

Introdução à Aprendizagem de Máquina

1 - Apresentação da Disciplina

Honovan Paz Rocha¹²

¹Instituto de Engenharia, Ciéncia e Tecnologia (IECT)
Universidade Federal dos Vales do Jequitinhonha e Mucuri (UFVJM)

²Programa de Pós-Graduaçao em Modelagem Computacional e Sistemas
(PPGMCS) Universidade Estadual de Montes Claros (Unimontes)

Introduction to Machine Learning

Outline

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Honovan Paz Rocha

Formação Acadêmica

- ▶ Graduação: Sistemas de Informação - FACET
- ▶ Especialização: MBA - Tecnologia e Gestão da Informação
- ▶ Mestrado: Engenharia Elétrica - UFMG
- ▶ Doutorado: Engenharia Elétrica (Inteligência Computacional) - UFMG

Experiência Profissional

- ▶ Analista de Sistemas / Arquiteto de Software: 2 Anos
- ▶ Projeto P&D - CEMIG: 2,5 Anos
- ▶ Estágio em Docência UFMG - 1,5 Anos
- ▶ Professor do DECOM (CEFET-MG) - Campus BH: 2 Anos
- ▶ Professor do IECT na UFVJM - Campus Janaúba: 9 Anos

Atuação Acadêmica

- ▶ Disciplinas: Linguagens de Programação, Algoritmos e Programação e Inteligência Artificial
- ▶ Pesquisas: Inteligência Artificial Clássica, Métodos de Busca, Machine Learning, Algoritmos Evolutivos, Metaheurísticas, Multiobjective Learning, Decision-Making.
- ▶ Outras atividades: Revisor em algumas das principais revistas da literatura em ML, Atividades do LITC (UFMG).

Conhecimentos Desejáveis para ML

- ▶ Conhecimentos elementares em computação
(Fundamentos, Algoritmos e Estruturas de Dados)
- ▶ Cálculo, Algebra, Probabilidade e Estatística.
- ▶ Modelagem computacional de problemas

Tópicos em Computação

- ▶ Projeto de Algoritmos.
- ▶ Estruturas de dados lineares, Árvores e Grafos.
- ▶ Linguagens de Programação (Python, R, Matlab, Octave, C e C++, Java, Scala, Rust)

Machine Learning

Tópicos em Algebra

- ▶ Soma e multiplicação entre matrizes e vetores.
- ▶ Matriz identidade, diagonal e matriz inversa.
- ▶ Produto interno e produto escalar.
- ▶ Norma, Distância entre vetores, Ângulos e Ortogonalidade.
- ▶ Normalização de matrizes e vetores.
- ▶ Transformações lineares, Autovalores e Autovetores.

Notações

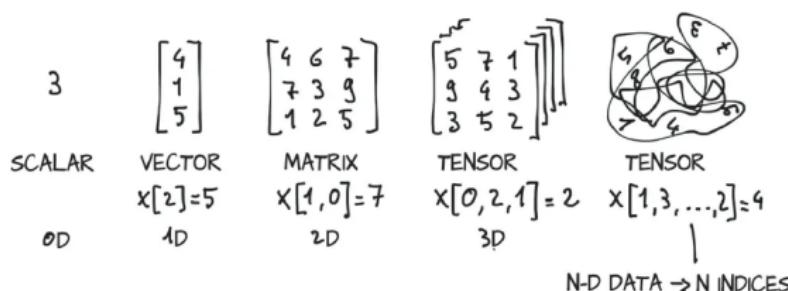


Figure: Fonte: Deep Learning with PyTorch - Eli Stevens

Machine Learning

Tópicos em Cálculo e Otimização

- ▶ Modelagem de Problemas e Função Objetivo.
- ▶ Definição de Problemas de Otimização.
- ▶ Busca de pontos extremos (Máximo ou Mínimo) em funções.
- ▶ Derivadas envolvendo Múltiplas Variáveis e Regra da Cadeia
- ▶ Métodos de Otimização Linear e Não-Linear.

Cálculo Vetorial e Gradiente

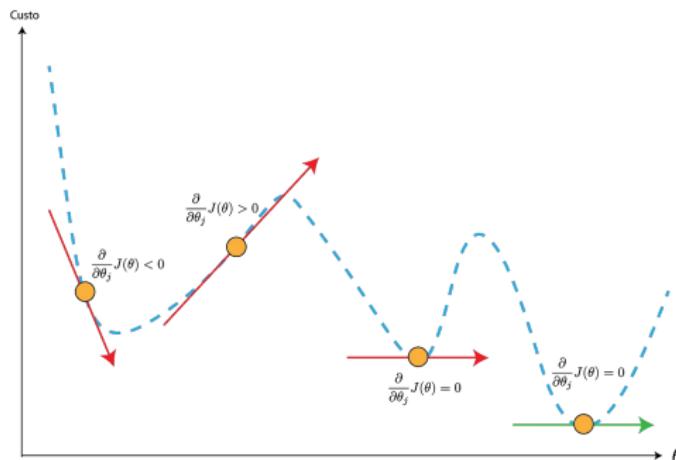
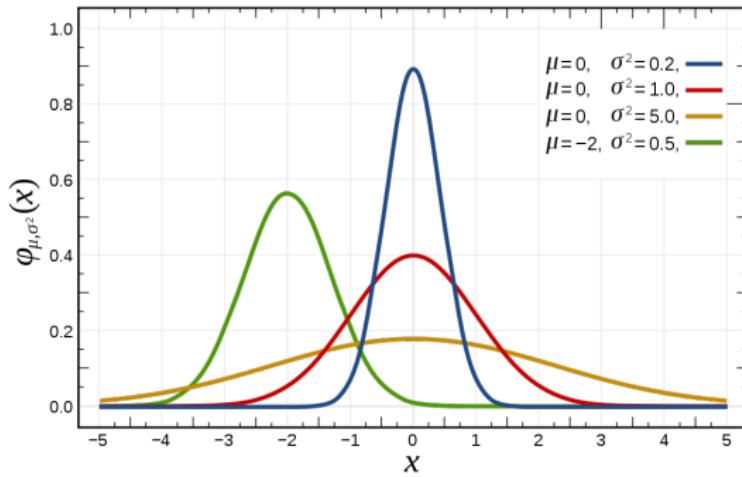


Figure: Fonte: <https://medium.com/luisfredgs/>

Tópicos em Probabilidade e Estatística

- ▶ Variável aleatória, Estatística Descritiva e Funções de Distribuição de Probabilidade.
- ▶ Testes Estatísticos Paramétricos e Não-Paramétricos.
- ▶ Teorema de Bayes e Análise de Séries Temporais.

Distribuição Normal



Qual a formação / Experiência de vocês?

Objetivos?

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Objetivo

Objetivo

A disciplina visa apresentar conceitos básicos, algoritmos e aplicações de Aprendizado de Máquina. Ao fim do curso o estudante estará capacitado para modelagem, desenvolvimento e aplicação de métodos de Aprendizado de Máquina.

Metas

- ▶ Modelar problemas a partir de dados.
- ▶ Utilizar técnicas para análise, extração e seleção de atributos.
- ▶ Desenvolver modelos lineares e não-lineares para regressão, classificação e agrupamento.
- ▶ Avaliar e selecionar modelos.

Ementa

Ementa

Conceitos básicos em Machine Learning e Reconhecimento de Padrões. Extração e Seleção de Características. Aprendizado Supervisionado e Não Supervisionado. Modelos Preditivos e Descritivos. Redes Neurais Artificiais. Avaliação de Modelos. Aplicações.

Conteúdo Programático

- ▶ Apresentação da Disciplina e Conhecimentos Básicos em Python
- ▶ Introdução ao Machine Learning
- ▶ Teoria de Aprendizado
- ▶ Modelos Lineares (Aprendizado Supervisionado)
- ▶ Aprendizado Bayesiano
- ▶ Aprendizado Baseado em Memória
- ▶ Árvores de Decisão
- ▶ Extração e Seleção de Características
- ▶ Avaliação de Modelos
- ▶ Aprendizado Não Supervisionado
- ▶ Introdução às Redes Neurais Artificiais
- ▶ Redes Perceptron de Múltiplas Camadas
- ▶ Aprendizado Multiobjetivo
- ▶ Introdução ao DeepLearning

Disciplinas PPGMCS

- ▶ Algoritmos e Estruturas de Dados
- ▶ Redes Neurais Artificiais
- ▶ Sistemas Nebulosos
- ▶ Algoritmos Evolutivos
- ▶ Otimização
- ▶ Otimização Multiobjetivo
- ▶ Visão Computacional
- ▶ Mineração de Dados
- ▶ Sistemas Inteligentes e Aplicações
- ▶ Probabilidade
- ▶ Introdução aos Processos Estocásticos
- ▶ Ciência de Dados
- ▶ Reconhecimento de Padrões
- ▶ DeepLearning

Bibliografia e Materiais em Geral

- ▶ Y.S. Abu-Mostafa, M. Magdon-Ismail, H.-T. Lin. Learning from data, AMLBook Singapore, 2012.
- ▶ BISHOP, Christopher M. Pattern Recognition and Machine Learning. New York: Springer, 2006.
- ▶ Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron. Deep Learning. MIT Press, 2017.
- ▶ MITCHELL, Tom M. Machine Learning. New York: McGraw-Hill, 1997.
- ▶ Duda, Hart, and Stork, Pattern Classification, 2nd Ed., 2002.
- ▶ HARRINGTON, P. Machine Learning in Action. Shelter Island, NY: Manning Publications Co, 2012.
- ▶ James, G., Witten, D., Hastie, T. e Tibshirani, An Introduction to Statistical Learning, 2013.
- ▶ BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDELMIR, T. B. Redes Neurais Artificiais: Teoria e Aplicações. 2 ed. LTC, 2007.
- ▶ Youtube

Recursos Necessários - Hardware e Software

- ▶ Pendrive, E-mail, Cloud...
- ▶ Python
- ▶ IDE: VScode, Spyder, JupyterLab, Google Colab ou equivalente
- ▶ Ambiente Anaconda

Avaliação

- ▶ Seminário: 15 pts
- ▶ Trabalhos Práticos: 60 pts
- ▶ Artigo Fundamento/Aplicação: 25 pts

Comunicação e Atendimento extraclasse

- ▶ Moodle e/ou Google Classroom/Meet
- ▶ Pessoalmente
- ▶ Dropbox ou GoogleDrive

Introdução à Aprendizagem de Máquina

- ▶ Dúvidas?
- ▶ Sejam Bem Vindos!!!

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Onde está contido o Aprendizado de Máquina?

Áreas da IA

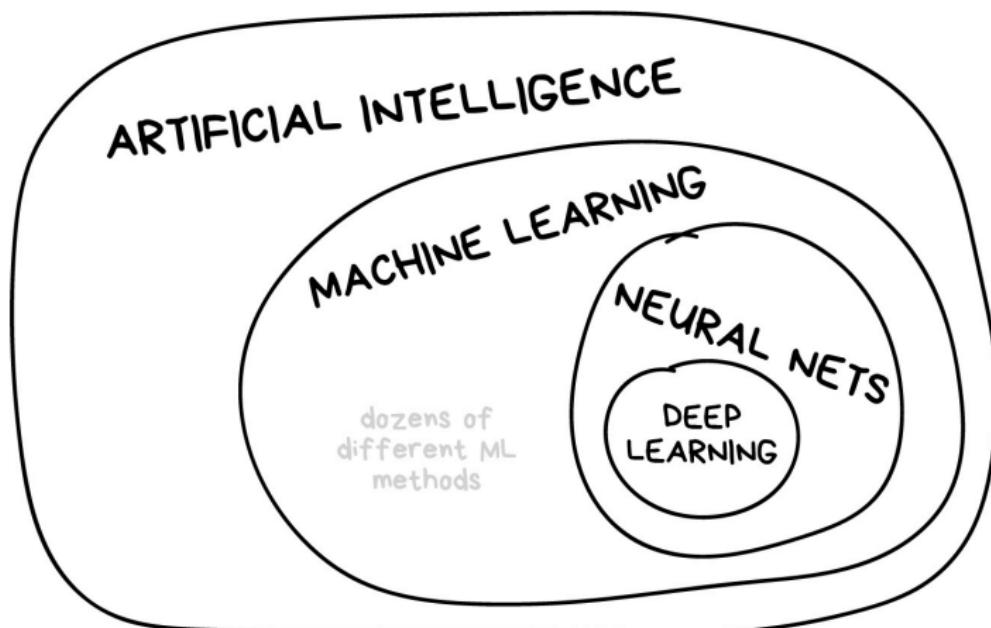


Figure: Fonte: https://vas3k.com/blog/machine_learning

Aprendizado Estatístico

- ▶ Um conjunto de ferramentas (técnicas) para modelagem e análise de dados denominado Aprendizado Estatístico.
- ▶ Uma abordagem estatística para o problema de ML.

Encontrando solução

Como encontrar o melhor modelo?

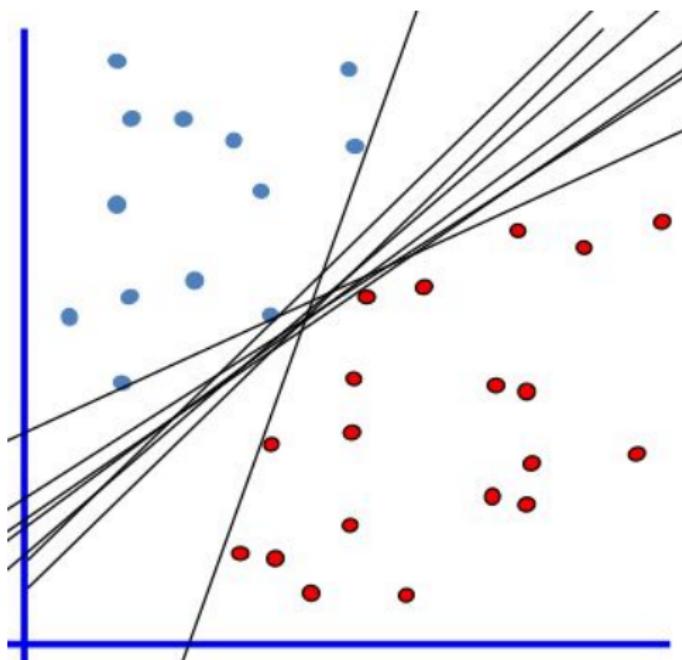


Figure: Fonte: David Menotti(UFPR)

Universo de Algoritmos

Algoritmos em ML

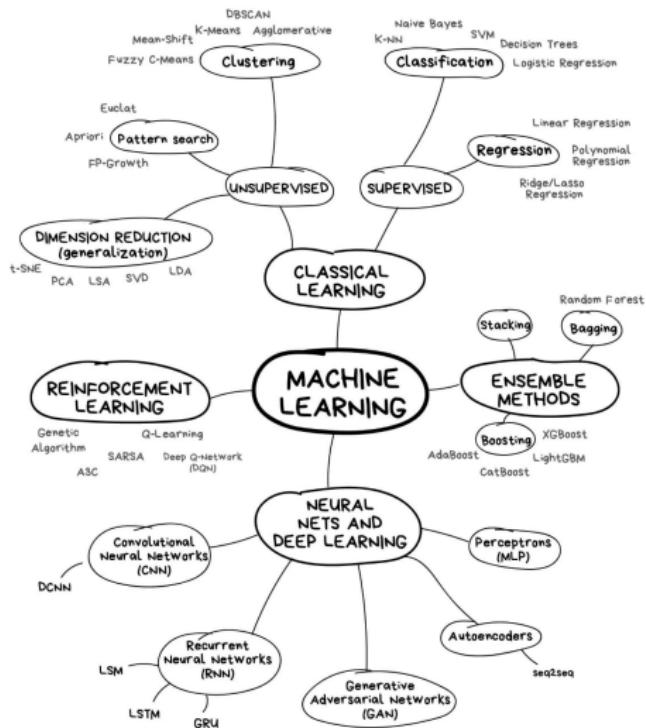


Figure: Fonte: https://vas3k.com/blog/machine_learning

Métodos Clássicos

Algoritmos Clássicos - Foco deste Curso

CLASSICAL MACHINE LEARNING

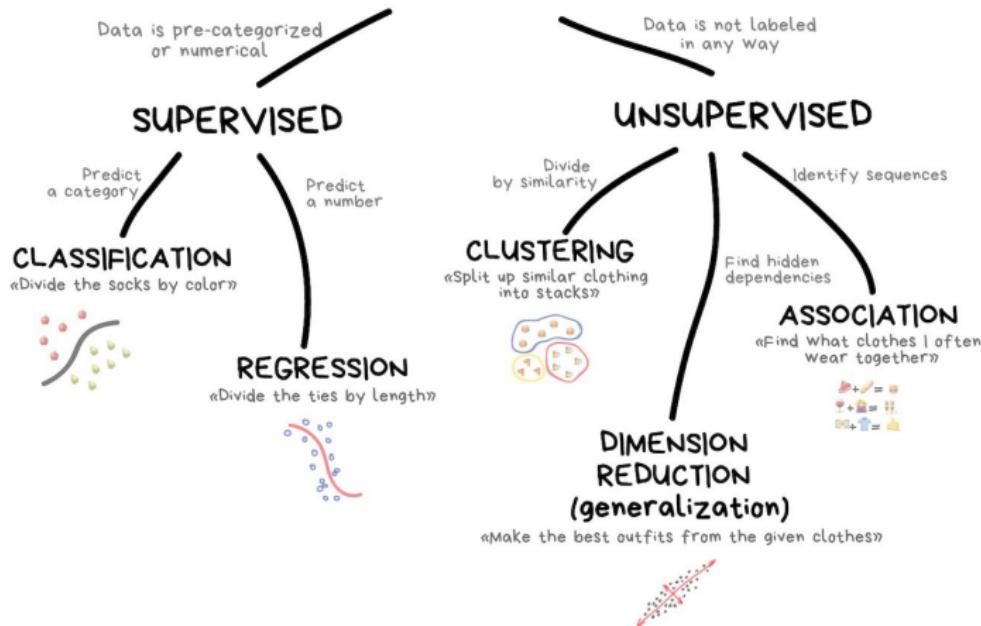


Figure: Fonte: https://vas3k.com/blog/machine_learning

Projeto Completo

Projeto de ML

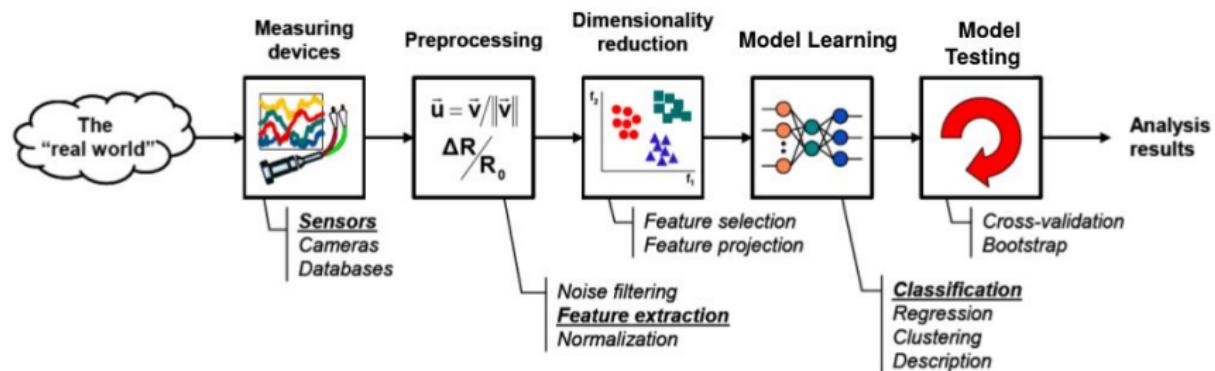


Figure: Fonte: David Menotti(UFPR)

O que a IA pode fazer hoje?

- ▶ **Veículos robóticos:** Um carro robótico sem motorista chamado STANLEY acelerou através do terreno acidentado do deserto Mojave a 22 mph, terminando o percurso de 212 quilômetros como o primeiro para ganhar o DARPA Grand Challenge 2005.

No ano seguinte, o BOSS da CMU ganhou o Urban Chalenge, dirigindo de forma segura no trânsito pelas ruas de uma base da força aérea fechada, obedecendo às regras de trânsito e evitando os pedestres e outros veículos.

<https://www.youtube.com/watch?v=4hFh100i8KI>

O que a IA pode fazer hoje?

- ▶ **Reconhecimento de voz:** Um viajante telefonando para a United Airlines para reservar um voo pode ter toda a conversa guiada por um sistema automático de reconhecimento de voz e de gestão de diálogo.

Produção de jogos e histórias interativas

- ▶ Como modelar o ambiente físico e o comportamento/personalidade dos personagens?
- ▶ Como permitir uma boa interação com usuário?



The Sims



FIFA Soccer

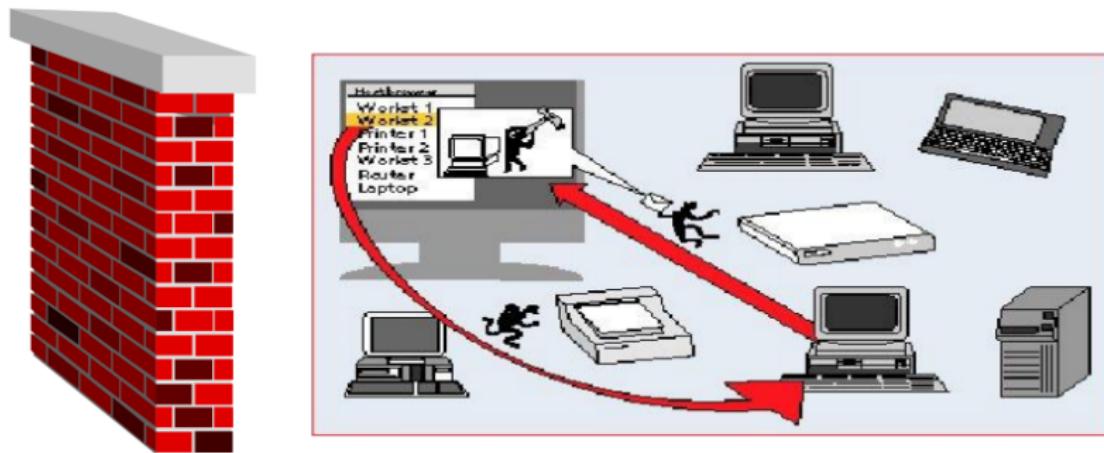
Previsão

- ▶ Como prever o valor do dólar (ou o clima) amanhã?
- ▶ Que dados são relevantes? Há comportamentos recorrentes?



Detecção de Intrusão e Filtragem de Spam

- ▶ Como saber se uma mensagem é lixo ou de fato interessa?
- ▶ Como saber se um dado comportamento de usuário é suspeito e como lidar com isto?



Interface

- ▶ Como dar ao usuário a ajuda que ele precisa?
- ▶ Como interagir (e quem sabe navegar na web) com celular sem ter que digitar (hands-free)?



Análise de Crédito

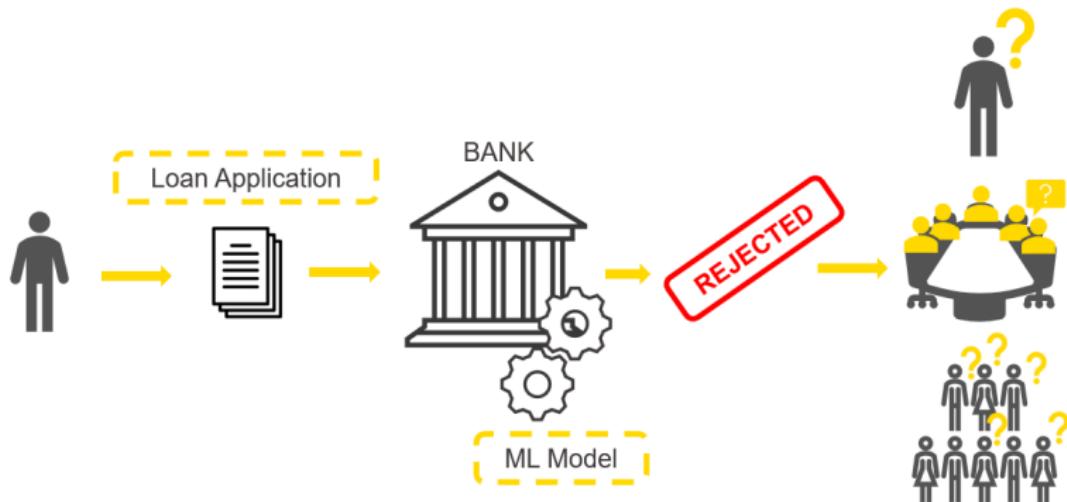


Figure: Fonte: <https://www.knime.com/blog/banks-use-xai-transparent-credit-scoring>

Predição de Covid-19

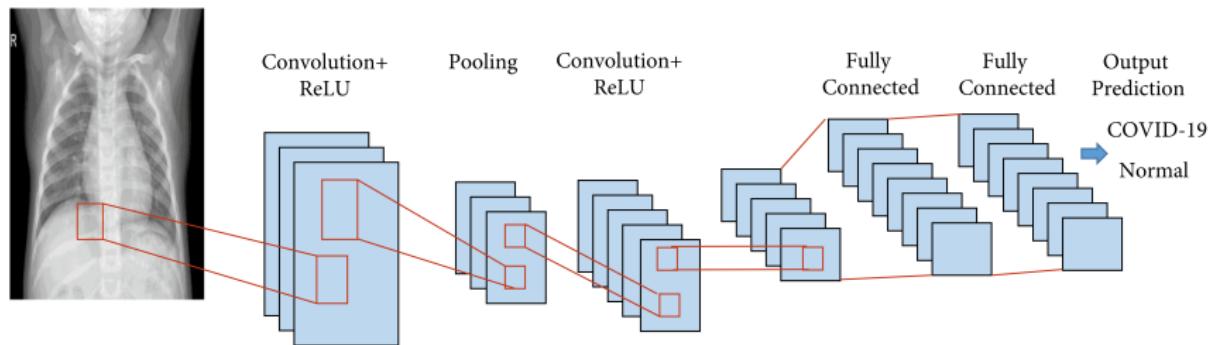
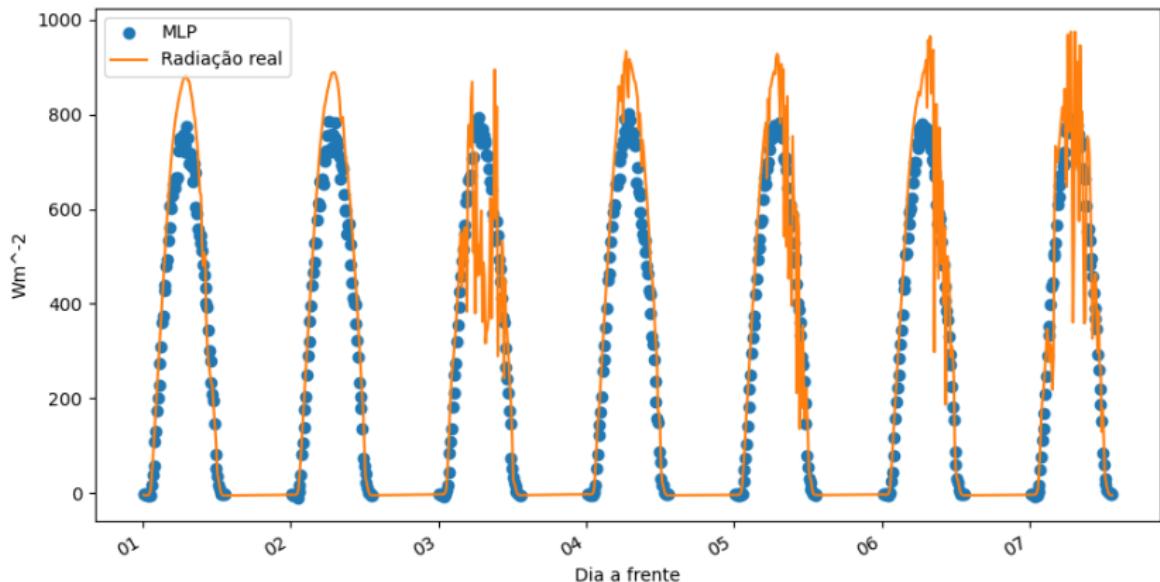


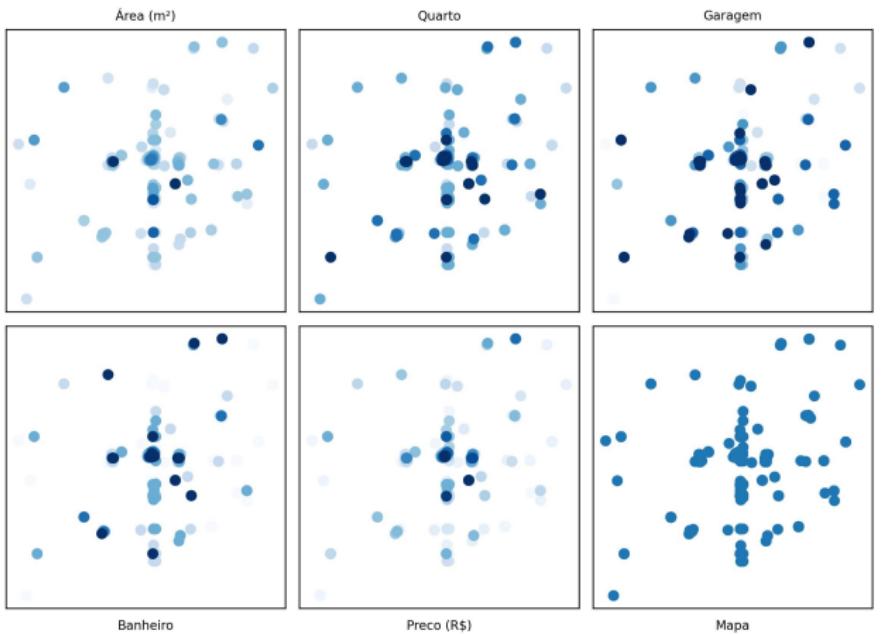
Figure: Fonte: <https://doi.org/10.1155/2022/6216273>

Predição de Radiação



Pesquisas Recentes

Predição Valor de Imóveis no Norte de MG



Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Ambiente de Desenvolvimento

Papel e caneta

Necessário para modelar os problemas e planejar o desenvolvimento da solução e o experimento computacional para validação.

IDE e Dependências

- ▶ Instalação Anaconda
- ▶ Jupyter Lab
- ▶ Instalação VScode

Linguagem Python

Porque Python?

Linguagem de alto nível com enorme comunidade de suporte.

Filosofia da Linguagem

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade.

"Código Python é considerado quase um Pseudocódigo"

Python

Porque usar Python

Python é a linguagem na vanguarda da computação científica com foco em análise de dados e inteligência artificial, onde será possível encontrar a maioria das estruturas e bibliotecas de machine learning, deep learning, etc.

Algumas Bibliotecas

- ▶ NumPy
- ▶ Matplotlib
- ▶ Pandas
- ▶ NLTK
- ▶ Scipy
- ▶ Scikit-learn
- ▶ Keras
- ▶ TensorFlow
- ▶ etc...

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Conceitos iniciais - Python

Variaveis

- ▶ Em Python tudo é objeto, incluindo-se qualquer tipo de variável.
- ▶ Variaveis em Python não guardam valores, guardam referências (endereços de memória).

Exemplo - Vamos abrir a IDE?

```
1 #Primeiro código em python
2 x = [1, 2, 3]
3 y = x
4 x.append(4)
5 print(y)
```

Executando o exemplo

```
Python 3.6.4 |Anaconda custom (64-bit)| (default, Jan 16 2018,  
Type "copyright", "credits" or "license" for more information.
```

```
IPython 6.2.1 -- An enhanced Interactive Python.
```

```
In [1]: x = [1, 2, 3]
```

```
In [2]: y = x
```

```
In [3]: x.append(4)
```

```
In [4]: print(y)  
[1, 2, 3, 4]
```

```
In [5]:
```

Tipos Básicos

Tipos

- ▶ **int, long, float, bool**
- ▶ **NoneType** é o tipo de **None**

Exemplos

```
1 a = 42 #decimal
2 b = 010 #octal
3 c = 0xA #hexadecimal
4 d = 0xDEFBDAEFBDAECBFBAEL #hexadecimal long
5 e = 6.02e23 #(o mesmo que 6.02 x 1023).
6 f = 1.14159265 #float
7 g = True #bool
8 h = False #bool
9 i = None #None type
```

STRING

Definição

- ▶ Para atribuirmos a uma variável uma referência do tipo **string**, basta que coloquemos entre aspas simples, duplas ou triplas

Exemplos

```
1 a = 'Isso é uma String com aspas Simples'
2 b = "Isso é uma String com aspas Duplas"
3 c = """Isso é Uma String com aspas Triplas"""
4 print(a + "\n" + b + "\n" + c)
```

STRING

Características

- ▶ Em Python, tudo é objeto, desta forma as Strings são objetos que tem embutidos vários métodos (funções)
- ▶ Strings são objetos **imutáveis**, ou seja, após a definição da string seus caracteres não podem ser alterados.

Vamos analisar este código!

```
1 #coding: utf-8
2 meu_nome = "Sintaxes Python"
3 meu_nick = 'honovan'
4 print ("Nome: %s, Nick: %s" % (meu_nome.upper(), meu_nick))
5 print ("Meu nome comeca com a letra ", meu_nome[0])
6 print ("Meu nome comeca com a letra ", meu_nome[0].lower())
7 print ("Meu primeiro nome é ", meu_nome[0:7])
```

STRING

string é uma classe

- ▶ Objetos(variáveis?) do tipo string contem diversos métodos úteis.

Exemplo

```
1          #coding: utf-8
2  s = "hello"
3  print(s.capitalize())    # Imprime: "Hello"
4  print(s.upper())        # Imprime: "HELLO"
5  print(s.rjust(7))       # Justifica à direita: " hello"
6  print(s.center(7))      # Centraliza; imprime: " hello "
7  print(s.replace('l', '(ell)')) # Imprime: "he(ell)(ell)o"
8  print(' world '.strip())  # Remove espaços: "world"
9
```

Tipos de formato

- ▶ **d, i** - Número inteiro escrito em decimal
- ▶ **o** - Número inteiro sem sinal escrito em octal
- ▶ **u** - Número inteiro sem sinal escrito em decimal
- ▶ **x** - Número inteiro sem sinal escrito em hexadecimal (minúsculas)
- ▶ **X** - Número inteiro sem sinal escrito em hexadecimal (maiúsculas)
- ▶ **e** - Número de ponto flutuante escrito em notação científica ('e' minúsculo)
- ▶ **E** - Número de ponto flutuante escrito em notação científica ('E' maiúsculo)
- ▶ **f, F** - Número de ponto flutuante escrito em notação convencional
- ▶ **g** - Mesmo que e se expoente é maior que 4. Caso contrario, igual a f
- ▶ **G** - Mesmo que E se expoente é maior que 4. Caso contrario, igual a F
- ▶ **c** - Caractere único (usado com inteiro ou string de tamanho 1)
- ▶ **s** - String

Operadores

Aritméticos	Comparação	Lógicos
+	==	and
-	!=	or
*	>	not
/ ou // (parte inteira)	<	
%	>=	
+ = - = * = / =	<=	
**	in in not	
is		

Entrada de Dados

Python fornece as funções `raw_input`(versões inferiores a 3.0) e `input` para entrada de dados.

Exemplo

```
1 import sys
2 nome = input("Digite seu nome: ")
3 idade = input("Digite sua idade: ")
4 print("Digite seu sexo: ")
5 sexo = sys.stdin.readline()
6 print("Nome:" + nome + "\n" + "Sexo: %s Idade: %s" % (sexo, idade))
```

Comandos de Decisão

Semelhante a outras linguagens

- ▶ Observe que chaves ({}) não são utilizadas
- ▶ Toda linha de código termina com quebra de linha (sem ;)
- ▶ Os comandos presentes no Python são: **IF**, **ELIF** e **ELSE**
- ▶ Para delimitar cada bloco de instruções basta identar o código:

```
if CONDIÇÃO:  
    BLOCO DE CÓDIGO  
elif CONDIÇÃO:  
    BLOCO DE CÓDIGO  
else:  
    BLOCO DE CÓDIGO
```

Comandos de Decisão

Exemplo - IF, ELIF, ELSE

```
1     dedos = int(input("Você tem quantos anos? "))
2
3     if dedos == 18:
4         print("Você tem 18 anos")
5     elif dedos > 18:
6         print("Você tem mais de 18 anos")
7     else:
8         print("Você é menor de idade?")
```

Comandos de Decisão

Exemplo - Opções

```
1 var1 = int(input("Digite um Número para var1"))
2 var2 = int(input("Digite um Número para var2"))
3 if var1 == 1:
4     print("Número var1 igual a 1")
5 elif var1 == 2 or var2 == 3:
6     print("var1 diferente de 1 ou var2 diferente de 2")
7 elif var1 >= 1000 or var2 <= -1000:
8     print("var1 maior que 1000 ou var2 menor que -1000")
9 else:
10    print("nenhuma das alternativas anteriores")
```

Laços

Função range()

Esta função retorna uma sequência de objetos.

```
1     print (range(10)) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2     print (range(5, 10)) # [5, 6, 7, 8, 9]
3     print (range(10, 0)) # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
4
5     i=int(input("Digite um Número"))
6     if i in range(0,10):
7         print("Está Contido")
8     else:
9         print("Não está Contido")
```

Laços

Comando for

Armazena valores de uma sequência de objetos na variável de controle.

Exemplo

```
1   for fruta in [ "banana", "maca ", "uva" ]:  
2       print ("Fruta : " + fruta)  
3       print ("-----")  
4  
5   for i in range (0,10): #(Inclui o Zero, NÃO INCLUI O 10)  
6       print ("i = " + str ( i ))  
7       print ("-----")  
8   for i in range(0,10):  
9       print ('Não realizado!')  
10      if i == 0:  
11          break  
12      print ('Realizado')
```

Laços

Comando while

Semelhante ao comando **for**, mas temos que alterar a variável de controle.

No exemplo a seguir a instrução **pass** faz **NADA**.

Exemplo

```
1 i = 0
2 while i < 10:
3     print ("i = ",i)
4     i += 1
5
6 while True:
7     pass
```

Exceções

Definição

As exceções podem ser tratadas utilizando o recurso do "try...catch".

O Python prefere que você identifique erros em tempo de execução, uma vez que prevenir estes erros pode ter um custo maior na média dos casos.

Exemplo

```
1  try:  
2      print (1 + 'st try')  
3  except:  
4      print ("Isto é uma exceção")
```

Exceções

Exemplo - Somatório Infinito

```
1 print('Digite os valores a serem somados: ')
2 print('Para encerrar digite apenas [ENTER].')
3 total = 0
4 while True:
5     try:
6         n = float(input('valor: '))
7         total = total + n
8     except:
9         break
10    print ('\nTOTAL: %s' % total)
```

Exceções

Exemplo 2 - Somatório Infinito

```
1 # Somador infinito - Versão melhorada
2 print('Digite os valores a serem somados: ')
3 print('Para encerrar digite apenas [ENTER].')
4 total = 0
5 while True:
6     try:
7         linha = input('valor: ')
8         n = float(linha)
9         total = total + n
10    except:
11        if len(linha) == 0:
12            break
13        elif ',' in linha:
14            print('Use o . (ponto) como separador decimal.')
15        else:
16            print('Isso não parece um número válido.')
17 print ('\nTOTAL: %s' % total)
```

Listas

Definição

- ▶ Consiste de um conjunto de objetos indexados por um número inteiro chamado de índice, que se inicia em zero
- ▶ Elementos da lista podem ser de qualquer tipo, até mesmo outras listas, e não precisam ser todos do mesmo tipo
- ▶ Uma lista é delimitada por colchetes e seus elementos separados por vírgulas. Listas são **mutáveis**.

Exemplo

```
1 lista = [1, 2, 3]
2 print ("Tamanho de lista:", len(lista))
3
4 lista[0]=9
5 print ("-----")
6 for i in range(len(lista)):
7     print (lista[i])
```

Acessando o conteúdo de listas

```
1 lista = [1, 2, 3]
2 print (lista[0])
3 print (lista[1])
4 print (lista[-1])
5 print (lista[-2])
6 print (lista[0:2])
7 print (lista[:2])
8 print (lista[1:])
```

Saída

Linha 2: 1

Linha 3: 2

Linha 4: 3

Linha 5: 2

Linha 6: [1, 2]

Linha 7: [1, 2]

Linha 8: [2, 3]

Listas

Exemplo de aplicação de função

```
1 frutas=['laranja', 'banana', 'abacaxi']
2 frutas.sort()
3 print (frutas)
4 frutas[0]='framboesa';
5 for item in frutas:
6     print (item.capitalize())
```

- ▶ A função **sort()** ordena os objetos da lista
- ▶ A função **capitalize()** deixa o primeiro caractere da string em caixa alta.

Listas

Adição e remoção na lista

```
1 convidados = [] #define uma lista vazia
2 convidados.append ('Exemplo 1')
3 convidados.append ('Exemplo 2')
4 convidados.append ('Exemplo 3')
5 convidados.append ('Exemplo 4')
6 print ("Tenho", len(convidados), "Convidados")
7 convidados.sort()
8 print ("Sao eles:")
9 print (convidados)
10 print ("O primeiro convidado eh o", convidados[0])
11 convidados.remove("Exemplo 4") #aqui tiramos o Exemplo 4 da lista
12 print ("Agora tenho somente", len(convidados), "Convidados")
13 print ("Sao eles:")
14 for convidado in convidados:
15     print (convidado)
```

Listas

List comprehensions

- ▶ Uma forma concisa de se criar listas. Característica chave em linguagens funcionais.

Exemplo - Criando uma lista tendo outra como base

```
1 nums = [0, 1, 2, 3, 4]
2 quadrados = []
3 for x in nums:
4     quadrados.append(x ** 2)
5 print(quadrados)    # Prints [0, 1, 4, 9, 16]
```

Exemplo - Utilizando List comprehensions

```
1 nums = [0, 1, 2, 3, 4]
2 quadrados = [x ** 2 for x in nums]
3 print(quadrados)    # Prints [0, 1, 4, 9, 16]
```

Listas

Exemplo - Listas com Strings

```
1 lista_palavras = ["Esta", "e", "uma", "lista", "de", "palavras"]
2 itens = [ palavra[0] for palavra in lista_palavras ]
3 print(itens) #imprime: ['E', 'e', 'u', 'l', 'd', 'p']
```

Exemplo - Pode-se utilizar condições

```
1 nums = [0, 1, 2, 3, 4]
2 quadrados_de_pares = [x ** 2 for x in nums if x % 2 == 0]
3 print(quadrados_de_pares) # Prints "0, 4, 16"
```

Exemplo condicional - Listas com Strings

```
1 string = "Hello 12345 World"
2 numeros = [x for x in string if x.isdigit()]
3 print(numeros) #imprime: ['1', '2', '3', '4', '5']
```

Tuplas e Dicionários

Tuplas

- ▶ Vimos até agora: strings, que são compostos de caracteres; e listas, que são compostas de elementos de qualquer tipo
- ▶ Vimos que strings são **imutáveis** e listas são **mutáveis**
- ▶ Há outro tipo em Python chamado tupla (tuple), que é similar a uma lista, exceto por ser **imutável**.
- ▶ Não pode-se incluir, remover ou alterar elementos de uma tupla

Exemplo

```
1 tupla = 'a', 'b', 'c', 'd', 'e'  
2 #Embora não necessário, é convencional usar tuplas entre parênteses:  
3 tupla = ('a', 'b', 'c', 'd', 'e')  
4 #Para criar tuplas com um único elemento, incluimos uma vírgula final:  
5 t1 = ('a',)  
6 type(t1) #Saída: <class 'tuple'>
```

Tuplas e Dicionários

Tuplas

- ▶ Tuplas podem ser utilizadas para retorno em funções. Um exemplo disso é o comando `divmod()` que retorna uma tupla contendo o valor do quociente e do resto de uma divisão.

Exemplo

```
1 x = 18
2 y = 4
3 quociente, resto = divmod(x, y)
4 print(quociente, resto)
```

Tuplas e Dicionários

Dicionários

- ▶ Dicionários em Python fornecem um mecanismo para mapeamento <chave, valor>.
- ▶ Um mapeamento é uma coleção de associações entre pares de valores, onde o primeiro elemento é chamado de chave (**imutável**) e o outro de conteúdo (valor).

Exemplo

```
1 veiculo = {}
2 veiculo['marca'] = 'Puma'
3 veiculo['modelo'] = 'GTB'
4 veiculo['ano'] = 1978
5 print(veiculo['marca'])
6 dicionario = {"chave": "17,532", "chave2": 17.365}
7 print(dicionario ["chave"]) #imprime valor (17,532)
8 print(dicionario ["chave2"]+5) #imprime valor (22.365)
9 print(type(dicionario)) #imprime: <class 'dict'>
```

Percorrendo um dicionário

```
1 d = {'pessoa': 2, 'gato': 4, 'aranha': 8}
2 for animal in d:
3     pernas = d[animal]
4     print('Um(a) %s tem %d pernas' % (animal, pernas))
5 # Prints "Um(a) pessoa tem 2 pernas", "Um(a) gato tem 4 pernas", ...
6 # "Um(a) aranha tem 8 pernas"
```

Dictionary comprehensions

- ▶ Similar a list comprehension. Permite criar dicionários facilmente, tomando uma lista como base.

Exemplo

```
1 nums = [0, 1, 2, 3, 4]
2 even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
3 print(even_num_to_square) # Prints "{0: 0, 2: 4, 4: 16}"
```

Funções

Sintaxe Básica

```
1 def nome(parametros):  
2     <bloco de comandos>
```

Exemplo 1 - Função sem parâmetros

```
1 def menu():  
2     print(""" Escolha um item  
3         1) Cadastrar aluno  
4         2) Consultar aluno  
5         3) Sair""")  
6  
7 menu() #Chamada
```

Ex 2 - Função com parâmetro e retorno

```
1 def somatorio(n):
2     soma = 0
3     for i in range(n, 0, -1):
4         soma += i
5     return soma
6
7 print(somatorio(5))
```

Ex 3 - Função com parâmetro e valor padrão

```
1 def curso(nome="IA"):
2     print("Curso de " + nome)
3
4 curso()
5 curso("Algoritmos")
6
7 #Saida >>
8 Curso de IA
9 Curso de Algoritmos
```

Funções

Funções como Objetos

Assim como tudo em Python, funções também são objetos.

Exemplo

```
1 def div_resto(x,y):  
2     return x // y, x % y #Retornando uma tupla  
3  
4 print(div_resto(28, 5)) #Saida >> (5, 3)  
5 print(div_resto)  
6 #Saida >> (5, 3)  
7 #<function div_resto at 0x7fbe84a90510>  
8  
9 d = div_resto  
10 print(d(28,5)) #Saida >> (5, 3)
```

Funções

Exemplo 2 - Passando função como parâmetro

```
1 def somatorio(n):
2     soma = 0
3     for i in range(n, 0, -1):
4         soma+=i
5     return soma
6
7 def produtorio(n):
8     prod = 1
9     for i in range(1, n+1):
10        prod*=i
11    return prod
12
13 def funcao_generica(funcao, n):
14     return funcao(n)
15
16 n = int(input("Insira um valor positivo: "))
17 print("\n\n", funcao_generica(somatorio,n))
18 print("\n\n", funcao_generica(produtorio, n))
```

Definição

- ▶ Para abrir um arquivo é necessário indicar o nome/path e o modo de abertura("w", "r").
- ▶ A função para abertura de arquivo gera um objeto para manipulação de arquivos.

Exemplo

```
1 f = open("teste.dat", "w")
2 print(type(f))
3 f.close()
```

Iterando sobre as linhas de um Arquivo

- ▶ É possível ler(read) as linhas de um arquivo através do objeto de manipulação de arquivos. O comando **for** pode ser utilizado.
- ▶ Cada linha do arquivo termina com \n.

Exemplo

```
1 arq = open("teste_leitura.txt", "r")
2
3 print("Resultado Avaliação:\n")
4 for linha in arq:
5     valores = linha.split()
6     print('\t', valores[0], valores[1], 'obteve a avaliação ',
7           valores[10])
8
9 arq.close()
```

Formas de leitura e gravação

write

Uso: **arq.write(s)** - Adiciona o string **s** no final do arquivo aberto para escrita ("w").

read(n)

Uso **arq.read()** - Lê e retorna um string de **n** caracteres ou o arquivo inteiro como um string se **n** não é fornecido.

readline(n)

Uso: **arq.readline()** - Retorna a próxima linha do arquivo com todo o texto e incluindo o caractere \n. Se **n** é fornecido como argumento então somente **n** caracteres são retornados.

readlines(n)

arq.readlines() - Retorna uma lista de strings, representando as linhas do arquivo. Se **n** é fornecido, então **n** caracteres serão lidos, sendo **n** arredondado para fornecer linhas inteiras.

Exemplo - readline()

```
1     arq = open("teste_leitura.txt", "r")
2     print("Resultado Avaliação:\n")
3     linha = arq.readline()
4     while linha:
5         valores = linha.split()
6         print('\t', valores[0], valores[1],
7               'obteve a avaliação ', valores[10] )
8         linha = arq.readline()
9
10    arq.close()
```

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Numpy

Numpy

- ▶ É a biblioteca central para computação científica em Python;
- ▶ Ela fornece um objeto de array multidimensional de alto desempenho e ferramentas para trabalhar com esses arrays;
- ▶ Se você tem familiaridade com o Matlab este artigo pode ser útil: http://scipy.github.io/old-wiki/pages/NumPy_for_Matlab_Users

Arrays

- ▶ Um array da **Numpy** é um grid de valores do mesmo tipo, indexados por um tupla de inteiros não negativos;
- ▶ o número de dimensões é o **rank** do array;
- ▶ O tamanho de cada dimensão do array é o **shape** dele.

Numpy

Iterando sobre as linhas de um Arquivo

- Pode-se inicializar arrays numpy a partir de listas. O acesso aos elementos pode ser realizado via colchetes;

Exemplo

```
1 import numpy as np
2
3 a = np.array([1, 2, 3])      # Cria um array de rank 1
4 print(type(a))              # Imprime: "<class 'numpy.ndarray'>"
5 print(a.shape)               # Imprime: "(3, )"
6 print(a[0], a[1], a[2])     # Imprime: "1 2 3"
7 a[0] = 5                    # Altera um valor no array
8 print(a)                    # Imprime: "[5, 2, 3]"
9
10 b = np.array([[1,2,3],[4,5,6]])    # Cria um array de rank 2
11 print(b.shape)                # Imprime: "(2, 3)"
12 print(b[0, 0], b[0, 1], b[1, 0])  # Imprime: "1 2 4"
```

Numpy

Funções para criação de arrays

```
1 import numpy as np
2 a = np.zeros((2,2))      # Cria um array de "zeros"
3 print(a)                 # Imprime: "[[ 0.  0.]
4 #                      [ 0.  0.]]"
5
6 b = np.ones((1,2))       # Cria um array de "1s"
7 print(b)                 # Imprime: "[[ 1.  1.]]"
8
9 c = np.full((2,2), 7)    # Cria um array constante
10 print(c)                # Imprime: "[[ 7.  7.]
11 #                      [ 7.  7.]]"
12
13 d = np.eye(2)            # Cria uma matriz identidade 2x2
14 print(d)                # Imprime: "[[ 1.  0.]
15 #                      [ 0.  1.]]"
16 e = np.random.random((2,2)) # Cria um array aleatório
17 print(e)                 # Pode imprimir: "[[ 0.91940167  0.08143941]
18 #                      [ 0.68744134  0.87236687]]"
```

Numpy

Formas para Indexar o array

Pode-se inicializar arrays numpy a partir de listas. O acesso aos elementos pode ser realizado via colchetes;

Formas para Indexar o array

```
1 #Array de rank 2 e shape (3,4)
2 a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
3 # [[ 1  2  3  4]
4 #  [ 5  6  7  8]
5 #  [ 9 10 11 12]]
6
7 b = a[:2, 1:3] #Podemos FATIAR(slicing) o array
8 # [[2 3]
9 #  [6 7]]
10
11 # Um slice de um array é apenas uma view(exibição) dos dados;
12 print(a[0, 1])    # Imprime: "2"
13 b[0, 0] = 77      # b[0, 0] é o mesmo a[0, 1]
14 print(a[0, 1])    # Imprime: "77"
```

Numpy

Copiando um Array

Utilizando indexação com valor inteiro, é possível copiar.

"Indexação sofisticada"

```
1 a = np.array([[1,2], [3, 4], [5, 6]])
2
3 # Indexando com inteiros. Array shape (3,)
4 print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]"
5
6 # O código acima é equivalente a este:
7 print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
8
```

Outra forma: Função copy

```
1 a = np.array([[1,2], [3, 4], [5, 6]])
2
3 b = a.copy()
```

Indexação Booleana

```
1 a = np.array([[1,2], [3, 4], [5, 6]])
2
3 bool_idx = (a > 2)      # Encontra elementos maiores que 2;
4
5 print(bool_idx)          # imprime: "[[False False]
6 #                           [ True  True]
7 #                           [ True  True]]"
8
9 # Utilizando o resultado para indexar
10 print(a[bool_idx])     # Imprime: "[3 4 5 6]"
11
12 # Tudo que está acima em uma linha (Lindo isso):
13 print(a[a > 2])        # Prints "[3 4 5 6]"
```

Operações matemáticas Básicas

```
1 x = np.array([[1,2],[3,4]], dtype=np.float64)
2 y = np.array([[5,6],[7,8]], dtype=np.float64)
3 # Soma par a par (Dois comandos possíveis)
4 print(x + y)
5 print(np.add(x, y)) #Resultado:    # [[ 6.0  8.0]
6                                # [10.0 12.0]]
7 print(x - y) # Diferença par a par
8 print(np.subtract(x, y))
9
10 print(x * y) # Produto par a par (!= matlab)
11 print(np.multiply(x, y))
12
13 print(x / y) # Divisão real par a par
14 print(np.divide(x, y))
15
16 print(np.sqrt(x)) # Raiz quadrada de cada elemento
```

Operações Matriciais

Para operações matriciais temos a função **dot**. Podemos fazer produtos internos, vetores X matrizes e matrizes X matrizes

```
1 x = np.array([[1,2],[3,4]])
2 y = np.array([[5,6],[7,8]])
3 v = np.array([9, 10])
4 w = np.array([11, 12])
5
6 print(v.dot(w)) # Produto interno de vetores - Imprime: 219
7 print(np.dot(v, w)) # Imprime: 219
8
9 # Produto entre Matriz e Vetor;
10 print(x.dot(v)) # Produz um array de rank 1 [29 67]
11 print(np.dot(x, v))
12
13 # Matrix X Matriz;
14 print(x.dot(y)) # Produz um array de rank 2
15 print(np.dot(x, y))      # [[19 22]
16 # [43 50]]
```

Numpy

Somatório de elementos em Array

Numpy fornece diversas funções úteis, **sum** é uma destas.
Outros podem ser encontradas na documentação.

```
1 import numpy as np
2
3 x = np.array([[1,2],[3,4]])
4
5 print(np.sum(x))    # Soma de todos os elementos; Imprime: "10"
6 print(np.sum(x, axis=0))  # Soma de cada coluna; Imprime: "[4 6]"
7 print(np.sum(x, axis=1))  # Soma de cada coluna; Imprime: "[3 7]"
```

Matriz Transposta

```
1 x = np.array([[1,2], [3,4]])
2 print(x)      # Prints "[[1 2]
3 #                 [3 4]]"
4 print(x.T)    # Prints "[[1 3]
5 #                 [2 4]]"
```

SciPy - Provê grande quantidade de funcionalidades

- ▶ Utiliza **Numpy** como base;
 - ▶ Útil para diferentes tipos de aplicações científicas.
 - ▶ Exemplo - Trabalhando com imagens:
-

```
1 from scipy.misc import imread, imsave, imresize
2
3 # Lendo a imagem do HD como um array Numpy
4 img = imread('testes/cat.jpg')
5 print(img.dtype, img.shape) # Imprime: "uint8 (400, 248, 3)"
6
7 # Alterando canais de cor imgem(400, 248, 3) * [1, 0.95, 0.9]
8 img_tinted = img * [1, 0.95, 0.9]
9
10 img_tinted = imresize(img_tinted, (300, 300)) # Alterando o tamanho
11
12 # Gravando a imagem no HD
13 imsave('testes/cat_2.jpg', img_tinted)
```

Exemplo



Matplotlib

Matplotlib

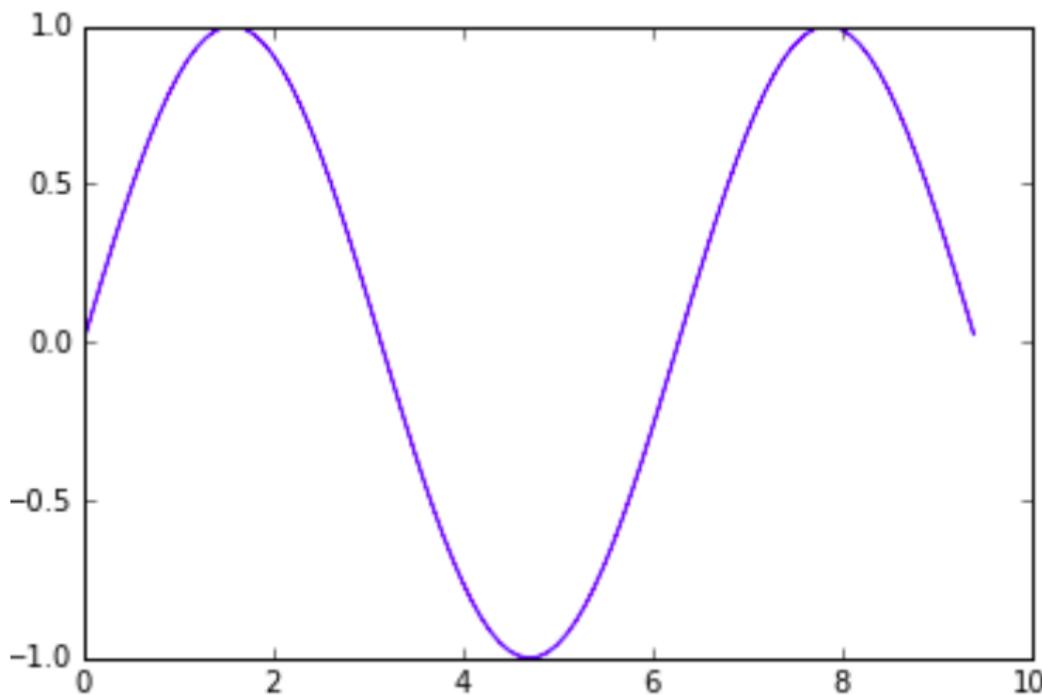
- ▶ É uma biblioteca para realização de **plots**;
- ▶ Um dos principais módulos é o **matplotlib.pyplot** que fornece um sistema de plots similar ao Matlab;
- ▶ A função mais importante deste módulo é a **plot**. Ela permite plots 2D;

Matplotlib

Exemplo

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Computa as coordenadas x e y para pontos em uma curva seno
5 x = np.arange(0, 3 * np.pi, 0.1)
6 y = np.sin(x)
7
8 # Plota os pontos usando a matplotlib
9 plt.plot(x, y)
10 # Pode-se chamar plt.show() para fazer o gráfico aparecer.
11 plt.show()
```

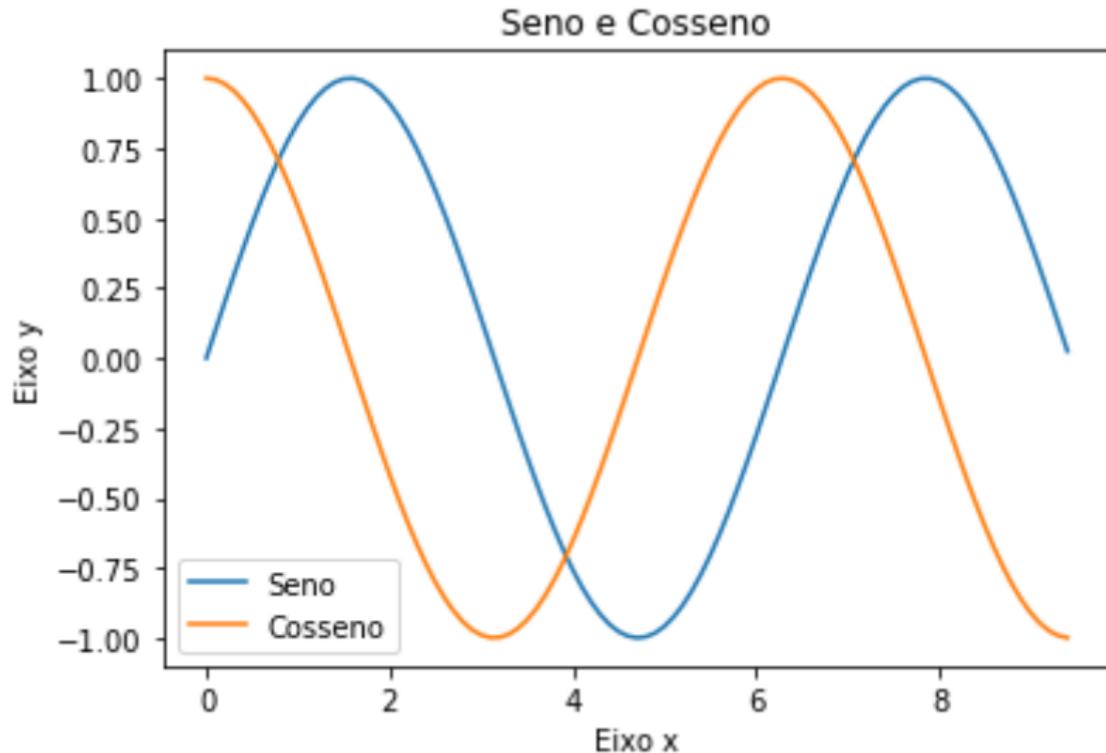
Resultado do exemplo



Melhorando o exemplo

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Gera as coordenadas x e y para pontos nas curvas seno e cosseno
5 x = np.arange(0, 3 * np.pi, 0.1)
6 y_sin = np.sin(x)
7 y_cos = np.cos(x)
8
9 # Plotando os pontos usando matplotlib
10 plt.plot(x, y_sin)
11 plt.plot(x, y_cos)
12 plt.xlabel('Eixo x')
13 plt.ylabel('Eixo y')
14 plt.title('Seno e Cosseno')
15 plt.legend(['Seno', 'Cosseno'])
16 plt.show()
```

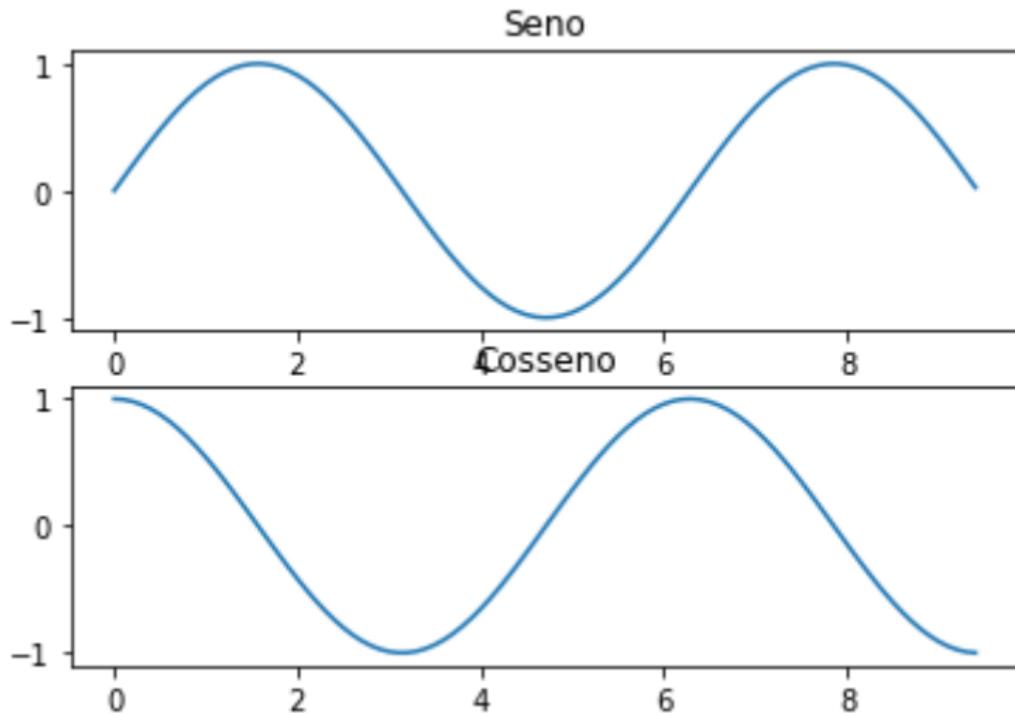
Resultado do exemplo



Subplots

```
1 # Gera as coordenadas x e y para seno e coseno
2 x = np.arange(0, 3 * np.pi, 0.1)
3 y_sin = np.sin(x)
4 y_cos = np.cos(x)
5
6 # Configura um grid subplot altura 2 e largura 1,
7 plt.subplot(2, 1, 1) # Primeira posição ativa
8
9 plt.plot(x, y_sin) #Gera o primeiro plot
10 plt.title('Seno')
11
12 plt.subplot(2, 1, 2) # Marca a segunda posição como ativa
13 plt.plot(x, y_cos) #Gera o segundo plot
14 plt.title('Cosseno')
15
16 plt.show() # Mostra o gráfico
```

Resultado do exemplo



Obrigado!

Dúvidas?

Obrigado!

Sumário

Apresentação

Organização da Disciplina

Primeiros Passos em ML

Aplicações

Conhecendo as ferramentas necessárias

Ferramentas para o Desenvolvimento de Soluções

Introdução Python

Variáveis e Tipos Básicos

Estruturas de Controle

Tratamento de Exceções

Tipos Não-Escalares

Funções

Arquivos

Bibliotecas - Computação Científica

Exercícios

Exercícios

1. Baixe de um repositório na web (UCI, Kaggle, etc...) a base de dados IRIS. Escreva um código Python para efetuar a leitura dos dados desta base. Adicionalmente, obtenha estatísticas descritivas sobre cada um dos atributos.
2. Utilizando a biblioteca Matplotlib, gere gráficos desta base de dados, evidenciando a disposição espacial de cada classe.
3. Utilizando a biblioteca Numpy, gere um conjunto sintético de dados, com 2 classes e 20 amostras em cada classe. pode-se utilizar qualquer distribuição para geração dos dados.