

Ciência de dados com



Professor Rodrigo Bossini


# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Pré-processamento de dados</b>	<b>2</b>
2.1	Bibliotecas . . . . .	2
2.2	Importando os dados . . . . .	3
2.3	Especificando as variáveis independentes e a variável dependente . .	3
2.4	Dados faltantes . . . . .	4
2.5	Codificação para variáveis categóricas . . . . .	5
2.6	Especificando coleções de dados para o treinamento e para o teste .	8
2.7	Feature scaling - Normalização . . . . .	8
2.8	Exercícios . . . . .	11
<b>3</b>	<b>Regressão Linear Simples</b>	<b>13</b>
3.1	Definição geral . . . . .	13
3.2	Expressão da Regressão Linear Simples . . . . .	14
3.3	Importando os dados . . . . .	14
3.4	Especificando as variáveis independente e dependente . . . . .	14
3.5	Dados para treinamento e para teste . . . . .	15
3.6	Executando o treinamento . . . . .	15
3.7	Calculando os valores de classe para as instâncias de teste . . . . .	15
3.8	Visualizando os dados de treinamento . . . . .	16
3.9	Visualizando os dados de teste . . . . .	16
3.10	Predição para uma única instância . . . . .	16
3.11	Visualizando a equação da reta . . . . .	17
3.12	Exercícios . . . . .	17
<b>4</b>	<b>Regressão Linear Múltipla</b>	<b>18</b>
4.1	Objetivo . . . . .	18
4.2	A coleção de dados . . . . .	18
4.3	Utilizando somente $v - 1$ variáveis criadas pela transformação One Hot Encoding . . . . .	19
4.4	Importando os dados . . . . .	20
4.5	Especificando as variáveis independentes e a variável dependente . .	21
4.6	One Hot Encoding para variáveis categóricas . . . . .	21
4.7	Dados para treinamento e para teste . . . . .	22

4.8	Realizando o treinamento . . . . .	22
4.9	Predição para a base de teste . . . . .	23
4.10	Comparando valores obtidos com valores reais . . . . .	23
4.11	Realizando a predição para uma única instância . . . . .	23
4.12	Exibindo os coeficientes . . . . .	24
4.13	Exercícios . . . . .	24
<b>5</b>	<b>Regressão Linear Polinomial</b>	<b>25</b>
5.1	As bibliotecas . . . . .	25
5.2	A base de dados . . . . .	25
5.3	Regressão Linear Simples para comparação . . . . .	26
5.4	Treinamento o modelo de regressão linear polinomial . . . . .	26
5.5	Visualizando os resultados com o modelo de regressão linear simples	26
5.6	Visualizando os resultados com o modelo de regressão linear poli- nomial . . . . .	27
5.7	Exercícios . . . . .	27
5.7.1	. . . . .	27
5.7.2	. . . . .	27
<b>6</b>	<b>Regressão por Vetor de Suporte</b>	<b>28</b>
6.1	Intuição . . . . .	28
6.2	As bibliotecas . . . . .	28
6.3	Importando os dados . . . . .	28
6.4	Ajustando o formato dos dados . . . . .	29
6.5	Feature scaling . . . . .	30
6.6	Realizando o treinamento . . . . .	30
6.7	Predição para uma instância . . . . .	31
6.8	Visualizando os resultados graficamente . . . . .	31
<b>7</b>	<b>Regressão por árvore de decisão</b>	<b>33</b>
7.1	Intuição . . . . .	33
7.2	Entropia . . . . .	36
7.3	As bibliotecas . . . . .	39
7.4	Importando os dados . . . . .	39
7.5	Realizando o treinamento . . . . .	39
7.6	Predição para nova instância . . . . .	40
7.7	Visualizando os resultados graficamente . . . . .	40
7.8	Exercícios . . . . .	41
	<b>Referências</b>	<b>42</b>

# Capítulo 1

## Introdução

Conforme previu Moore em 1965, ao longo das décadas seguintes, a capacidade computacional dos computadores cresceu exponencialmente. Em contraste, o preço das memórias neste mesmo período cairia drasticamente. Tais acontecimentos tornaram viável o armazenamento e processamento de grandes volumes de dados, fenômeno esse muitas vezes denominado "Big Data". Além disso, estima-se que, ao longo da próxima década, a quantidade de dados disponíveis e acessíveis por meio da Internet irá dobrar a cada quinze minutos. Esses dados incluem interações de usuários em redes sociais, hábitos de motoristas coletados por seus carros inteligentes, hábitos de consumo de pessoas coletados por suas casas inteligentes, variações de temperatura e umidade no campo coletados por sensores etc. Diante da constatação de que tais dados possuem informações neles a princípio ocultas e que são de alto valor para a sociedade, fica evidente a necessidade de técnicas computacionais e matemáticas apropriadas para sua devida extração. Neste material abordam-se tópicos de estatística, probabilidade e algoritmos de mineração de dados com foco na extração de informações a partir de grandes volumes de dados brutos. Tais técnicas, quando aplicadas ao cenário descrito, constituem parte daquilo que hoje é conhecido como Ciência de Dados. Os algoritmos serão implementados utilizando-se a linguagem  python, uma das mais utilizadas para este fim atualmente.


# Capítulo 2

## Pré-processamento de dados

As técnicas que estudaremos consistem em analisar grandes coleções de dados a fim de encontrar possíveis padrões de interesse. Antes de implementá-las, entretanto, é comum a necessidade de alguns tipos de preparos prévios.

- O que fazer com dados faltantes?
- Como lidar com variáveis categóricas?
- Uma vez construído um modelo, qual coleção de dados utilizar para testá-lo?
- O que fazer quando o intervalo de uma variável é muito maior que o das demais?

Neste capítulo veremos algumas técnicas comumente utilizadas para resolver esses problemas.

**2.1 Bibliotecas**  possui muitas bibliotecas próprias para o processamento de dados.

- **numpy** - Disponibiliza objetos multidimensionais e uma coleção de funções eficientes para a sua manipulação, o que inclui ordenação, entrada e saída de dados, álgebra linear, operações estatísticas básicas etc.
- **matplotlib** - Viabiliza a visualização de dados por meio da criação de gráficos estáticos, animados e interativos.
- **pandas** - Fornece a implementação de estruturas de dados eficientes, em particular para dados relacionais e/ou rotulados.
- **scikit-learn** - Possui implementações de diversos algoritmos de aprendizado de máquina, além de ferramentas para pré-processamento de dados, validação etc.

O Bloco de Código 2.1.1 mostra como importá-las.

## Bloco de Código 2.1.1

---

```
#utilizamos "as" para dar um "apelido"  
# às bibliotecas e simplificar seu uso  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

---

**2.2 Importando os dados** Faça o download do arquivo com os dados. Para importá-los, usaremos a biblioteca **pandas**, em particular a sua função **read\_csv**. Veja o Bloco de Código 2.2.1.

## Bloco de Código 2.2.1

---

```
#o prefixo r vem de raw  
#caracteres de escape são ignorados  
#pode ser útil caso queira usar \  
dataset = pd.read_csv(r'Data.csv')
```

---

**2.3 Especificando as variáveis independentes e a variável dependente**

Algoritmos de classificação têm como finalidade gerar um modelo que descreve como o valor de uma variável pode ser descrito em função dos valores das demais variáveis. Muitas vezes, essa variável é chamada de variável **dependente** ou de **classe**. As demais são chamadas de variáveis **independentes** ou **features**. Vamos utilizar a biblioteca **pandas** para, a partir da coleção de dados completa, especificar quais são as variáveis independentes e qual é a variável dependente. A Figura 2.3.1 mostra a coleção de dados com que estamos trabalhando no momento.

Figura 2.3.1

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Os dados incluem algumas características de consumidores de uma loja. A ideia é obter um modelo que prevê se um determinado consumidor irá comprar um produto ou não, dadas as demais características. O Bloco de Código 2.3.1 mostra como construir uma estrutura contendo as variáveis independentes e uma outra estrutura contendo a variável dependente.

Bloco de Código 2.3.1

---

```

#iloc: integer ou index location
#primeiro range: todas as linhas
#segundo range: todas as colunas, exceto a última
#iloc devolve instâncias associadas a índices
#values pega somente os valores
features = dataset.iloc[:, :-1].values
classe = dataset.iloc[:, -1].values

```

---

**2.4 Dados faltantes** Note que algumas instâncias (cada linha é uma instância) não possuem valores para algumas de suas variáveis. Isso pode ter impacto significativo no resultado final. Por isso, é preciso aplicar algum tipo de processamento para lidar com esses casos. Quando a coleção de dados é suficientemente grande e a quantidade de instâncias com valores faltantes é pequena, a sua simples **remoção** pode ser suficiente. Uma outra técnica simples consiste em **substituir o valor faltante pela média dos valores das demais instâncias**. O Bloco de Código

2.4.1 mostra como implementar essa última utilizando a classe `SimpleImputer` da biblioteca `scikit-learn`.

Bloco de Código 2.4.1

---

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer

. . .

imputer = SimpleImputer(missing_values=np.nan,
                        strategy="mean")
# todas as linhas
# somente as colunas numéricas
# manter as categóricas pode resultar em erros
# fit: sobre quais colunas operar?
imputer.fit(features[:, 1:3])
# transform: faz a operação e gera um objeto alterado
features[:, 1:3] = imputer.transform(features[:, 1:3])
```

---

**2.5 Codificação para variáveis categóricas** Algoritmos de aprendizado de máquina podem lidar com variáveis categóricas. No entanto, muitos deles requerem que elas sejam convertidas para valores numéricos, o que pode ser feito utilizando-se um **sistema de codificação**. Um sistema muito simples consiste em **substituir os valores textuais por números naturais em sequência**, como na Figura 2.5.1. Ele leva o nome de **Codificação por Rótulo**.

Figura 2.5.1

Country	Code
France	0
Spain	1
Germany	2

Embora possa funcionar em alguns casos, esse método pode dar origem a interpretações errôneas por parte de alguns algoritmos, devido à ordem natural dos números. No exemplo que estamos utilizando, podemos dizer que um país é, de alguma forma, “maior” do que o outro e, por isso, recebe um número maior? Isso depende da natureza dos dados e da análise que se deseja fazer. Outra técnica consiste em **converter a variável categórica para  $n$  variáveis**, sendo  $n$  igual ao número de valores existentes nela. Para cada instância, cada nova variável re-



cebe o valor 0 ou o valor 1. O valor 1 aparece somente na coluna referente ao valor existente naquela instância. Veja o exemplo da Figura 2.5.2.

Figura 2.5.2

Country	France	Spain	Germany
France	1	0	0
Spain	0	1	0
Germany	0	0	1
Spain	0	1	0
Germany	0	0	1
France	1	0	0
Spain	0	1	0
France	1	0	0
Germany	0	0	1
France	1	0	0

Com esse método nos livramos da ordenação imposta pelo método anterior. Entretanto, ele pode levar alguma desvantagem pelo fato de ter um número de colunas potencialmente muito maior, o que pode ter impacto no desempenho dos algoritmos. Ele leva nomes diferentes, com **One Hot Encoding** sendo bastante comum. O Bloco de Código 2.5.1 mostra a sua aplicação à variável *Country*.

## Bloco de Código 2.5.1

---

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

. . .

# transformers: queremos fazer codificação
# do tipo One Hot Encoding
# na coluna de índice zero
# remainder: o que fazer com as demais colunas
# sem esse parâmetro elas seriam excluídas
# np.array: converte o resultado para um array numpy,
#           esperado pelos algoritmos que usaremos
columnTransformer = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [0])],
    remainder='passthrough')
features =
    np.array(columnTransformer.fit_transform(features))

```

---

A variável dependente - *Purchased* - também é categórica e também será convertida. Utilizaremos a Codificação por Rótulo. Veja o Bloco de Código 2.5.2.

## Bloco de Código 2.5.2

---

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

. . .

labelEncoder = LabelEncoder()
classe = labelEncoder.fit_transform(classe)

```

---

## 2.6 Especificando coleções de dados para o treinamento e para o teste

Uma vez que um modelo tenha sido obtido, é muito importante testar o seu desempenho. Não estamos nos referindo ao seu tempo de execução ou consumo de memória, por exemplo. Estamos nos referindo, neste momento ainda informalmente, ao seu percentual de acerto quando seus resultados são comparados com uma coleção de dados composta por instâncias já classificadas. O teste, obviamente, não pode ser realizado utilizando-se a mesma base utilizada durante a fase de construção do modelo (o treinamento). Afinal, o modelo incorpora informações desta base e seu percentual de acerto seria provavelmente muito alto neste caso. Precisamos de uma outra coleção de dados para realizar os testes. Uma técnica bastante comum consiste em separar a coleção de dados em duas sub-coleções: uma que será utilizada somente na fase de treinamento e uma outra que será utilizada sobre na fase de testes. O Bloco de Código 2.6.1 mostra uma possível implementação.

Bloco de Código 2.6.1

---

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

. . .

#obtemos bases para treinamento e teste
#test_size: 20% das instâncias serão usadas para teste
#random_state: a cada execução as bases geradas são iguais
features_treinamento, features_teste, classe_treinamento,
    classe_teste = train_test_split(features, classe,
    test_size = 0.2, random_state=1)
```

---

**2.7 Feature scaling - Normalização** Cada variável independente tem um intervalo próprio. Devido à própria natureza dos dados, pode acontecer de uma delas ter um intervalo muito maior do que o das demais. Isso pode ser um problema para a construção de modelos de predição, pois variáveis com intervalo maior tendem a dominar as demais. Alguns algoritmos utilizam, por exemplo, medidas de distância entre instâncias para encontrar aquelas que são mais similares. Caso uma das variáveis independentes tenha intervalo muito maior do que o das demais, a distância entre duas instâncias será definida quase que exclusivamente em

função desta variável específica. Para resolver esse problema, aplicamos técnicas de **normalização**. A ideia consiste em fazer com que o intervalo de cada variável independente seja igual. Ele pode ser  $[0, 1]$ ,  $[-1, 1]$  etc. Tudo depende da natureza dos dados. Um dos métodos de **normalização** mais simples se chama **min-max**. Ele pode ser implementado como mostra a Equação 2.7.1, em que *norm* é o valor normalizado e *real* é o valor original.

$$norm = \frac{real - \min(real)}{\max(real) - \min(real)} \quad (2.7.1)$$

A Figura 2.7.1 mostra um exemplo em que as variáveis *Age* e *Salary* são normalizadas. As instâncias com valores faltantes para alguma dessas variáveis foram removidas.

Figura 2.7.1

Country	Age	Salary	Purchased	Age_Norm	Salary_Norm
France	44	72000	No	0.74	0.69
Spain	27	48000	Yes	0.00	0.00
Germany	30	54000	No	0.13	0.17
Spain	38	61000	No	0.48	0.37
France	35	58000	Yes	0.35	0.29
France	48	79000	Yes	0.91	0.89
Germany	50	83000	No	1.00	1.00
France	37	67000	Yes	0.43	0.54

Uma outra técnica de normalização bastante utilizada leva o nome de **standardização** ou **padronização**. Ela pode ser implementada como mostra a Equação 2.7.2, em que *norm* é o valor normalizado, *real* é o valor original,  $\mu$  é a média da variável e  $\sigma$  é o seu desvio-padrão. O valor obtido indica a quantidade de desvios-padrão o valor original está acima (quando positivo) ou abaixo (quando negativo) da média.

$$norm = \frac{real - \mu}{\sigma} \quad (2.7.2)$$

A Figura 2.7.2 mostra um exemplo em que a padronização foi aplicada às variáveis *Age* e *Salary*.

Figura 2.7.2

Country	Age	Salary	Purchased	Age_Std	Salary_Std
France	44	72000	No	0.699858	0.589891
Spain	27	48000	Yes	-1.51365	-1.5075
Germany	30	54000	No	-1.12303	-0.98315
Spain	38	61000	No	-0.08138	-0.37141
France	35	58000	Yes	-0.472	-0.63359
France	48	79000	Yes	1.220683	1.20163
Germany	50	83000	No	1.481095	1.551195
France	37	67000	Yes	-0.21158	0.152935

Lembre-se que o objetivo da padronização é ajustar o intervalo de variáveis evitando o domínio de uma sobre as demais. As variáveis binárias que adicionamos para codificar as variáveis categóricas não devem, portanto, passar por esse processo. O Bloco de Código 2.7.1 mostra o uso da classe `StandardScaler` da biblioteca `scikit-learn`.

## Bloco de Código 2.7.1

---

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

. . .

standardScaler = StandardScaler()
# padronização na coleção usada para o treinamento
# fit: calcula a média e o desvio padrão de cada variável
# transform: faz o cálculo

features_treinamento[:, 3:] = standardScaler.fit_transform(
    features_treinamento[:, 3:])
#fazemos somente transform para a coleção de teste
#assim a média e desvio-padrão da população completa são
    usados
#Em produção, os dados de teste serão novos dados
#Suas variáveis devem ser padronizadas usando as mesmas
#medidas da população
features_teste[:, 3:] = standardScaler.transform(
    features_teste[:, 3:])
```

---

**2.8 Exercícios** Faça o download do arquivo de dados próprio para os exercícios. A ideia é ajustar os dados para que, no futuro, seja possível elaborar um modelo que decide se haverá jogo ou não em função das variáveis independentes. Escreva

um programa com  python que

- Faz a leitura do arquivo *CSV* usando *pandas*.
- Cria uma base contendo as variáveis independentes e uma base contendo a variável dependente.
- Substitui dados faltantes pela média da respectiva variável.
- Codifica todas as variáveis categóricas independentes com **One Hot Encoding**.
- Codifica a variável dependente com **Codificação por Rótulo**.
- Separa a base em duas partes: uma para treinamento e outra para testes. Use 85% das instâncias para o treinamento.
- Normaliza as variáveis *temperatura* e *humidade* usando padronização.

# Capítulo 3

## Regressão Linear Simples

Esta técnica é aplicada quando desejamos explicar a variação de uma variável dependente em função de uma única variável independente. Pressupõem-se que há uma relação linear (a qual pode ser verificada por diferentes métodos, os quais não são parte do escopo deste capítulo) entre as variáveis, daí o nome **linear**. O modelo é dito **simples** pelo fato de basear-se somente em uma única variável independente.

**3.1 Definição geral** A expressão geral de um modelo de regressão linear é exibida pela Equação 3.1.1. , em que  $x$  é a variável dependente,  $a_1, a_2, \dots, a_k$  são as variáveis independentes e  $w_0, w_1, \dots, w_k$  são os coeficientes que desejamos encontrar, calculados na fase de treinamento. Note que  $k = 1$  quando o modelo é “simples”. O peso  $w_0$  está sendo utilizado para simplificar a notação. Considere que ele está associado a um atributo  $a_0$  cujo valor é sempre 1.

$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + w_k a_k \quad (3.1.1)$$

Suponha que  $a_1^{(i)}, a_2^{(i)}, \dots, a_k^{(i)}$  são os valores das variáveis independentes da  $i$ -ésima instância da coleção de dados e que a sua classe ou variável dependente é  $x^{(i)}$ . O valor de  $x^{(i)}$  obtido pelo método de regressão linear é exibido na Equação 3.1.2.

$$w_0 a_0^{(i)} + w_1 a_1^{(i)} + \dots + w_k a_k^{(i)} = \sum_{j=0}^k w_j a_j^{(i)} \quad (3.1.2)$$

A intenção é calcular coeficientes de modo que a diferença entre o valor obtido pelo modelo e o valor real existente nos dados de treinamento sejam minimizado. A Equação 3.1.3 mostra o valor a ser minimizado, considerando que  $n$  é o número de instâncias na coleção de dados.

$$\sum_{i=1}^n \left( x^{(i)} - \sum_{j=0}^k w_j a_j^{(i)} \right)^2 \quad (3.1.3)$$

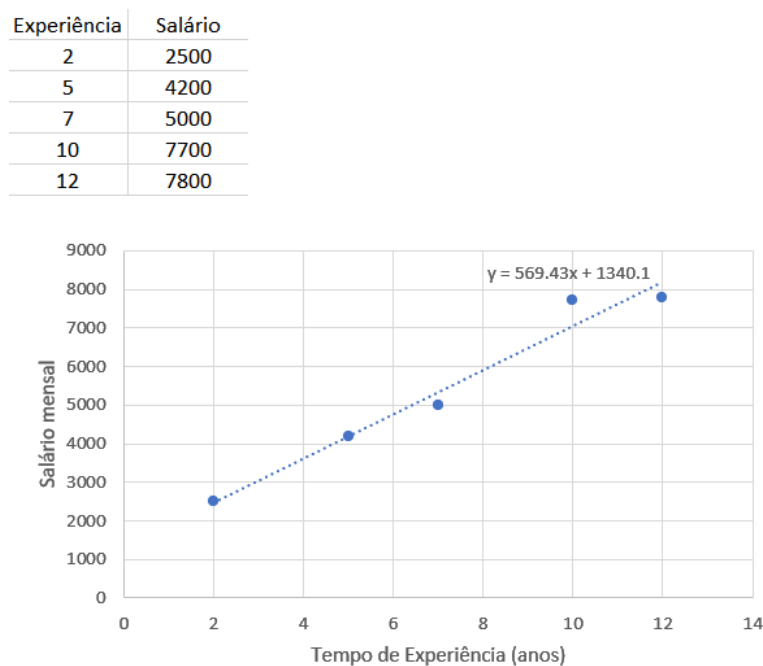


**3.2 Expressão da Regressão Linear Simples** Utilizaremos a expressão exibida pela Equação 3.2.1, em que  $y$  é a variável dependente,  $b_0$  é o coeficiente linear a ser encontrado e  $b_1$  é o coeficiente angular a ser encontrado.

$$y = b_0 + b_1x_1 \quad (3.2.1)$$

Graficamente, a Equação 3.2.1 dá origem a uma reta. Como mostra a Figura 3.2.1, o objetivo é traçar-la minimizando a soma dos quadrados das diferenças entre os valores preditos e os valores reais para cada instância.

Figura 3.2.1



**3.3 Importando os dados** Faça o download do arquivo de dados do ambiente da disciplina. A seguir, importe as bibliotecas necessárias, como mostra o Bloco de Código 3.3.1.

Bloco de Código 3.3.1

---

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

---

**3.4 Especificando as variáveis independente e dependente** Nossa coleção de dados possui somente duas variáveis. O Bloco de código 3.4.1 mostra como importar os dados e separar cada variável em uma estrutura de dados independente.

## Bloco de Código 3.4.1

---

```
dataset = pd.read_csv ('dados.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

---

**3.5 Dados para treinamento e para teste** Uma vez obtido um modelo, desejamos verificar o seu desempenho quando aplicado a uma base de dados diferente daquela utilizada no treinamento. Por isso, vamos separar a base de dados em duas: uma será utilizada para o treinamento e a outra para os testes. O Bloco de Código 3.5.1 mostra como fazê-lo.

## Bloco de Código 3.5.1

---

```
. . .
from sklearn.model_selection import train_test_split
. . .
x_treinamento, x_teste, y_treinamento, y_teste =
    train_test_split(x, y, test_size=0.2,
                    random_state=0)
```

---

**3.6 Executando o treinamento** A biblioteca `scikit-learn` possui uma classe que nos permite obter modelos de regressão linear simples. Ela se chama `LinearRegression`. Uma vez importada, construímos um objeto e utilizamos o método `fit` para realizar o treinamento, ou seja, a obtenção do vetor de coeficientes (neste caso temos apenas um). Veja o Bloco de Código 3.6.1.

## Bloco de Código 3.6.1

---

```
. . .
from sklearn.linear_model import LinearRegression
. . .
linearRegression = LinearRegression()
linearRegression.fit(x_treinamento, y_treinamento)
```

---

**3.7 Calculando os valores de classe para as instâncias de teste** Uma vez obtido o modelo, desejamos verificar os valores de classe que ele irá calcular para cada instância de teste. Isso pode ser feito utilizando-se o método `predict`. Veja o Bloco de Código 3.7.1.

## Bloco de Código 3.7.1

---

```
. . .
y_pred = linearRegression.fit(x_treinamento, y_treinamento)
```

---

**3.8 Visualizando os dados de treinamento** A biblioteca `matplotlib` viabiliza a visualização de dados. Utilizaremos métodos de `pyplot` para visualizar os dados de treinamento e a reta obtida em função dos coeficientes calculados. Os dados podem ser visualizados utilizando-se um gráfico de dispersão, que pode ser obtido com o método `scatter`. A reta será construída com o método `plot`. Veja o Bloco de Código 3.8.1.

Bloco de Código 3.8.1

---

```
. . .
plt.scatter(x_treinamento, y_treinamento, color="red")
plt.plot(x_treinamento,
         linearRegression.predict(x_treinamento),
         color="blue")
plt.title("Salário x Tempo de Experiência (Treinamento)")
plt.xlabel("Anos de Experiência")
plt.ylabel("Salário")
plt.show()
```

---

**3.9 Visualizando os dados de teste** A visualização dos dados de teste é análoga à visualização dos dados de treinamento. Basta trocar os dados. Veja o Bloco de Código .

Bloco de Código 3.9.1

---

```
. . .
plt.scatter(x_teste, y_teste, color="red")
#os coeficientes são únicos, assim não faz diferença
#trocar as coleções na hora de exibir
plt.plot(x_treinamento,
         linearRegression.predict(x_treinamento),
         color="blue")
plt.title("Salário x Tempo de Experiência (Teste)")
plt.xlabel("Anos de Experiência")
plt.ylabel("Salário")
plt.show()
```

---

**3.10 Predição para uma única instância** Podemos realizar a predição para uma única instância usando o método `predict`. Note que ele espera um vetor de vetores, já que pode operar sobre uma coleção de instâncias, cada qual com uma coleção de atributos. Veja o Bloco de Código 3.10.1.

## Bloco de Código 3.10.1

---

```

. . .
# qual o salario de alguem com 15.7 anos de experiencia
print(linearRegression.predict([[15.7]]))
#quanto deve ganhar alguem que acabou de entrar na empresa
print(linearRegression.predict([[0]]))

```

---

**3.11 Visualizando a equação da reta** Podemos obter os coeficientes angular e linear obtidos utilizando os métodos `coef_` e `intercept_`, respectivamente. Veja o Bloco de Código 3.11.1.

## Bloco de Código 3.11.1

---

```

. . .
print(
f'y = {linearRegression.coef_[0]:.2f}x +
      {linearRegression.intercept_:.2f}')


```

---

**3.12 Exercícios** Vasilhe o portal de dados brasileiro (Link 3.12.1) e encontre uma base de dados interessante, própria para a obtenção de um modelo de regressão linear simples. Caso a base tenha mais de duas variáveis independentes, escolha uma delas e remova as demais.

Link 3.12.1

<https://dados.gov.br/>

Escreva um programa com  python que

- Faz a leitura do arquivo *CSV* usando *pandas*.
- Cria uma base contendo as variáveis independentes e uma base contendo a variável dependente.
- Separa a base em duas partes: uma para treinamento e outra para testes. Use 85% das instâncias para o treinamento.
- Constrói um modelo de regressão linear simples em função dos dados de treinamento.
- Exibe a reta obtida e os dados de treinamento em um mesmo gráfico. Inclua a equação obtida no título do gráfico.
- Exibe a reta obtida e os dados de teste em um mesmo gráfico. Inclua a equação obtida no título do gráfico.

# Capítulo 4

## Regressão Linear Múltipla

O método conhecido como Regressão Linear Múltipla difere da Regressão Linear Simples, vista no Capítulo 3, no número de variáveis independentes.

**4.1 Objetivo** O objetivo é estimar o valor de uma única variável em função do valor de  $k > 1$  variáveis independentes. Se  $x$  é a variável dependente e  $a_i (i = 1, 2, \dots, k)$  são as variáveis independentes, deseja-se encontrar valores  $w_0, w_1, \dots, w_k$  tais que

$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + a_k w_k \quad (4.1.1)$$

O treinamento é feito em função da coleção da base de dados e o objetivo é minimizar a soma dos quadrados das diferenças entre os valores estimados e seus respectivos valores reais.

**4.2 A coleção de dados** A base de dados com que trabalharemos é exibida na Figura 4.2.1.

Figura 4.2.1

Gastos com pesquisa e desenvolvimento	Gastos com administração	Gastos com marketing	Estado	Lucro
165349.2	136897.8	471784.1	São Paulo	192261.83
162597.7	151377.59	443898.53	Rio Grande do Sul	191792.06
153441.51	101145.55	407934.54	Ceará	191050.39
144372.41	118671.85	383199.62	São Paulo	182901.99
142107.34	91391.77	366168.42	Ceará	166187.94
131876.9	99814.71	362861.36	São Paulo	156991.12
134615.46	147198.87	127716.82	Rio Grande do Sul	156122.51
130298.13	145530.06	323876.68	Ceará	155752.6
120542.52	148718.95	311613.29	São Paulo	152211.77

Ela contém dados de gastos de uma série de empresas. Elas têm gastos com **Pesquisa e Desenvolvimento**, **Aspectos administrativos** e **Divulgação**. Uma vez obtidos os pesos  $w_i$ , uma resposta que se pode encontrar é em qual

dessas áreas vale mais à pena investir visando melhores lucros. Note que a base possui uma variável categórica: **o Estado em que cada empresa está localizada**. Caso seja de interesse utilizá-la, será necessário aplicar algum tipo de transformação, tornando viável o seu uso em uma expressão matemática. Dois tipos de transformação utilizados são

- **Codificação por rótulo** Interessante quando há uma ordem natural entre os valores da variável categórica. Basta associar um número natural sequencial a cada valor da variável categórica. Veja a Figura 4.2.2.

Figura 4.2.2

Gastos com pesquisa e desenvolvimento	Gastos com administração	Gastos com marketing	Estado	CodigoEstado	Lucro
165349.2	136897.8	471784.1	São Paulo	0	192261.83
162597.7	151377.59	443898.53	Rio Grande do Sul	1	191792.06
153441.51	101145.55	407934.54	Ceará	2	191050.39
144372.41	118671.85	383199.62	São Paulo	0	182901.99
142107.34	91391.77	366168.42	Ceará	2	166187.94
131876.9	99814.71	362861.36	São Paulo	0	156991.12
134615.46	147198.87	127716.82	Rio Grande do Sul	1	156122.51

Uma de suas vantagens é ser um método muito simples. Por outro lado, alguns algoritmos podem ter o seu funcionamento comprometido pois podem considerar a ordenação dos valores como algo relevante. Dependendo da natureza dos dados, isso nem sempre será verdade.

- **One Hot Encoding** Este método cria uma nova variável binária para cada valor existente na variável categórica sendo codificada. Veja a Figura 4.2.3.

Figura 4.2.3

Gastos com pesquisa e desenvolvimento	Gastos com administração	Gastos com marketing	Estado	São Paulo	Rio Grande do Sul	Ceará	Lucro
165349.2	136897.8	471784.1	São Paulo	1	0	0	192261.83
162597.7	151377.59	443898.53	Rio Grande do Sul	0	1	0	191792.06
153441.51	101145.55	407934.54	Ceará	0	0	1	191050.39
144372.41	118671.85	383199.62	São Paulo	1	0	0	182901.99
142107.34	91391.77	366168.42	Ceará	0	0	1	166187.94
131876.9	99814.71	362861.36	São Paulo	1	0	0	156991.12
134615.46	147198.87	127716.82	Rio Grande do Sul	0	1	0	156122.51

Com este método, a ordenação possivelmente incondizente com a base de dados sob análise deixa de existir. Por outro lado, a sua implementação pode ser mais trabalhosa e o desempenho dos algoritmos pode sofrer impacto devido ao aumento número de variáveis.

**4.3 Utilizando somente  $v - 1$  variáveis criadas pela transformação One Hot Encoding** Há uma propriedade muito importante envolvendo as variáveis criadas pela transformação One Hot Encoding. Dada a sua definição, é sempre verdade que qualquer uma delas pode ser o seu valor calculado em função das demais. Veja um exemplo na Figura 4.3.1. Adicionamos uma nova variável a

fim de comparar seus valores com os valores de uma das variáveis criadas pela transformação. Note que os valores de ambas são sempre iguais. Além disso, a nova variável tem seus valores calculados em função das demais.

Figura 4.3.1

Gastos com pesquisa e desenvolvimento	Gastos com administração	Gastos com marketing	Estado	São Paulo	Rio Grande do Sul	Ceará	1 - São Paulo - Ceará	Lucro
165349.2	136897.8	471784.1	São Paulo	1	0	0	0	192261.83
162597.7	151377.59	443898.53	Rio Grande do Sul	0	1	0	0	191792.06
153441.51	101145.55	407934.54	Ceará	0	0	1	1	191050.39
144372.41	118671.85	383199.62	São Paulo	1	0	0	0	182901.99
142107.34	91391.77	366168.42	Ceará	0	0	1	1	166187.94
131876.9	99814.71	362861.36	São Paulo	1	0	0	0	156991.12
134615.46	147198.87	127716.82	Rio Grande do Sul	0	1	0	0	156122.51

Note, portanto, que se  $v$  é o número de valores existentes em uma variável categórica ( $a_k$ ) e se todas as variáveis criadas  $c_1, c_2, \dots, c_v$  pela transformação One Hot Encoding forem incluídas no modelo com os pesos  $w_{c_1}, w_{c_2}, \dots, w_{c_v}$ , a expressão final se torna

$$\begin{aligned}
 x &= w_0 + w_1 a_1 + w_2 a_2 + \dots + w_{k-1} a_{k-1} + \sum_{i=1}^v w_{c_i} c_i \\
 &= w_0 + w_1 a_1 + w_2 a_2 + \dots + w_{k-1} a_{k-1} + \sum_{i=1}^{v-1} w_{c_i} c_i + (w_{c_v} 1 - \sum_{i=1}^{v-1} w_{c_i} c_i)
 \end{aligned} \tag{4.3.1}$$

Ou seja, as variáveis  $c_1, c_2, \dots, c_{v-1}$  aparecem duas vezes cada na equação. Isso indica que, quando esse método é utilizado, uma das variáveis criadas precisa ser ignorada. Seu uso é irrelevante pois a sua importância já está expressada pelo uso das demais. Cabe ao cientista de dados observar esse detalhe. A variável a ser removida pode ser qualquer uma. O valor de  $w_0$  inclui o seu significado. A Equação 4.3.2 mostra um exemplo concreto usando a base de dados sob análise.

$$\begin{aligned}
 x &= w_0 + w_1 pd + w_2 adm + w_3 mkt + w_4 sp + w_5 rs + w_6 ce \\
 &= w_0 + w_1 pd + w_2 adm + w_3 mkt + w_4 sp + w_5 rs + w_6 (\times 1) - (w_4 sp + w_5 rs)
 \end{aligned} \tag{4.3.2}$$

**4.4 Importando os dados** Faça o download do arquivo de dados do ambiente da disciplina. A seguir, importe as bibliotecas necessárias, como mostra Bloco de Código 4.4.1.

Bloco de Código 4.4.1

---

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

---

Use um dataframe Pandas para abrigar os dados, como no Bloco de Código 4.4.2. Ele mostra também exemplos de acesso à coleção.

## Bloco de Código 4.4.2

---

```
dataset = pd.read_csv('dados_empresas.csv')
#dataset inteiro
print (dataset)
#duas primeiras linhas, por exemplo
print (dataset[0:2])
```

---

#### 4.5 Especificando as variáveis independentes e a variável dependente

Desejamos explicar os valores da variável **lucro** em função dos valores das demais. Ou seja, ela é a variável dependente. Assim, vamos montar duas bases: uma que contém as variáveis independentes e uma outra que contém somente a variável dependente. Usamos o método `iloc` para isso. Veja o Bloco de Código 4.5.1.

## Bloco de Código 4.5.1

---

```
#todas as linhas e todas as colunas, exceto a última
independent = dataset.iloc[:, :-1].values
dependent = dataset.iloc[:, -1].values
#exemplos extras
#primeira linha, todas as colunas
print(dataset.iloc[0:1, :])
#10 primeiras linhas, primeira coluna
print(dataset.iloc[0:11, 0])
```

---

**4.6 One Hot Encoding para variáveis categóricas** Como a base sob análise possui uma variável categórica de interesse para o estudo, iremos aplicar uma técnica de codificação para que ela possa ser utilizada por algoritmos que manipulam valores numéricos. Aplicaremos a técnica **One Hot Encoding**. Veja o Bloco de Código 4.6.1.



### Bloco de Código 4.6.1

---

```

. . .
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
. . .
#One Hot Encoding feito na coluna 3
#passthrough: não mexer com as demais
transformer = ColumnTransformer(transformers=[('encoder',
                                             OneHotEncoder(), [3])], remainder='passthrough')
independent =
    np.array((transformer.fit_transform(independent)))
#exiba com e sem a opção remainder=passthrough para ver o
    resultado
print (independent)

```

---

**4.7 Dados para treinamento e para teste** Como de costume, utilizamos um percentual significativo das instâncias na fase de treinamento e reservamos o restante para a fase de teste. O Bloco de Código 4.7.1 utiliza a função `train_test_split` para esse fim.

### Bloco de Código 4.7.1

---

```

. . .
from sklearn.model_selection import train_test_split
. . .
#test_size=0.2: 20 %das instâncias para teste
#random_state=0: resultados diferentes a cada execução
ind_train, ind_test, dep_train, dep_test =
    train_test_split(independent, dependent,
                    test_size=0.2, random_state=0)

```

---

**4.8 Realizando o treinamento** O treinamento para a Regressão Linear Múltipla será realizado pela mesma classe que utilizamos para a Regressão Linear Simples: `LinearRegression`.

---

**Nota.** As bibliotecas que estamos utilizando se encarregam de resolver problemas comuns encontrados na análise de dados. Não precisamos nos preocupar em remover uma das variáveis geradas em função da variável categórica pois isto será feito automaticamente.

---

Veja o Bloco de Código 4.8.1.

## Bloco de Código 4.8.1

---

```

. . .
from sklearn.linear_model import LinearRegression
. . .
#calcula os coeficientes
#efetivamente realiza o treinamento
linearRegression.fit(ind_train, dep_train)

```

---

**4.9 Predição para a base de teste** Uma vez feito o treinamento, realizamos a predição para as instâncias existentes na base de teste. Veja o Bloco de Código 4.9.1.

## Bloco de Código 4.9.1

---

```

#a partir das variáveis independentes de teste
#obtemos as predições para a variável dependente
#para cada instância de teste
dep_pred = linearRegression.predict(ind_test)

```

---

**4.10 Comparando valores obtidos com valores reais** Exibiremos dois vetores para fins de comparação: o vetor de valores reais, existentes na base de teste e o vetor de valores preditos, obtidos pelo algoritmo. O Bloco de Código 4.10.1.

## Bloco de Código 4.10.1

---

```

#somente duas casas decimais
np.set_printoptions(precision=2)
#exibindo os dois vetores concatenados
#reshape para exibir o vetor de len(dep_pred) linhas e 1
coluna
#reshape para exibir o vetor de len(dep_test) linhas e 1
coluna
#axis=0: concatenacao vertical
#axis=1: concatenacao horizontal
#note que o primeiro parâmetro de concatenate é uma tupla de
vetores
print (np.concatenate((dep_pred.reshape(len(dep_pred), 1),
                        dep_test.reshape(len(dep_pred), 1) ), axis=1))

```

---

**4.11 Realizando a predição para uma única instância** O método predict pode ser utilizado para realizar a predição de valores para uma quantidade arbitrária de instâncias. O Bloco de Código 4.11.1 mostra como fazê-lo para uma única

instância.

Bloco de Código 4.11.1

---

```
print (linearRegression.predict([[1, 0, 0, 120000, 100000,
15000]]))
```

---

**4.12 Exibindo os coeficientes** Os coeficientes obtidos se encontram nas variáveis `coef_` e `intercept_`. Veja o Bloco de Código 4.12.1.

Bloco de Código 4.12.1

---

```
for c in linearRegression.coef_:
    print (f'{c:.2f} ')
print (f'{linearRegression.intercept_:.2f}')
```

---

**4.13 Exercícios** Faça o download da base de dados disponível no Link 4.13.1.

Link 4.13.1

<https://www.kaggle.com/jsphyg/weather-dataset-rattle-package>

Escolha cinco variáveis e as utilize como variáveis independentes para a construção de um modelo de Regressão Linear Múltipla. A predição deve ter como alvo a variável “RainTomorrow”.

## Capítulo 5

# Regressão Linear Polinomial

O modelo conhecido como Regressão Polinomial admite trabalharmos com relações não necessariamente lineares. Veja a Equação 5.0.1. Embora o polinômio possa ter grau  $k > 1$ , ainda se trata de um modelo linear pois estamos preocupados em expressar o valor de  $x$  em função dos coeficientes  $w_i$ .

$$x = w_0 + w_1 a_1 + w_2 a_1^2 + \dots + w_k a_1^k \quad (5.0.1)$$

**5.1 As bibliotecas** Importe as bibliotecas como mostra o Bloco de Código 5.1.1.

Bloco de Código 5.1.1

---

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

---

**5.2 A base de dados** Obtenha uma cópia da base de dados com que trabalharemos no ambiente da disciplina. Ela é exibida na Figura 5.2.1.

Figura 5.2.1

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Ela pode ser importada como mostra o Bloco de Código 5.2.1. Note que iremos deixar de utilizar a variável categórica.

#### Bloco de Código 5.2.1

---

```
dataset = pd.read_csv('dados.csv')
ind = dataset.iloc[:, 1:-1].values
dep = dataset.iloc[:, -1].values
```

---

**5.3 Regressão Linear Simples para comparação** Iremos obter um modelo deregressão linear simples e, a seguir, um modelo de regressão linear polinomial a fim de comparar seus resultados. Comece construindo o primeiro, como no Bloco de Código 5.3.1.

#### Bloco de Código 5.3.1

---

```
. . .
from sklearn.linear_model import LinearRegression
. . .
linearRegression = LinearRegression()
linearRegression.fit(ind, dep)
```

---

**5.4 Treinamento o modelo de regressão linear polinomial** O primeiro passo para obter um modelo de regressão linear polinomial é aplicar uma transformação à variável independente. Isso pode ser feito com a classe `PolynomialFeatures` da `scikit-learn`. Quando o fazemos, escolhemos o grau do polinômio desejado. A seguir, construímos o modelo, o que pode ser feito usando a classe `LinearRegression`. Veja o Bloco de Código 5.4.1.

#### Bloco de Código 5.4.1

---

```
. . .
from sklearn.preprocessing import PolynomialFeatures
. . .
poly_features = PolynomialFeatures (degree= 2)
ind_poly = poly_features.fit_transform(ind)
polyLinearRegression = LinearRegression()
polyLinearRegression.fit(ind_poly, dep)
```

---

### 5.5 Visualizando os resultados com o modelo de regressão linear simples

O Bloco de Código 5.5.1 mostra a construção do gráfico em que os resultados obtidos com a regressão linear simples podem ser visualizados.

## Bloco de Código 5.5.1

---

```
plt.scatter(ind, dep, color="red")
plt.plot(ind, linearRegression.predict(ind), color="blue")
plt.title("Regressão Linear Simples")
plt.xlabel("Nível")
plt.ylabel("Salário")
plt.show()
```

---

Note que a reta obtida é muito inapropriada para os dados sob análise.

**5.6 Visualizando os resultados com o modelo de regressão linear polinomial** O Bloco de Código 5.6.1 mostra a construção do gráfico em que os resultados obtidos com a regressão linear simples podem ser visualizados.

## Bloco de Código 5.6.1

---

```
plt.scatter(ind, dep, color="red")
plt.plot(ind, polyLinearRegression.predict(ind_poly),
         color="blue")
plt.title("Regressão Linear Polinomial")
plt.xlabel("Nível")
plt.ylabel("Salário")
plt.show()
```

---

## 5.7 Exercícios

**5.7.1** Use a base de dados disponível no Link 5.7.1 para obter modelos de regressão linear simples e polinomial.

Link 5.7.1

[https://www.kaggle.com/sakshamjn/  
heightvsweight-for-linear-polynomial-regression](https://www.kaggle.com/sakshamjn/heightvsweight-for-linear-polynomial-regression)

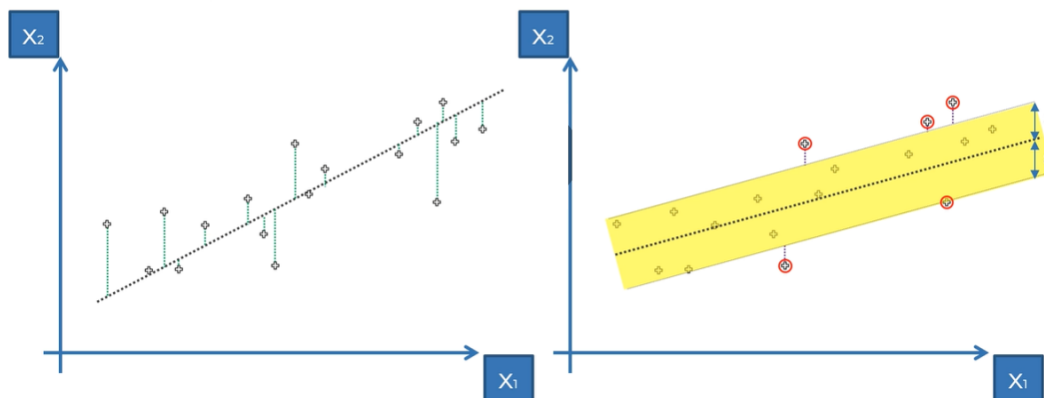
**5.7.2** Obtenha um novo modelo de regressão polinomial utilizando `degree=4`. Exiba o gráfico a seguir.

## Capítulo 6

# Regressão por Vetor de Suporte

**6.1 Intuição** A regressão por vetor de suporte se baseia na ideia de um “tubo de tolerância”. Especificamos um valor  $\epsilon$  que indica o erro que nos é aceitável. As instâncias “dentro do tubo” não são consideradas para o erro final do modelo. Veja a Figura 6.1.1. À esquerda ela mostra a ideia de uma regressão linear simples. À direita, o modelo de regressão por vetor de suporte é ilustrado.

Figura 6.1.1



**6.2 As bibliotecas** Importe as bibliotecas iniciais como feito no Bloco de Código 6.2.1.

Bloco de Código 6.2.1

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

**6.3 Importando os dados** Faça o download do arquivo de dados disponível no ambiente da disciplina. Ela é exibida na Figura 6.3.1.

Figura 6.3.1

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

A seguir, importe os dados usando a biblioteca `pandas`, como no Bloco de Código 6.3.1.

Bloco de Código 6.3.1

---

```
dataset = pd.read_csv('dados.csv')
#ignorando a variável categórica
ind = dataset.iloc[:, 1:-1].values
dep = dataset.iloc[:, -1].values
```

---



---

**Nota.** Para este exemplo, usaremos a base inteira na fase de treinamento.

---

**6.4 Ajustando o formato dos dados** As coleções `ind` e `dep` têm formatos diferentes. Vamos ajustar a coleção `dep` para que, assim como `ind`, ela passe a ser uma matriz de um único vetor na vertical. Veja o Bloco de Código 6.4.1.

Bloco de Código 6.4.1

---

```
#ind é uma matriz com um vetor na vertical
#transformando dep para que tenha o mesmo formato de ind
#len(dep) linhas, 1 coluna
dep = dep.reshape(len(dep), 1)
#agora ambas as coleções são uma matriz de um vetor na
#vertical
print ("dep:\n", dep)
print ("ind:\n", ind)
```

---



**6.5 Feature scaling** O intervalo de valores da variável independente é muito diferente do intervalo da variável dependente. Nestes casos é preciso aplicar alguma técnica de *feature scaling* para a variável dependente também. Caso contrário, o modelo irá ignorar as features cujos intervalos sejam muito diferentes de seu intervalo. Veja o Bloco de Código 6.5.1.

Bloco de Código 6.5.1

---

```
. . .
from sklearn.preprocessing import StandardScaler
. . .
#objeto para feature scaling da variável independente
indScaler = StandardScaler()
ind = indScaler.fit_transform(ind)
#precisamos de outro scaler, caso contrário a média
# de ind faria parte das contas
depScaler = StandardScaler()
dep = depScaler.fit_transform(dep)
print ("dep:\n", dep)
print ("ind:\n", ind)
```

---

**6.6 Realizando o treinamento** Nesta seção realizaremos a construção do modelo de regressão por vetor de suporte. Este é um tipo específico de **Máquina de Vetor de Suporte**, um modelo mais geral sobre o qual trataremos adiante. O Bloco de Código 6.6.1 mostra que, na **scikit-learn** ele vem de um pacote cujo nome é **svm**, de **Support Vector Machine**. Este tipo de modelo requer a especificação de um **kernel**, que nada mais é do que uma função matemática que expressa o tipo de relação que desejamos encontrar (linear, polinomial etc). Para este exemplo, vamos usar uma função chamada **Radial Basis Function**. Ela se baseia na noção de **distância** entre instâncias da base de dados.

Bloco de Código 6.6.1

---

```
. . .
from sklearn.svm import SVR
. . .
#radial basis function
svr = SVR(kernel='rbf')
#treinamento ocorre aqui
svr.fit(ind, dep)
```

---

**6.7 Predição para uma instância** A fim de realizar a predição para uma nova instância, precisamos realizar os seguintes passos.

- Aplicar feature scaling à variável independente usando o objeto apropriado, construído em função da variável independente dos dados de treinamento.
- Realizar a predição.
- Remover o efeito do feature scaling do resultado obtido, usando o objeto apropriado, construído em função da variável dependente dos dados de treinamento.

Veja o Bloco de Código 6.7.1.

Bloco de Código 6.7.1

---

```
#predição para uma instância
instancia = [[6.5]]
#aplicar feature scaling
instancia_com_feature_scaling = indScaler.transform(instancia)
#predicao obtida com feature scaling
valor_predito = svr.predict(instancia_com_feature_scaling)
#voltar ao valor original, sem feature scaling
valor_predito_sem_feature_scaling =
depScaler.inverse_transform(valor_predito)
print(valor_predito_sem_feature_scaling)
```

---

Não deixe de comparar o resultado obtido com a base de dados utilizada no treinamento.

**6.8 Visualizando os resultados graficamente** Utilizaremos um gráfico de dispersão para visualizar as instâncias da base de treinamento. Além disso, o gráfico irá mostrar a curva de predição obtida pelo modelo. Veja o Bloco de Código 6.8.1.

### Bloco de Código 6.8.1


---

```
#gráfico de dispersão para visualizar os pontos
#removemos o efeito do feature scaling \
plt.scatter(indScaler.inverse_transform(ind),
            depScaler.inverse_transform(dep), color='red')
#curva da predição, remover o feature scaling
plt.plot(indScaler.inverse_transform(ind),
         depScaler.inverse_transform(svr.predict(ind)),
         color='blue')
plt.title("Regressão por Vetor de Suporte")
plt.xlabel('Nível')
plt.ylabel('Salário')
plt.show()
```

---

# Capítulo 7

## Regressão por árvore de decisão

O método de regressão por árvore de decisão consiste na construção de uma árvore em que cada nó é uma - como o nome do modelo sugere - regra de decisão obtida a partir da coleção de dados. Neste capítulo, veremos as técnicas mais comumente utilizadas para a sua construção e uma implementação em  python .

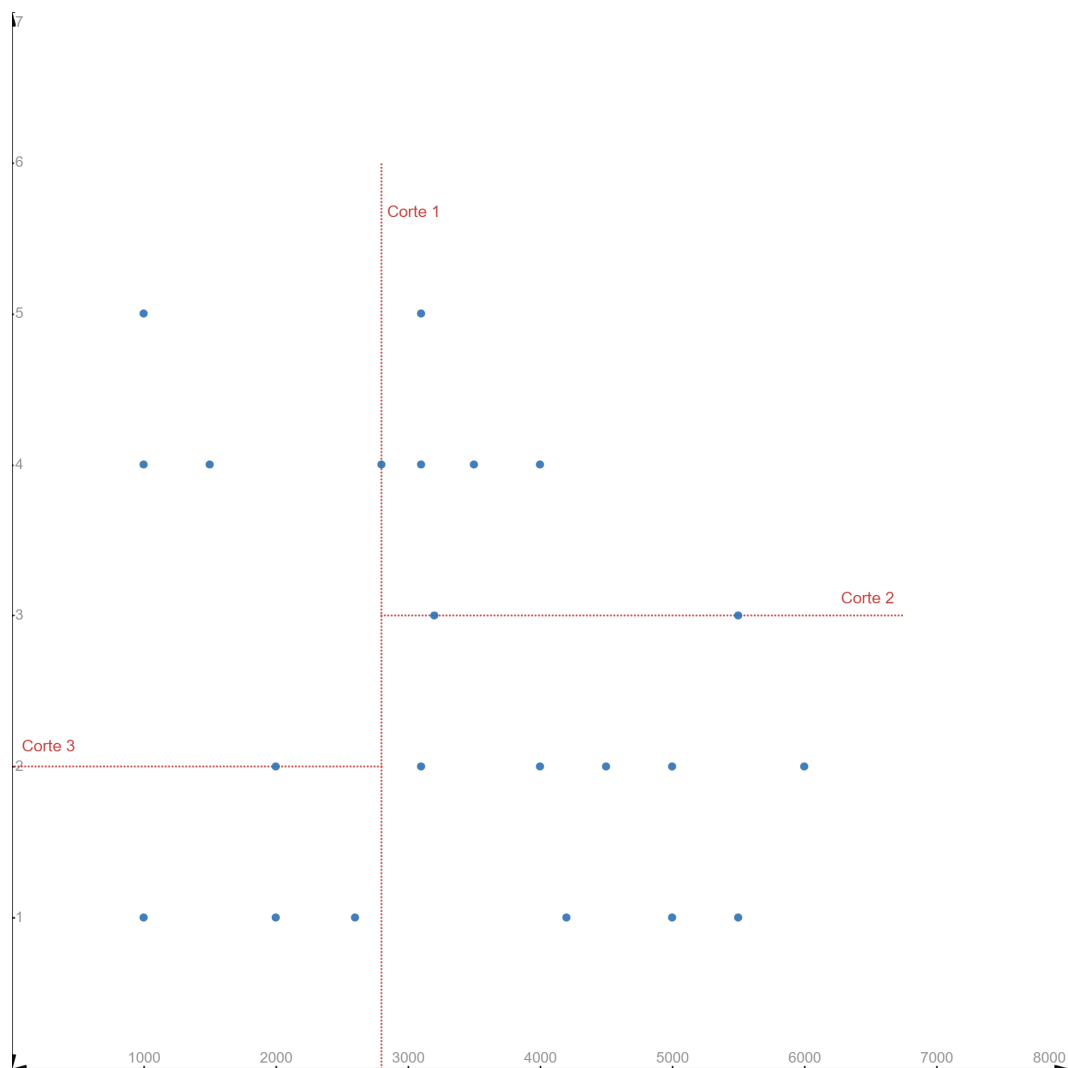
**7.1 Intuição** Para entender como se dá a construção de uma árvore de decisão, considere a base de dados exibida pela Figura 7.1.1.

Figura 7.1.1

Renda Familiar	Número de pessoas em casa	Gastos
1000	1	800
2000	1	1400
2600	1	1900
4200	1	4300
5000	1	4000
5500	1	4700
2000	2	2100
3100	2	3100
4000	2	3900
4500	2	4000
5000	2	5100
6000	2	4500
3200	3	4000
5500	3	5600
1000	4	1700
1500	4	2250
2800	4	3100
3100	4	3200
3500	4	3500
4000	4	3900
1000	5	2700
3100	5	4000

Suponha que desejamos explicar a variável *Gastos* em função das demais, ou seja, ela é a variável dependente. O gráfico da Figura 7.1.2 mostra as variáveis independentes e sugestões de **cortes** a serem realizados, obtendo sub-grupos de instâncias.

Figura 7.1.2



Os cortes representados por linhas horizontais e verticais indicam decisões envolvendo as variáveis **número de pessoas** e **gastos**, respectivamente. A Figura 7.1.3 mostra a árvore de decisão referente a tais cortes.

Figura 7.1.3



Uma vez que a árvore tenha sido obtida, a regressão é feita por meio do cálculo da média dos valores da variável dependente das instâncias de cada sub-grupo.

**7.2 Entropia** Uma das partes mais importantes na construção de uma árvore de decisão é a escolha dos atributos a serem envolvidos em cada regra de decisão. Ela se baseia em um conceito matemático conhecido como **entropia**[1]. Em uma árvore de decisão, cada nó tem a sua entropia. É um número que representa a “quantidade” de informação inerente àquele nó. Ele é **inversamente proporcional** à probabilidade de o evento representado por seu nó ocorrer. A Equação 7.2.1 mostra a sua definição formal. Considere que  $X$  é a variável cuja entropia desejamos calcular e  $x_i (i = 1, 2, \dots, n)$  são os valores que  $X$  pode assumir.

$$I(X) = - \sum_{i=1}^n P(x_i) \log_2 P(x_i) \quad (7.2.1)$$

---

#### Exemplo

---

Considere a variável “lançamento de um dado”. Há dois resultados possíveis: cara e coroa. Caso a moeda envolvida no experimento seja “honesta”, a probabilidade de cada resultado é igual:  $p(cara) = p(coroa) = \frac{1}{2}$ . Neste caso,

$$\begin{aligned} I(X) &= - \sum_{i=1}^2 P(x_i) \log_2 P(x_i) \\ &= - \left( \frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2} \right) \\ &= - \left( \frac{1}{2} \times -1 + \frac{1}{2} \times -1 \right) \\ &= - \left( -\frac{1}{2} + \left( -\frac{1}{2} \right) \right) = -(-1) = 1 \end{aligned}$$

Entretanto, não necessariamente ambos os resultados têm probabilidade igual. Pode ser que estejamos lidando com uma moeda viciada. Se  $p(cara) = 0,8$  - e portanto  $p(coroa) = 0,2$  - temos

$$\begin{aligned} I(X) &= - \sum_{i=1}^2 P(x_i) \log_2 P(x_i) \\ &= -(0,8 \log_2 0,8 + 0,2 \log_2 0,2) \\ &= -(0,8 \times (-0,32) + 0,2 \times (-2,32)) \\ &= -(-0,256 + (-0,46)) = -(-0,72) = 0,72 \end{aligned}$$


---

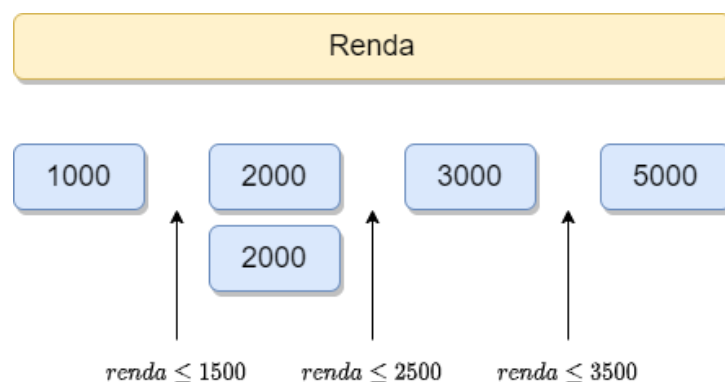
Como mais um exemplo, considere a coleção de dados da Figura 7.2.1.

Figura 7.2.1

Id	Renda	Pessoas	Gastos
a	1000	1	800
b	2000	1	1300
c	2000	3	2200
d	3000	5	4200
e	5000	4	5600

Como construir uma árvore de decisão para realizar regressão a partir desses dados? Começamos calculando a entropia associada a cada possível corte de cada variável. Aquele que resultar em menor entropia é escolhido. Para a variável **Renda**, temos três possíveis cortes, como ilustra a Figura 7.2.2.

Figura 7.2.2



As entropias de cada corte são as seguintes.

$$I(\text{Renda} \leq 1500) = -\left(\frac{1}{5} \log_2 \frac{1}{5} + \frac{4}{5} \log_2 \frac{4}{5}\right) = 0.72$$

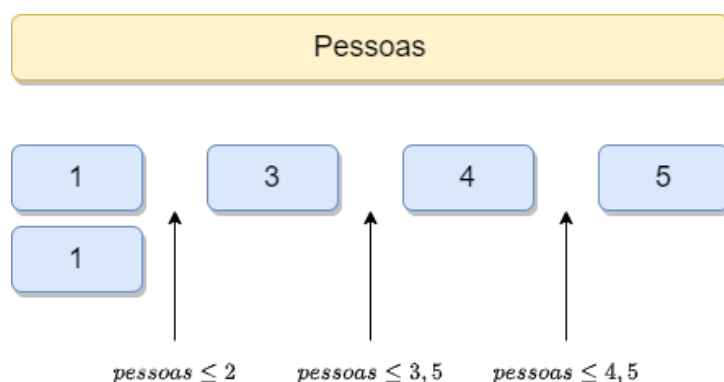
$$I(\text{Renda} \leq 2500) = -\left(\frac{3}{5} \log_2 \frac{3}{5} + \frac{2}{5} \log_2 \frac{2}{5}\right) = 0.97$$

$$I(\text{Renda} \leq 4000) = -\left(\frac{4}{5} \log_2 \frac{4}{5} + \frac{1}{5} \log_2 \frac{1}{5}\right) = 0.72$$

Para a variável **Pessoas** há também três possíveis cortes, como mostra a Figura 7.2.3.



Figura 7.2.3



As entropias de cada corte são as seguintes.

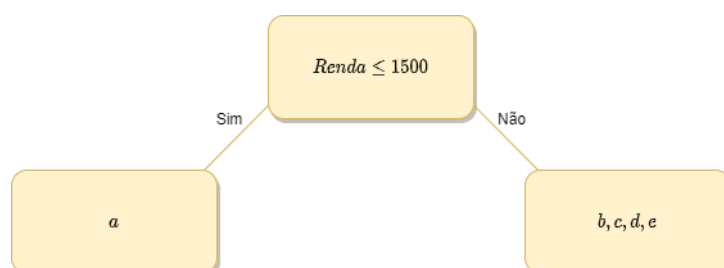
$$I(Pessoas \leq 2) = -\left(\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5}\right) = 0.97$$

$$I(Pessoas \leq 3,5) = -\left(\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5}\right) = 0.97$$

$$I(Pessoas \leq 4,5) = -\left(\frac{4}{5} \log_2 \frac{4}{5} + \frac{1}{5} \log_2 \frac{1}{5}\right) = 0.72$$

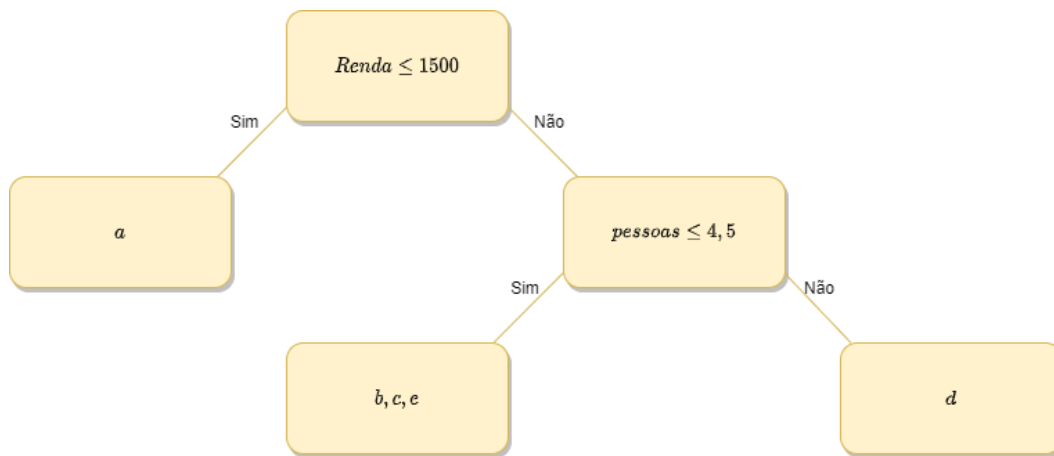
Para a primeira regra de decisão, todas aquelas de menor entropia (0.72) são candidatas. Escolhendo arbitrariamente uma delas, a árvore inicial é aquela dada pela Figura 7.2.4.

Figura 7.2.4



Prosseguindo de maneira análoga, podemos encontrar a árvore da Figura 7.2.5. O procedimento pode ser encerrado utilizando-se critérios diferentes. Podemos limitar a altura da árvore. Podemos também especificar um número de instâncias pertencentes a cada nó a partir do qual a construção já nos é suficiente.

Figura 7.2.5



Observe que **quanto menor for a entropia de um nó da árvore, maior a “quantidade” de informação inerente a ele**. A construção de uma árvore de decisão é feita recursivamente. A cada passo opta-se por utilizar a variável de menor entropia.

**7.3 As bibliotecas** Importe as bibliotecas como exibe o Bloco de Código 7.3.1.

Bloco de Código 7.3.1

---

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

---

**7.4 Importando os dados** A coleção de dados que utilizaremos é exibida na Figura 7.1.1. Faça o download do arquivo no ambiente da disciplina. O Bloco de Código 7.4.1 mostra como importar a base.

Bloco de Código 7.4.1

---

```
dataset = pd.read_csv('dados.csv')
ind = dataset.iloc[:, 0:-1].values
dep = dataset.iloc[:, -1].values
```

---

**7.5 Realizando o treinamento** O treinamento pode ser realizado utilizando a classe `DecisionTreeRegressor` da `scikit-learn`. Veja o Bloco de Código 7.5.1.

---

**Nota.** A construção de uma árvore de decisão se baseia em “cortes” envolvendo as variáveis independentes de interesse. Ao final, o cálculo para novas instâncias é a simples média dos valores das instâncias existentes no sub-grupo em que ela melhor se encaixar. Não há, portanto, uma expressão do tipo  $y = ax_1 + bx_2 + \dots + nx_n$  em que alguma variável  $x_i$  poderia dominar as demais. Isso quer dizer que **não será necessário realizar feature scaling**.

---

#### Bloco de Código 7.5.1

---

```
. . .
from sklearn.tree import DecisionTreeRegressor
. . .
#random_state = 0 (seed igual para todas as execuções garante
#resultado igual)
decisionTreeRegressor =
    DecisionTreeRegressor (random_state = 0)
decisionTreeRegressor.fit(ind, dep)
```

---

**7.6 Predição para nova instância** O Bloco de Código 7.6.1 mostra a predição para algumas novas instâncias.

#### Bloco de Código 7.6.1

---

```
. . .
print (decisionTreeRegressor.predict([[1000, 1]]))
print (decisionTreeRegressor.predict([[2000, 1]]))
print (decisionTreeRegressor.predict([[1000, 2]]))
print (decisionTreeRegressor.predict([[1000, 5]]))
print (decisionTreeRegressor.predict([[5000, 3]]))
. . .
```

---

**7.7 Visualizando os resultados graficamente** Podemos visualizar os resultados em duas dimensões, mantendo uma das variáveis independentes no eixo horizontal. Também podemos gerar uma visualização 3d. Veja o Bloco de Código 7.7.1.

## Bloco de Código 7.7.1

---

```

#renda vs gastos
plt.scatter(ind[:, 0], decisionTreeRegressor.predict(ind),
            color="red")
plt.xlabel('Renda')
plt.ylabel('Gastos')
plt.title('Renda vs Gastos')
plt.show()

#número pessoas vs gastos
plt.scatter(ind[:, 1], decisionTreeRegressor.predict(ind),
            color="red")
plt.xlabel('Pessoas')
plt.ylabel('Gastos')
plt.title('Pessoas vs Gastos')
plt.show()

#número pessoas vs renda vs gastos
fig = plt.figure()
subplot = fig.add_subplot(111, projection='3d')
subplot.scatter(ind[:, 0], ind[:, 1],
                decisionTreeRegressor.predict(ind), color="red")
plt.show()

```

---

**7.8 Exercícios** Obtenha a base de dados disponível no Link 7.8.1.

Link 7.8.1

<https://www.kaggle.com/uciml/>[red-wine-quality-cortez-et-al-2009?select=winequality-red.csv](https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009?select=winequality-red.csv)

Construa uma árvore de decisão para regressão. Considere que “quality” é a variável dependente. Exiba um gráfico de duas dimensões de “alcohol” versus “quality”. Exiba um gráfico de três dimensões de “alcohol” versus “density” versus “quality”.

# Referências

- [1] Claude E. Shannon. “A mathematical theory of communication.” Em: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. URL: <http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48>.