Application des ondes EEG

avec le Machine Learning

Quentin PETIT

June 29, 2021

Problématique

Peut-on aider un handicapé moteur à acquérir une certaine autonomie ?

Sommaire

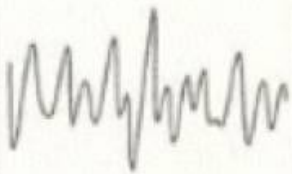
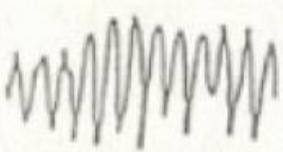
1 - Analyse du signal électroencéphalographique

Signal cérébral - Général

Ondes électroencéphalographiques

Fluctuation du potentiel électrique entre différentes parties du cerveau qui est mesuré à l'aide d'un EEG

Chaque bande de fréquence décrit un type d'onde:

DELTA	THETA	ALPHA	BETA	GAMMA
Inférieure à 4Hz	4 à 8 Hz	8 à 13 Hz	13 à 35 Hz	Supérieure à 35Hz
Sommeil profond	Somnolence	Relaxation	Une attention à son maximum	Exitation
				

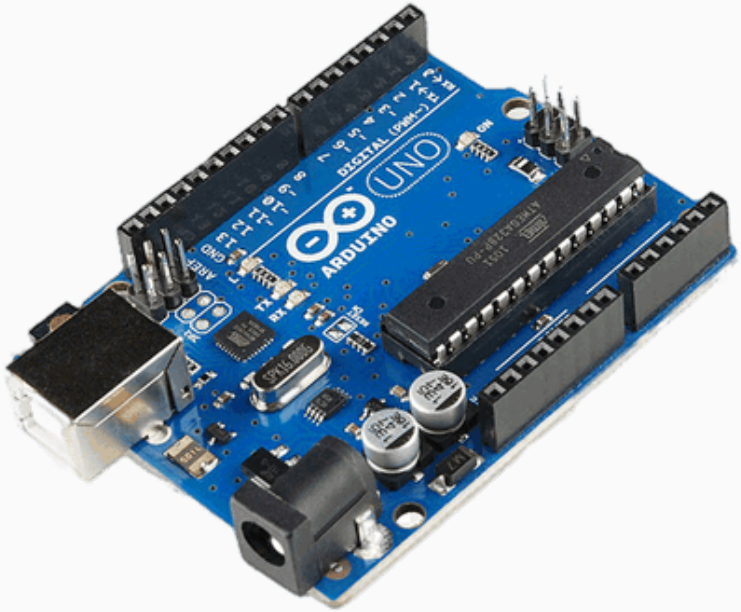
Ondes EEG - Notre cas

Neurosky Mindwave mobile 2 qui va nous permettre de mesurer l'activité cérébrale du cerveau

8 types d'ondes envoyées par le casque NeuroSky

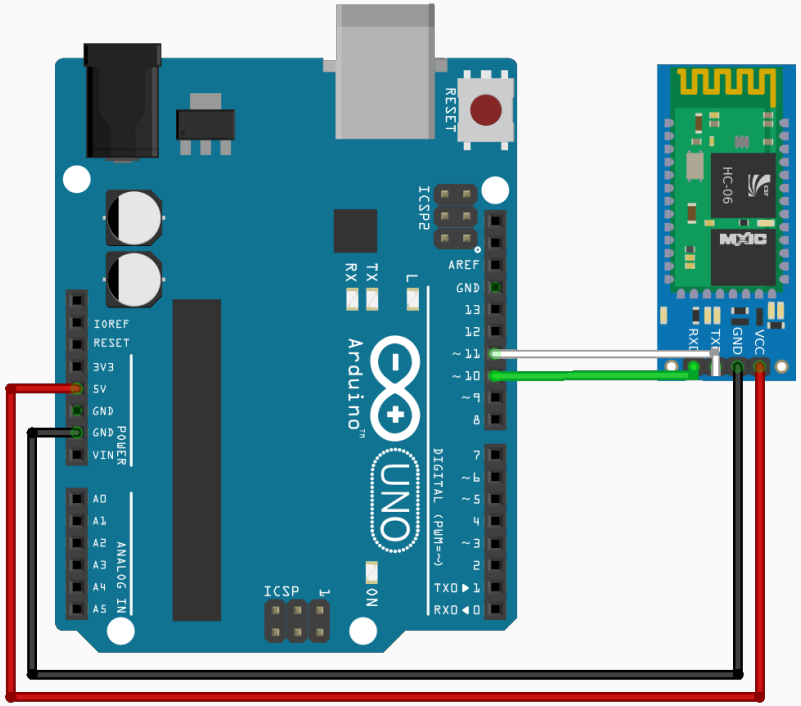
2 - Communication entre appareils - IOT

Réception des données



Arduino qui permettra la liaison entre le casque et le fauteuil

Module bluetooth



Liaison entre un arduino et un module bluetooth

Commandes AT

Commandes AT ou de Hayes

Un langage de commandes permettant la communication entre modems

Par exemple:

AT+NAME = "moduleNeuroSky" - Définir le nom du module
AT+UART = "115200" - Définir la valeur du baud
AT+ROLE = "1/0" - Définir le rôle du module
AT+PSWD = "1234" - MDP de connexion au NeuroSky
AT+BIND = "c464e3e8d699 - Adresse MAC du NeuroSky

Paraméter les commandes AT

```
1
2     #include <SoftwareSerial.h>
3
4     SoftwareSerial hc05(10, 11);
5
6     void setup() {
7       // Initialiser le Serial Monitor
8       Serial.begin(115200);
9       Serial.println("Commandes AT:");
10      // Initialiser le port Bluetooth
11      hc05.begin(9600);
12    }
13
14    void loop() {
15      // Transferer les donnees du HC05 vers le SM
16      if (hc05.available()) {
17        Serial.write(hc05.read());
18     }
19    }
20
```

Données envoyées par le casque

Exemple de 3 mesures faites par le casque:

170,170,2,0,131,24,0,34,157,0,17,204,0,22,83,0,61,215,0,38,69,0,39,147,0,
27,207,0,13,143,4,83,5,75

170,170,2,0,131,24,1,87,254,1,171,58,0,54,34,0,62,163,0,41,112,0,35,139,0,
22,202,0,17,207,4,78,5,67

170,170,2,0,131,24,6,159,233,0,168,132,0,55,15,0,15,95,0,21,167,0,13,159,0,
25,200,0,3,4,4,63,5,64

Récupération des données

Code Arduino:

- Détecter le début de la mesure (170,170)
- Récupérer la qualité du signal (2)
- Récupérer les ondes EEG (131)
- Récupérer l'attention et la méditation (4,5)
- Traiter la liste des ondes EEG en décalant les bits de poids fort pour retrouver la valeur réelle

Créer notre premier dataset

Séparer nos 3 possibilités en 3 classes distinctes:

- 0: Aller en avant
- 1: Tourner à droite
- 2: Tourner à gauche

Delta	Theta	Low-Alpha	High-Alpha	Low-Beta	High-Beta	Low-Gamma	Mid-Gamma	Classe
-284	-15910	-9739	31751	16123	4749	4314	3416	0
-14052	-28842	-6319	-26752	5576	7634	4265	3334	0
-29270	23831	21658	-2631	13445	5833	2762	3732	0
-12431	-10620	-17204	17598	5535	5816	6433	3607	0
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
17611	17614	-29253	-4853	16492	12324	5692	1851	1
-21160	26639	29278	26943	12190	4378	6935	3724	1
-18280	14938	27829	19019	3285	957	2738	496	1
-2816	-16536	-20764	17884	8893	3435	3124	2916	1
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
-3129	8727	29893	-13241	14242	6292	6657	2576	2
29742	18106	25842	15904	17214	6916	6279	2925	2
19616	18443	15787	-19235	16277	17733	14361	3792	2
5482	10057	25381	-28572	30941	28101	13168	2545	2

3 - Traiter nos données - Préprocessing

Importer nos données

```
1
2       def csv(self, nom):
3           dataset = list()
4           with open(nom, 'r') as fichier:
5               csv_reader = reader(fichier)
6               for l in csv_reader:
7                   if not l:
8                       continue
9                   dataset.append(l)
10          return dataset
11  .
```

Importer nos données

```
1
2       # Importer le dataset
3       data = knn.csv("data.csv")
4
5       # Supprimer le header & transformer les strings en int
6       del data[0]
7       data= [[int(float(j)) for j in i] for i in data]
8   .
```

Normaliser les données

L'objectif de la normalisation est de recentrer l'ensemble des valeurs d'un ensemble fini de réels dans [0,1]

Définition - Normalisation

Soit E un ensemble fini de réels qui admet donc un minimum X_{min} et un maximum X_{max}, X $\in$ E, où l'on cherche à normaliser X par rapport à E

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Normaliser les données

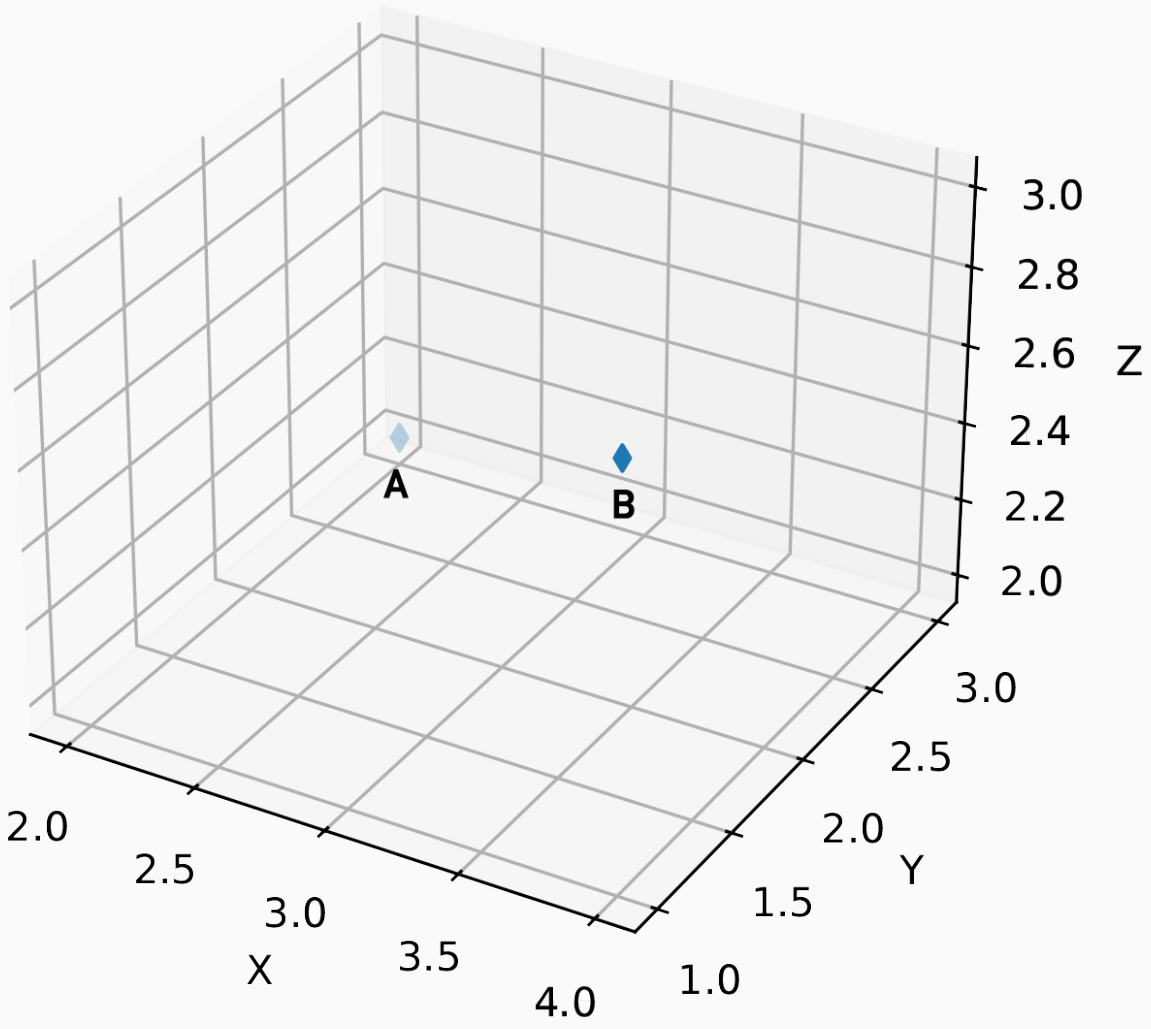
```
1
2       def minMax(dataset):
3           minmax = []
4           for i in range(len(dataset[0])-1):
5               col = [l[i] for l in dataset]
6               vMin = min(col)
7               vMax = max(col)
8               minmax.append([vMin, vMax])
9           return minmax
10  .
```

Normaliser les données

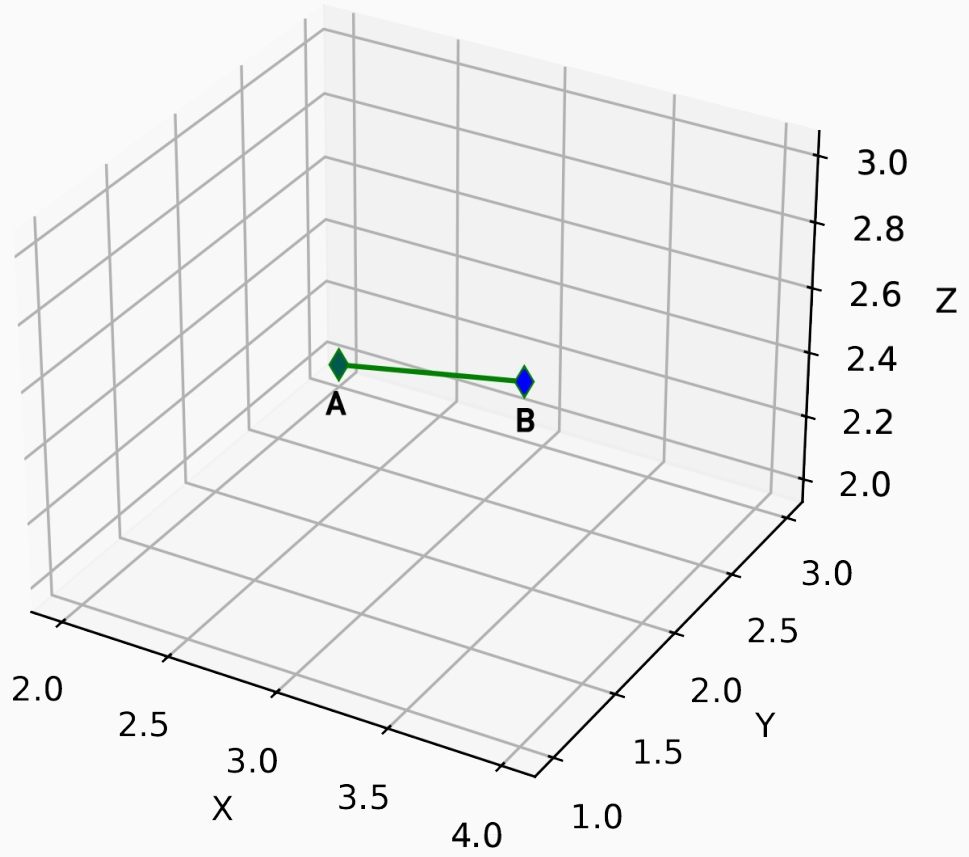
```
1
2       def normaliser(dataset, minmax):
3           lNormaliser = dataset.copy()
4           for k in range (len(dataset)):
5               for i in range (len(dataset[k])-1):
6                   lNormaliser[k][i] = (dataset[k][i] -
7                   minmax[i][0])/(minmax[i][1] - minmax[i][0])
8           return lNormaliser
9   .
```

4 - Algorithme des k plus proches voisins

Mesurer une distance - Cas en 3 dimensions



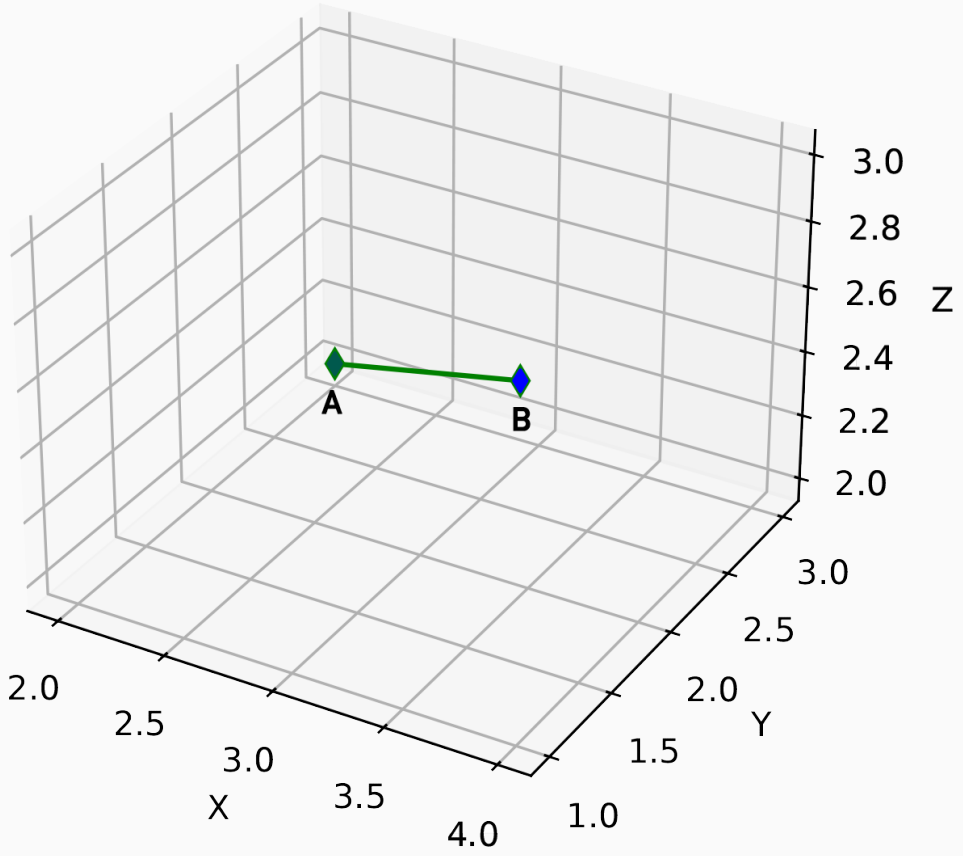
Mesurer une distance - Cas en 3 dimensions



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 p=2:

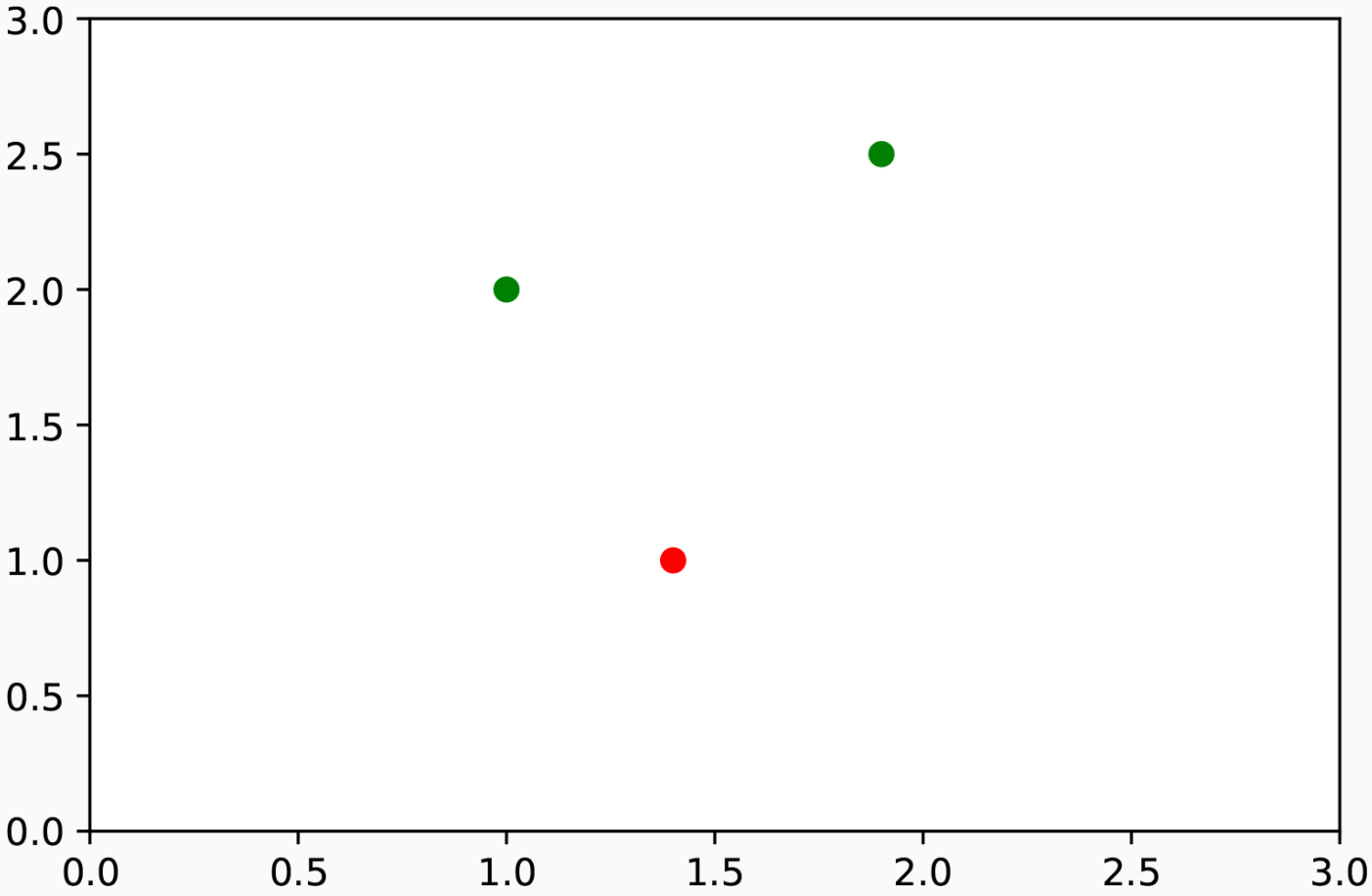
$$d_2(\overrightarrow{u}, \overrightarrow{v}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

Distance euclidienne

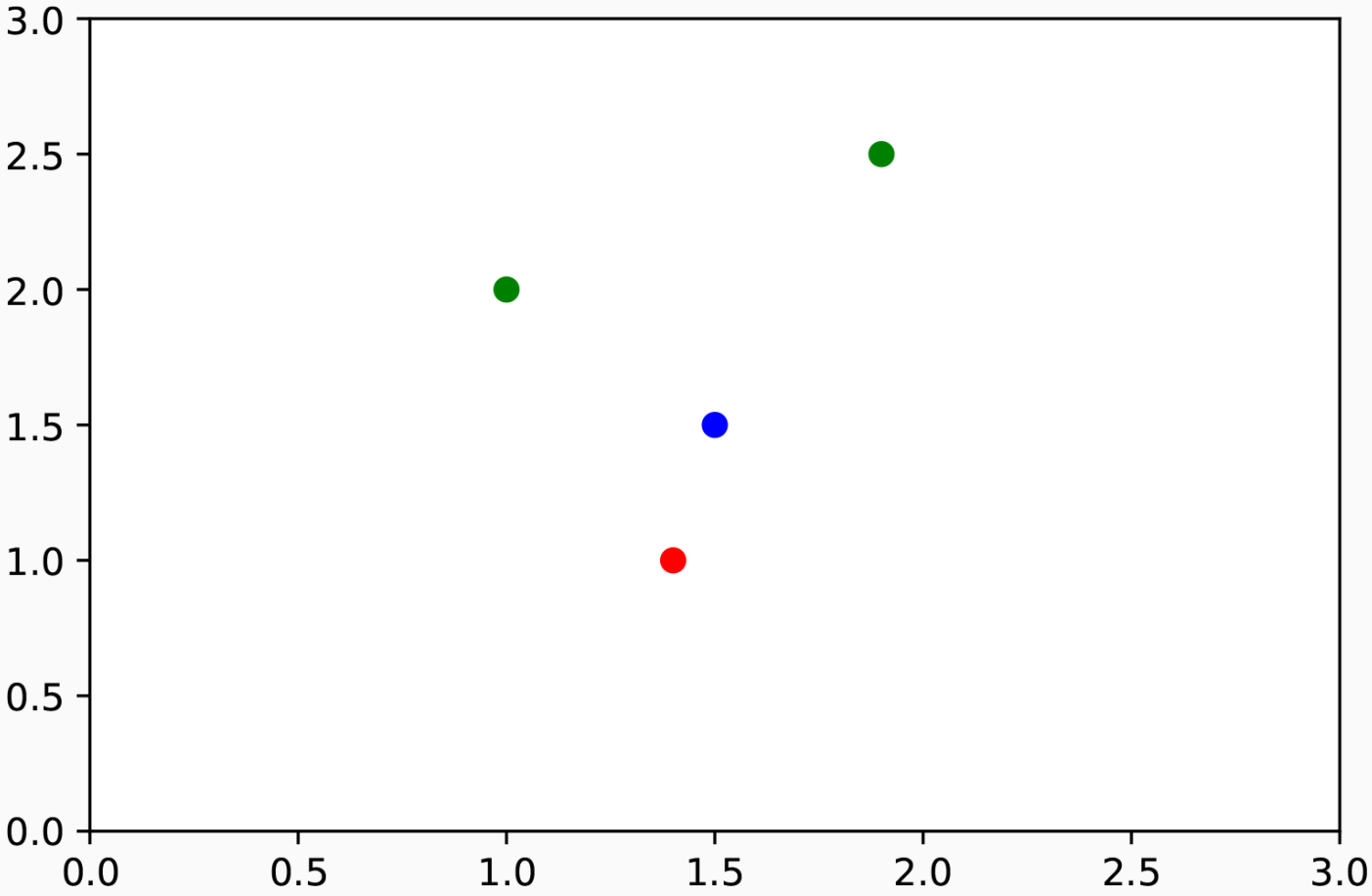
En Python:

```
1
2       def distance_euclidienne(l1, l2):
3           # Initialisation de la distance
4           distance = 0
5           # On calcule la distance euclidienne à n dimension
6           for k in range(len(l1)-1):
7               distance += (l1[k] - l2[k])**2
8           return sqrt(distance)
9   .
```

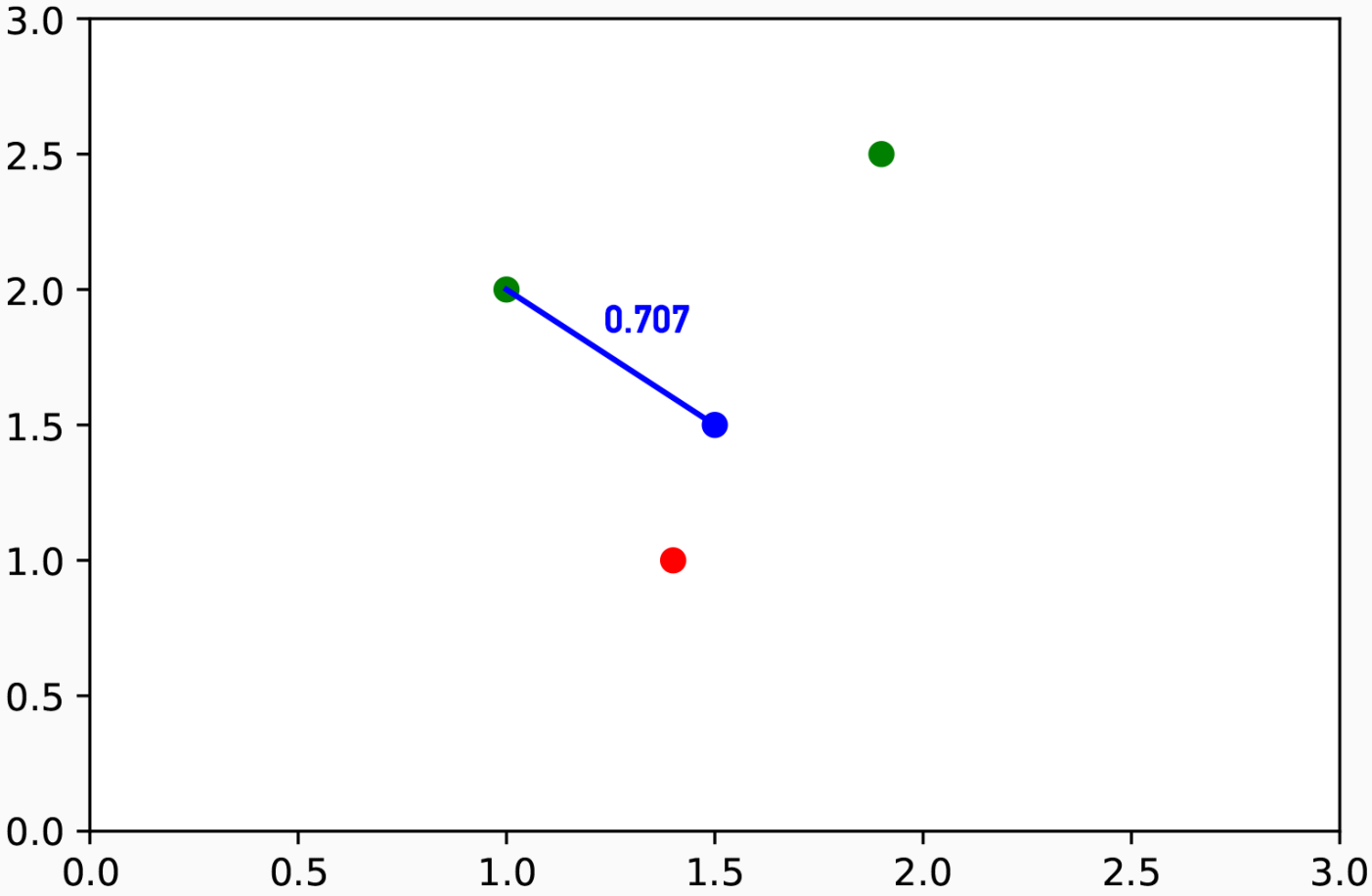
Algorithme du plus proche voisin



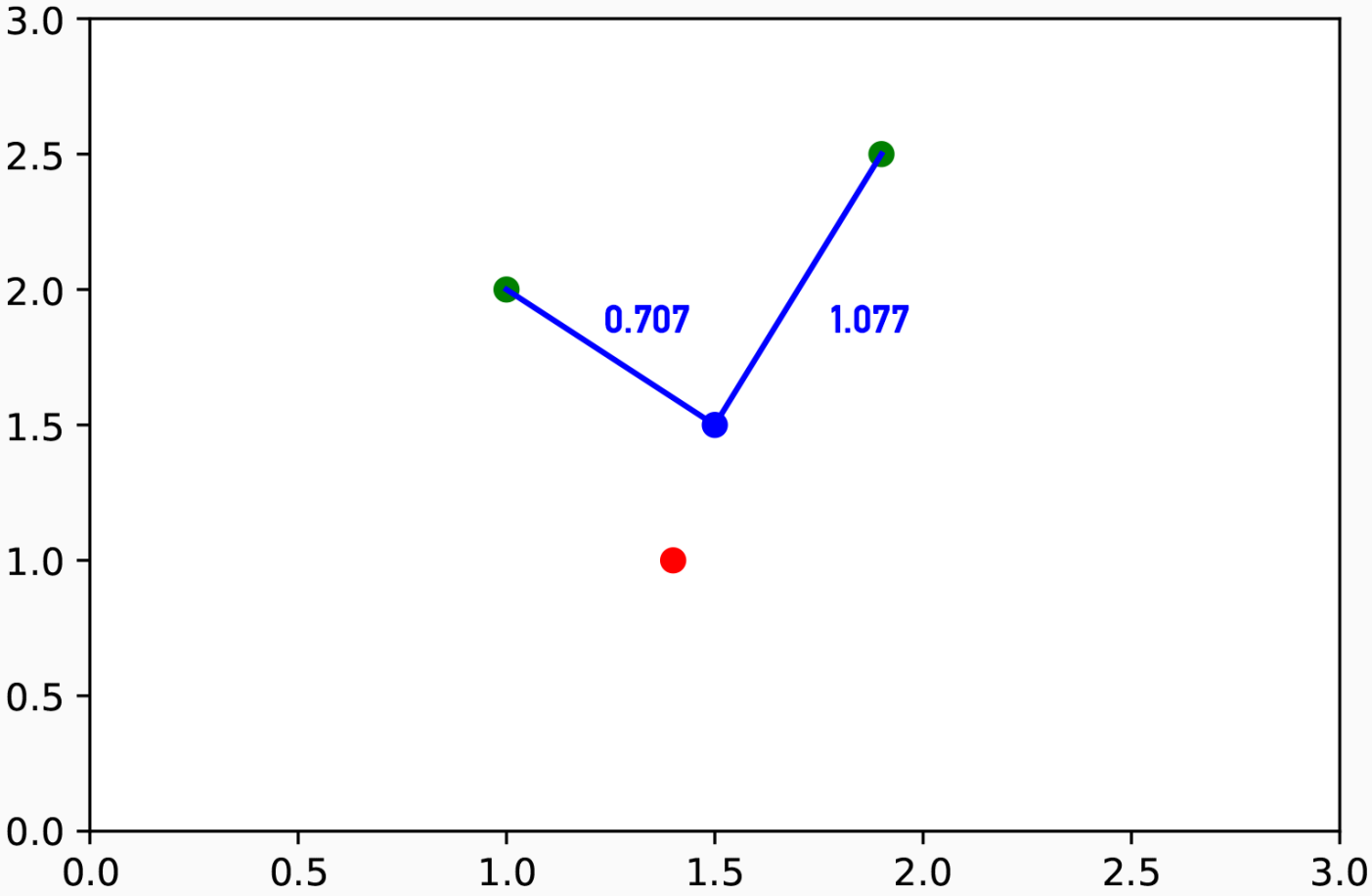
Algorithme du plus proche voisin



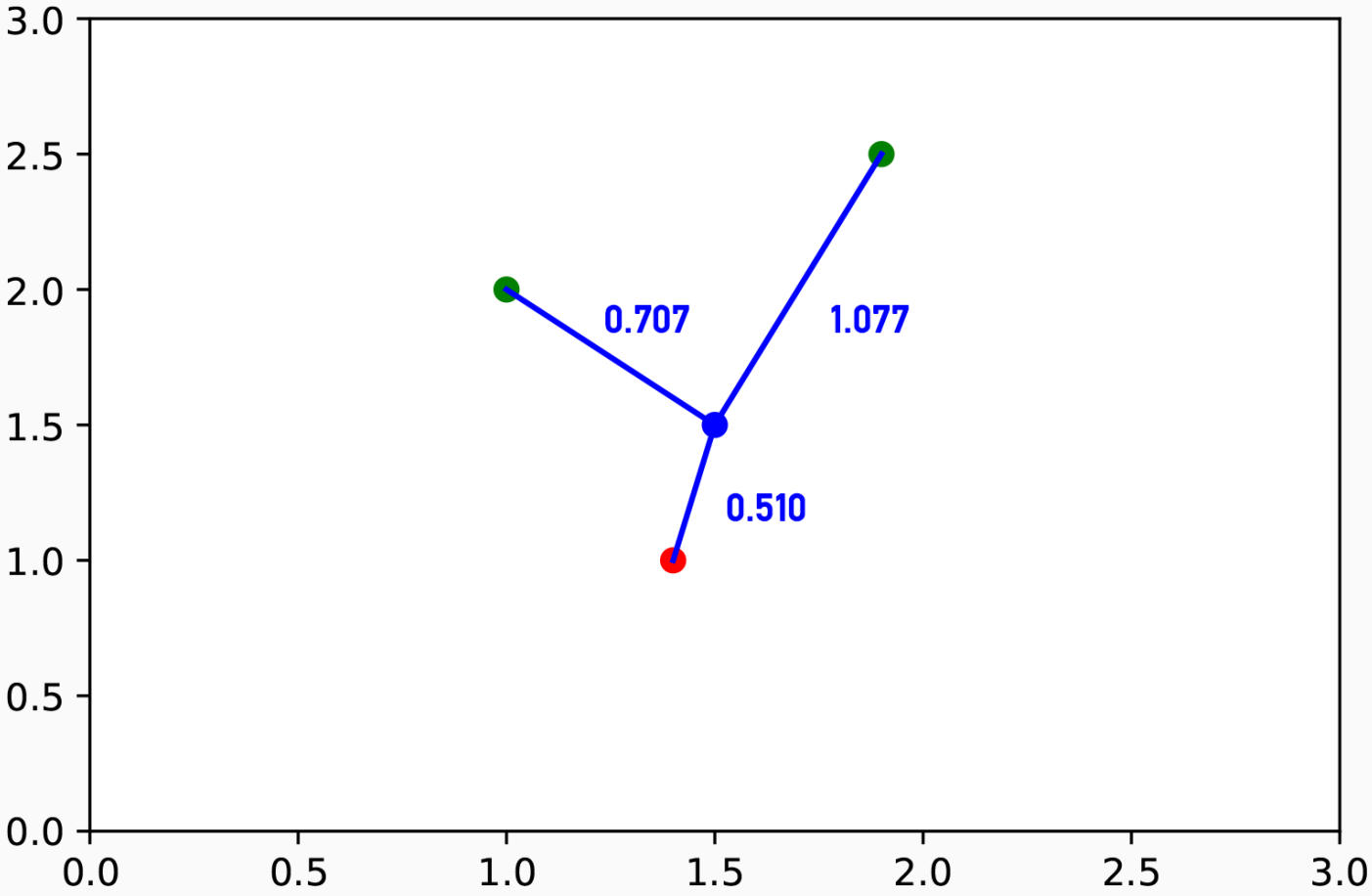
Algorithme du plus proche voisin



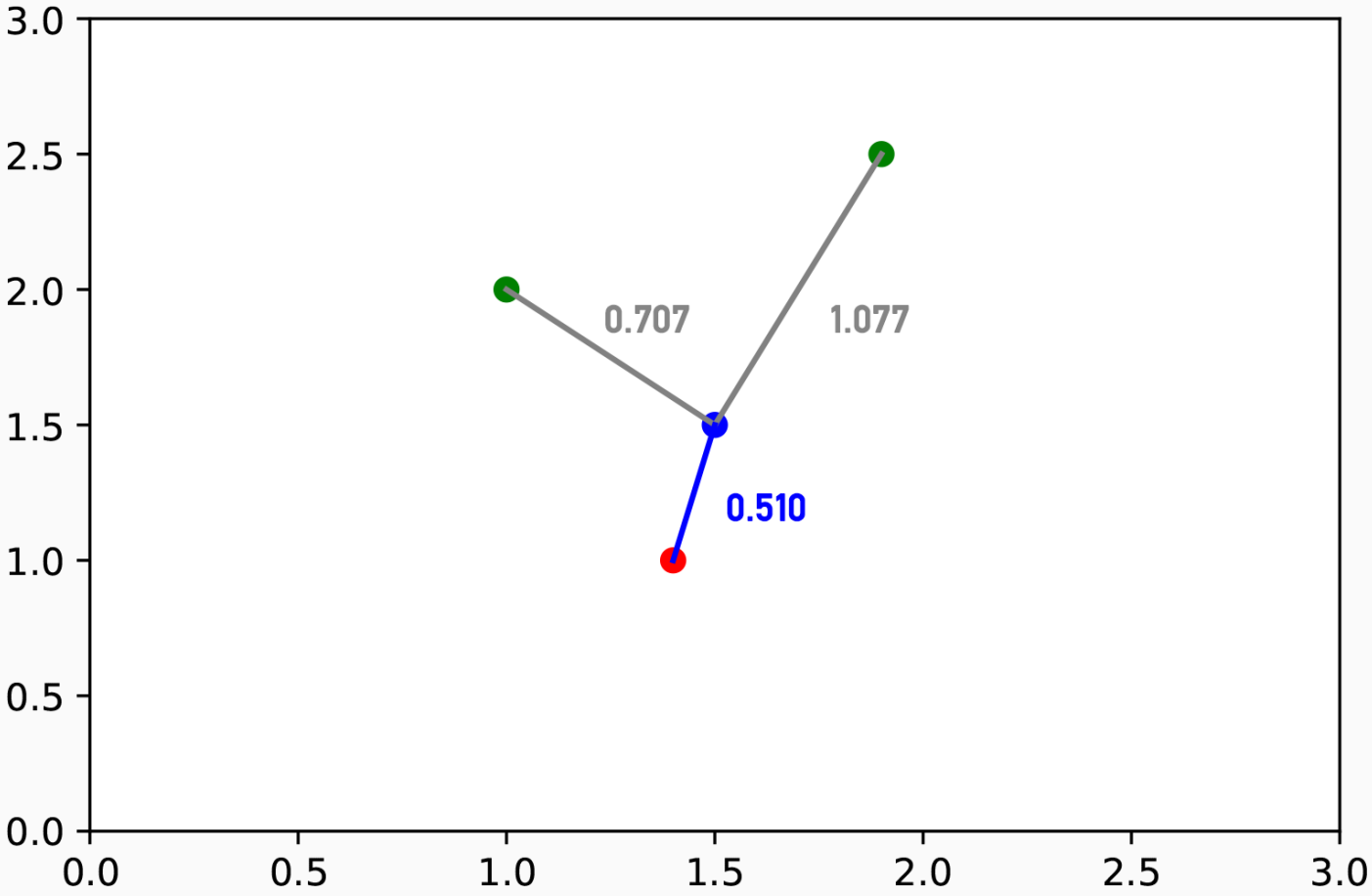
Algorithme du plus proche voisin



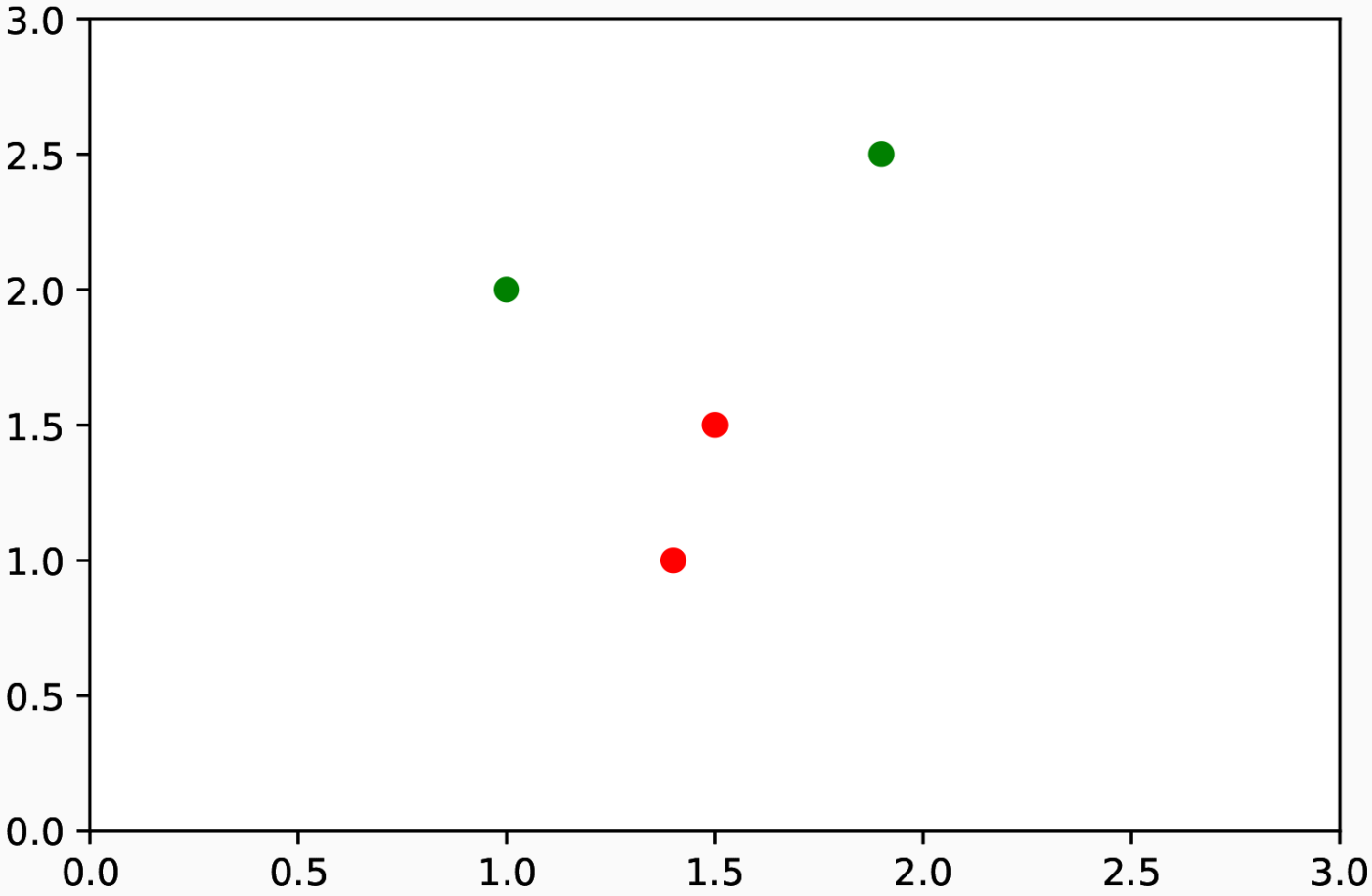
Algorithme du plus proche voisin



Algorithme du plus proche voisin



Algorithme du plus proche voisin



Algorithme du plus proche voisin

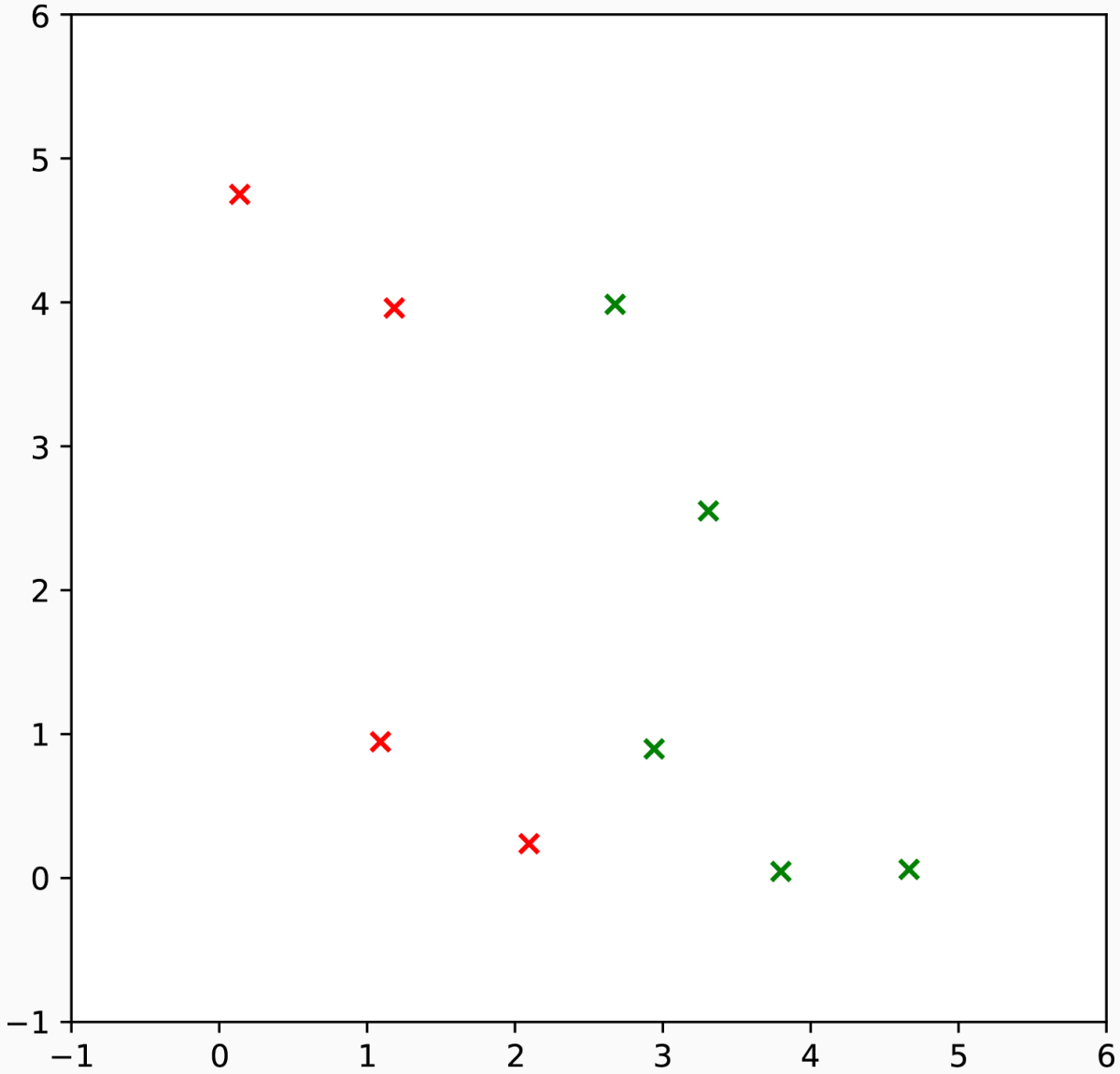
Plus proche voisin

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

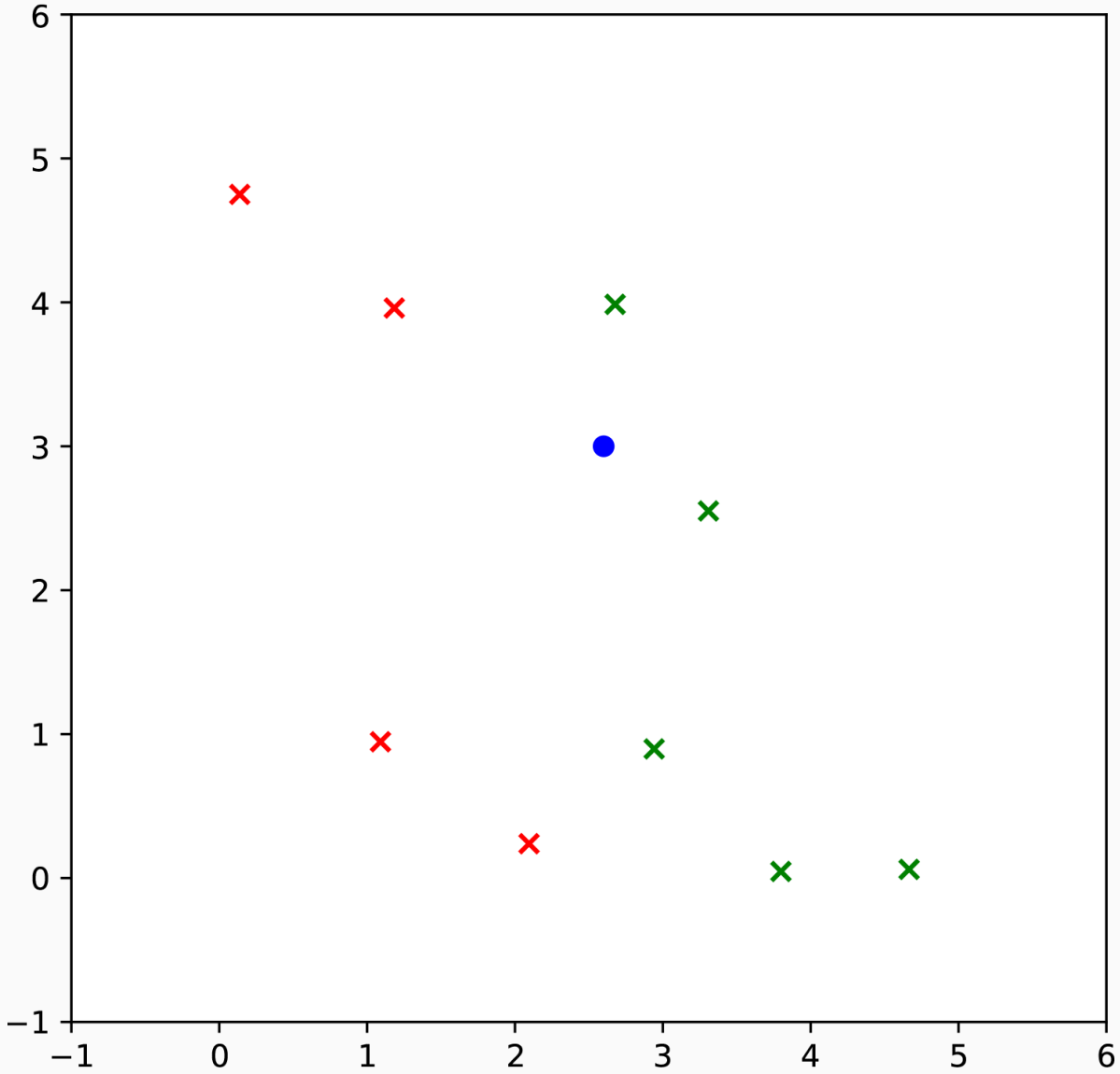
$$f(\overrightarrow{x}) = y_k$$

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

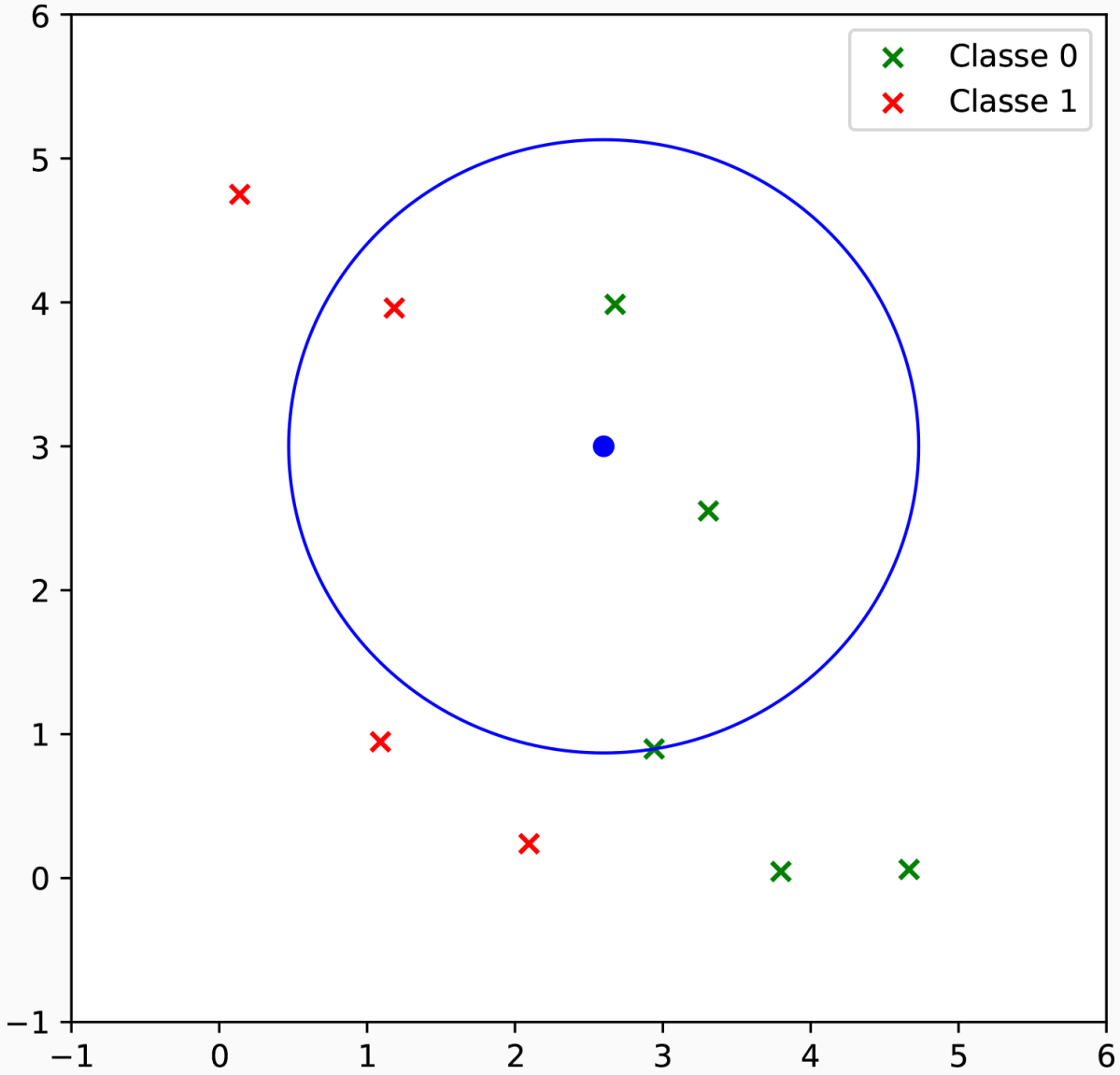
k plus proches voisins



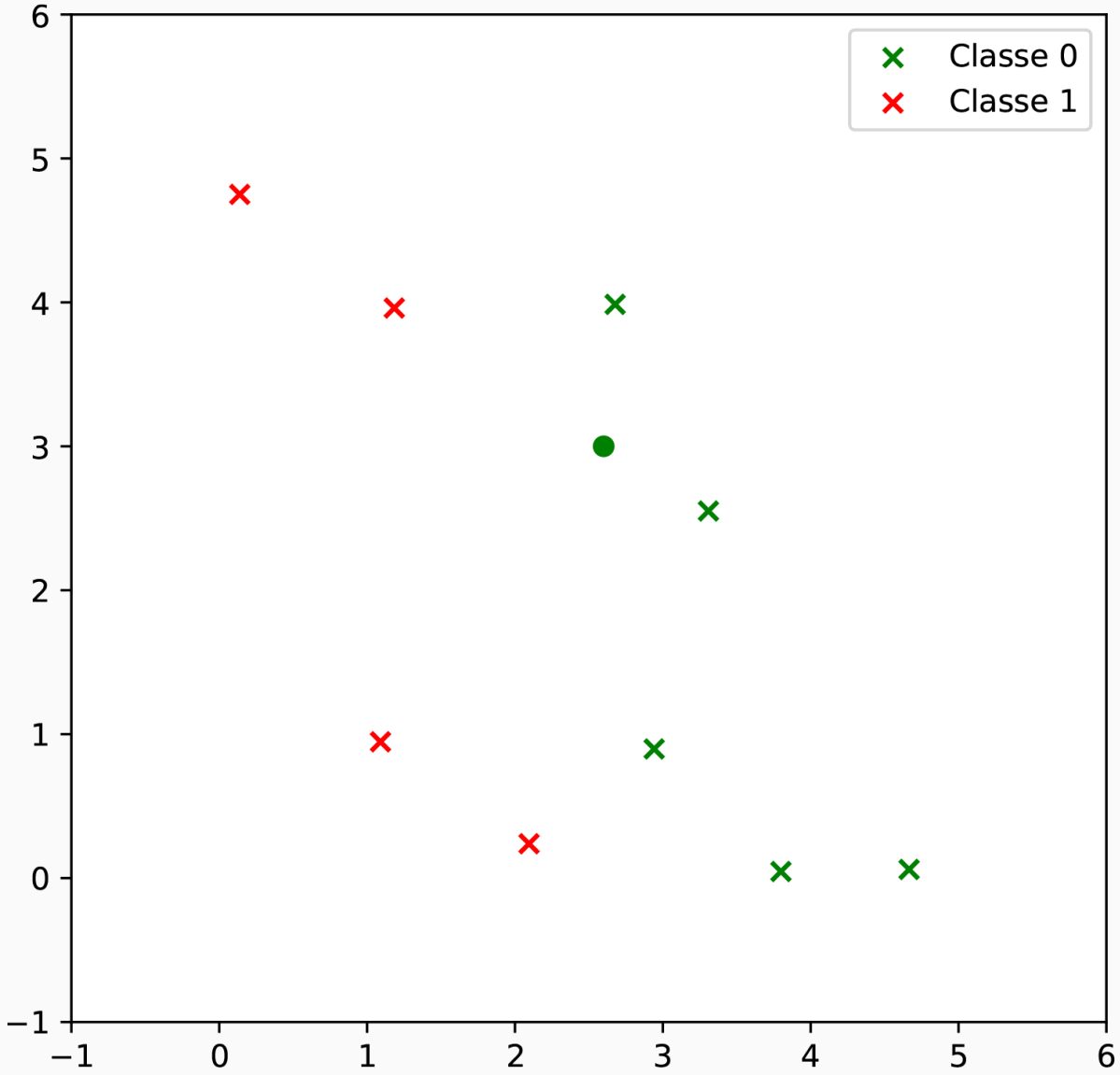
k plus proches voisins



k plus proches voisins - k=4



k plus proches voisins



k plus proches voisins

```
1
2  def prochesVoisins(train, testL, k):
3      # Initialisation de la distance qui contiendra après
4      # la boucle, les distances euclidiennes entre
5      # train et testL
6      distances = []
7      for trainL in train:
8          distance = distance_euclidienne(testL, trainL)
9          distances.append([trainL, distance])
10     # On tri la liste dans l'ordre croissant par rapport
11     # au deuxieme élément de la liste ex:
12     # [(5,6),(10,3),(5,5)] devient [(10, 3), (5, 5), (5, 6)]
13     distances.sort(key=lambda lis: lis[1])
14     voisins = []
15     # On stocke pour k voisins, la classe du plus petit
16     # k correspondant dans la liste distances
17     for i in range(k):
18         voisins.append(distances[i][0])
19     return voisins
20 .
```

k plus proches voisins

```
1
2       def classification(train, testL, k):
3           # On récupere l
4           voisins = prochesVoisins(train, testL, k)
5           sortie = [l[-1] for l in voisins]
6           prediction = max(sortie, key=sortie.count)
7           return prediction
8   .
```

Déterminer le k le plus performant

```
1
2   def kPerformance(self,train,test):
3     listeK = [i for i in range(1,len(train))]
4     listePerformance=[] # Pourcentage de réussite
5     for k in listeK:
6         somme,j=0,0
7         for testL in test:
8             self.k=k
9             if self.classification(train,testL)[0] ==
10            train[j][-1]:
11                somme+=1
12            j+=1
13        listePerformance += [somme/len(train)]
14    plt.ylim(0,1)
15    plt.plot(listeK,listePerformance,color='r')
16    return listePerformance,
17    listeK[listePerformance.index(max(listePerformance))]
18 .
```

Mise en pratique - Données brutes

Delta	Theta	Low-Alpha	High-Alpha	Low-Beta	High-Beta	Low-Gamma	Mid-Gamma	Classe
-284	-15910	-9739	31751	16123	4749	4314	3416	0
-14052	-28842	-6319	-26752	5576	7634	4265	3334	0
-29270	23831	21658	-2631	13445	5833	2762	3732	0
-12431	-10620	-17204	17598	5535	5816	6433	3607	0
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
17611	17614	-29253	-4853	16492	12324	5692	1851	1
-21160	26639	29278	26943	12190	4378	6935	3724	1
-18280	14938	27829	19019	3285	957	2738	496	1
-2816	-16536	-20764	17884	8893	3435	3124	2916	1
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
-3129	8727	29893	-13241	14242	6292	6657	2576	2
29742	18106	25842	15904	17214	6916	6279	2925	2
19616	18443	15787	-19235	16277	17733	14361	3792	2
5482	10057	25381	-28572	30941	28101	13168	2545	2

Mise en pratique - Traiter

```
1
2       # Importer le fichier csv
3       dataset = csv("data.csv")
4       # Supprimer le header
5       del dataset[0]
6       # Transformer les strings en entier
7       dataset= [[int(float(j)) for j in i] for i in dataset]
8       # Prédiction
9       k=3
10      x=[3070,340,438,273,620,427,284,466,0]
11      print("La prédiction est", classification(dataset,x,k))
12      #-> La prédiction est 0
13  .
```

5 - Performances et comparaisons

Matrice de confusion

Matrice de confusion

Soit une matrice M de dimension n (dans notre cas, 3 classes), permet de modéliser la réussite d'un modèle de classification qui y est associé sous la forme:

$$\begin{pmatrix} LL & FL & RL \\ LF & FF & RF \\ LR & FR & RR \end{pmatrix}$$

- Classe réelle selon les colonnes
- Classe prédite selon les lignes

Matrice de confusion

On définit:

TP (True Positive) : Le modèle prédit correctement la classe positive comme positive.

TN (True Negative) : Le modèle prédit correctement la classe négative comme négative.

FP (False Positive) : Le modèle prédit incorrectement la classe négative comme positive.

FN (False Negative) : Le modèle prédit incorrectement la classe positive comme négative.

- A calculer pour chaque classe

Evaluer notre algorithme

Etablissons la matrice de confusion de notre classification en effectuant 500 mesures pour chaque classe:

$$\begin{pmatrix} 439 & 12 & 49 \\ 18 & 454 & 28 \\ 4 & 10 & 486 \end{pmatrix}$$

Pour la classe Left:

- TP $= 439$
- TN $= 454 + 486 + 10 + 28$
- FP $= 12 + 49$
- FN $= 18 + 4$

Evaluer notre algorithme

Calculons les différents indices de performances pour chaque classe

Pour la classe Left:

$$\begin{pmatrix} 940(TN) & 22(FN) \\ 61(FP) & 439(TP) \end{pmatrix}$$

Evaluer notre algorithme

Précision (ou Positive predictive value)

La précision représente la proportion de prédictions correctes parmi les prédictions positives.

$$Precision = \frac{TP}{TP + FP}$$

Evaluer notre algorithme

Calculons les différents indices de performances pour chaque classe

Pour la classe Left:

$$\begin{pmatrix} 940(TN) & 22(FN) \\ 61(FP) & 439(TP) \end{pmatrix}$$

Precision $= \frac{439}{439+61} = 0.878$

Problème: si on fait peu de prédictions positives

Evaluer notre algorithme

Rappel (ou Recall)

Le rappel représente le taux de vrais positifs, c'est à dire la proportion de prédiction positive correctement identifiés

$$Rappel = \frac{TP}{TP + FN}$$

Evaluer notre algorithme

Calculons les différents indices de performances pour chaque classe

Pour la classe Left:

$$\begin{pmatrix} 940(TN) & 22(FN) \\ 61(FP) & 439(TP) \end{pmatrix}$$

Precision $= \frac{439}{439+61} = 0.878$

Recall $= \frac{439}{439+22} = 0.953$

Problème: si on prédit pour chaque valeur qu'elles sont positives.

Evaluer notre algorithme

F_{score}

Le F_{score} est la moyenne harmonique de la précision et du rappel:

$$F_{score} = 2 * \frac{Precision * Rappel}{Precision + Rappel}$$

Evaluer notre algorithme

Calculons les différents indices de performances pour chaque classe

Pour la classe Left:

$$\begin{pmatrix} 940(TN) & 22(FN) \\ 61(FP) & 439(TP) \end{pmatrix}$$

$\text{Precision} = \frac{439}{439+61} = 0.878$

$\text{Recall} = \frac{439}{439+22} = 0.953$

$F_{score} = 2 * \frac{0.878*0.953}{0.878+0.953} = 0.912$

Evaluer notre algorithme

Classe	**Précision**	**Rappel**	F_{score}
Aller à Gauche	0.878	0.953	0.912
Aller en Face	0.773	0.858	0.813
Aller à Droite	0.790	0.793	0.791

$$F_{scoreT} = \frac{F_{score0} + F_{score1} + F_{score2}}{3} = 0.83$$

Comparaison avec la librairie scikit-learn


```
1
2       # Importation des librairies
3       import numpy as np
4       import matplotlib.pyplot as plt
5       import pandas as pd
6       from sklearn.neighbors import KNeighborsClassifier
7
8       # Importation des données du casque
9       data = pd.read_excel('data.xlsx')
10      data.head()
11
12      # Parsing des données
13      y = data['Classe']
14      X = data.drop('Classe', axis=1)
15  .
```

Comparaison avec la librairie scikit-learn

```
1
2       # Creation du KNN
3       model = KNeighborsClassifier(3)
4       model.fit(X,y)
5       model.score(X,y)
6
7       # Donnée que l'on veut classer
8       x = np.array([329,415,243,153,393,317,260,123])
9       .reshape(1,8)
10
11      # Affichage de la prediction & probabilité
12      model.predict(x)
13      print(model.predict(x))
14      print(model.predict_proba(x))
15  .
```

6 - Conclusion

Aller plus loin..

- Acquérir un casque plus performant

- Remplir davantage la base de donnée

- Tester de nouveaux algorithmes: Random Forest, SVM...

- Améliorer la complexité pour pouvoir rajouter des données

- Réduction de la dimensionalité

Code

L'ensemble des fichiers présentés ici et nécessaires à l'élaboration de ce projet est publié sur mon github

`https://github.com/QuentinPTT/Just-Think-It`

Questions?