Metodologias de Aprendizado de Máquina

Deep Learning

Rafael Ris-Ala José Jardim

2021

Metodologia de Aprendizado de Máquina

Deep Learning

Rafael Ris-Ala José Jardim

Sumário

Capítulo 1. Metodologias de Aprendizado de Máquina

Aprendizado supervisionado e não supervisionado

O Aprendizado de Máquinas é uma linha de pesquisa da Inteligência Artificial, que visa criar programas capazes de aprender uma determinada tarefa aplicando um conjunto de dados ou medida de desempenho.

Ao invés de criar um programa determinando os passos para executar sua tarefa, no aprendizado de máquinas usamos algoritmos que aprendem uma tarefa conforme seu treinamento.

Modelo

O algoritmo treinado para aprender a executar uma tarefa, recebe o nome de Modelo, e segundo Sakurai (2021), um modelo pode ser treinado para prever:

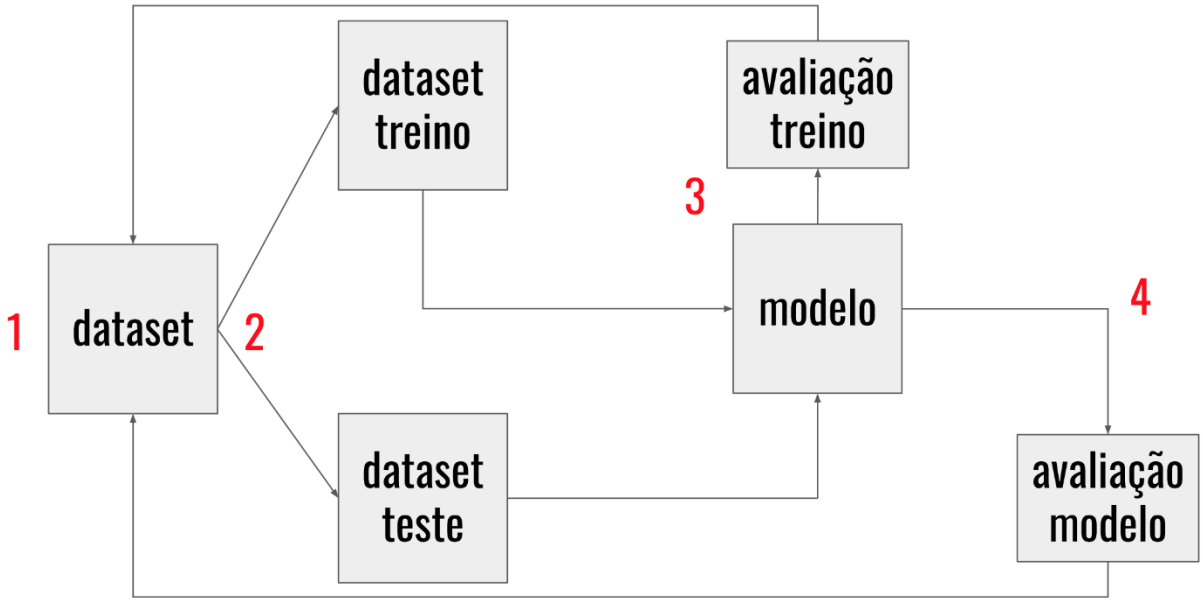
- Se um comentário é positivo, neutro ou negativo;
- Se uma compra no cartão de crédito é real ou fraude;
- Quais produtos podem ser oferecidos para um cliente com base no seu histórico de compras;
- Com base no histórico de sites acessados por um usuário, quais propagandas fará o usuário clicar e comprar o produto anunciado;
- Qual o próximo movimento deve ser feito para aumentar as chances de vencer um jogo.

O treinamento dos modelos geralmente é feito utilizando uma das seguintes abordagens:

- Aprendizado supervisionado;
- Aprendizado não supervisionado;
- Aprendizado semi-supervisionado;
- Aprendizado por reforço.

Na Figura 1 temos a representação de um fluxo resumido do funcionamento, de modo geral, do treinamento de um modelo:

Figura 1 - Aprendizado a partir dos dados



Fonte: Rafael Sakurai, 2021.

- **Passo 1.** Dado seu conjunto de dados (dataset) já coletado, a primeira parte que precisa ser analisada é quais características (variáveis) devem ser utilizadas; aqui também ocorre um tratamento dos dados para padronizar o seu formato, preencher as informações que estão faltando, gerar novas características a partir das características iniciais, remover dados inválidos (ou outliers, normalmente informações que atrapalham o aprendizado do modelo), se houver um valor para a saída esperada (também conhecido como rótulo ou classe), padronizar esses valores (por exemplo: definir todas as saídas como numéricas e em ordem crescente), etc.;
- **Passo 2.** O dataset é dividido normalmente em duas partes: uma parte (algo em torno de 70% a 80% do dataset) é usada para realizar o treino do modelo e a segunda parte (que o modelo não conheceu durante o treinamento), é usada para avaliar a tarefa que o modelo aprendeu;
- **Passo 3.** A escolha do modelo (Decision Tree, kNN, Naive Bayes, SVM, Redes Neurais etc.) ocorrerá de acordo com o dataset e objetivo (classificação, regressão, agrupamento etc.). Nesse ponto ocorre o treinamento do modelo e normalmente fazemos uma avaliação cruzada com os dados de treino e se

necessário voltamos ao Passo 1 para tentar melhorar as características usadas no treino;

- **Passo 4.** Depois que realizamos o treino e com os dados obtemos um resultado satisfatório na tarefa que o modelo deve aprender, passamos para o passo de teste do modelo, utilizando dados que são desconhecidos por ele e nesse ponto tentamos obter um resultado mais real de qual será o comportamento do modelo quando for disponibilizado em produção. Se os resultados não forem satisfatórios, normalmente voltamos ao Passo 1 e começamos todo o processo novamente para obter uma acurácia melhor. (SAKURAI, 2021, online).

Aprendizado supervisionado

Neste tipo de treinamento, os dados (também conhecidos como amostras ou exemplos) já estão rotulados e, portanto, existem informações sobre um exemplo e sua representação (rótulo / classe).

Utilizando um dataset, um modelo é treinado com a finalidade de identificar padrões nas características que melhor especificam os rótulos de cada exemplo. Após o treinamento espera-se que o modelo realize o reconhecimento das características de um novo conjunto de dados e rotule-os da melhor forma possível. Quanto maior a variabilidade do dataset de treinamento, mais generalizado será o treinamento, podendo prever com mais acurácia novos conjuntos de dados. (SAKURAI, 2021, online).

O canal Zurubabel explica nesse vídeo o que é Aprendizado supervisionado.

Aprendizado não supervisionado

No aprendizado não supervisionado o dataset não está rotulado, sendo assim o aprendizado é feito nos padrões identificados e uma medida de avaliação informa se a tarefa está sendo feita da melhor maneira possível. A tarefa mais habitual é o agrupamento (clustering).

Aprendizado semi-supervisionado

No aprendizado semi-supervisionado parte do dataset está rotulado, portanto procura-se generalizar o treinamento da melhor maneira possível com estes dados para tentar rotular os demais.

Aprendizado por reforço

O aprendizado por reforço consiste em um treinamento onde o modelo é avaliado a cada execução de tarefa e recebe um reforço positivo (quando a tarefa é realizada de maneira igual ou similar ao esperado) ou reforço negativo (quando realiza algo de maneira errada).

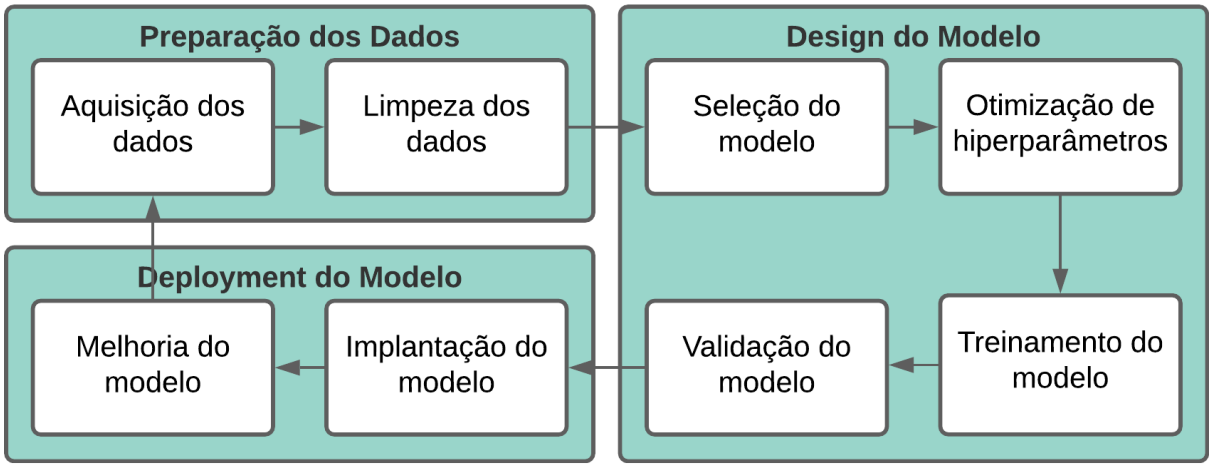
Guilherme de Lazari explica neste vídeo o que é Aprendizado por reforço e apresentando alguns exemplos.

Workflow de Data Science

A experimentação em Ciência de Dados (*Data Science*) compreende a realização de estudos científicos para se aprender com os dados, conforme discutido por David Donoho (2017). O fluxo de trabalho realizado é apresentado na Figura 2, uma adaptação do workflow de Ciência de Dados de Dakuo Wang (et al., 2019).

Figura 2: Fluxo de trabalho de Ciência de Dados executado na realização deste projeto.

Fonte: Prof. Rafael Jardim



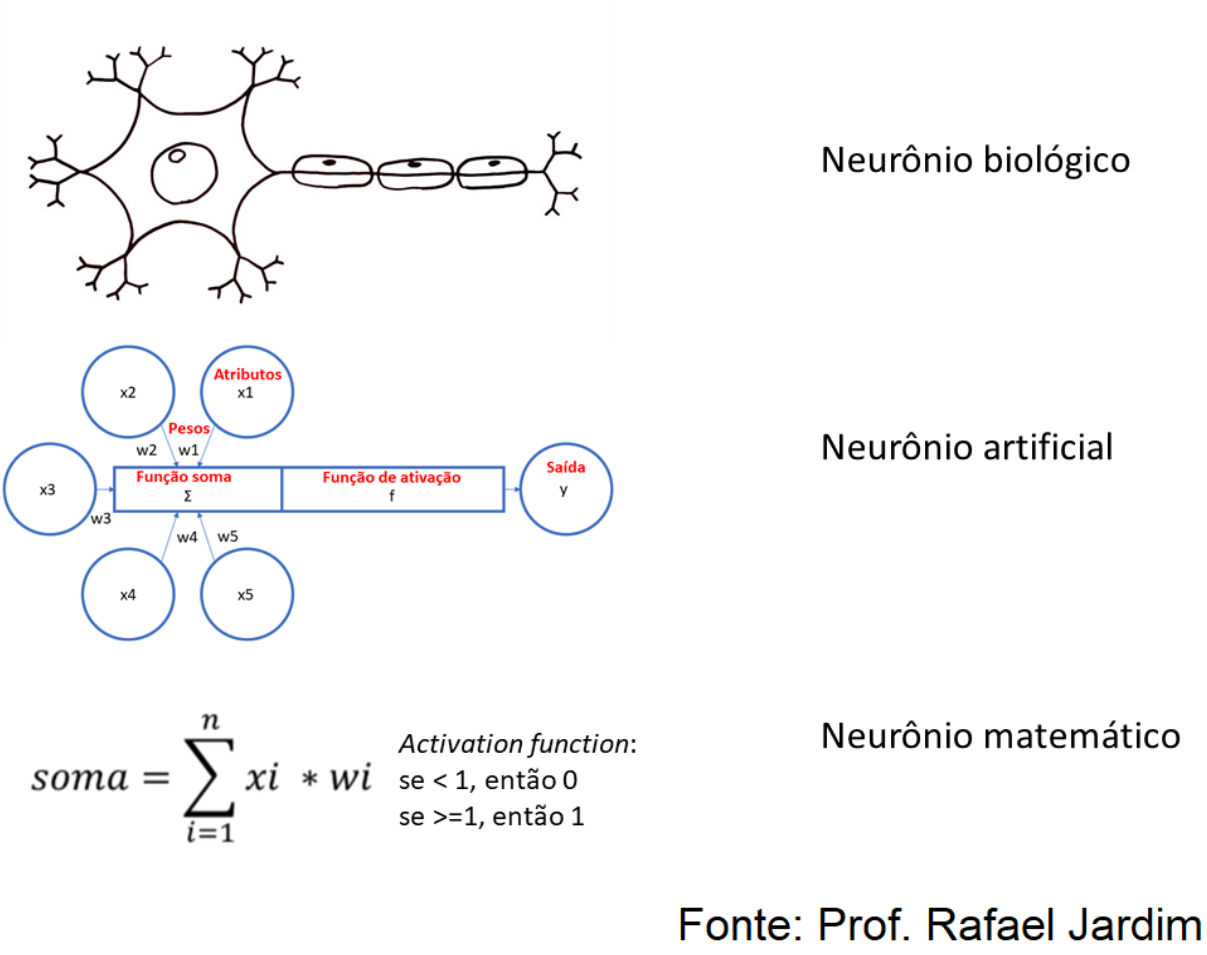
Fonte: Dakuo Wang et al., 2019

Capítulo 2. Algoritmos de Redes Neurais Artificiais

Perceptron

Observe a comparação entre os neurônios na Figura 3.

Figura 3 - Comparação entre o neurônio biológico, neurônio artificial e o neurônio matemático.



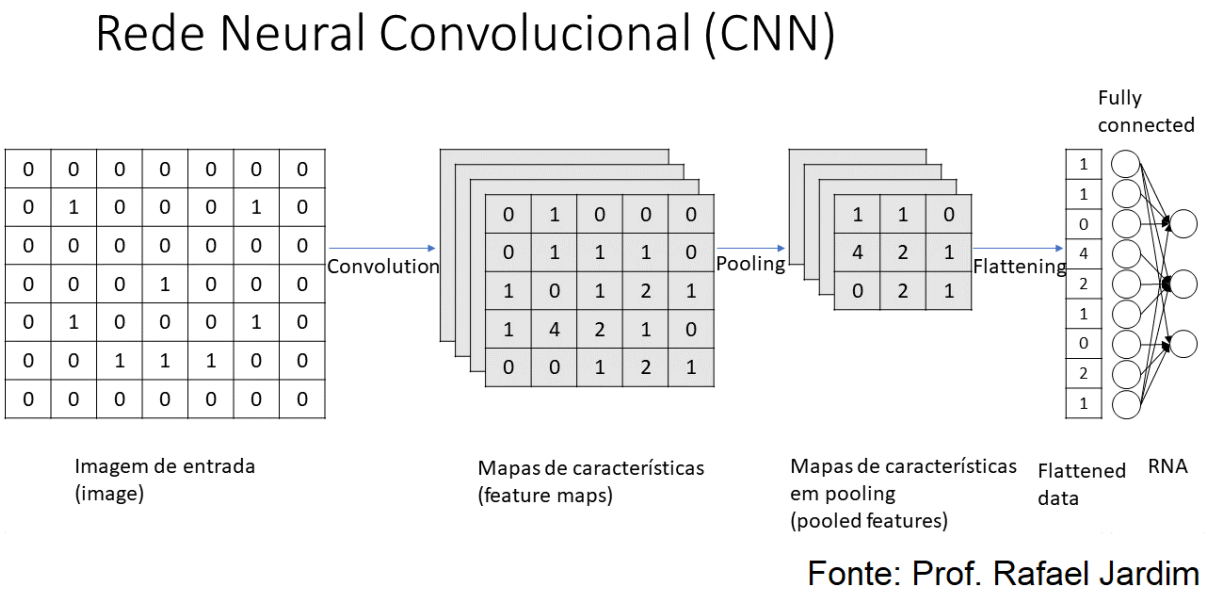
Fonte: Elaborado pelo autor

As Redes Neurais Convolucionais (CNN) são modelos de Redes Neurais comumente utilizados para a classificação de imagens. Tem-se uma imagem de entrada, que é submetida à CNN e que faz uma classificação desta imagem a partir da estimativa da probabilidade de pertencimento a uma classe ou a outra.

Em um processo de aprendizado supervisionado, imagens previamente etiquetadas com as respectivas classes são passadas à CNN para reconhecer, aprender os padrões nessa imagem e equalizar seus parâmetros a fim de acertar a classe. Posteriormente, em uma fase de teste, novas imagens sem as respectivas classes são passadas à CNN, que tenta determinar a classe das imagens (LECUN et al., 1998).

As CNN passam por 4 etapas: *Convolution*, *Pooling*, *Flattening* e *Full connected*, apresentado na Figura 4.

Figura 4: Arquitetura da Rede Neural Convolucional com detalhes em todas as etapas.



Fonte: Elaborado pelo autor

As imagens do mundo real são não-lineares, ou seja, não seguem um padrão na mesma direção. Não existe uma linearidade, os objetos estão misturados (e possuem bordas). Quando se utiliza uma CNN, corre-se o risco de gerar muita linearidade. O propósito de se utilizar a função de ativação *Rectified Linear Unit* (ReLU) é diminuir a linearidade gerada pela convolução, removendo-se os valores negativos.

O objetivo é destacar as características mais importantes. Existem vários tipos de pooling, sendo o "*max pooling*" o mais usual. Nesse caso, é estabelecido um quadro que percorre o "mapa de característica", registrando o maior valor dentro desse quadro em um "mapa de característica em *pooling*". Isso permite preservar as características mais importantes dos objetos, reduzir o tamanho dos dados originais e, consequentemente, reduzir a quantidade de dados que 'serão introduzidos na Rede Neural.

Na etapa de *flattening* (achatamento), a matriz do "mapa de características em *pooling*" é transformada em um vetor. Esses valores serão as entradas da Rede Neural Artificial. Observe que todo o trabalho até agora foi para realizar um tratamento do dado a ser introduzido em uma RNA. Até poderia ser passada uma outra técnica para o processamento dos dados obtidos, mas a RNA tem uma capacidade muito boa de determinar as características relevantes na classificação das imagens.

Nessa etapa é que as redes neurais são treinadas. Em CNN, essa RNA é totalmente conectada (*fully connected*), o que significa que todas os neurônios da última camada estão conectados à camada de saída. Nem sempre uma RNA é totalmente conectada, porém, no caso das CNN, elas têm essa característica. Os dados obtidos na etapa de *flattening* são passados para a camada de entrada da RNA e processados pelas camadas ocultas, para equacionar os pesos de cada atributo. A camada de saída, por sua vez, determina a probabilidade de a imagem passada pertencer a cada umas das classes.

Conforme apresentado, as CNN foram tradicionalmente desenvolvidas para trabalharem com imagens. De forma semelhante, houve um aprimoramento para sua aplicação em textos. A diferença fica por conta de que a matriz de entrada ao invés

de ser uma imagem é o texto, onde a sequência de palavras da frase são as linhas e os valores das dimensões dos vetores da palavra são as colunas.

Capítulo 3. Python para Aprendizado de Máquina

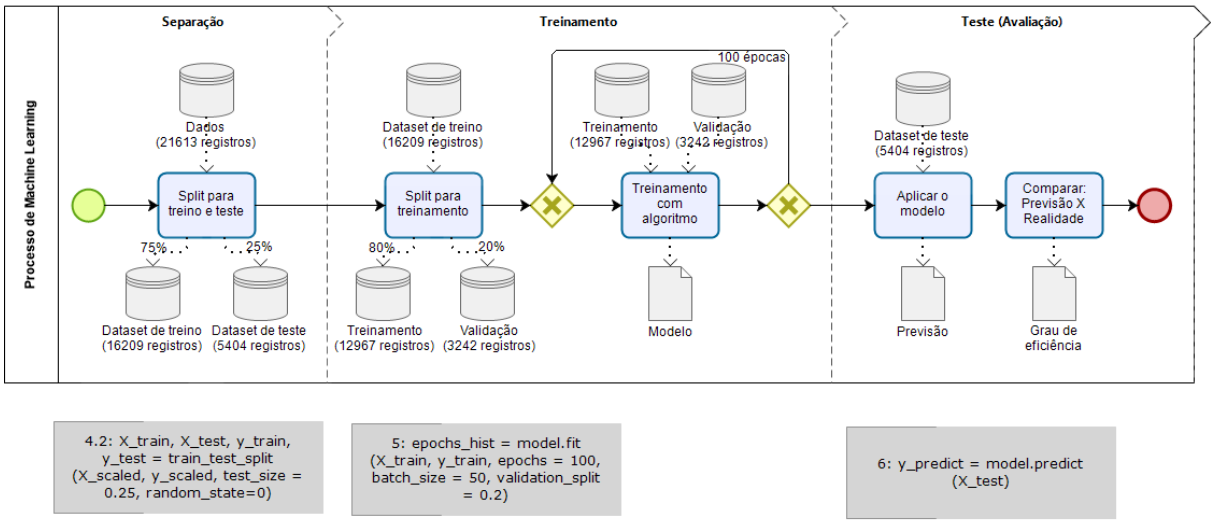
A Figura 5 ilustra o processo de Aprendizado de Máquina realizada na atividade prática:

- (https://colab.research.google.com/drive/1BzkEaSKKIbyaWN5ER7s0uYh17EOh7onS).

O momento das divisões dos dataset podem variar, podendo ocorrer por holdout ou cross-validation. Observe que abaixo da figura há o código referente à forma com que os dados foram separados.

Figura 5 – Processo de Aprendizado de Máquina

Fonte: Prof. Rafael Jardim



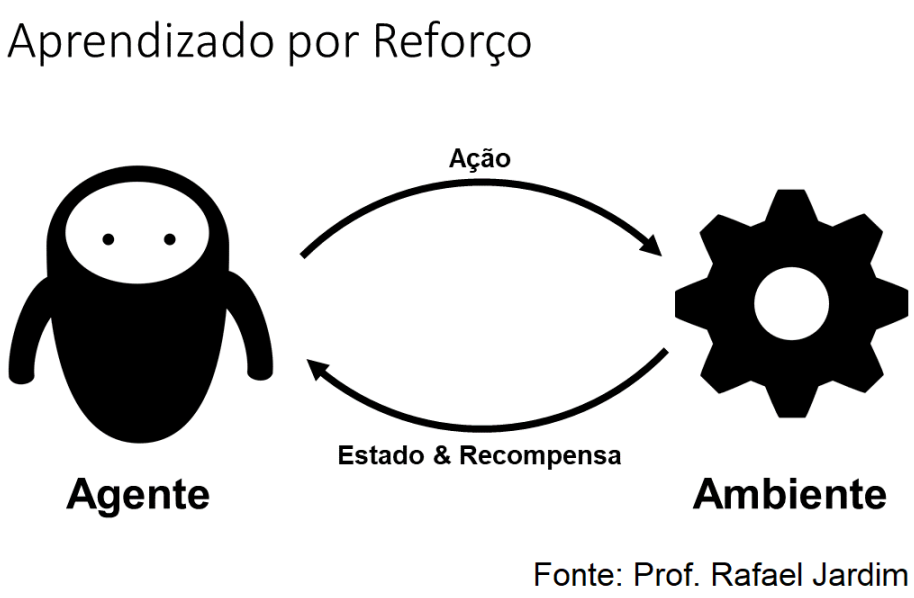
Fonte: Elaborado pelo autor

Capítulo 4. Aprendizado de Máquina por Reforço

De forma geral, na Aprendizagem de Máquina (Machine Learning) o aprendizado ocorre através de exemplos, ou seja, exemplos são passados e há o reconhecimento de padrões. Já o Aprendizado por Reforço (*Reinforcement Learning*) vai além, pois assume-se que não existe exemplos, ou seja, assume-se que o agente (sistema) poderá interagir com o ambiente e aprender por meio de recompensas e prejuízos.

Isso acontece de forma semelhante ao treinamento de um cachorro que ganha um biscoito quando cumpre um comando, reforçando um comportamento. A Figura 6 ilustra o Aprendizado por Reforço.

Figura 6 – Aprendizado por Reforço



Fonte: Elaborado pelo autor

Desta forma, a ideia do aprendizado por reforço é desenvolver um programa que poderá interagir com o ambiente, tentar coisas arbitrárias e que quando fizer a coisa certa receberá uma recompensa. Um exemplo muito utilizado é em jogos no qual o personagem é penalizado quando cai em um buraco ou recompensado quando pega uma moeda.

Capítulo 5. Manipulação de Dados em Python

Funções e recursividades

- **Função: def**

```
# def: definição de uma função
def primeiraFunc():
      print('Hello World')

primeiraFunc()
```

```
Hello World
```

```
# Segunda função
def segundaFunc(argumento):
    print('Hello %s' %(argumento))

segundaFunc('Bruna')
```

```
Hello Bruna
```

```
# Terceira função:
def terceiraFunc():
    nome = input('Escreva seu nome: ')
    print('Entendi, seu nome é ' + nome)

terceiraFunc()
```

```
Escreva seu nome: Rafael
Entendi, seu nome é Rafael
```

```
# Definindo uma função que verifica a letra "R"
def temLetraR():
    frase = input('Escreva seu nome: ')
    if 'R' in frase:
        print('Você utilizou a letra R!')
    else:
        print('Você não utilizou a letra R!')
temLetraR()
```

```
Escreva seu nome: Bruna
Você não utilizou a letra R!
```

```
# Definindo uma função que soma o quadrado de dua variáveis
def somaQuadrados(a,b):
    somaQ = a**2 + b**2
    return soma

somaQuadrados(3,3)
```

```
18
```

- **Biblioteca - Módulo - Função**

```
# Importo a biblioteca assim:
import math
# E uso uma função dessa biblioteca assim:
math.factorial(5)
```

```
120
```

```
# Observe que posso importar uma biblioteca inteira para utilizá-la:
import datetime
# E utilizá-la de acordo com a seguinte estrutura: biblioteca.módulo.função()
datetime.date.today()
```

```
datetime.date(2021, 9, 15)
```

```
# Ou eu posso importar somente o módulo dentro da biblioteca:
from datetime import date
# Também na hora de usar, posso ser mais direto:
date.today()
```

```
datetime.date(2021, 9, 15)
```

- **Recursividade**

Recursividade é um método de simplificação que consiste em dividir um problema em subproblemas do mesmo tipo. Os subproblemas são resolvidos e seus resultados combinados. Uma função é recursiva quando ela chama a si própria!

1. Exemplo de recursividade do cálculo de fatorial

O fatorial de um número corresponde ao produto desse número pelo fatorial de seu número anterior. Assim, temos a fórmula:

fatorial(num) = num X fatorial(num -1)

sendo que, o fatorial de 0 e de 1 é1. Ou: n! = n.(n –1), sendo que, o fatorial de 0 e de 1 é 1.

- Representação descritiva: Fatorial de 5
- Representação matemática: 5!
- Representação lógica: 5 x 4 x 3 x 2 x 1 = 120

```
# Construir a função para se calcular o fatorial
def fatorial(num):
    if num == 0 or num == 1:
        return 1
    else:
        return num * fatorial(num - 1)

# Construir variável para receber o número passado pelo usuário
x = int(input("Digite um número para calcular seu fatorial: "))
# Executar a função e exibir o resultado:
resposta = fatorial(x)
print(resposta)
```

```
Digite um número para calcular seu fatorial: 5
120
```

```
# Função recursiva do cálculo fatorial em Python
def fatorial(num):
    if num == 0 or num == 1:
        return 1
    else:
        return num * fatorial(num - 1)
x = int(input("Digite um número para calcular seu fatorial: "))
resposta = fatorial(x)
print("O fatorial de %d é %d" % (x, resposta))
```

```
Digite um número para calcular seu fatorial: 5
O fatorial de 5 é 120
```

2. Exemplo de recursividade do cálculo de Fibonacci

O cálculo de Fibonacci também é uma operação recursiva. Ele corresponde à soma dos dois valores anteriores.

Ex: 0, 1, 1, 2, 3, 5, 8, 13, 21...

Assim, temos a fórmula:

F(num) = F(num -1) + F(num -2), para (num >1)

Qual o 4º número Fibonacci?

f(4) = f(3) + f(2) = 2 + 1 = 3

f(3) = f(2) + f(1) = 1 + 1 = 2

f(2) = f(1) + f(0) = 1 + 0 = 1

f(1) = 1

f(0) = 0

```
# Função recursiva do cálculo de Fibonacci:
def fibonacci(num):
#    if num == 0 or num == 1:
#        return num
    if num <= 1:
        return num
    else:
        return fibonacci(num-1) + fibonacci(num-2)
x = int(input("Você quer saber o número Fibonacci na posição: "))
resposta = fibonacci(x)
print("O Fibonacci na posição %d é %d" % (x,resposta))
```

```
Você quer saber o número Fibonacci na posição: 4
O Fibonacci na posição 4 é 3
```

Armazenamento: Lista [ ]

Listas: conjunto de dados mutáveis (pode ter elementos alterados)

```
lista = ['banana', 'amora', 'damasco', 'carambola']
print(lista)
```

```
['banana', 'amora', 'damasco', 'carambola']
```

Armazenamento: Tupla ( )

Tuplas: conjunto de dados imutáveis (não pode ter elementos alterados e usado para estruturas que não devem ser modificadas)

Ex.: dias da semana ou datas de um calendário.

```
tupla = ('segunda', 'terça', 'quarta', 'quinta', 'sexta')
print(tupla.index("sexta"))
```

```
4
```

Armazenamento: Dicionário { }

Dicionário é um mapa de associações

dicionário = {"chave":valor}

```
dicionario = {'Amanda': 18, 'Bruno': 22, 'Carlos': 19}
print(dicionario['Amanda'])
```

```
18
```

Armazenamento: Conjunto { }

Set são como os conjuntos da matemática:

- não possui ordenação nem índice;
- não permitem valores duplicados.

```
conjunto = set(['lápis', 'borracha', 'caneta', 'lápis'])
print(conjunto)
```

```
{'borracha', 'caneta', 'lápis'}
```

Strings e Operações

Tipos de variáveis:

- Interger: número inteiro. Ex.: 123
- Float: número fracionado. Ex.: 123.5
- String: texto. Ex.: palavra
- Boolean true: predicado verdadeiro. Ex.: True
- Boolean false: predicado falso. Ex.: False
- None: sem valor, é o valor padrão para variáveis não iniciadas. Ex.: None
- Para verificar o tipo de uma variável, execute: type(a)

```
# Criando uma variável
# Obs: Use tralha para adicionar um comentário que não é executado
"""
Use 3 aspas duplas para
comentar váras linhas
"""

a = 10
print(a)
```

```
10
```

Operação aritmética: +, -, * , **, / e %

```
# Adição
a = 2
b = 3

soma = a + b

print(soma)
```

```
5
```

Operação relacional: ==, !=, <, <=, >, >=

```
# Igual
a = 2
b = 3

print(a == b)
```

```
False
```

Operação lógica: and, or e not

```
# and: verifica se as duas condições são verdadeiras
a = 3
b = 3
c = 7

print(a == b and b == c)
```

```
False
```

Manipulação de Arquivos

Os objetos "file" contém métodos e atributos úteis para manipular arquivos de texto.

Sintaxe: manipulador = open(arquivo, modo)

Modo:

- w: escrita
- r: leitura
- a: adição

```
# Criar o arquivo
escrita = open("arquivo.txt", "w")
escrita.write("Este é o conteúdo do arquivo")
escrita.close()
```

Métodos de leitura:

- read(): Lê o arquivo inteiro
- readline(): Lê uma linha
- readlines(): Lê o arquivo e o armazena em uma lista

```
# Ler o arquivo
escrita = open("arquivo.txt", "r")
escritaCompleta = escrita.read()
print(escritaCompleta)
```

```
Este é o conteúdo do arquivo
```

Orientação a objetos: classes

Orientação a Objeto

Uma classe é representada por atributos e métodos. Observe como verificar atributos e métodos de um objeto:

- objeto.atributo
- objeto.método(parâmetro)

```
lista = [1,2,3,4,5]
print(lista)
```

```
[1, 2, 3, 4, 5]
```

```
# Exibir todos os métodos e atributos disponíveis de um objeto
dir(lista)
```

```
...
```

```
lista.remove(3)
print(lista)
[1, 2, 4, 5]


# help: demonstra como utiliar um método de um objeto
# help(objeto.método)
help(lista.remove)
```

```
Help on built-in function remove:

remove(value, /) method of builtins.list instance
    Remove first occurrence of value.

    Raises ValueError if the value is not present.
```

Classes

```
Classes representam conceitos. Classifica entidades que tenham propriedades
similares.

Classe define objetos que possuem os mesmos atributos e operações.

Classe: pessoa

Objeto: Rafael
```

Construindo uma classe

```
Métodos são funções dentro de classes. Atributos são variáveis dentro das
classes.

Métodos representam o comportamento. Atributos representam o estado.

Obs.: Qnd se instancia uma classe por meio de um objeto os "métodos" são
compartilhados com todos os objetos criados a partir da mesma classe. Mas os
"atributos" não são compartilhados com os outros objetos criados a partir da
mesma classe.
def teste(v, i):                    # definindo uma função
    valor = v
    incremento = i
    resultado = valor + incremento
    return resultado

teste(2, 2)

# Observe que a variável resultado só existe dentro da função
resultado
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-6212350f06e5> in <module>()
      1 # Observe que a variável resultado só existe dentro da função
----> 2 resultado

NameError: name 'resultado' is not defined

# Vamos transformar em uma classe:
class Didatica:
    def teste(self, v, i):              # A única diferença foi q add "self"
        valor = v
        incremento = i
        resultado = valor + incremento
        return resultado

Didatica()
```

```
<__main__.Didatica at 0x7f3825865f90>
```

Tratamento de exceções

Tratamento de exceções

- SyntaxError: a sintaxe digitada é inválida e não reconhecida pelo interpretador
- TypeError: os tipos de dados são incompatíveis

- IndexError: o índice está fora da coleção
- ZeroDivisionError: o erro foi causado pela divisão por zero
- NameError: a variável não foi definida
- RuntimeError: ocorreu um erro de execução

1) Exemplo SyntaxError

```
1 = 2
```

```
File "<ipython-input-1-c0ab9e3898ea>", line 1
1 = 2
^
SyntaxError: can't assign to literal
```

2) Exemplo TypeError

```
10/"laranja"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-f5cf184247e7> in <module>()
----> 1 10/"laranja"

TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

3) Exemplo IndexError

```
lista = [1, 2, 3, 4, 5]
lista
```

```
[1, 2, 3, 4, 5]
```

```
lista[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-47b17e44d0b2> in <module>()
----> 1 lista[10]

IndexError: list index out of range
```

4) Exemplo ZeroDivisionError

```
# Operação funcionando
a = 4
b = 2
print(a/b)
print("Executando outro comando")
```

```
2.0
Executando outro comando
```

```
# Operação com erro
a = 4
b = 0
print(a/b)
print("Executando outro comando")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-6-bd35a3a26a76> in <module>()
      2 a = 4
      3 b = 0
----> 4 print(a/b)
      5 print("Executando outro comando")

ZeroDivisionError: division by zero
```

```
a = 4
b = 0

try:
    print(a/b)
except:
    print("Não é permitido a divisão por zero")
print("Executando outro comando")
```

```
Não é permitido a divisão por zero
Executando outro comando
```

5) Exemplo NameError

```
# Observe que como a variável "c" não está definida, ocorre o "NameError"
print(c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-89d5e5580f17> in <module>()
      1 # Observe que como a variável "c" não está definida, ocorre o
"NameError"
----> 2 print(c)

NameError: name 'c' is not defined
```

```
# Observe como fica com o tratamento de exceção:
try:
    print(c)
except:
    print("Deu erro!")
```

```
Deu erro!
```

```
# Observe como fica com o tratamento de exceção:
try:
    print(d)
except NameError as erro:
    print("Erro do desenvolvedor. Detalhe do erro:", erro)
```

```
Erro do desenvolvedor. Detalhe do erro: name 'd' is not defined
```

6) Exemplo NameError 2

```
# Gerando erro de index:
try:
    a = []
    print(a[1])
except NameError as erro:
    print("Erro do desenvolvedor. Detalhe do erro:", erro)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-12-1034ebc323da> in <module>()
      2 try:
      3     a = []
----> 4     print(a[1])
      5 except NameError as erro:
      6     print("Erro do desenvolvedor. Detalhe do erro:", erro)

IndexError: list index out of range
```

```
# Tratando o erro de index:
try:
    a = []
    print(a[1])
except NameError as erro:
    print("Erro do desenvolvedor. Detalhe do erro:", erro)
except IndexError as erro:
    print("Erro de index. Detalhe:", erro)
```

```
Erro de index. Detalhe: list index out of range
```

```
# Tratando vários erros:
try:
    a = [1, 2, 3]
    print(a[1])
except NameError as erro:
    print("Erro do desenvolvedor. Detalhe do erro:", erro)
except IndexError as erro:
    print("Erro de index. Detalhe:", erro)
else:
    print("Seu código foi executado com sucesso!!")
```

```
2
Seu código foi executado com sucesso!!
```

```
# Código adicional, independentemente de ter erro ou não:
try:
    a = [1, 2, 3]
    print(a[1])
except NameError as erro:
    print("Erro do desenvolvedor. Detalhe do erro:", erro)
except IndexError as erro:
    print("Erro de index. Detalhe:", erro)
else:
    print("Seu código foi executado com sucesso!!")
finally:
    print("Aprenda sempre!")
```

```
2
Seu código foi executado com sucesso!!
Aprenda sempre!
```

7) Exemplo KeyboardInterrupt

```
while True:
  try:
    p = int(input('Digite um número inteiro'))
  except ValueError:
    print('Valor inválido')
  except KeyboardInterrupt:
    print('Usuário interrompeu a execução')
    break
  else:
    print(f'Valor digitado é {p}')
    break
```

```
Usuário interrompeu a execução
```

pass, break e continue

```
pass: deixa passar as instruções do programa
break: interrompe as instruções do programa
continue: continua as instruções do programa
# pass e break
counter = 0
while counter < 100:
    if counter == 4:
        break
    else:
        pass
    print(counter)
    counter = counter + 1
```

```
0
1
2
3
```

```
for verificador in "Python":
    if verificador == "h":
        continue
    print(verificador)
```

```
P
y
t
o
n
```

Desenvolvimento de Classes

- Classes: modela entidades que tenham propriedades similares. Representa conceitos!
- Atributos: são as variáveis dentro das classes. Representa estados!
- Métodos: são funções dentro de classes. Representa comportamentos!

Obs.: quando se instancia uma classe por meio de um objeto os "métodos" são compartilhados com todos os objetos criados a partir da mesma classe. Mas os "atributos" não são compartilhados com os outros objetos criados a partir da mesma classe.

```
# Criação da classe Pessoa
class Pessoa:
    '''Classe para gerar o status de uma pessoa'''
    def __init__(self, nome, idade): #construtor usado para instanciar classe
        self.nome = nome
        self.idade = idade

    def Estuda(self):
        print(self.nome + " está estudando...")

    def Anda(self, destino):
        print(self.nome + " está andando até " + destino)
```

Capítulo 6. Deploy

O *deployment*, no contexto da administração de rede, refere-se ao processo de configuração de um novo computador ou sistema até o ponto em que ele esteja pronto para o trabalho produtivo em um ambiente ativo. Ao configurar um site, você sempre terá seu site ativo, que é chamado de ambiente ativo ou ambiente de produção.

Se você deseja fazer alterações sem que elas afetem seu site ativo, você pode adicionar ambientes adicionais. Esses ambientes são chamados de ambientes de desenvolvimento ou ambientes de implantação. Os ambientes de desenvolvimento adicionais normalmente serão um ambiente local, um ambiente de desenvolvimento e um ambiente de teste (também conhecido como um site de teste). A quantidade de ambientes de que você precisa depende de você e da complexidade do projeto em que está trabalhando.

Embora os modelos de implantação possam variar, o mais comum é o modelo clássico de implantação “da esquerda para a direita” ao trabalhar com vários ambientes de implantação. Nesse modelo, as alterações são feitas em ambientes locais, de desenvolvimento ou de teste (dependendo da configuração) e enviadas da esquerda para a direita nos diferentes ambientes que terminam no ambiente ao vivo. Depois que esse processo de implantação for concluído, as novas alterações ficarão visíveis no ambiente ativo.

Há diversas etapas envolvidas no processo de *deployment*. A Figura 8 exibe os comandos básicos para a etapa de versionamento do código no GIT e no SVN.

Figura 8 - Comandos básicos para a etapa de versionamento do código no GIT e no SVN.

Operations	Git command	Svn command
Initial checkout from existing repo	git clone <url> git checkout <origin/branch>	svn checkout <url>
Update locally from repo	git fetch git pull (does a fetch and merge)	svn update
Diff locally changed file	git diff <filename>	svn diff <filename>
Revert locally changed file	git checkout <filename>	svn revert <filename>
Revert all local changes	git reset --hard HEAD	svn revert . R
Commit changes to repo	git commit -m “message” <filename> git push	svn -ci m “message” <filename>

Fonte: http://ananthchellathurai.blogspot.com/2013/05/git-for-beginners.html

Referências

SAKURAI, Rafael. **Introdução ao aprendizado de máquinas**. São Bernardo do Campo, SP, 2021. Site Rafael Sakurai. Disponível em: http://www.sakurai.dev.br/introducao-aprendizado-maquinas/. Acesso em: 08 de ago. 2021.

DONOHO, David. 50 Years of Data Science. **Journal of Computational and Graphical Statistics**, London, v. 26, n. 4, p. 745–766, 2017. Disponível em: https://www.tandfonline.com/doi/pdf/10.1080/10618600.2017.1384734?needAccess=true. Acesso em: 09 ago. 2021.

WANG, Dakuo et al. Human-AI collaboration in Data Science: exploring data scientists'perceptions of automated AI. **Proceedings of the ACM on Human_Computer Interaction**, New York, v. 3, n. CSCW, 2019. Disponível em: https://arxiv.org/pdf/1909.02309.pdf. Acesso em: 08 ago. 2021.

LECUN, Yann et al. Gradient-Based Learning Applied to Document Recognition. **Proceedings of the IEEE**, [S. l.], v.86, n.11, nov. 1998. Disponível em: https://ieeexplore.ieee.org/document/726791. Acesso em: 09 ago. 2021.

JARDIM, Rafael R. J. **Desenvolvimento de um modelo classificador de questões para o cenário educacional brasileiro fundamentado em Ciência de Dados**. 2021. Dissertação (Mestrado em Informática). Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, 2021.