Application des ondes EEG

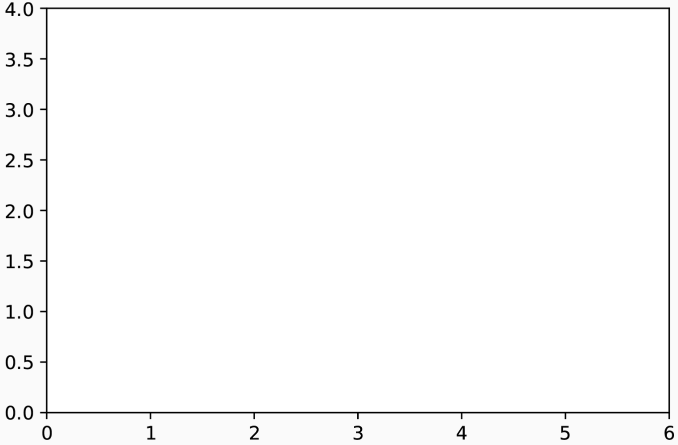
avec le Machine Learning

Quentin PETIT

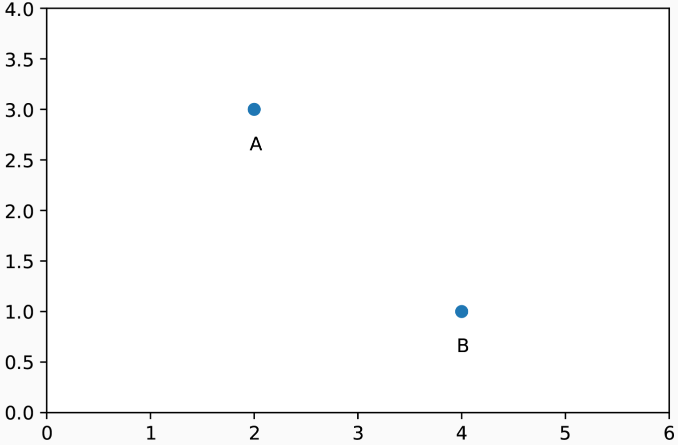
April 25, 2021

2 - Algorithme des k plus proches voisins

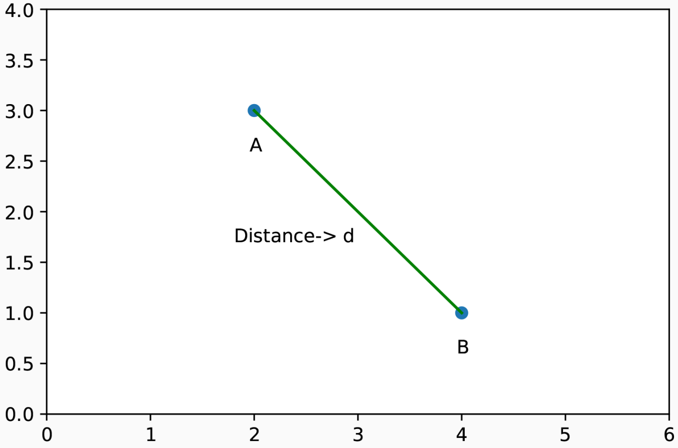
Mesurer une distance - Cas en 2 dimensions



Mesurer une distance - Cas en 2 dimensions

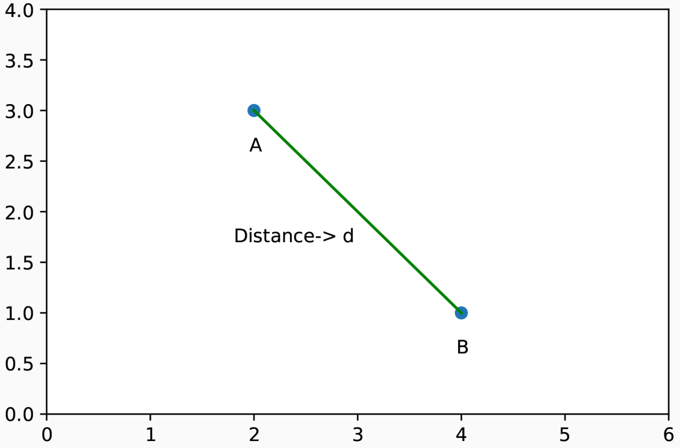


Mesurer une distance - Cas en 2 dimensions



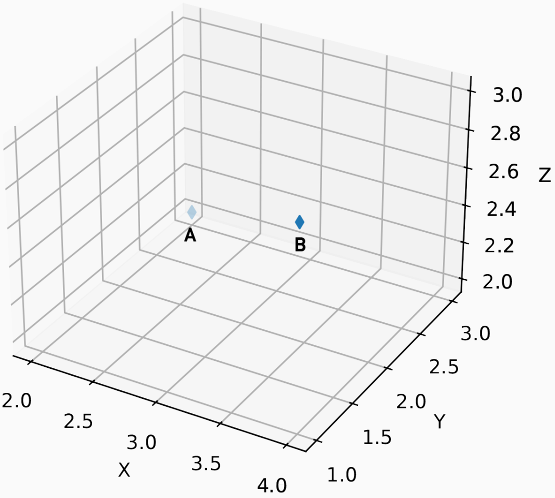
Soit A(2,3) et B(4,1), $d(A, B) = \sqrt{(2-4)^2 + (3-1)^2} = 2\sqrt(2)$

Mesurer une distance - Cas en 2 dimensions

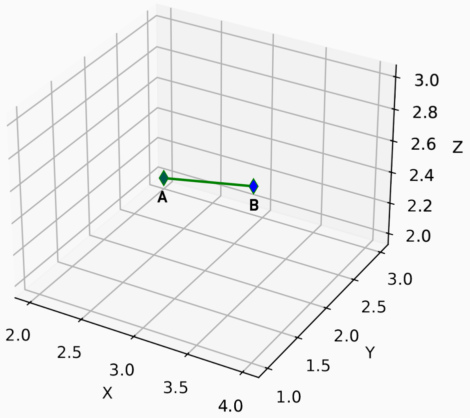


Soit A(x_1, y_1) et B(x_2, y_2), $d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Mesurer une distance - Cas en 3 dimensions



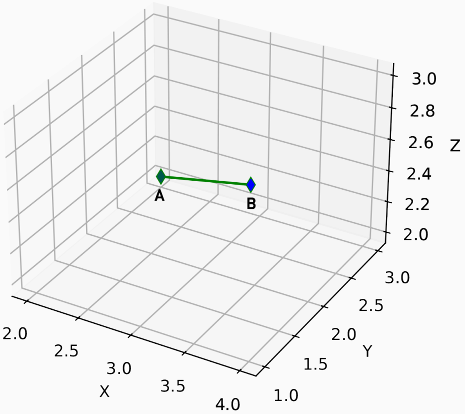
Mesurer une distance - Cas en 3 dimensions



Soit A(2,3,2) et B(4,1,3),

$$d(A,B) = \sqrt{(2-4)^2 + (3-1)^2 + (2-3)^2} = 3$$

Mesurer une distance - Cas en 3 dimensions



Soit A(x_1, y_1, z_1) et B(x_2, y_2, z_2),

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Distance de Minkowski

Distance de Minkowski

Soit p fixé, p>=1, cette distance est définie par

$$d_p : \mathbb{R}^n * \mathbb{R}^n \to \mathbb{R}$$

$$\overrightarrow{u}, \overrightarrow{v} \mapsto \sqrt[p]{\sum_{k=1}^{n} |x_k - y_k|^p}$$

Cas particulier:

La distance euclidienne est le cas où q=2:

$$d_2(\overrightarrow{u}, \overrightarrow{v}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

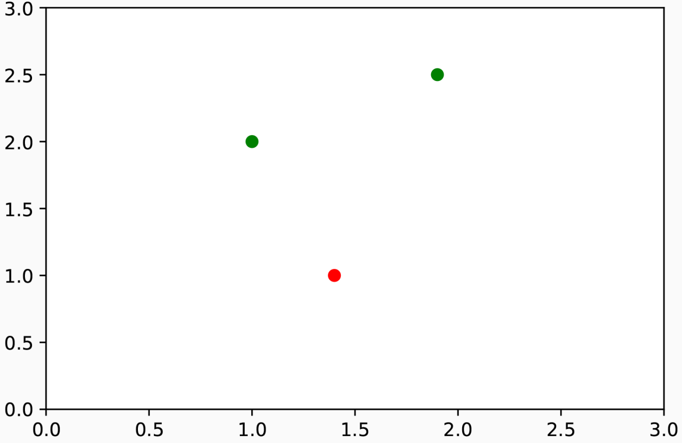
Distance euclidienne

En Python:

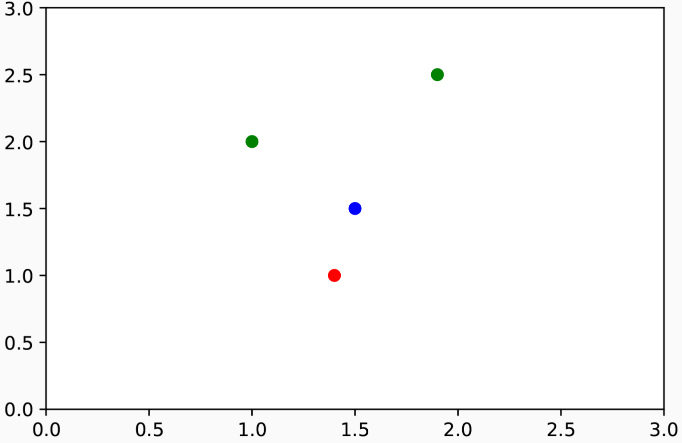
```python

    def distance_euclidienne(l1, l2):
        # Initialisation de la distance
        distance = 0
        # On calcule la distance euclidienne à n dimension
        for k in range(len(l1)-1):
            distance += (l1[k] - l2[k])**2
        return sqrt(distance)
.
```

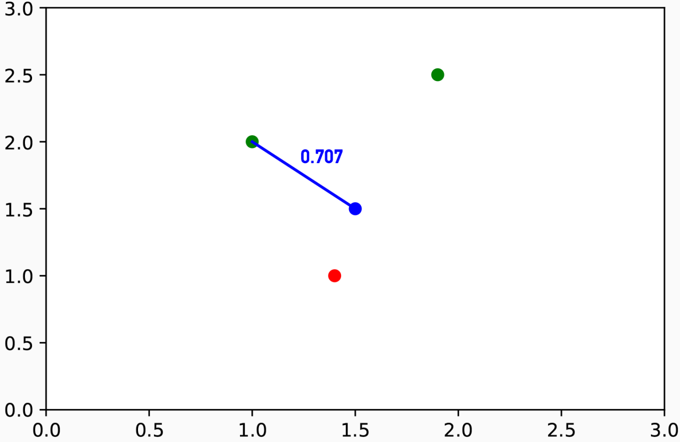
Algorithme du plus proche voisin



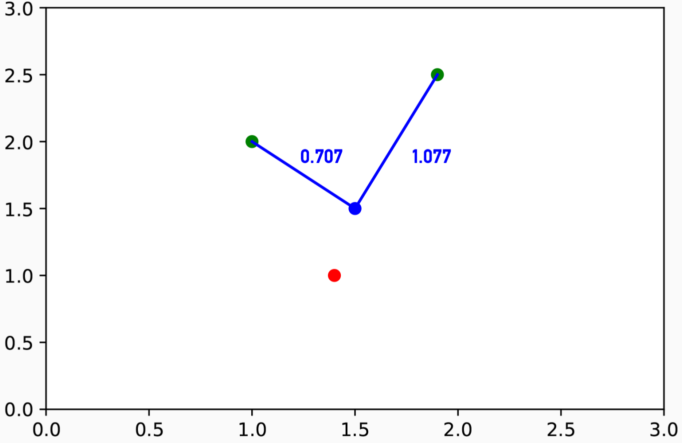
Algorithme du plus proche voisin



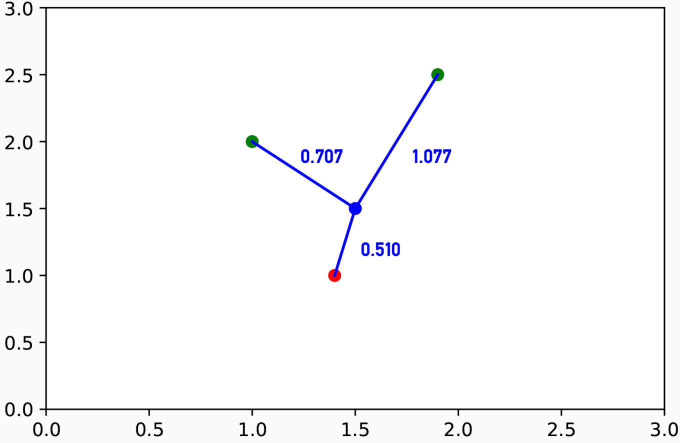
Algorithme du plus proche voisin



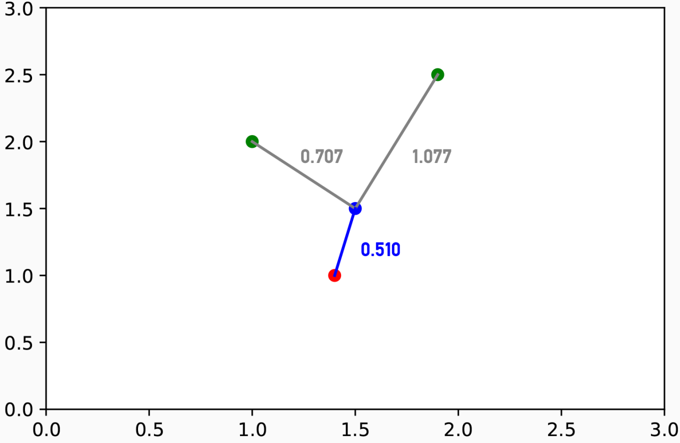
Algorithme du plus proche voisin



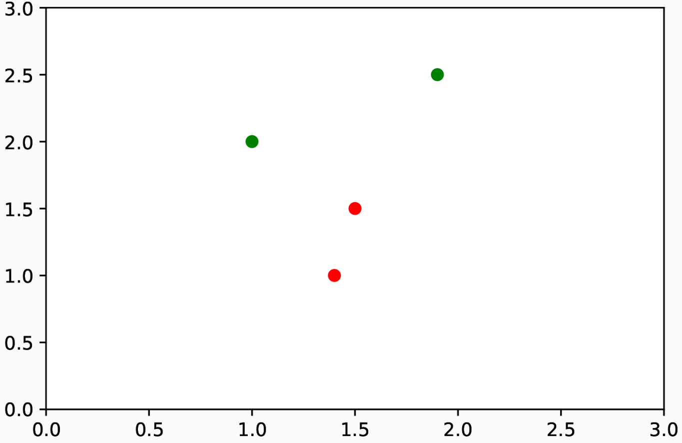
Algorithme du plus proche voisin



Algorithme du plus proche voisin



Algorithme du plus proche voisin



Algorithme du k plus proche voisin

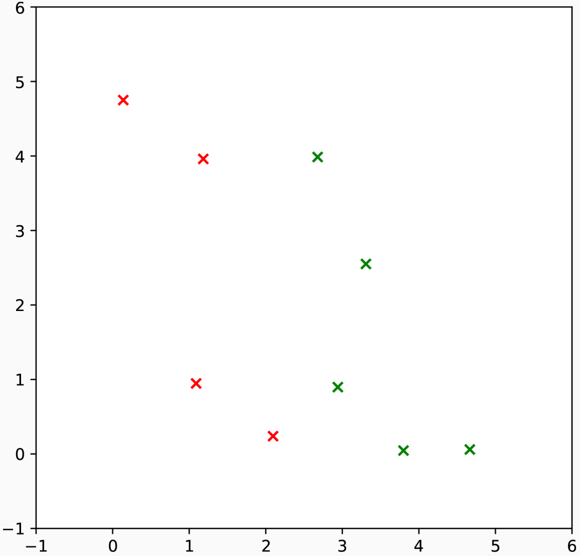
k plus proche voisin

Soit un jeu de données D $= (\overrightarrow{x_i}, y_i)_{1 \leq i \leq n}$ de quantité n, l'algorithme du k plus proche voisin assigne à une nouvelle situation l'étiquette du point le plus proche

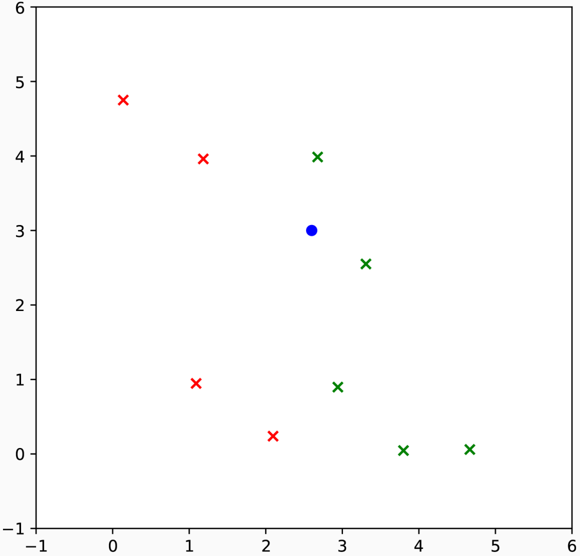
$$f(\overrightarrow{x}) = y_k$$

où k est la classe du vecteur $\overrightarrow{x_i}$ qui réalise le $min(d(\overrightarrow{x_i}, \overrightarrow{x}))$

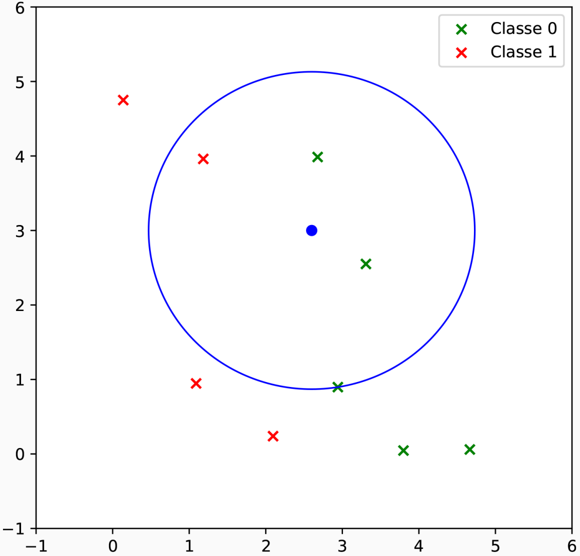
k plus proches voisins



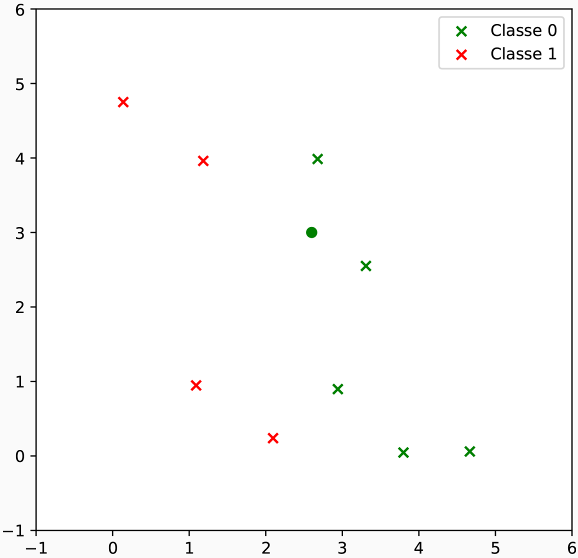
k plus proches voisins



k plus proches voisins - k=4



k plus proches voisins



k plus proches voisins

```python

def prochesVoisins(train, testL, k):
    # Initialisation de la distance qui contiendra après
    # la boucle, les distances euclidiennes entre
    # train et testL
    distances = []
    for trainL in train:
        distance = distance_euclidienne(testL, trainL)
        distances.append([trainL, distance])
    # On tri la liste dans l'ordre croissant par rapport
    # au deuxieme élément de la liste ex:
    # [(5,6),(10,3),(5,5)] devient [(10, 3), (5, 5), (5, 6)]
    distances.sort(key=lambda lis: lis[1])
    voisins = []
    # On stocke pour k voisins, la classe du plus petit
    # k correspondant dans la liste distances
    for i in range(k):
        voisins.append(distances[i][0])
    return voisins
.
```

k plus proches voisins

```

    def classification(train, testL, k):
        # On récupere l
        voisins = prochesVoisins(train, testL, k)
        sortie = [l[-1] for l in voisins]
        prediction = max(sortie, key=sortie.count)
        return prediction
.
```

Mise en pratique - données brutes

Delta	Theta	Low-Alpha	High-Alpha	Low-Beta	High-Beta	Low-Gamma	Mid-Gamma	Classe
524	504	357	75	469	481	259	254	1
355	154	445	538	416	221	351	218	1
732	494	729	228	341	273	282	102	1
472	351	282	545	466	284	272	183	1
355	776	181	368	281	195	85	127	1
121	256	374	505	298	191	334	240	1
641	414	476	378	410	409	230	116	1
132	545	296	456	340	194	387	50	1
276	501	534	325	195	391	286	252	1
180	1585	966	1767	379	315	381	307	1

Mise en pratique - traiter

```

    # Importer le fichier csv
    dataset = csv("data.csv")
    # Supprimer le header
    del dataset[0]
    # Transformer les strings en entier
    dataset= [[int(float(j)) for j in i] for i in dataset]
    # Prédiction
    k=3
    x=[3070,340,438,273,620,427,284,466,0]
    print("La prédiction est", classification(dataset,x,k))
    #-> La prédiction est 0
.
```

Evaluer notre algorithme

- Zero

Comparaison avec la librairie scikit-learn

```

    # Importation des librairies
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    from sklearn.neighbors import KNeighborsClassifier

    # Importation des données du casque
    data = pd.read_excel('data.xlsx')
    data.head()

    # Parsing des données
    y = data['Classe']
    X = data.drop('Classe', axis=1)
.
```

Comparaison avec la librairie scikit-learn

```python

    # Creation du KNN
    model = KNeighborsClassifier(3)
    model.fit(X,y)
    model.score(X,y)

    # Donnée que l'on veut classer
    x = np.array([329,415,243,153,393,317,260,123])
    .reshape(1,8)

    # Affichage de la prediction & probabilité
    model.predict(x)
    print(model.predict(x))
    print(model.predict_proba(x))
.
```