

AULA LABORATORIAL N.º 4
APRENDIZAGEM AUTOMÁTICA - SKLEARN

1. Objetivos

Esta ficha pretende apresentar a ferramenta em linguagem Python e scikit-learn (sklearn), para aplicação a problemas de classificação em “machine learning (ML)”

Os objetivos de aprendizagem consistem em:

- Utilizar a Linguagem Python para resolução de problemas de aprendizagem automática;
- Conhecer os elementos principais da linguagem;
- Conhecer a aplicar a biblioteca scikit-learn;
- Aplicar os métodos a problemas de classificação;
- Implementar em python um *pipeline* para classificação.

2. A Linguagem Python

O Python consiste atualmente na linguagem mais usada no domínio da aprendizagem automática, pelo facto de combinar a facilidade de utilização de linguagens script (tal como o Matlab ou R) com as capacidades associadas às linguagens de programação orientadas a objetos ou funcionais.

A facilidade de aplicação a domínios específicos é potenciada pela disponibilização de bibliotecas para análise e visualização de dados, incluindo processamento de imagem, texto, sequências, ferramentas de análise estatística entre outras (Muller and Guido, 2017; <http://bit.ly/intro-machine-learning-python>).

Para iniciar a programação em Python recomenda-se nesta ficha a instalação da distribuição Anaconda, disponível para Windows, MacOS e Linux, que já incorpora outras bibliotecas necessárias para a construção das primeiras aplicações de aprendizagem automática (Figura 1).

A instalação pode ser realizada a partir de:

- <https://www.anaconda.com/>

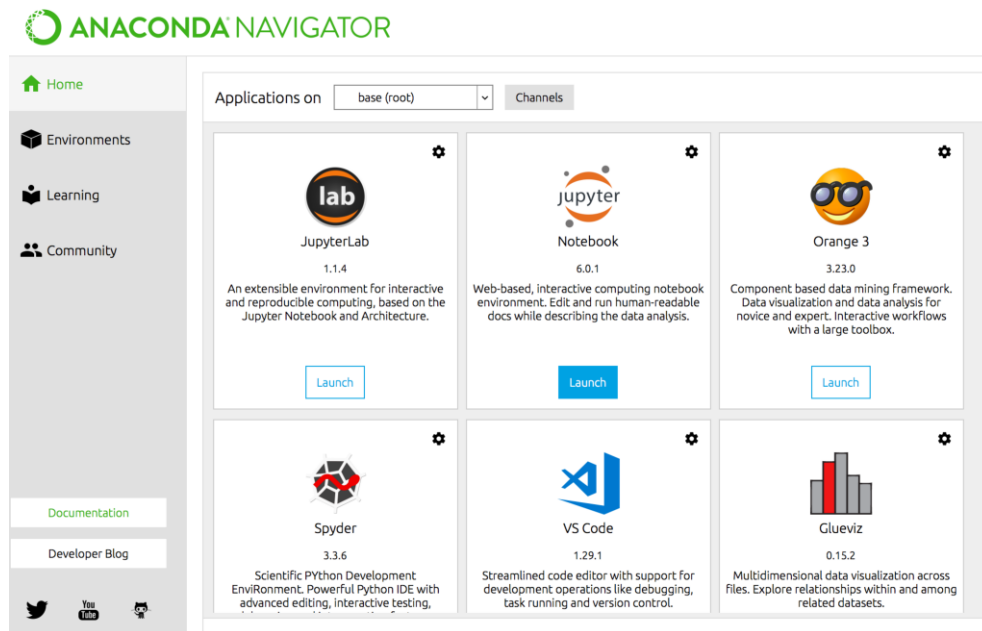


Figura 1. Distribuição Anaconda – Navigator screen.

Para o desenvolvimento de programas recomenda-se o IDE Spyder ou em alternativa o Jupyter Notebook, ambos incluídos na distribuição Anaconda. Para a construção de fluxos de execução interativos, sem necessidade de programação, poderá utilizar-se a framework Orange.

Para iniciar o estudo da Linguagem Python sugere-se:

- Python Tutorial
<https://docs.python.org/3/tutorial/index.html>
- Python Reference
<https://docs.python.org/3/library/index.html>
- LearnPython – Tutorial Interativo de Python
<https://www.learnpython.org/>
- Python Tutor (permite a visualização de execução do código, útil em particular, para compreensão da manipulação de listas e dicionários)
 - <http://pythontutor.com/visualize.html#mode=edit>

Todas as referências para a documentação associada às bibliotecas também podem ser acedidas via Anaconda:

<https://docs.anaconda.com/anaconda/packages/pkg-docs>

2.1 O meu primeiro Programa

Tradicionalmente, o primeiro programa numa nova linguagem é o designado "Hello, World!". Para a construção deste primeiro programa em Python, que apresente a mensagem "Hello World":

- Inicie o IDE spyder (Figura 2) e escreva:

```
#!/usr/bin/env python3
print("Hello World")
```

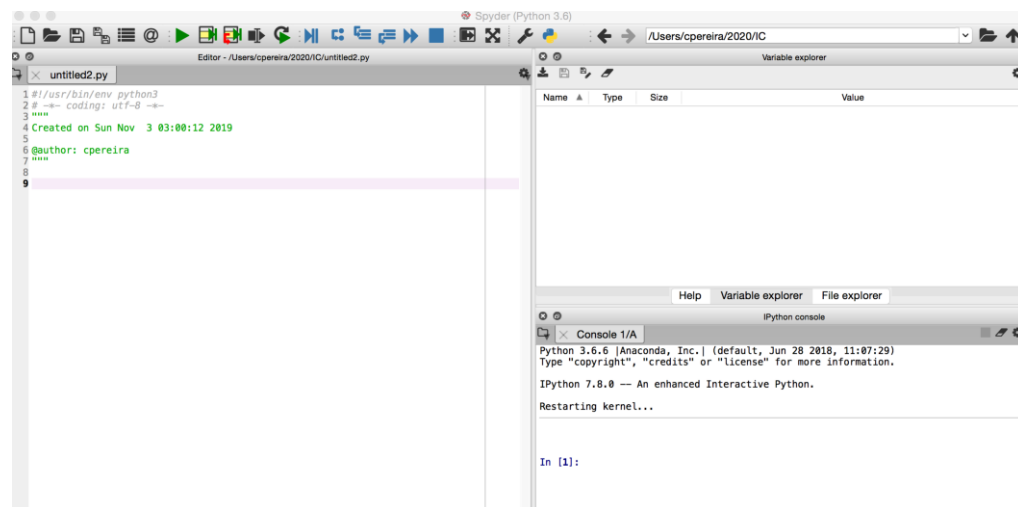


Figura 2. IDE Spyder

- Em alternativa à criação do script, sendo o Python uma linguagem interpretada, na linha de comando do editor digite
>>> print("Hello World")

```
In [1]: print("hello world")
hello world
In [2]:
```

Figura 3. Linha de comando

- Como terceira alternativa, pode usar o “Jupyter Notebook”:
 - Inicie um novo kernel Python 3
 - Digite o comando “print print(“Hello World”)” na primeira célula, em modo de edição.
 - Clique em “Run” para executar o código na célula (Figura 4).

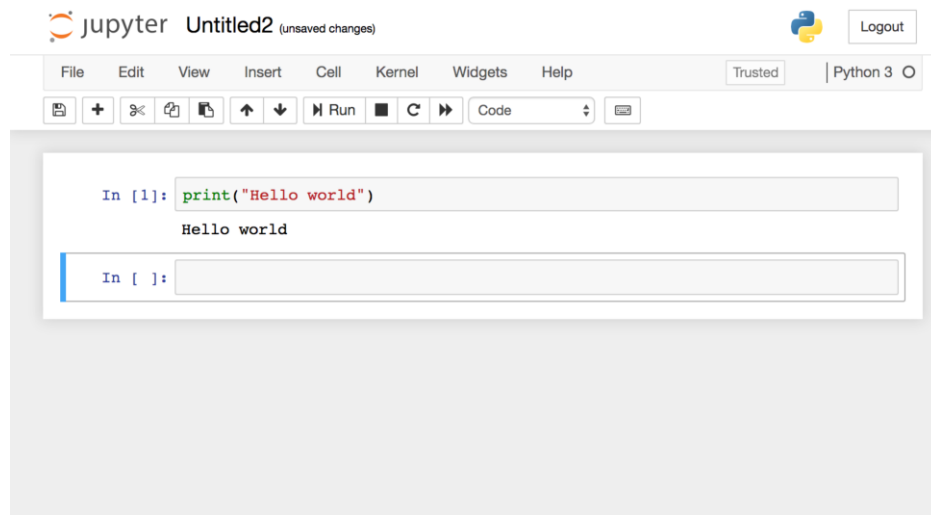


Figura 4. Jupyter Notebook.

- Finalmente, podemos construir uma interface gráfica, recorrendo à livreria *tkinter* (Figura 5):

<https://docs.python.org/3/library/tk.html>

```
from tkinter import *  
  
root = Tk() #Toplevel widget of Tk , main window of an app  
w=Label(root,text="Hello,world!")#Labelwidget  
w.pack()  
root.mainloop()
```

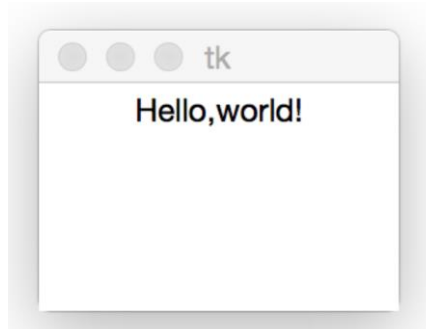


Figura 5. Tkinter.

3. Fundamentos da Linguagem

O Python é considerado uma linguagem interpretada, existindo dois modos de usar o interpretador: modo interativo (linha de comando) e modo de script (com base num editor).

Esta ficha tem como objetivo apenas o de fornecer os fundamentos básicos para a programação de algoritmos de ML em ambiente Python.

Sugere-se a utilização do tutorial disponível em learnpython.org e execução dos capítulos da secção “Learn the Basics”. Neste documento apresenta-se um resumo baseado no Jupyter notebook apoiado pela visualização de código no “Python Tutor”.

3.1 Declaração de Variáveis e Tipos

No Python não é necessário declarar variáveis, ou o seu tipo, antes de usá-las e sendo orientado a objetos, qualquer variável é considerada um objeto.

A linguagem suporta números inteiros, float e complexos. Abaixo ilustra-se a utilização de um inteiro e um float;

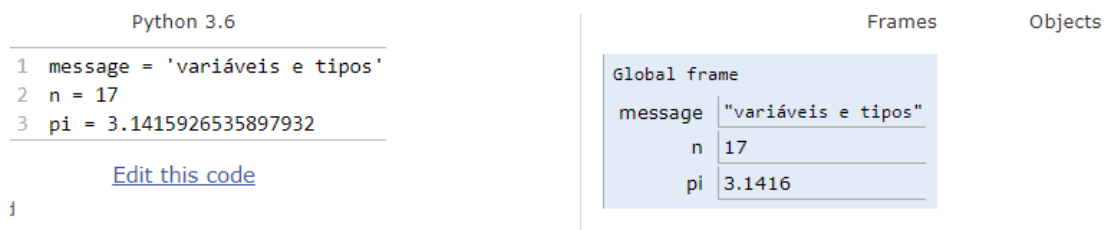


Figura 6. Declaração de variáveis numéricas (python tutor).

3.2 Strings

As cadeias de caracteres (strings) são definidas com aspas simples ou duplas.

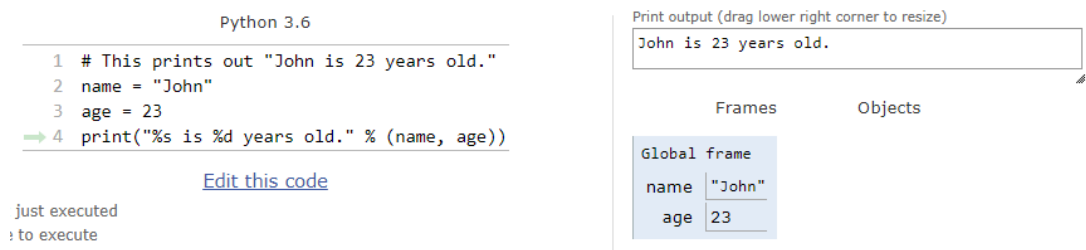


Figura 7. strings.

3.3 Operadores

Podemos aplicar os operadores básicos a números, strings e listas.



Figura 8. Operadores.

3.4 Funções

As funções em python são definidas usando a palavra-chave "def", seguida do nome da função:

```
def my_function():
    print("Hello From My Function!")

def my_function_with_args(username, greeting):
    print("Hello, %s , From My Function!, I wish you %s"%(username, greeting))

def sum_two_numbers(a, b):
    return a + b

# print(a simple greeting)
my_function()

#prints - "Hello, John Doe, From My Function!, I wish you a great year!"
my_function_with_args("John Doe", "a great year!")

# after this line x will hold the value 3!
x = sum_two_numbers(1,2)
```

Print output (drag lower right corner to resize)

```
Hello From My Function!  
Hello, John Doe , From My Function!, I wish you a great year!
```

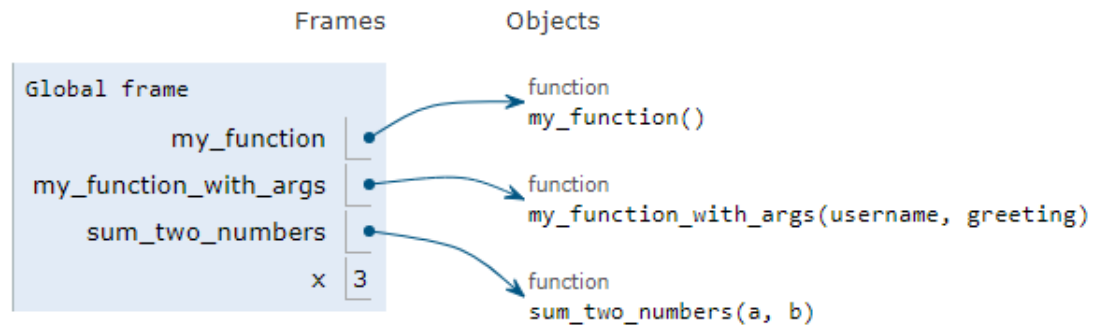


Figura 9. Funções.

3.5 Condições e Ciclos

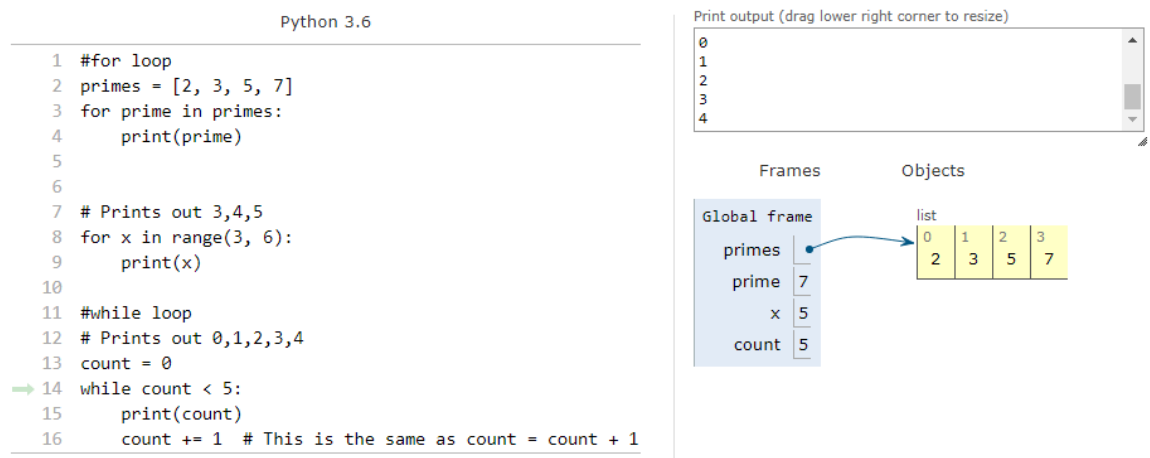


Figura 10. Ciclos for e while.

3.6 Listas

As listas são semelhantes às matrizes, podem conter qualquer tipo de variável e podem ser iteradas de forma simples:

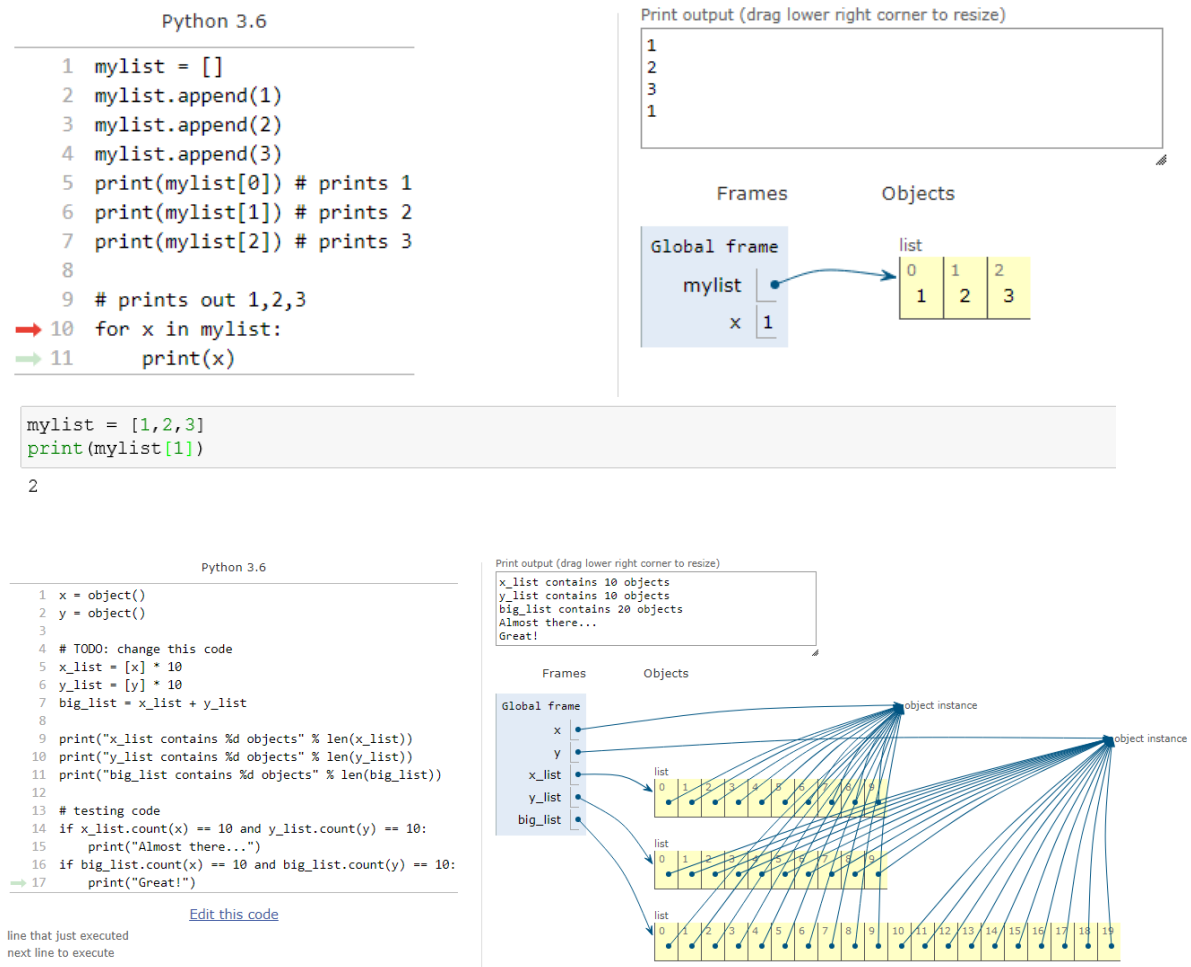


Figura 11. Operadores (python tutor).

3.7 Dicionários

Um dicionário consiste num tipo dados semelhante a uma matriz, no entanto funciona com 'chaves e valores' em vez de 'índices'. Cada valor armazenado num dicionário pode ser acedido através de uma chave, que por sua vez consiste em qualquer tipo de objeto (sequência, número, lista etc.) - a chave substitui o 'índice' habitual nas matrizes para aceder aos seus valores.

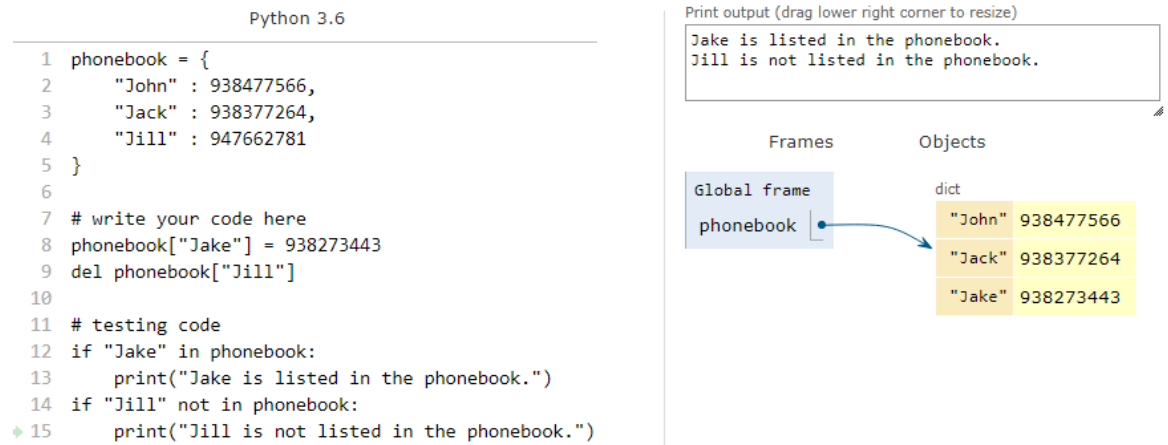


Figura 12. Dicionários.

3.8 Classes e Objetos

O Python é uma linguagem orientada a objetos, que representam um encapsulamento de variáveis e funções numa entidade.

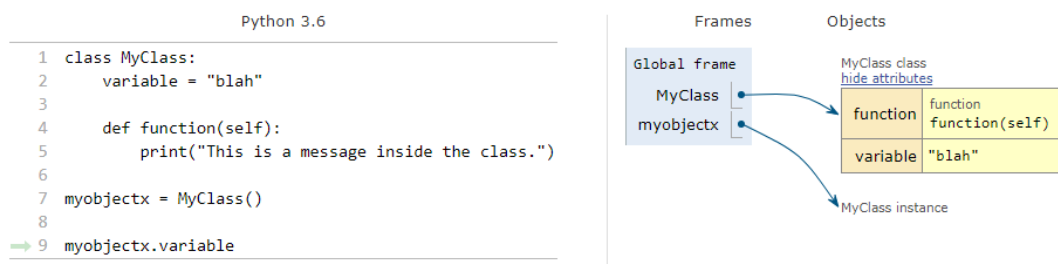


Figura 13. Classes e objetos.

4. Scikit-Learn

Esta biblioteca está incluída na distribuição Anaconda, no entanto poderá ser instalada, como qualquer outra *package* a partir do comando pip (*\$ pip install scikit-learn*).

Baseia-se na utilização de outras *packages* em particular:

4.1 NumPy

O Numpy é a das *packages* principais para cálculo científico em Python, oferecendo as funcionalidades para:

- Manipulação de matrizes multidimensionais;
- Funções matemáticas, tal como álgebra linear, transformadas de Fourier entre outras.

O núcleo funcional consiste na classe “ndarray”, que define, tal como o nome indica, um array de dimensão n. O scikit-learn manipula dados neste formato. Assim, quaisquer dados manipulados devem ser convertidos para um array numpy. Todos os elementos de um array devem ser do mesmo tipo (int, float, string, ...). Como exemplo, consideremos a definição de um array de inteiros 2*3:

```
In [1]: import numpy as np

In [2]: x = np.array([[1, 2, 3], [4, 5, 6]])

In [3]: print("x:\n{}".format(x))

x:
[[1 2 3]
 [4 5 6]]

In [ ]:
```

4.2 SciPy

Esta package consiste num repositório de funções para cálculo científico, incluindo:

- Álgebra linear;
- Otimização
- Processamento de sinal
- Análise estatística.

Como exemplo, considere-se a representação esparsa de matrizes bidimensionais, que permite a compactação e possível armazenamento em memória de matrizes “com muitos zeros” (consulte www.scipy-lectures.org).

```
In [7]: from scipy import sparse
md = np.eye(4)
print("NumPy array:\n{}".format(eye))

NumPy array:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

In [11]: #Conversão para uma matriz esparsa:
sparse_md = sparse.csr_matrix(md)
print("representação esparsa\n{}".format(sparse_md))

representação esparsa
(0, 0) 1.0
(1, 1) 1.0
(2, 2) 1.0
(3, 3) 1.0
```

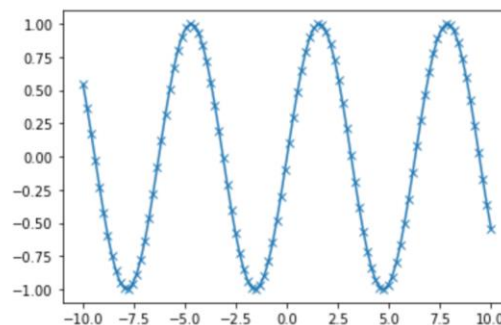
4.3 Matplotlib

A Matplotlib trata-se da livreria mais importante para visualização de gráficos ou histogramas.

```
In [12]: import matplotlib.pyplot as plt
# array de números de -10..+10 (100 passos)
x = np.linspace(-10, 10, 100)
# define vetor de valores com sin(x)
y=np.sin(x)
```

```
In [13]: #plot y vs x
plt.plot(x, y, marker="x")
```

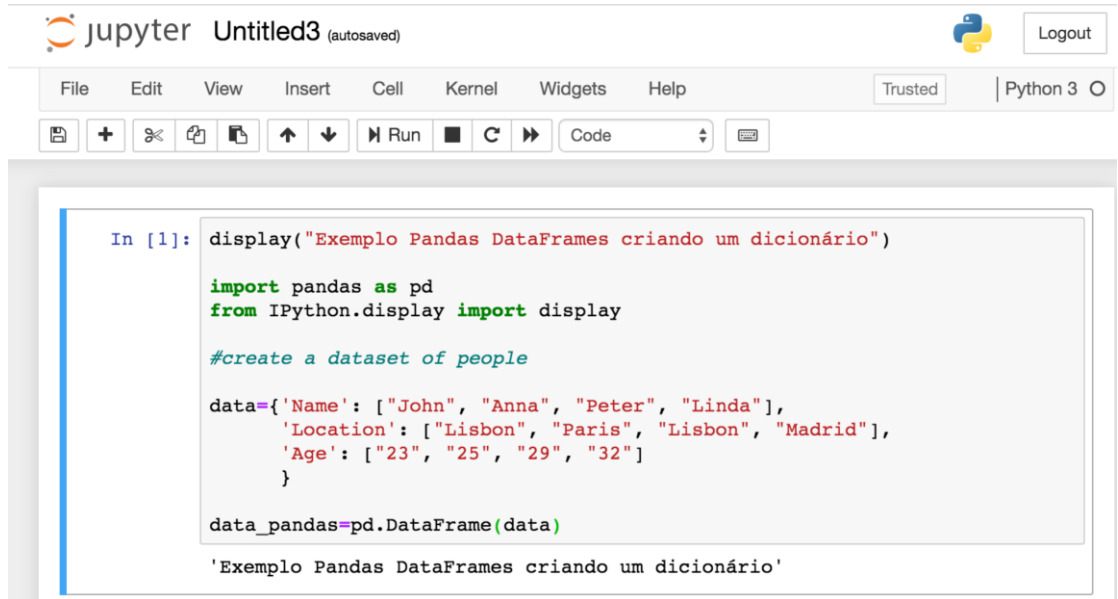
```
Out[13]: [<matplotlib.lines.Line2D at 0x1162d56d8>]
```



4.4 Pandas

Consiste numa livreria python para representação e análise de dados, centrada no conceito de “dataframe”. O conceito de dataframe foi inspirado na linguagem R e Matlab, e consiste essencialmente numa tabela “tipo excel”. Fornece também os métodos necessários para manipular estas tabelas, incluindo comandos similares a SQL. Permite (contrário do numpy) que uma na mesma coluna coexistem variáveis de diferentes tipos (inteiros, float, strings, date, etc..), Outra vantagem é a sua capacidade em consumir dados em formato SQL, Excel, CSV, entre outros.

Exemplo:



The image shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, undo, redo, and running code. The code cell contains the following Python code:

```
In [1]: display("Exemplo Pandas DataFrames criando um dicionário")

import pandas as pd
from IPython.display import display

#create a dataset of people

data={'Name': ["John", "Anna", "Peter", "Linda"],
      'Location': ["Lisbon", "Paris", "Lisbon", "Madrid"],
      'Age': ["23", "25", "29", "32"]}

data_pandas=pd.DataFrame(data)

'Exemplo Pandas DataFrames criando um dicionário'
```

Vamos selecionar os habitantes de Lisboa:

```
In [2]: print(data_pandas)
print("\n Habitants in Lisbon")
display(data_pandas[data_pandas.Location == "Lisbon"])
```

Name Location Age

0	John	Lisbon	23
1	Anna	Paris	25
2	Peter	Lisbon	29
3	Linda	Madrid	32

Habitants in Lisbon

	Name	Location	Age
0	John	Lisbon	23
2	Peter	Lisbon	29

e os habitantes com mais de 25 anos:

```
In [3]: print("\n Older then 25")
display(data_pandas[data_pandas.Age >= "25"])
```

Older then 25

	Name	Location	Age
1	Anna	Paris	25
2	Peter	Lisbon	29
3	Linda	Madrid	32

5. Exemplo – Classificação da Iris

Vamos aplicar as metodologias disponíveis no scikit-learn para classificação do conhecido benchmark “iris flower dataset”. O pipeline típico de um classificador consiste nas fases:

- Análise do Problema
- Carregar e Processar dados
- Formação dos conjuntos de treino, validação e teste
- Construção do classificador
- Avaliação do classificador

5.1 Carregar e processar os dados

5.1.1 Carregar os dados pré-definidos no sklearn

```
# load dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Keys of iris dataset: \n{}".format(iris_dataset.keys()))
print(iris_dataset['DESCR'][:193] + "\n...")

Keys of iris_dataset:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
.. _iris_dataset:

Iris plants dataset
-----

**Data Set Characteristics:**

: Number of Instances: 150 (50 in each of three classes)
: Number of Attributes: 4 numeric, pre

...

print("Target names: {}".format(iris_dataset['target_names']))

Target names: ['setosa' 'versicolor' 'virginica']

print("Feature names: \n{}".format(iris_dataset['feature_names']))

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

5.1.2 Analisar os atributos (features) e classes a representar (labels ou targets):

[illegible]

5.2 Preparar os dados para treino e teste

Com os valores usados, a função “train_test_split()” devolve 75% dos exemplos para treino e 25% para teste. As matrizes devolvidas são do formato “arrays NumPy”.

O método baralha o conjunto de dados usando um gerador de números pseudo-aleatório – “pseudorandom number generator”, de forma a garantir a representação das três classes nos conjuntos de treino e teste.

Com ‘random_state=0’, a divisão é feita de forma determinística (assim, com esta opção, gera sempre a mesma divisão).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))

print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

5.3 Análise do problema – Inspeção de dados

Antes de construir o classificador devemos inspecionar os dados de forma a entender a sua complexidade (linear ou não linear? Quantos parâmetros?) a definir para o classificador para separação de dados.

A melhor forma de inspeção consiste na sua visualização. Vamos recorrer à biblioteca Pandas e “plots” bidimensionais – “one feature against the others”. Esta abordagem é razoável quando o número de features é reduzido.

```
In [3]: import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris_dataset = load_iris()

X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

#transformar array numpy para pandas dataframe
#As colunas ficam com label=names (strings com nome da classe)

iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
```

```
In [5]: grr = pd.scatter_matrix(iris_dataframe, c=y_train,
                                figsize=(15, 15), marker='o', hist_kws={'bins': 20},
                                s=60, alpha=.8, cmap=mglearn.cm3)
```

5.4 Primeiro Classificador – KNN

Vamos construir um classificador básico, o K-NN. Classifica um exemplo de teste com base nas etiquetas atribuídas no conjunto de treino aos “k” vizinhos mais próximos. Não constrói propriamente um modelo, simplesmente armazena o conjunto de treino.

```
from sklearn.neighbors import KNeighborsClassifier

#K=1 - apenas um vizinho
knn = KNeighborsClassifier(n_neighbors=1)

#treina
knn.fit(X_train, y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
```

Podemos criar uma nova instância e verificar a classe atribuída:

```
#nova instância
X_new = np.array([[5, 2.9, 1, 0.2]])

#determina resposta do modelo
prediction = knn.predict(X_new)
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))

Predicted target name: ['setosa']
```

5.5 Avaliação do desempenho do Classificador

Podemos calcular a taxa de acerto para o conjunto de teste (conjunto independente, não usado para o treino):

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))

Test set score: 0.97

#podemos confirmar o resultado usando também o método score do classificador
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

Test set score: 0.97
```

Verificamos assim uma taxa de acerto de 97% para o conjunto de teste.

6. Trabalho Experimental

6.1 Iris

Considere o dataset “Iris”

- i) Para o classificador K-NN, calcule as métricas de “precisão” e “recall”. Determine igualmente a curva de ROC e AUC.
- ii) Treine e avalie uma rede neuronal MLP.
- iii) Treine e avalie um classificador SVM linear.

6.2 OptDigits

Considere o dataset “Optdigits” usado na segunda aula prática para classificação de dígitos.

Construa e avalie os seguintes classificadores:

- KNN;
- MLP;
- SVM linear e com funções gaussianas.

6.3 MNIST Digits Dataset

Considere o dataset “Digits Mnist” usado na segunda aula prática para classificação de dígitos.

- i) Construa e avalie os seguintes classificadores:
 - a. KNN;
 - b. MLP;
 - c. SVM linear.
- ii) Transforme o problema num classificador binário escolhendo o dígito ‘9’ como classe alvo.
 - a. Procure o classificador com melhor valor de AUC.
 - b. Que relação encontra entre o balanceamento dos dados (número de exemplos positivos vs negativos) e desempenho do classificador. No caso de dados não balanceados, a ‘accuracy’ poderá ser usada como uma métrica eficiente? Justifique.
- iii) Otimize o desempenho da uma SVM não linear – função gaussiana, determinado de forma automática os melhores valores para C e gamma (inverso da largura da função gaussiana).

Referências:

- Müller, Andreas C., and Sarah Guido. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.", 2016.
- Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2017.
- <https://greenteapress.com/wp/think-python/>
- Kenneth A. Lambert, The Fundamentals of Python: First Programs, 2011, Cengage Learning, ISBN: 978-1111822705.