

Uso de Machine Learning para previsão de doenças

Projeto Integrado - Engenharia Informática - UÉvora

Ana Sapata, 42255
José Azevedo, 45414
Raquel Lopes, 42075

October 14, 2019

1 Introdução

Este trabalho é realizado no âmbito da disciplina de Projeto Integrado, da licenciatura de Engenharia Informática da Universidade de Évora.

Como objetivos do mesmo pretende-se aprender a utilizar bibliotecas específicas do Python como o Numpy, Matplotlib e Scikit-Learning, utilizadas em projetos relacionados com Machine Learning. Sendo assim, o trabalho consiste na realização de um projeto na área de Machine Learning, mais concretamente na implementação do artigo Building meaningful machine learning models for disease prediction (<https://shiring.github.io/machine.learning/2017/03/31/webinar.code>) em Python, através das bibliotecas anteriormente referidas.

Primeiramente irão ser apresentados conceitos básicos de Python, bem como das bibliotecas a utilizar, passando de seguida à implementação do projeto.

2 Python

2.1 Conceitos Básicos

- Definição de variáveis:

```
<nome_variavel> = <valor>
```

Exemplos:

```
string = "Olá!"  
inteiro = 4  
decimal = 3.14
```

- Arrays

Os arrays são criados com parentesis retos [], estando os elementos do mesmo dentro destes separados por vírgulas.

```
<nome_array> = [elem1, elem2, ..., elemN]
```

Exemplo:

```
array = [1, 2, 3, "a dog"]
```

- Adicionar elementos

Para adicionar elementos a um array utiliza-se o método append, seguido do elemento que se pretende adicionar.

Exemplo:

```
array.append(5)  
print(array)  
[1, 2, 3, "a dog", 5]
```

- Selecionar elementos de um array

Para se selecionar elementos de um array, utiliza-se o índice do elemento que pretendemos selecionar dentro de parentesis retos, a seguir ao nome do array.

Exemplo:

```
#Selecionar a string "a dog" do array, sendo o seu indice 3  
print(array[3])  
a dog
```

```
#selecionar o numero 3 do array, sendo o seu indice 2, e atribuir
#o mesmo à variável num
num = array[2]
print(num)
3
```

- Eliminar elementos de um array

Para eliminar elementos de um array é utilizada a função `del` sendo dado como o seu parametro o nome do array seguido do indice do elemento que se pretende eliminar entre parentesis retos `[]`.

Exemplo:

```
#queremos apenas que o array contenha numeros, pelo que se irá eliminar
#a string "a dog" contido no indice 3
del(array[3])
print(array)
[1, 2, 3, 5]
```

- Adicionar elementos num indice especifico

Para se adicionar elementos ao array, num indice especifico é utilizado o método `insert` aplicado ao array seguido do indice onde se pretende adicionar e o elemento que se pretende adicionar no mesmo.

```
<nome_array>.insert(<indice>, <elemento>)
```

Exemplo:

```
#inserir novamente a string "a dog" no array no indice 3,
#onde estava anteriormente
array.insert(3, "a dog")
print(array)
[1, 2, 3, "a dog", 5]
```

- Selecionar um subconjunto do array

```
<nome_array>[indice_inicial:indice_final]
```

Exemplo:

```
array[2:4]
[3, "a dog"]
```

- Ciclo for

```
for <condição>:
    <desenvolvimento do que fará quando se verifica a condição>
```

Exemplo:

```
for x in range (0,3):
    print "We're on time" + x
```

Quando x pertence ao intervalo `[0;3[` (`range(a,b)` cria o intervalo/sequencia `[a.b]`) escreve/imprime a frase `We're on time x`, substituindo x pelo respetivo valor

- Ciclo while

```
for <condição>:
    <desenvolvimento do que fará quando se verifica a condição>
```

Exemplo:

```
x = 1
while x<3:
    print "We're on time" + x
    x += 1
```

Enquanto x for inferior a 3, imprime We're on time x, substituindo x pelo respetivo valor, incrementando de seguida 1 ao mesmo

- Condicionais if/else

```
if <condição>:
    <o que faz se condição verdade>
else:
    <o que faz se condição falsa>
```

Exemplo:

```
num = 3
if num >= 0:
    print("Positivo ou zero")
else:
    print("Negativo")
```

Se a variavel num for maior ou igual que zero imprime a frase "Positivo ou zero", caso contrário imprime a frase "Negativo"

- Definição de funções:

```
def <nome> (<args>):
    <corpo da função>
    termina com return
```

Exemplo:

```
def soma (a, b):
    return a + b
```

2.2 Numpy

URL Tutorial:

<https://numpy.org/devdocs/user/quickstart.html>

Para utilizar a biblioteca numpy comea-se por utilizar o comando

```
#importação da biblioteca numpy passando a ser denominado por np daqui para a frente
import numpy as np
```

Podem criar-se matrizes de zeros ou com os elementos que pretendemos. Para se criar uma matriz de zeros com n linhas e m colunas é utilizado o comando

```
x = np.zeros((n, m))
```

Para se criar uma matriz com os elementos pretendidos, é utilizada a função array da biblioteca numpy, sendo dados como argumentos as linhas da matriz.

Exemplos:

```
#array de zeros com 2 linhas e 3 colunas
x = np.zeros((2, 3))
print(x)
[[0. 0. 0.]
 [0. 0. 0.]]

y = np.array([[1, 2], [0,3.2], [1, 7]])
print(y)
[[1. 2. ]
 [0. 3.2]
 [1. 7. ]]
```

Para se saber as dimensões da matriz aplica-se o método shape à matriz em questão.

```
#a matriz x   composta por 2 linha e 3 colunas
x.shape
(2,3)

#a matrix y   composta por 3 linhas e 2 colunas
y.shape
(3,2)
```

Se pretendermos saber o número total de elementos existentes na matriz aplica-se o método size à mesma

```
x.size
6

y.size
6
```

Para além de matrizes também é possível a criação de arrays, uma vez que estes são considerados matrizes com apenas uma linha, como é possível verificar no seguinte exemplo

```
a = np.array([2, 3, 4])
print(a)
[2 3 4]
```

Sendo assim possível a construo com números num determinado intervalo, para tal é utilizada a função arange que recebe como argumentos o valor inicial do intervalo, o valor final e o passo utilizado entre cada elemento.

Exemplo:

```
#array cujos elementos começam no valor 10 e terminam no 25, uma vez que o 30 já
#não irá entrar no intervalo, variando de 5 em 5
b = np.arange(10, 30, 5)
print(b)
[10 15 20 25]
```

Também é possível dizer apenas os valores iniciais e finais que pretendemos e o número de elementos que irão constituir o array, usando para tal a função linspace.

Exemplo:

```
#array cujos elementos começam no valor 0 e terminam no 2, sendo o mesmo composto
#por 9 elementos
c = np.linspace(0, 2, 9)
```

```
print(c)
[0. 0.25 0.5 0.75 1. 1.25 1.5 1.75 2.]
```

2.3 Pandas

URL Getting Started:

https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html

Tutorial Pandas, implementar

2.4 Matplotlib

URL Tutorial:

<https://matplotlib.org/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py>

2.5 scikit-learn

URL Tutorial:

<https://scikit-learn.org/stable/tutorial/index.html>

3 Uso de Machine Learning para previsão de doenças

Os dados utilizados no projeto podem ser obtidos a partir do repositório de Machine Learning da UCI (<http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>), estando os mesmos relacionados com o diagnostico de cancro.

3.1 Tratamento e análise dos dados

Inicialmente comeou por se fazer o download dos dados, e transformar os mesmo no formato de data frame, de modo a facilitar a sua posterior análise.

Para os dados serem carregados no python e ao mesmo tempo transformados numa data frame, foi utilizado o comando *read_csv* e guardado o resultado na variavel *df*.

```
#importação da biblioteca pandas, quando for necessário usar a mesma irá ser utilizado
#pd em vez de pandas
import pandas as pd

# Leitura do ficheiro dos dados, especificando que o mesmo não tem nome para as colunas
#(header = None)

# Comando read_csv da biblioteca Pandas o equivalente ao read.table do R, uma vez que
# temos o ficheiro em formato csv
df = pd.read_csv('/home/anasapata/Personal/ProjetoIntegrado/Uso-de-Machine-Learning
-para-previs-o-de-doen-as/breast-cancer-wisconsin.data.csv',
                header = None)

# Mostra as primeiras 5 linhas do ficheiro/data frame
print(df.head())
```

Após guardados os dados na variavel *df*, uma vez que as colunas da data frame não tinha qualquer nome associado foi necessário atribuir os respetivos nomes a estas.

```
# Uma vez que o ficheiro não tem nome para as colunas, tal como acontece
# posteriormente com a data frame então necessário atribuir os respetivos
# nomes às mesmas para tal
df.columns = ['sample_code_number',
              'clump_thickness',
              'uniformity_of_cell_size',
```

```

        'uniformity_of_cell_shape',
        'marginal_adhesion',
        'single_epithelial_cell_size',
        'bare_nuclei',
        'bland_chromatin',
        'normal_nucleoli',
        'mitosis',
        'classes']

# Mostrar novamente as primeira 5 linhas de modo a confirmar que os nomes das
# colunas lhes foram atribuidos
print(df.head())

```

De seguida foi tratada a coluna *classes* de modo a ter o valor "benign" quando esta era 2, "malignant" quando era 4 e NA nos restantes casos.

```

# Quando classes tem o valor 2 deverá torna-se "benign", quando tem o valor 4
# deverá tornar-se "malignant" e nos restantes casos NA
df.classes.replace([2, 4], ['benign', 'malignant'], inplace = True)

# Verificar que alterou os valores
print(df.head())
print(df.tail())

```

Uma vez organizada a coluna *classes*, pretende-se agora verificar a existência de valores NA. Sabe-se que existem células cujo seu valor é '?' pelo que primeiramente estes valores terão de passar a NA. O equivalente ao NA na biblioteca numpy é o NaN, tendo então utilizado este valor quando as células tinham o valor '?'.

```

import numpy as np

# Quando existe o valor ? atribuido ao mesmo o valor NaN (equivalente ao NA)
df.replace('?', np.NaN, inplace = True)

```

Após efetuada a alteração, verificou-se quais as colunas que continham valores NaN e quantas linhas existiam com os mesmos.

```

# Verifica quais as colunas com valores nulos
null_columns = df.columns[df.isnull().any()]

# Conta o nmero de celulas com valores nulos
print(df[null_columns].isnull().sum())

```
