
array([0.11268079, 0.105916  , 0.0537743 , 0.02426665, 0.02038544,
       0.02698114, 0.01489455, 0.49003817, 0.13925262, 0.        ,
       0.00168352, 0.01012681])
```

```
df_importancia = pd.DataFrame({'Variáveis':list(dt.feature_names_in_),
                               'Importância':list(dt.feature_importances_)})
df_importancia = df_importancia.sort_values(by='Importância', ascending=False, ignore_index=True).copy()
```

Figura 3.21: Obtenção dos valores do índice Gini e construção de *dataframe* para esses valores

A tabela exibida na Figura 3.22 mostra todos os valores do índice Gini para cada variável em ordem decrescente.

	Variáveis	Importância
0	median_income	0.490038
1	INLAND	0.139253
2	longitude	0.112681
3	latitude	0.105916
4	housing_median_age	0.053774
5	population	0.026981
6	total_rooms	0.024267
7	total_bedrooms	0.020385
8	households	0.014895
9	NEAR OCEAN	0.010127
10	NEAR BAY	0.001684
11	ISLAND	0.000000

Figura 3.22: Valores do índice Gini para cada variável da árvore de regressão

Uma maneira ainda mais simples de ver esses resultados é por meio de um gráfico de barras. As figuras (Figura 3.23 e Figura 3.24) abaixo mostram a implementação e exibem esse gráfico.

```
plt.figure(figsize=(8,6))
sns.barplot(data=df_importancia, x='Importância', y='Variáveis')
plt.xlabel('Importância', fontsize=12, fontweight='bold')
plt.ylabel('Variáveis', fontsize=12, fontweight='bold')
plt.title('Importância das variáveis na árvore de regressão (Gini)', fontsize=12, fontweight='bold')
```

Figura 3.23: Implementação de um gráfico de barras utilizando as bibliotecas *matplotlib* e *seaborn*

No trecho de código acima a biblioteca *matplotlib* é utilizada para controlar aspectos da figura e o método *barplot* da biblioteca *seaborn* efetivamente faz o gráfico a partir dos valores exibidos na tabela contida na Figura 3.24.

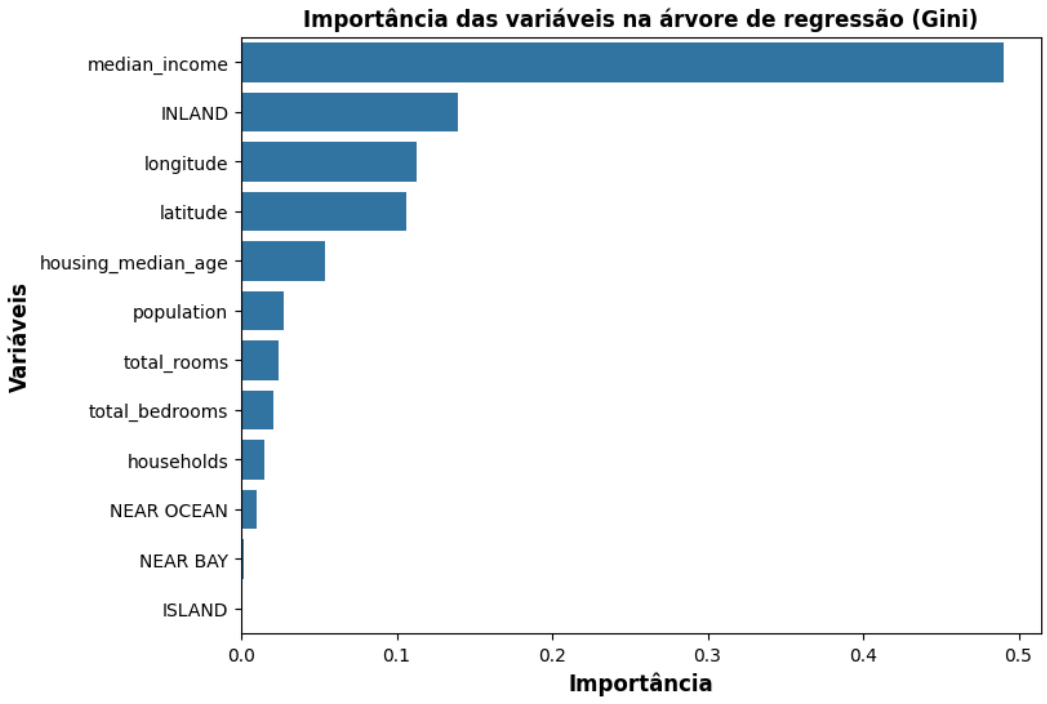


Figura 3.24: Importância das variáveis em no algoritmo árvore de regressão

Vale ressaltar que o índice Gini e até mesmo os coeficientes da regressão linear vistos anteriormente não são as únicas maneiras de avaliar a importância das variáveis. Existem diversas outras técnicas, dentre as quais pode-se destacar os valores de *shapley* (*shap*) (Lundberg & Lee, 2017). Esse texto não irá abordar essas outras estratégias, mas fica a cargo do leitor buscar e aprofundar.

3.4 MODELO DE REGRESSÃO BASEADO NO ALGORITMO DOS K VIZINHOS MAIS PRÓXIMOS

O algoritmo dos k vizinhos mais próximos (KNN) funciona com base no cálculo de distâncias entre os dados. O valor de k é justamente a quantidade de vizinhos que serão utilizados na comparação da distância. O KNN pode ser utilizado tanto para problemas de classificação quanto regressão. Um ponto importante a se destacar é que esse algoritmo funciona por memorização dos dados de treinamento e seus respectivos resultados, não sendo um treinamento direto com ajuste de parâmetros com base em alguma estratégia. Na fase de operação (classificação ou predição) a distância entre os dados de teste é comparada aos dados memorizados no treinamento e a resposta é gerada com base na menor distância.

Para clarificar seu funcionamento, vamos fazer uma simplificação do problema de predição do preço de uma casa já abordado no tópico anterior. A Tabela 3.3 abaixo mostra cinco pares de registros fictícios contendo cada um a área construída de uma casa e seu respectivo preço. Os dados dessa tabela são o conjunto de treinamento do algoritmo.

Área (m^2)	Preço
100	150000
120	170000
150	200000
180	220000

200	250000

Tabela 3.3: Valores fictícios de área construída e preço de casas

O objetivo agora é prever o preço de uma casa que possui 160 m^2 de área construída utilizando o KNN. O primeiro passo aqui é determinar o valor de k, que nesse caso, será igual a 3. O segundo passo é calcular a distância euclidiana entre cada amostra apresentada no conjunto de treinamento e a variável disponível no conjunto a ser predito. Vale ressaltar aqui que para esse exemplo simples tem-se apenas uma variável (Área), portanto a comparação será feita entre cada valor de área da Tabela 3.3 (conjunto de treinamento) e a área de 160 fornecido como variável para predição.

Se houvesse mais variáveis relacionadas às características da casa como no exemplo da seção anterior, a distância seria calculada entre cada variável disponível. A Equação 3.4 (Faceli et al., 2023) abaixo é a base do algoritmo.

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d}(x_i^l - x_j^l)^2} \qquad \text{Equação 3.4}$$

Na equação acima x_i e x_j são vetores e x_i^l e x_j^l são elementos desses vetores. O índice *l* indica cada atributo contido no vetor. Na prática, o vetor com índice *i* representa as características (variáveis) no conjunto de treinamento e o vetor com índice *j* representa as características (variáveis) no conjunto de teste.

Fazendo os cálculos utilizando os dados da Tabela 3.3 temos a Tabela 3.4 a seguir:

Área (m^2)	Preço	Cálculo	Distância
100	150000	$\sqrt{(100-160)^2}$	60
120	170000	$\sqrt{(120-160)^2}$	40
150	200000	$\sqrt{(150-160)^2}$	10
180	220000	$\sqrt{(180-160)^2}$	20
200	250000	$\sqrt{(200-160)^2}$	40

Tabela 3.4: Cálculo das distâncias para o algoritmo KNN

Agora que as distâncias foram calculadas, utiliza-se a quantidade de vizinhos para fazer o cálculo da média dos valores finais. Escolhendo as três (k=3) menores distâncias calculadas tem-se: 10, 20 e 40. Dessa forma os preços correspondentes às menores distâncias são: 200000, 220000 e 170000. Calculando a média desses três preços resulta em 196666,66. Portanto, o valor predito para uma casa com 160 m^2 de área é R$ 196666,00. Esse exemplo pode ser facilmente implementado em Python como mostra o trecho de código abaixo na Figura 3.25:

```python
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
```

```python
df = pd.DataFrame({'area':[100, 120, 150, 180, 200],
                   'preco':[150000, 170000, 200000, 220000, 250000]})
df
```

	area	preco
0	100	150000
1	120	170000
2	150	200000
3	180	220000
4	200	250000

```python
knn = KNeighborsRegressor(n_neighbors=3)
knn.fit(df[['area']], df['preco'])
knn.predict(pd.DataFrame({'area':[160]}))
```

```
array([196666.66666667])
```

Figura 3.25: Implementação do KNN para o exemplo do preço da casa em função da área em Python

Vamos agora aplicar o algoritmo KNN ao problema dos preços das casas na Califórnia já apresentado na seção anterior. Já foram apresentadas métricas de qualidade dos algoritmos de regressão, importância de variáveis e alguns gráficos, mas ainda não ficou explícita a comparação entre os dados reais e os dados preditos por um algoritmo de regressão, e por isso, vamos implementar agora. O procedimento é exatamente o mesmo que foi adotado para a árvore de regressão, a única diferença é que agora aplica-se o método KNN que também é implementado pela biblioteca *scikit-learn* e sua importação é mostrada na Figura 3.25. A Figura 3.26 abaixo compara no mesmo gráfico os valores reais de casas na California e os valores preditos pelo modelo KNN.

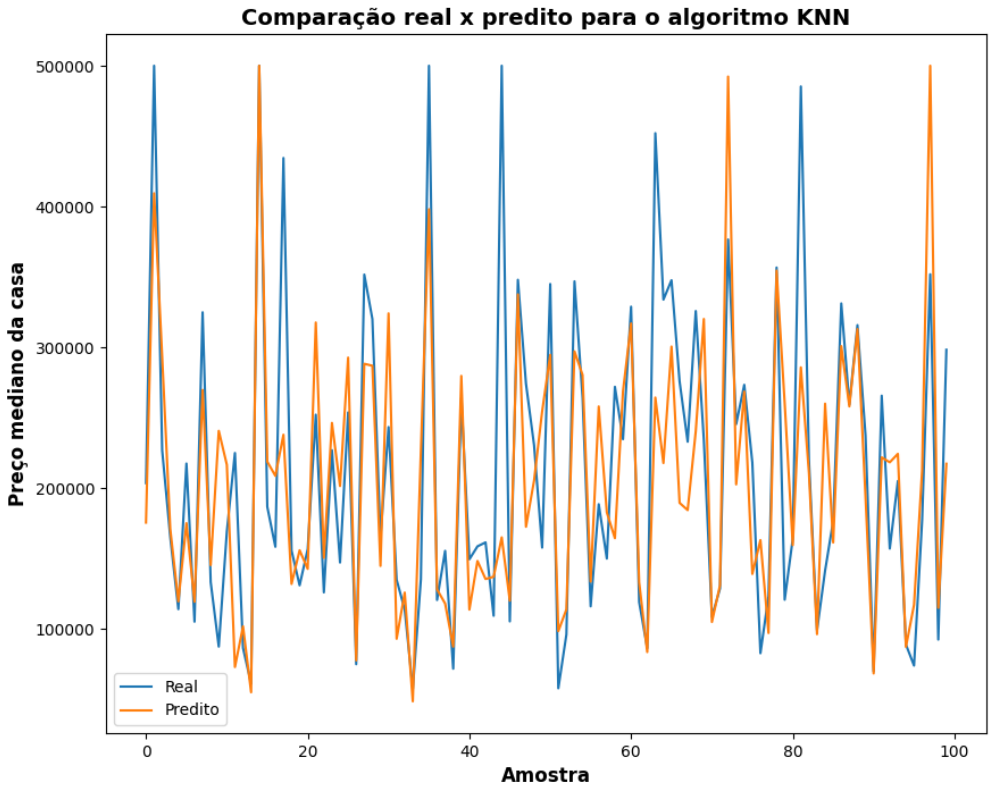


Figura 3.26: Valores reais e preditos (KNN) para o problema do preço das casas na California

Esse conjunto de dados possui 20640 amostras disponíveis das quais 14448 foram utilizadas para treinamento do algoritmo e 6192 para teste. Foram utilizadas as mesmas variáveis que entraram na árvore de regressão anteriormente. Para fins de exibição apenas as 100 últimas amostras foram apresentadas no gráfico da Figura 3.26, visto que mostrar o resultado de 6192 amostras iria deixar a representação muito poluída visualmente e pouco esclarecedora. Mas o que se percebe na Figura 3.26 é que os resultados preditos pelo KNN (linha laranja) são próximos dos resultados reais (linha azul), o que indica que essa abordagem é boa para capturar as relações entre os dados desse problema. Vale reforçar que o bom comportamento do gráfico não deve ser o único fator levado em consideração para decidir por um algoritmo ou outro,

mas dá um indicativo. O R2 calculado para o KNN foi de aproximadamente 68,76% e o MAPE = 23,27%. Em comparação ao algoritmo de árvore de regressão da seção anterior o KNN é levemente superior em relação exclusivamente a esses indicadores.

Um fator importante de ser apresentado nesse texto é que o algoritmo KNN implementado pela biblioteca *scikit-learn* não possui o cálculo de importância das variáveis de forma direta como na árvore de regressão. Para contornar isso, existe uma outra metodologia, também disponível na biblioteca *scikit-learn*, que avalia a contribuição das variáveis para qualquer modelo, não só o KNN, e por isso será utilizada aqui. O trecho de código abaixo exibe a implementação.

```
from sklearn.inspection import permutation_importance
importancia = permutation_importance(knn, X_test_transformed, y_test, n_repeats=10, random_state=0)
```

```
df_importancia = pd.DataFrame({'Variáveis':list(knn.feature_names_in_),
                               'Importância':list(importancia['importances_mean'])})
df_importancia = df_importancia.sort_values(by='Importância', ascending=False, ignore_index=True).copy()
df_importancia
```

	Variáveis	Importância
0	median_income	0.732127
1	latitude	0.250754
2	population	0.227673
3	longitude	0.204455
4	INLAND	0.139697
5	housing_median_age	0.101845
6	households	0.099066
7	total_bedrooms	0.083511
8	total_rooms	0.067120
9	NEAR OCEAN	0.024994
10	NEAR BAY	0.015632
11	ISLAND	0.000213

Figura 3.27: Implementação do cálculo da importância de variáveis por meio de permutação

A Figura 3.27 é apenas um trecho da implementação completa do problema, no qual já foi feito o pré-processamento dos dados, divisão em treino e teste, transformação e ajuste do modelo. A primeira célula da figura mostra a importação da biblioteca e a chamada ao método que efetivamente calcula a importância. É possível controlar a quantidade de repetições para o processo por meio do parâmetro *n_repeats* (segunda linha da primeira célula na Figura 3.27). A segunda célula mostrada na figura monta um *dataframe* para armazenar a importância e já dispões as variáveis em ordem decrescente, uma vez que quanto maior o valor, mais importante é a variável.

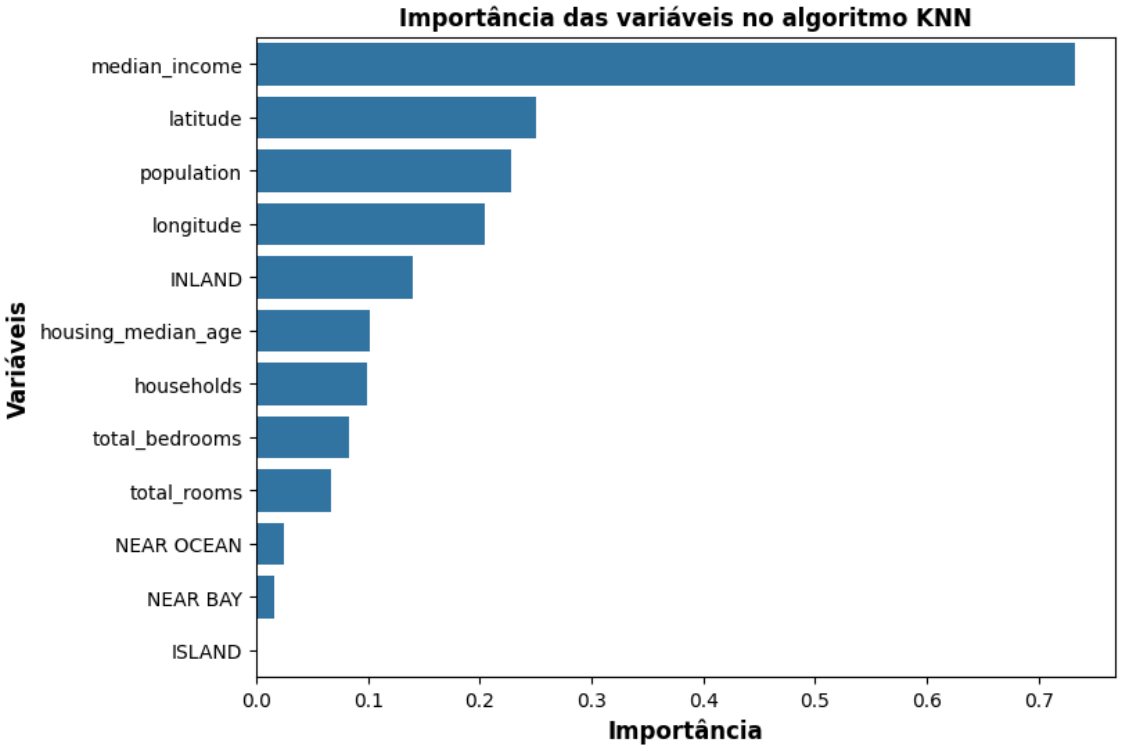


Figura 3.28: Importância das variáveis no algoritmo KNN

A Figura 3.28 acima (implementada da mesma maneira que a Figura 3.24) apresenta de maneira gráfica a importância das variáveis no modelo KNN assim como feito na árvore de regressão. Aqui foi utilizada a metodologia de permutação e é perceptível que a variável mais impactante no modelo é a

mediana do salário, seguida pela latitude. Em projetos de ML é importante, além de bons resultados, conseguir explicá-los, entender as variáveis que impactam os modelos e tentar inferir a causa dessa importância. Esse tipo de conhecimento é essencial para a área de negócios de uma empresa que está utilizando análise avançada para evoluir. Aqui nesse texto não iremos aprofundar nas causas dessa importância de variáveis e a explicabilidade dos modelos, ficando a cargo do leitor mergulhar nos estudos e conseguir aplicar os conceitos em diferentes cenários.

3.5 MODELO DE REGRESSÃO BASEADO NO ALGORITMO DOS K VIZINHOS MAIS PRÓXIMOS

A expressão florestas aleatórias é bastante sugestiva a partir do momento em que modelos baseados em árvores já foram explorados. Resumidamente, o algoritmo floresta aleatória (Random Forest – RF) é constituído por várias árvores, ou seja, são vários modelos de árvore compondo um conjunto e que trabalham para realizar uma predição ou uma classificação. Essa abordagem de aprendizado em conjunto (*Ensemble Learning*) é bastante poderosa, uma vez que obtém os resultados de todos os modelos e faz uma média entre todos para gerar o resultado (isso quando se trata de um problema de regressão). É possível criar *ensembles* com outros modelos de ML, sejam todos do mesmo tipo dentro do grupo ou mesmo de tipos distintos.

Uma maneira de tornar os algoritmos ainda mais robustos é a aplicação da técnica *bagging*, que em tradução literal seria algo como ensacamento. Essa abordagem separa aleatoriamente o conjunto de dados de treinamento em diversos subconjuntos que são aplicados aos diferentes modelos dentro do *ensemble*. Cada modelo é treinado em um subconjunto e faz sua predição para no fim, serem combinadas (média dos valores preditos) em uma resposta.

Os trechos de código a seguir mostram a implementação completa, desde leitura e pré-processamento dos dados até o treinamento e avaliação do modelo RF. A estratégia seguida aqui é a mesma utilizada anteriormente para os outros modelos de regressão. Diferentemente do que foi mostrado até aqui, esse exemplo foi implementado utilizando um arquivo único com extensão .py que é o padrão do *Python*. Além disso, foi utilizado o *PyCharm* como editor de código.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler,
MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,
mean_absolute_percentage_error

#Ler os dados
df = pd.read_csv('california_housing.csv')

#Remover dados nulos
imputer = SimpleImputer(strategy='mean')
clean_data = imputer.fit_transform(df[['total_bedrooms']])
df['total_bedrooms'] = clean_data

#Criar variáveis dummy a partir da variável categórica
ocean_proximity
df_ocean_proximity = pd.get_dummies(df['ocean_proximity'],
drop_first=True, dtype=int)
df = pd.concat([df, df_ocean_proximity], axis=1)
df.drop('ocean_proximity', axis=1, inplace=True)

#Preparação dos dados para inserir no modelo
X = df.copy()
y = X.pop('median_house_value')



#Separar os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

```
#Transformar os dados
scaler = StandardScaler()

data_transformed_train =
scaler.fit_transform(X_train.drop(['INLAND', 'ISLAND',
'NEAR BAY', 'NEAR OCEAN'], axis=1))
X_train_transformed =
pd.DataFrame(data=data_transformed_train,
columns=X_train.drop(['INLAND', 'ISLAND', 'NEAR BAY',

'NEAR OCEAN'], axis=1).columns)
X_train_transformed = pd.concat([X_train_transformed,
X_train[['INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']]],
axis=1)

data_transformed_test =
scaler.transform(X_test.drop(['INLAND', 'ISLAND', 'NEAR
BAY', 'NEAR OCEAN'], axis=1))
X_test_transformed =
pd.DataFrame(data=data_transformed_test,
columns=X_test.drop(['INLAND', 'ISLAND', 'NEAR BAY',

'NEAR OCEAN'], axis=1).columns)
X_test_transformed = pd.concat([X_test_transformed,
X_test[['INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']]],
axis=1)

#Instanciar e treinar o modelo de floresta aleatória
rf = RandomForestRegressor(random_state=0)
rf.fit(X_train_transformed, y_train)

#Predições do modelo utilizando os dados de teste
y_pred = rf.predict(X_test_transformed)

#Gerar métricas para avaliar a qualidade do modelo
r2_test = np.round(r2_score(y_test, y_pred),4)
mape_test = np.round(mean_absolute_percentage_error(y_test,
y_pred),4)

print(f'R2: {r2_test}\n')
print(f'MAPE: {mape_test}\n')
```

```
#Gerar importância das variáveis
df_importancia =
pd.DataFrame({'Variáveis':list(rf.feature_names_in_),

'Importância':list(rf.feature_importances_)})
df_importancia =
df_importancia.sort_values(by='Importância',
ascending=False, ignore_index=True).copy()
#Gerar gráfico de importância
plt.figure(figsize=(8,6))
sns.barplot(data=df_importancia, x='Importância',
y='Variáveis')
plt.xlabel('Importância', fontsize=12, fontweight='bold')
plt.ylabel('Variáveis', fontsize=12, fontweight='bold')
plt.title('Importância das variáveis no algoritmo Random
Forest', fontsize=12, fontweight='bold')
plt.savefig('./rf_importancia_variaveis_california_house.pn
g', bbox_inches='tight')

#Gerar gráfico de real x predito
plt.figure(figsize=(10,8))
plt.plot(y_test.iloc[-100:].values)
plt.plot(y_pred[-100:,])
plt.title('Comparação real x predito para o algoritmo
Random Forest', fontsize=14, fontweight='bold')
plt.xlabel('Amostra', fontsize=12, fontweight='bold')
plt.ylabel('Preço mediano da casa', fontsize=12,
fontweight='bold')
plt.legend(['Real', 'Predito'])
plt.savefig('./rf_real_x_predito_california_house.png')
```

O resultado foi positivo comparado aos anteriores (árvore de regressão e KNN). Avaliando as métricas, o RF obteve R2 = 81,54% e MAPE = 17,95% indicando desempenho superior na resolução do problema de preços das casas na California.

A Figura 3.28 abaixo apresenta a comparação entre os valores reais e preditos para o preço mediano das casas na California. Comparando com a Figura 3.26 referente ao KNN não é possível ver uma diferença perceptível na qualidade das respostas, ou seja, as predições do RF parecem acompanhar os dados reais tão bem quanto as predições do KNN.

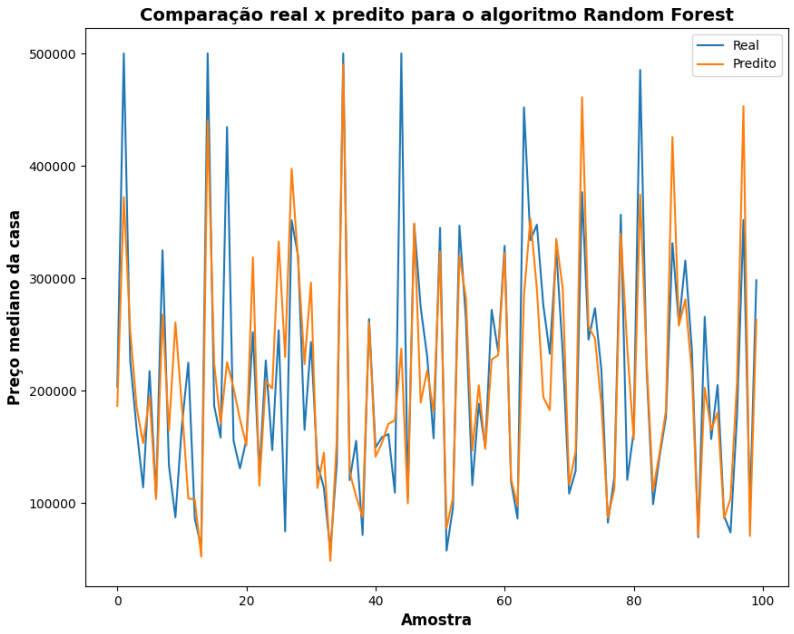


Figura 3.28: Valores reais e preditos (RF) para o problema do preço das casas na California

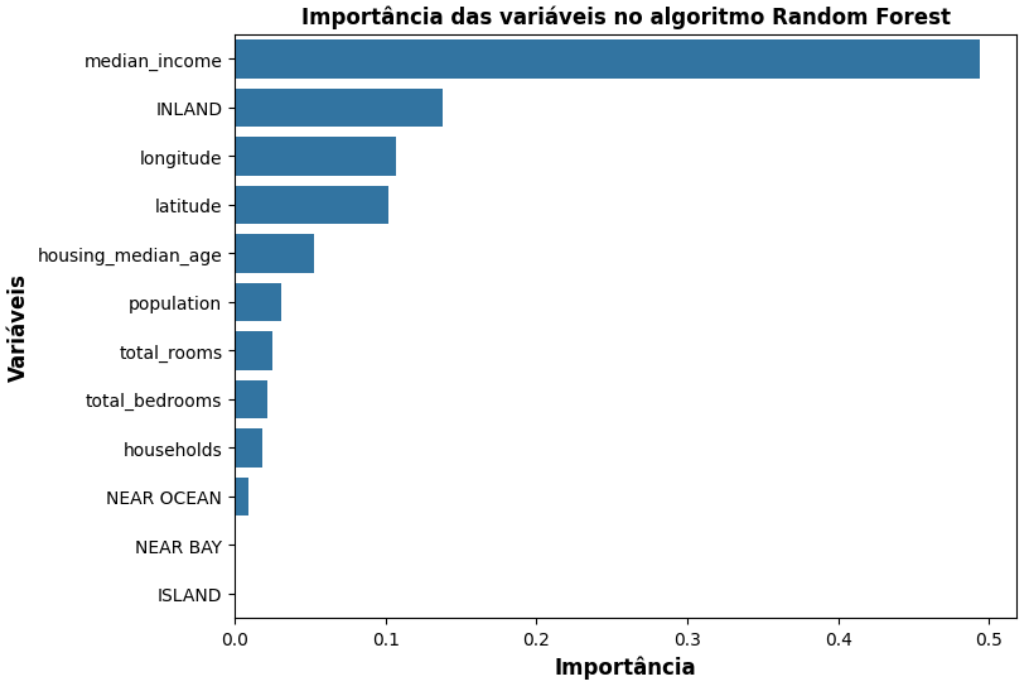


Figura 3.29: Importância das variáveis no algoritmo RF

UNIDADE 4 APRENDIZADO SUPERVISIONADO – CLASSIFICAÇÃO PARTE 1

Nesta unidade, exploraremos os princípios da classificação, uma das tarefas executadas por modelos de *machine learning* supervisionados. Alguns conceitos relacionados à avaliação da qualidade dos modelos serão explorados como a curva ROC e as métricas

OBJETIVOS DA UNIDADE 4

Ao final dos estudos, você deverá ser capaz de:

- Implementar algoritmos de classificação
- Analisar resultados e comparar diferentes abordagens de *machine learning*

4.1. REGRESSÃO LOGÍSTICA

Algoritmos de classificação trabalham com o objetivo de separar os dados em classes distintas. No geral, eles aplicam funções aos dados e conseguem calcular a probabilidade de os dados pertencerem a uma classe. A regressão logística faz exatamente isso por meio da função *sigmoid* (Equação 4.1).

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \text{Equação 4.1}$$

A representação gráfica dessa função é mostrada na Figura 4.1 abaixo.

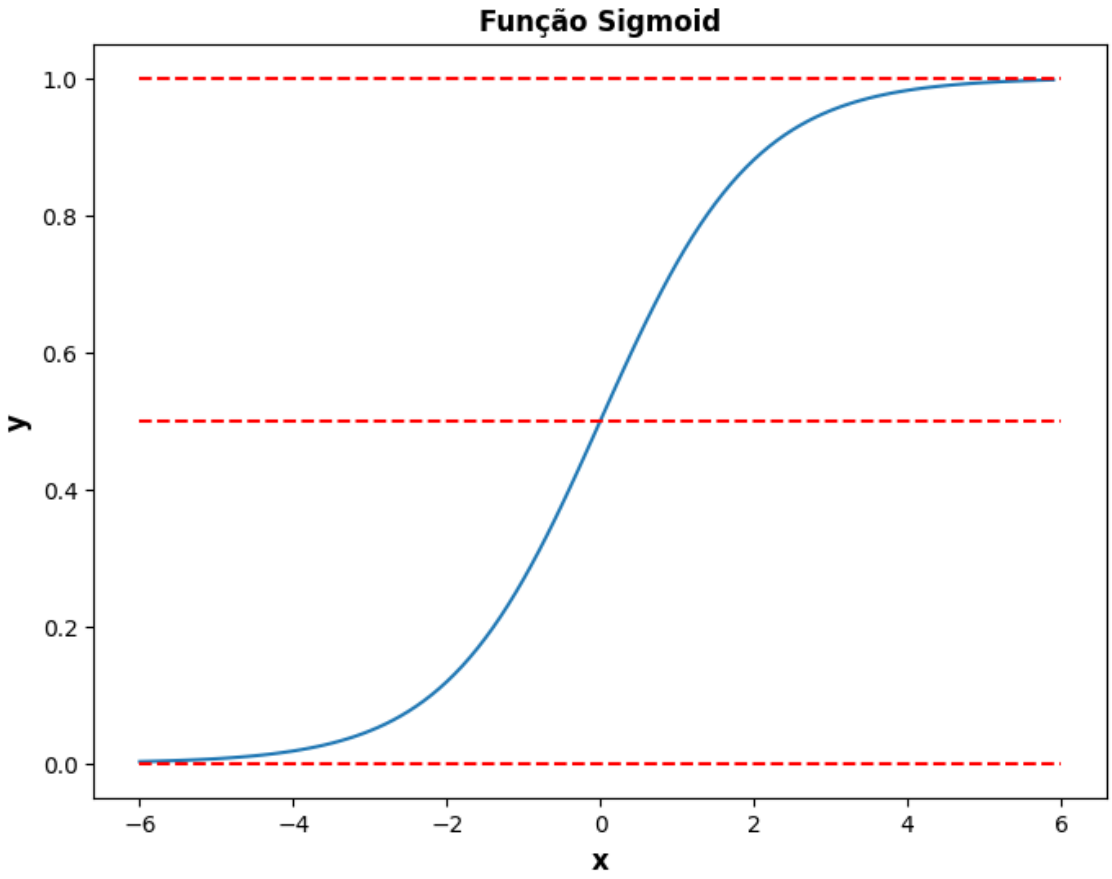


Figura 4.1: Gráfico da função *sigmoid*

Essa função possui uma característica muito interessante que é possuir valores no intervalo [0, 1]. Esse fato a torna muito boa para fazer classificações binárias. De uma maneira muito simples e sem entrar nos detalhes matemáticos mais complexos, é possível entender que se estamos tratando de um valor x que irá resultar em uma resposta y, a medida em que esse valor x aumenta, a resposta estará mais próxima de 1, e, ao diminuir os valores de x, a resposta tende a ficar mais próxima de 0. Veja novamente a curva na Figura 4.1 e faça uma reflexão a respeito disso. Dessa forma, a expressão abaixo resume esse raciocínio.

$$y = \begin{cases} 0 \; se \; x < 0.5 \\ 1 \; se \; x \geq 0.5 \end{cases}$$ Equação 4.2

Vale reforçar que existe todo um formalismo matemático envolvendo probabilidades e cálculos para o desenvolvimento do algoritmo de regressão logística que não são objetivo desse texto. O leitor pode buscar se aprofundar nos conceitos em outros materiais (Morettin & Singer, 2023).

Essa reflexão em cima do gráfico é simples e ajuda a entender o racional por trás do modelo de regressão logística, mas vale destacar que problemas com diversas variáveis podem ser resolvidos e não apenas uma como exibido no racional da Equação 6. Em resumo, esse modelo é ajustado para calcular a probabilidade de um conjunto de dados serem pertencentes a uma classe ou outra.

Será apresentado agora um exemplo prático de implementação da regressão logística para classificar um conjunto de dados de celulares. O *dataset* (Figura 4.2) apresenta uma série de características relativas aos telefones como capacidade da bateria, velocidade do processador, se possui suporte a dois chips, se possui 3G, se possui 4G, quantidade de *pixels* na vertical, quantidade de *pixels* na horizontal, quantidade de memória *ram*, quantidade de memória de armazenamento, se possui tela sensível ao toque, se possui *wi-fi* dentre algumas

outras. A variável alvo é a faixa de preço à qual o celular pertence. O *dataset* possui quatro faixas distintas, mas para fins práticos desse texto, iremos utilizar apenas duas (classificação binária). Vale ressaltar que é possível classificar mais de duas classes, mas não será abordado aqui.

	battery_power	blue	clock_speed	dual_sim	four_g	int_memory	m_dep	mobile_wt	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.2	0	0	7	0.6	188	20	756	2549	9	7	19
1	1021	1	0.5	1	1	53	0.7	136	905	1988	2631	17	3	7
2	563	1	0.5	1	1	41	0.9	145	1263	1716	2603	11	2	9
3	615	1	2.5	0	0	10	0.8	131	1216	1786	2769	16	8	11
4	1821	1	1.2	0	1	44	0.6	141	1208	1212	1411	8	2	15

three_g	touch_screen	wifi	price_range
0	0	1	1
1	1	0	2
1	1	0	2
1	0	0	2
1	1	0	1

Figura 4.2: *Dataset* contendo dados para o problema de classificação de faixa de preço de celulares

A Figura 4.2 mostra os dados disponíveis para treinar e testar o modelo de regressão logística sendo a última coluna (*price_range*) o alvo. O conjunto de dados possui 2000 amostras, mas como dito anteriormente, houve a remoção de dados pertencentes a duas classes e com isso, o novo conjunto de dados possui 1000 amostras e 17 variáveis. Abaixo segue a implementação do código que faz o pré-processamento dos dados, treina e testa a regressão logística.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
precision_score, f1_score, recall_score, roc_auc_score,
```

```
roc_curve, RocCurveDisplay, ConfusionMatrixDisplay

excluir = ['n_cores', 'fc', 'pc']
categoricas = ['blue', 'dual_sim', 'four_g', 'three_g',
'touch_screen', 'wifi']

#Ler os dados
df = pd.read_excel('data.xlsx', engine='openpyxl')

#Excluir colunas
df.drop(excluir, axis=1, inplace=True)

# Excluir dados de price_range = 0 e price_range = 3
df = df[df['price_range'].isin([1, 2])].copy()
df.reset_index(drop=True, inplace=True)


#Converter a variável alvo price_range de valores iguais a
1 e 2 para 0 e 1
df_price_range = pd.get_dummies(df['price_range'],
drop_first=True, dtype=int)
df = pd.concat([df, df_price_range], axis=1)
df.drop('price_range', axis=1, inplace=True)
df.rename(columns={2:'price_range'}, inplace=True)

#Preparação dos dados para inserir no modelo
X = df.copy()
y = X.pop('price_range')

#Separar os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

```
#Transformar os dados
scaler = StandardScaler()

data_transformed_train =
scaler.fit_transform(X_train.drop(categoricas, axis=1))
X_train_transformed =
pd.DataFrame(data=data_transformed_train,
columns=X_train.drop(categoricas, axis=1).columns)
X_train_transformed = pd.concat([X_train_transformed,
X_train[categoricas]], axis=1)

data_transformed_test =
scaler.transform(X_test.drop(categoricas, axis=1))
X_test_transformed =
pd.DataFrame(data=data_transformed_test,
columns=X_test.drop(categoricas, axis=1).columns)
X_test_transformed = pd.concat([X_test_transformed,
X_test[categoricas]], axis=1)

#Instanciar e treinar o modelo de regressão logística
lr = LogisticRegression(random_state=0)
lr.fit(X_train_transformed, y_train)

#Predições do modelo utilizando os dados de teste
y_pred = lr.predict(X_test_transformed)

#Gerar métricas para avaliar a qualidade do modelo
accuracy_test = np.round(accuracy_score(y_test, y_pred),4)
precision_test = np.round(precision_score(y_test,
y_pred),4)
f1_test = np.round(f1_score(y_test, y_pred),4)
recall_test = np.round(recall_score(y_test, y_pred),4)
roc_test = roc_auc_score(y_test,
lr.predict_proba(X_test_transformed)[:, 1])

print(f'Accuracy: {accuracy_test}\n')
print(f'Precision: {precision_test}\n')
print(f'F1: {f1_test}\n')
print(f'Recall: {recall_test}\n')
print(f'ROC AUC: {roc_test}\n')
```

```
#Gerar importância das variáveis
df_importancia =
pd.DataFrame({'Variáveis':list(lr.feature_names_in_),

'Importância':list(lr.coef_[0])})
df_importancia =
df_importancia.sort_values(by='Importância',
ascending=False, ignore_index=True).copy()

#Gerar gráfico de importância
plt.figure(figsize=(8,6))
sns.barplot(data=df_importancia, x='Importância',
y='Variáveis')
plt.xlabel('Importância', fontsize=12, fontweight='bold')
plt.ylabel('Variáveis', fontsize=12, fontweight='bold')
plt.title('Importância das variáveis no algoritmo regressão
logística', fontsize=12, fontweight='bold')
plt.savefig('./lr_feature_importance_phone_classification.p
ng', bbox_inches='tight')

#Gerar gráfico da curva ROC
x_diagonal = np.arange(0, 1.1, 0.1)
y_diagonal = np.arange(0, 1.1, 0.1)

fig, ax_roc = plt.subplots(figsize=(8, 6))
RocCurveDisplay.from_estimator(lr, X_test_transformed,
y_test, ax=ax_roc)
ax_roc.plot(x_diagonal, y_diagonal, color='r', linestyle='-
-')
ax_roc.set_title('Receiver Operating Characteristic (ROC)
curve', fontsize=12, fontweight='bold')
ax_roc.grid(linestyle='--')
fig.savefig('./roc_curve_logistic_regression.png',
bbox_inches='tight')

#Gerar gráfico da matriz de confusão
fig, ax = plt.subplots(figsize=(8, 6))
ConfusionMatrixDisplay.from_estimator(lr,
X_test_transformed, y_test, ax=ax)
ax.set_title('Matriz de confusão', fontsize=12,
fontweight='bold')
ax.set_xlabel('Classes preditas' , fontsize=12,
fontweight='bold')
ax.set_ylabel('Classes verdadeiras' , fontsize=12,
fontweight='bold')
fig.savefig('./matriz_confusao_logistic_regression.png',
bbox_inches='tight')
```

Problemas de classificação precisam ser avaliados com métricas de qualidade distintas das métricas utilizadas em problemas de regressão. Aqui serão introduzidas acurácia (*accuracy*), precisão (*precision*), *f1*, revocação (*recall*), *receiver operating characteristic* (ROC), área debaixo da curva (AUC) e matriz de confusão. A métrica mais direta para medir a qualidade de um algoritmo de classificação é a sua acurácia que mede o percentual de acerto de forma direta, ou seja, a quantidade de acertos e erros e contabilizada e então os percentuais são gerados. O processo de avaliação da regressão logística é semelhante aos modelos de regressão. Após o treinamento do modelo, dados de teste separados são aplicados ao modelo que retorna as classificações e, a partir disso, sua performance é mensurada. O modelo de regressão logística implementado para classificar os celulares obteve acurácia = 97,67% no conjunto de dados de teste, o que é muito alto, indicando que o modelo acertou quase todas as classes. Veja a Figura 4.3 abaixo e será feita a discussão a respeito da matriz de confusão.

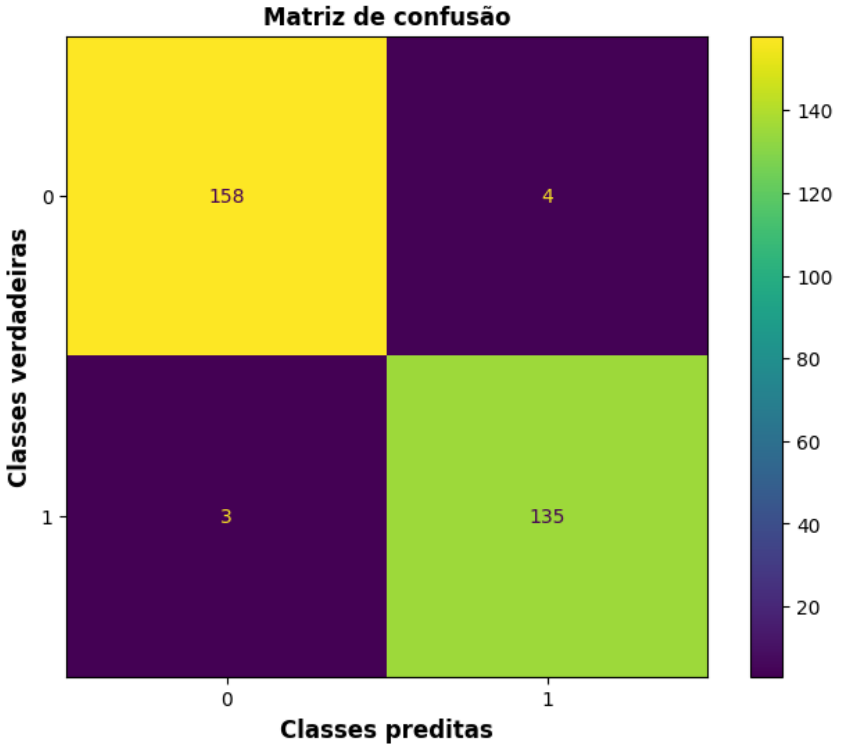


Figura 4.3: Matriz de confusão para o problema de classificação de celulares

A matriz de confusão é uma ferramenta muito útil na interpretação dos resultados de um modelo de classificação. Primeiramente, veja que a soma dos valores na matriz é igual à quantidade de dados no conjunto de teste (158+4+3+135 = 300), portanto, cada resposta do modelo foi contabilizada para a construção dessa matriz. O eixo x representa os valores preditos pelo modelo e o eixo y representa os valores reais. Dessa forma, a diagonal principal dessa matriz (158 e 135) representa as classes que o modelo acertou (seja para 0 ou para 1). O valor 158 (na parte superior esquerda) é lido como: a classe verdadeira é 0 e o modelo classificou como 0, isso é um verdadeiro positivo (VP). O valor 135 (na parte inferior direita) é lido como: a classe verdadeira é 1 e o modelo classificou como 1, isso é um verdadeiro negativo (VN). O valor 4 (na parte superior direita) é lido como: a classe verdadeira é 0 e modelo classificou como 1, isso é um falso negativo (FN). O valor 3 (na parte inferior esquerda) é lido como: a classe verdadeira é 1 e o modelo classificou como 0, isso é um falso positivo (FP).

Uma outra maneira de verificar essa matriz e compreender melhor as nomenclaturas positivas e negativas é substituir o 0 por “+” e o 1 por “-“ tanto no eixo y quanto no eixo x, dessa forma a leitura fica mais simples. Isso se deve à representação teórica, entretanto, aqui está sendo apresentado um exemplo mais prático com classes definidas em 0 e 1. De forma direta, a leitura dessa matriz indica a contabilização das amostras no conjunto de teste em que o modelo acerta e erra as classes (0 ou 1).

A partir dessas definições (VP, FN, FP, VN) fica mais simples explicar as próximas métricas. No exemplo atual obteve-se precisão = 97,12%, que também é um valor muito alto. De acordo com (Faceli et al., 2023) tem-se a definição de que precisão é a proporção de exemplos positivos classificados corretamente entre todos aqueles preditos como positivos. Seu cálculo pode ser feito por meio da Equação 4.3 abaixo.

$$precision = \frac{VP}{VP+FP} \qquad \text{Equação 4.3}$$

A próxima métrica a ser vista aqui é a revocação, que nesse exemplo foi igual a 97,83% e que corresponde à taxa de acerto na classe positiva, também é conhecida como sensibilidade ou taxa de verdadeiros positivos (Faceli et al., 2023) (Equação 4.4).

$$recall = \frac{VP}{VP+FN} \qquad \text{Equação 4.4}$$

A média harmônica ponderada das duas métricas anteriores (precisão e revocação) é conhecida como F e quando se dá o mesmo peso à ambas, tem-se o *f1* (Faceli et al., 2023). A Equação 4.5 abaixo ilustra o cálculo de *f1*.

$$f1 = \frac{2*precision*recall}{precision+recall} \qquad \text{Equação 4.5}$$

A precisão pode ser compreendida como a capacidade do modelo de classificar corretamente os elementos em suas respectivas classes e a revocação, na prática é a aptidão do modelo em identificar elementos da classe positiva.

A análise ROC é uma importante análise gráfica para avaliação de modelos de classificação binária. Para facilitar o entendimento, vale relembrar que após o cálculo dos valores na função *sigmoid* obtém-se uma resposta contínua, e essa, por sua vez, é comparada em um limiar de 0.5 (Equação 4.2). Para calcular os valores que serão utilizados para fazer o gráfico da curva ROC, esse limiar de 0.5 é alterado e o respectivo resultado é armazenado. Dessa forma, tem-se um conjunto de resultados, um para cada limiar de probabilidade distinto. Para cada um desses resultados, são calculados VP, FN, FP e VN. A curva ROC pode então ser gerada ao colocar no eixo x os valores de FP e no eixo y os valores de VP. Lembrando que cada valor corresponde a um limiar de

probabilidade distinto. A Figura 4.4 abaixo mostra a curva ROC para o problema de classificação de celulares. A linha diagonal tracejada em vermelho representa um modelo de classificação que produz resultados aleatórios. Modelos que possuam suas curvas abaixo dessa linha são ainda piores. O melhor dos casos é quando a curva está próxima do canto superior esquerdo da figura. Modelos que possuem curvas próximas à parte superior direita produzem somente resultados positivos e os que possuem curvas próximas à parte inferior esquerda produzem somente resultados negativos (Faceli et al., 2023). A curva ROC da Figura 39 está quase perfeita de acordo com o explicado acima, portanto, temos mais um indicativo de que a regressão logística é realmente muito boa em predizer as faixas de preço de celulares.

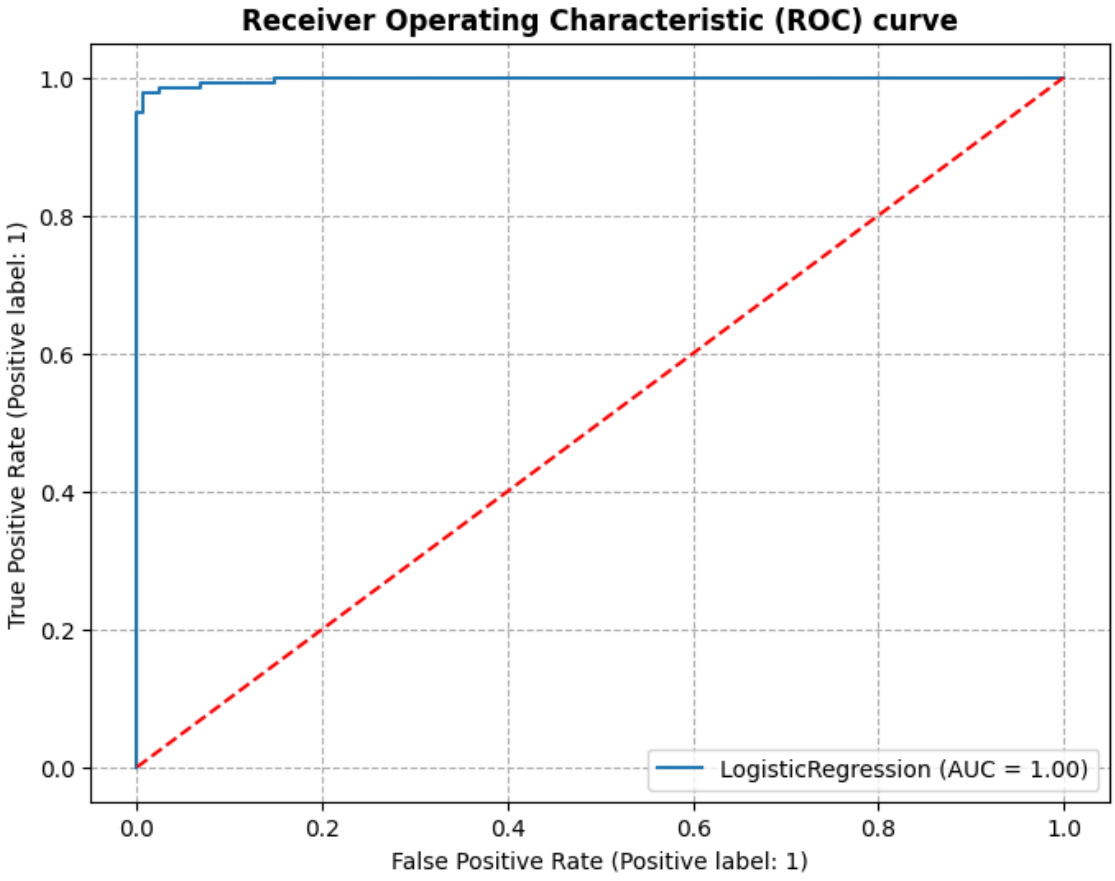


Figura 4.4: Curva ROC para o problema de classificação de celulares

Como última métrica a ser apresentada na análise desse exemplo temos a área debaixo da curva ROC. Como visto no gráfico da Figura 4.4 a AUC é praticamente 1, mais uma vez indicando um modelo excepcional na tarefa de classificação binária. O conceito de área debaixo de uma curva remete à teoria matemática de integrais estudada na disciplina de cálculo. Para um classificador de ML, quanto maior e mais próxima de 1 for sua AUC, melhor ele é.

Para finalizar a análise do modelo de regressão logística vejamos como ficou o gráfico contendo a importância das variáveis. Reforça-se que essa é uma possível metodologia de análise de importância e fica a cargo do leitor aprofundar em demais abordagens. A Figura 4.5 apresenta um gráfico de barras contendo o valor da importância (coeficientes da regressão logística) para cada variável do problema de classificação de faixas de preço de celulares.

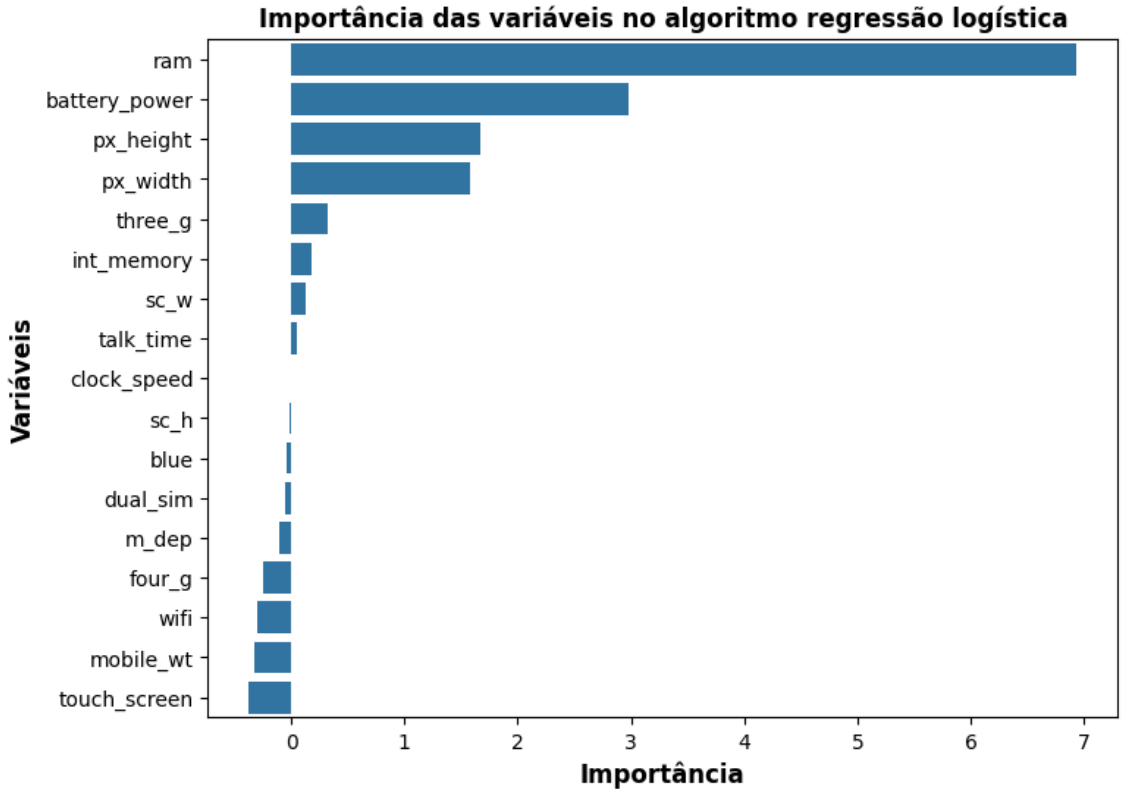


Figura 4.5: Importância das variáveis no algoritmo regressão logística

É possível ver no gráfico acima que a variável memória *ram* do celular é a mais impactante no modelo, seguida pela capacidade da bateria. Algumas variáveis possuem valores negativos de importância o que significa que seu impacto será na classe oposta e terão um efeito inversamente proporcional ao efeito das variáveis com valor positivo de importância.

4.1. MODELO DE CLASSIFICAÇÃO BASEADO EM ÁRVORE

O conceito de um modelo baseado em árvore já foi discutido nas seções anteriores relativas à regressão. A Figura 3.16 mostra um formato básico para um modelo de árvore e vale recapitular. Especificamente para a tarefa de classificação, o que muda é que agora, o modelo passa a se chamar árvore de decisão e sua saída não é mais constituída por um número contínuo, mas sim, por um valor que representa uma classe (0 ou 1). É perfeitamente possível que tenhamos ainda, problemas de classificação que envolvam mais do que duas classes distintas e uma árvore de classificação é capaz de lidar com tal situação. No mais, as métricas de avaliação da performance do modelo permanecem as mesmas, porém com alguns ajustes, especialmente na curva ROC. Tais ajustes não serão tratados nesse texto, mas o trecho de código abaixo implementa de forma básica o algoritmo para resolver o problema das faixas de preço de celulares. Dessa vez as quatros faixas de valores (alvo) foram mantidas no intuito de demonstrar um problema multiclasse.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report

excluir = ['n_cores', 'fc', 'pc']
categoricas = ['blue', 'dual_sim', 'four_g', 'three_g',
'touch_screen', 'wifi']

#Ler os dados
df = pd.read_excel('data.xlsx', engine='openpyxl')

#Excluir colunas
df.drop(excluir, axis=1, inplace=True)
```

```
#Preparação dos dados para inserir no modelo
X = df.copy()
y = X.pop('price_range')

#Separar os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)


#Transformar os dados
scaler = StandardScaler()

data_transformed_train =
scaler.fit_transform(X_train.drop(categoricas, axis=1))
X_train_transformed =
pd.DataFrame(data=data_transformed_train,
columns=X_train.drop(categoricas, axis=1).columns)
X_train_transformed = pd.concat([X_train_transformed,
X_train[categoricas]], axis=1)

data_transformed_test =
scaler.transform(X_test.drop(categoricas, axis=1))
X_test_transformed =
pd.DataFrame(data=data_transformed_test,
columns=X_test.drop(categoricas, axis=1).columns)
X_test_transformed = pd.concat([X_test_transformed,
X_test[categoricas]], axis=1)

#Instanciar e treinar o modelo de árvore de decisão
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train_transformed, y_train)

#Predições do modelo utilizando os dados de teste
y_pred = dt.predict(X_test_transformed)

#Gerar métricas para avaliar a qualidade do modelo
accuracy_test = np.round(accuracy_score(y_test, y_pred),4)

print(f'Accuracy: {accuracy_test}\n')
print(classification_report(y_test, y_pred))
```

```
#Gerar importância das variáveis
df_importancia =
pd.DataFrame({'Variáveis':list(dt.feature_names_in_),

'Importância':list(dt.feature_importances_)})
df_importancia =
df_importancia.sort_values(by='Importância',
ascending=False, ignore_index=True).copy()

#Gerar gráfico de importância
plt.figure(figsize=(8,6))
sns.barplot(data=df_importancia, x='Importância',
y='Variáveis')
plt.xlabel('Importância', fontsize=12, fontweight='bold')
plt.ylabel('Variáveis', fontsize=12, fontweight='bold')
plt.title('Importância das variáveis no algoritmo árvore de
decisão', fontsize=12, fontweight='bold')
plt.savefig('./dt_feature_importance_phone_classification_f
ull_classes.png', bbox_inches='tight')
```

Interessante perceber que a biblioteca *scikit-learn* consegue entender e tratar a variável alvo com valores inteiros representando as diferentes classes do problema (0, 1, 2 e 3), e, portanto, não há necessidade de aplicar *one-hot-encoding* à variável alvo.

A acurácia da árvore de decisão ao classificar quatro classes distintas foi igual a 84,83% que é um bom resultado. Além disso, *scikit-learn* fornece um método pronto para calcular as métricas de performance do modelo (*classification_report*), facilitando o cálculo. A Figura 4.6 abaixo mostra o relatório contendo os resultados das métricas do modelo.

	precision	recall	f1-score
0	0.93	0.91	0.92
1	0.77	0.84	0.80
2	0.79	0.75	0.77
3	0.90	0.88	0.89
accuracy			0.85
macro avg	0.85	0.85	0.85
weighted avg	0.85	0.85	0.85

Figura 4.6: Relatório de classificação para a árvore de decisão

As métricas de precisão, revocação e *f1* são calculadas em relação à cada uma das classes, o que adiciona alguns detalhes ao cálculo e como já dito, tais modificações não serão tratadas nesse texto. O algoritmo de árvore de decisão faz um bom trabalho de classificação para esse problema das faixas de preço dos celulares, visto que todas as métricas estão relativamente próximas de 1. A Figura 4.7 abaixo mostra a importância das variáveis.

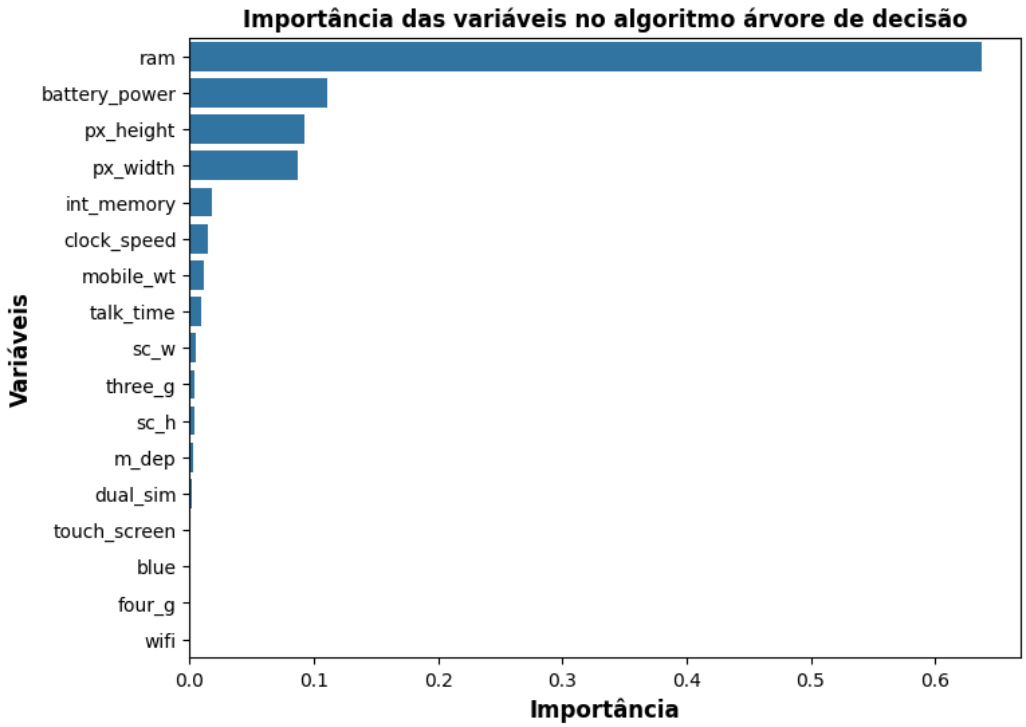


Figura 4.7: Importância das variáveis no algoritmo árvore de decisão

Novamente a memória *ram* e a bateria possuem muito impacto nos resultados do modelo de árvore de decisão. Como já explicado anteriormente, a importância das variáveis deve ser sempre avaliada com cautela, inclusive por outras estratégias, mas devido às mesmas variáveis aparecem como importantes em diferentes modelos leva a crer realmente que temos um caminho bem estabelecido para descrição dos resultados. A árvore de decisão forneceu resultados bastante consistentes mesmo ao avaliar os dados das quatro classes disponíveis (faixas de preços de celulares) mostrando que é um modelo de ML bastante poderoso e capaz. Um comentário adicional em relação ao conjunto de dados é que ele é perfeitamente balanceado, uma vez que das 2000 amostras disponíveis, existem 500 para cada classe. Em problemas da vida real esse equilíbrio é pouquíssimo provável e por isso as métricas de precisão, revocação e *f1* auxiliam no entendimento dos resultados do modelo em conjuntos com amostras desbalanceadas, indicando se o modelo está propenso a classificar mais para um lado ou outro.

UNIDADE 5 APRENDIZADO SUPERVISIONADO – CLASSIFICAÇÃO PARTE 2

Nessa unidade daremos sequência ao estudo dos modelos supervisionados de classificação explorando alguns detalhes a mais que complementam a unidade anterior.

OBJETIVOS DA UNIDADE 5

Ao final dos estudos, você deverá ser capaz de:

- Implementar novos modelos supervisionados de classificação
- Aplicar diferentes técnicas para mensurar a performance dos modelos

5.1. MODELO DE CLASSIFICAÇÃO BASEADO EM FLORESTA ALEATÓRIA

Seguindo o racional empregado na árvore de classificação, modelos baseados em floresta aleatória possuem um funcionamento muito semelhante ao já apresentado na seção sobre regressão, com exceção de que aqui também a saída será uma classe e não um valor contínuo. Um classificador baseado em floresta aleatória possui várias árvores de classificação em seu interior (*ensemble*) que irão gerar cada uma um resultado. Nesse momento uma votação é feita de modo que cada árvore indica a classe à qual cada amostra pertence e a classe vencedora (com mais votos) será o resultado do modelo. A diferença básica é que no modelo direcionado para problemas de regressão é feita uma média dos valores contínuos resultantes de cada árvore e não a votação.

O trecho de código abaixo exemplifica a implementação de um modelo de floresta aleatória utilizando a biblioteca *scikit-learn*. O problema é o mesmo utilizado anteriormente (classificação de faixas de preços de celulares) e o código contém poucas alterações em relação ao da árvore de decisão.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,
classification_report

excluir = ['n_cores', 'fc', 'pc']
categoricas = ['blue', 'dual_sim', 'four_g', 'three_g',
'touch_screen', 'wifi']

#Ler os dados
df = pd.read_excel('data.xlsx', engine='openpyxl')

#Excluir colunas
df.drop(excluir, axis=1, inplace=True)
```

```
#Preparação dos dados para inserir no modelo
X = df.copy()
y = X.pop('price_range')

#Separar os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)

#Transformar os dados
scaler = StandardScaler()

data_transformed_train =
scaler.fit_transform(X_train.drop(categoricas, axis=1))
X_train_transformed =
pd.DataFrame(data=data_transformed_train,
columns=X_train.drop(categoricas, axis=1).columns)
X_train_transformed = pd.concat([X_train_transformed,
X_train[categoricas]], axis=1)

data_transformed_test =
scaler.transform(X_test.drop(categoricas, axis=1))
X_test_transformed =
pd.DataFrame(data=data_transformed_test,
columns=X_test.drop(categoricas, axis=1).columns)
X_test_transformed = pd.concat([X_test_transformed,
X_test[categoricas]], axis=1)

#Instanciar e treinar o modelo de árvore de decisão
rf = RandomForestClassifier(random_state=0)
rf.fit(X_train_transformed, y_train)

#Predições do modelo utilizando os dados de teste
y_pred = rf.predict(X_test_transformed)

#Gerar métricas para avaliar a qualidade do modelo
accuracy_test = np.round(accuracy_score(y_test, y_pred),4)

print(f'Accuracy: {accuracy_test}\n')
print(classification_report(y_test, y_pred))
```

```
#Gerar importância das variáveis
df_importancia =
pd.DataFrame({'Variáveis':list(rf.feature_names_in_),

'Importância':list(rf.feature_importances_)})
df_importancia =
df_importancia.sort_values(by='Importância',
ascending=False, ignore_index=True).copy()

#Gerar gráfico de importância
plt.figure(figsize=(8,6))
sns.barplot(data=df_importancia, x='Importância',
y='Variáveis')
plt.xlabel('Importância', fontsize=12, fontweight='bold')
plt.ylabel('Variáveis', fontsize=12, fontweight='bold')
plt.title('Importância das variáveis no algoritmo floresta
aleatória', fontsize=12, fontweight='bold')
plt.savefig('./rf_feature_importance_phone_classification_f
ull_classes.png', bbox_inches='tight')
```

O modelo de floresta aleatória obteve uma acurácia levemente superior ao modelo de árvore, sendo de 88,83% aqui contra aproximadamente 84% anterior. O resultado aqui leva em consideração as quatro classes (0, 1, 2, 3). A Figura 5.1 abaixo apresenta os resultados das métricas.

	precision	recall	f1-score
0	0.94	0.97	0.95
1	0.86	0.79	0.83
2	0.81	0.85	0.83
3	0.94	0.93	0.93
accuracy			0.89
macro avg	0.89	0.89	0.89
weighted avg	0.89	0.89	0.89

Figura 5.1: Relatório de classificação para a floresta aleatória

Esse relatório apresenta valores levemente superiores aos da árvore de classificação, ou seja, um pouco mais próximos de 1. Em uma comparação direta somente levando em consideração as métricas apresentadas o algoritmo baseado em floresta aleatória leva vantagem, mesmo que pequena e se mostra um pouco melhor na tarefa de classificação das faixas de preço dos celulares. Essa superioridade faz sentido uma vez que a floresta conta com diversas árvores e seu algoritmo possui técnicas não exploradas nesse texto que ajudam a torná-lo mais robusto. Um ponto negativo é que modelos mais complexos, mesmo tendendo a gerar melhores predições são mais difíceis de serem explicados e avaliados. Em muitos casos, vale mais a pena utilizar um modelo mais simples que seja facilmente explicável (porque está classificando dessa forma e a importância de suas variáveis) do que algo complexo.

5.2. MODELO DE CLASSIFICAÇÃO BASEADO NO ALGORITMO DOS K VIZINHOS MAIS PRÓXIMOS

O algoritmo KNN aplicado à um problema de classificação segue um racional muito parecido com o já apresentado na parte de regressão. Como já explicado anteriormente, o KNN memoriza os exemplos de treinamento e ao realizar o teste em novos dados calcula a distância entre esses pontos. Com base na quantidade de vizinhos escolhidos na configuração do algoritmo a classe pertencente de cada nova amostra é escolhida. Essa escolha é feita com base na classe já conhecida dos dados memorizados (treinamento), de maneira que o sistema seja uma votação na qual o resultado é baseado na maioria. Para ficar mais claro, veja a Figura 5.2 abaixo.

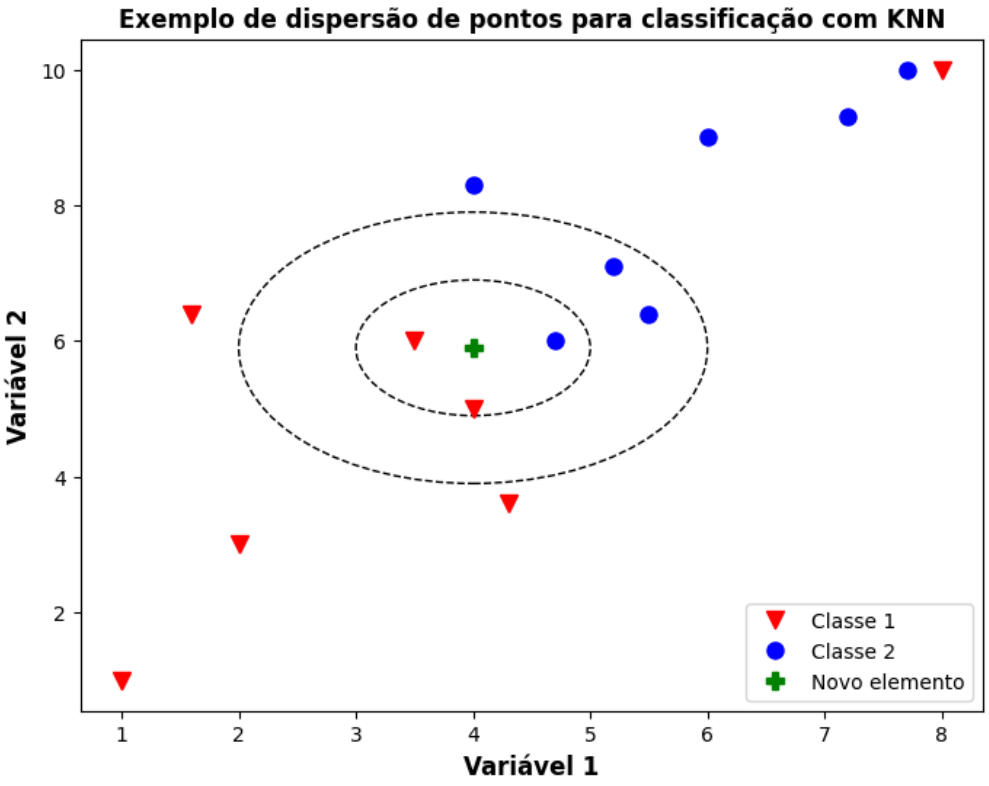


Figura 5.2: Dispersão de pontos em um plano 2D e as regiões de decisão do algoritmo KNN

Nesse exemplo simples estamos interessados em classificar um novo elemento que está representado pelo símbolo +. O plano apresenta pontos dispersos provenientes das variáveis 1 e 2. Os pontos com marcadores de triângulo vermelho e com marcadores de círculos azuis são os dados memorizados (treinamento) pelo algoritmo. O novo elemento é inserido no algoritmo e sua distância é calculada em relação a todos os pontos. Nesse momento, o valor de k (quantidade de vizinhos) já deve ter sido escolhido. Para k = 3 veja que o círculo tracejado menor possui três pontos além do novo elemento, sendo dois vermelhos e um azul e, portanto, o novo elemento é classificado com pertencente à Classe 1 (triângulos vermelhos). Para k = 5 veja que o círculo tracejado maior possui cinco pontos além do novo elemento, sendo três azuis e dois vermelhos e, portanto, o novo elemento é classificado com pertencente à Classe 2 (círculos azuis). Após analisar esse exemplo, fica claro que a escolha do valor k altera o resultado do algoritmo. Além disso, recomenda-se que o valor de k seja um número ímpar (3, 5, 7, ...) para evitar o empate entre classes distintas.

O KNN para problemas de classificação difere do modelo de regressão apenas pela questão da votação da maioria de amostras na vizinhança, visto que na regressão é feita uma média dos valores dos vizinhos mais próximos. É um método relativamente simples, mas que produz ótimos resultados. A biblioteca scikit-learn possui a implementação do KNN para classificação, sendo necessário instanciar o objeto *KNeighborsClassifier*(). A implementação é muito semelhante à dos demais classificadores apresentados anteriormente.

FINALIZAR

Após a leitura desse material, espero que o aprendizado dos conceitos relacionados à *machine learning* esteja mais fácil. Como dito anteriormente, essa foi uma introdução ao assunto e abrirá caminho para diversos outros conceitos.

As técnicas utilizadas nesse material são básicas e de caráter educativo, visto que o leitor precisa se familiarizar com os conteúdos e amadurecer aos poucos para conseguir compreender tópicos mais complexos. Os modelos de *machine learning* apresentados aqui foram todos utilizados com suas configurações padrão. O leitor pode buscar se aprofundar em técnicas que buscam os melhores parâmetros dentro de um conjunto definido para cada modelo.

Além disso, várias questões relacionadas ao aprendizado dos modelos e reamostragem dos dados não foram abordadas também, visto que são um pouco mais complexas para esse início de jornada.

Espero que esse material tenha despertado a curiosidade no leitor para continuar investigando esse mundo maravilhoso dos algoritmos de machine learning. Parabenizo o esforço empregado até aqui e convido o leitor a mergulhar de cabeça nos estudos pois vale a pena.

Prof. Dr. Thiago Santana Lemes

Sobre o autor

Thiago Santana Lemes é doutor em Engenharia Elétrica e de Computação pela Universidade Federal de Goiás (UFG), Cientista de dados na Ernst Young (EY). Experiência docente no ensino básico atuando na disciplina de matemática e nos cursos de Engenharia e Sistemas de Informação atuando nas disciplinas de cálculo diferencial, álgebra linear, cálculo numérico, equações diferenciais, probabilidade e estatística, algoritmos dentre outras. Desenvolve pesquisas na área de inteligência artificial com modelos de Machine Learning (implementação de modelos de regressão, classificação e recomendação), Deep Learning (redes convolucionais, redes recorrentes) e modelos de otimização (algoritmos genéticos e modelos lineares).

Referências Bibliográficas

Faceli, K., Lorena, A. C., Gama, J., Almeida, T. A. de, & Carvalho, A. C. P. L. F. de. (2023). *Inteligência Artificial Uma Abordagem de Aprendizado de Máquina* (2nd ed.). LTC.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems 30*, 4765–4774.

Morettin, P. A., & Singer, J. da M. (2023). *Estatística e Ciência de Dados* (1st ed.). LTC.

Pedregosa, F. and V., G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, & M. and Duchesnay. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

The pandas development team. (2024). *Pandas*. Https://Pandas.Pydata.Org.

Waskom, M. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021