O objetivo deste trabalho é implementar diversas métricas com algumas aplicações elementares. Vamos considerar dois tipos de dados: numéricos e binários.

1 Dados numéricos

O ficheiro `cars.csv` é constituído por 8 atributos numéricos e notamos por $\mathcal{A} = A_1 \times \cdots \times A_8$ o espaço dos atributos enquanto $D = (x^n)_{n=1,\cdots,N}$ representa o conjunto de dados, $x_i^n \in A_i = \mathbb{R}$. Para ler os dados, usamos a biblioteca `pandas` da maneira seguinte

```
import scipy.spatial.distance as dist
import pandas as pd
import numpy as np
#-----------------
cars = pd.read_csv('../csv/cars.csv',sep=',')
A=np.array(cars.values[:,1:8],dtype=np.float)
```

A é uma matrix onde `A[n]` corresponde ao vetor de atributos do elemento n da base de dados.

1.1 Implementação das distâncias

Vão implementar em funções distintas as distâncias e dessemelhanças seguintes: euclidiana, Manhattan, Chebyshev, Canberra e cosine.

```
def Euclidian(x,y):
    ...
    blablabla
    ...
    return distance
```

onde `x,y` são do tipo `numpy array` e retorna o valor `distance`, a distância entre os dois vetores. Testar cada função com os vetores `A[0], A[1]`. Comparar os seus resultados com as métricas dadas pela biblioteca `scipy`. Por exemplo `dist.euclidean(A[0],A[1])` retorna a distância euclidiana.

1.2 Visualizar uma bola

Depois da fase de validação, vamos determinar e visualizar a bola centrada no ponto x^1 =`A[0]` e de raio `r`. Recordamos que esta bola corresponde aos elementos de D tais que

$$B = \{x^n \in D,\ d(x^1, x^n) < r\}$$

onde a distância $d(x, x')$ depende da escolha da métrica.
Para simplificar e permitir uma visualização tridimensional, vamos tratar os dados **unicamente com três atributos**. Para isso, definimos `B=A[:,[3,4,5]]` que reduz a dimensão do espaço dos atributos para três componentes.

1. Implementar uma rotina que determine as componentes dos vetores `xe[],ye[],ze[]` que correspondem aos pontos da bola de raio `re` usando a distância euclidiana. Visualizar num mesmo gráfico, em azul claro todos os dados e em vermelho apenas os dados que pertencem à bola.

```
fig = plt.figure()
ax = fig.add_subplot(221, projection='3d')
ax.scatter(B[:,0],B[:,1], B[:,2], c='b', marker='.')
ax.scatter(xe, ye, ze, c='r', marker='o')
```

Escolher $re = 100$ no exemplo.

2. Fazer o mesmo trabalho mas com as outras dessemelhanças: $rm = 100$ para Manhattan, $r = 0.2$ para Canberra e $r = 0.00001$ para o cosine.

2 Dados binários

O ficheiro `SCADI.csv` representa um exemplo de dado do ICF-CY: *International classification of functioning, disability and health : children & youth version* onde cada atributo indica a presença (1) ou ausência (0) de uma capacidade da criança. O espaço dos atributos é $\mathcal{A} = \{0, 1\}^{206}$ que obtemos como

```
import scipy.spatial.distance as dist
import pandas as pd
import numpy as np
#SCADI = pd.read_csv('../csv/SCADI.csv',sep=',')
A=np.array(SCADI.values[:,2:-1],dtype='int32')
```

A matriz A é constituidas dos vetores `A[n]` correspondentes aos eventos $n = 1, \cdots, N$. Por exemplo, o vetor `A[1]` contem os 206 valores binarios associados ao evento 1.

2.1 Implementação das distâncias

As dessemelhaças que usamos são todas baseadas na matriz de confusão `CM` que contabiliza quatro tipos de situações. As linhas (primeira componente) representa a realidade enquanto as colunas (segunda componente) representa a predicção. Logo temos `CM[1,1]` true positive, `CM[0,0]` true negative, `CM[0,1]` false positive, `CM[1,0]` false negative. Implementar uma função que retorna a matriz de confusão entre dois vetores binarios.

```
# convention: CM[1,1]=yes-yes, CM[0,1]=no-yes, CM[1,0]=yes-no,CM[0,0]=no-no
def confusion_matrix(x,y):
    ...
    blablabla
    ...
    return CM
```

onde `x` é o vetor considerado como correto (true ou false) enquanto `y` é o vetor de predicção (positive, negative).
A partir da matriz de confusão, construir as distâncias seguintes: matching (ou Hamming), Dice, Jaccard e Sokal-Michener usando o formato seguinte

```
def matching_distance(CM):
    ...
    return distance
```

onde `CM` é a matriz de confusão.

2.2 Path-based clustering

Um caminho $l(x, y) = \{z^0, z^1, \cdots, z^K\}$ entre dois pontos x, y de D é uma lista ordenada de elementos de D tal como $x = z^0$ e $y = z^K$. Para qualquer distância (ou métrica) d, notamos por $|l(x, y)|$ a distância do caminho como a distância máxima entre dois pontos sucessivos,

$$|l(x, y)| = \max_{k=1}^{K} d(z_{k-1}, z^k).$$

Seja $\varepsilon > 0$ um parâmetro positivo. Dois pontos x, y fazem parte do mesmo cluster C se existe um caminho $l(x, y)$ tal que $|l(x, y)| < \varepsilon$. Por outras palavras, existe uma sucessão de pontos de D, de distância sucessiva inferior a ε que liga x para y. **Neste trabalho,usamos** $\varepsilon = 0.5$.

Na base desta definição, construimos uma partição do conjuntos de dados em subconjuntos (ou path-based clusters). O objetivo desta parte é construir e identificar os clusters com os dados do SCADI.

1. Construir a matriz de dessemelhança `DT`:

```
def build_dissimilarity_table(A):
    ...
    blablabla
    ...
    return DT
```

   cujas entradas `DT[n,m]` são as distâncias entre x^n e x^m.

2. Seja $\varepsilon > 0$ dado, construir a matriz de adjacência (adjacency Matrix) M de entradas `M[n,m]=1` se `DT[n,m]`$< \varepsilon$, 0 senão.

3. Para ligar os elementos de um mesmo cluster entre eles, precisamos de um algoritmo de tipo: "the friends of my friends are my friends", o que se chama em Matemática *transitive closure*. Basicamente, precisamos de construir uma matriz constituida por 0 e 1 que indica quais são os elementos ligados entre si via um caminho de distância inferior a ε. O algoritmo (não otimal) seguinte realiza esta operação

```
def transitive_closure(M):
    n=M.shape[0]
    for i in range(n):
        for j in range(i+1,n):
            if M[i,j]:
                for k in range(n):
                    if M[i,k]:
                        M[j,k]=1
                        M[k,j]=1
    for i in range(n):
        ii=n-i-1
        for j in range(i+1,n):
            jj=n-j-1
            if M[ii,jj]:
                for k in range(n):
                    if M[ii,k]:
                        M[jj,k]=1
                        M[k,jj]=1
    return M
```

   Como deduzir os clusters com esta rotina?

4. A última coluna do ficheiro `SCADI.csv` contem a classe associada a cada evento, com $C = \{1, \cdots, 7\}$, onde os valores representam o nível de dependência de uma criança.
   - para cada cluster determinar a sua média de dependência.
   - Realizar uma função que retorna a estatística de cada cluster, seja o número de eventos associados a cada nível de dependência. Como definir a classe de um cluster?