

Introduction à l'algorithme des k plus proches voisins

L'algorithme des k plus proches voisins est l'un des algorithmes utilisés dans le domaine de l'intelligence artificielle.

C'est un algorithme d'apprentissage automatique supervisé qui attribue une catégorie à un élément en fonction de la classe majoritaire de ses plus proches voisins dans l'échantillon d'entraînement.

Son principe peut être résumé par cette phrase : *Dis-moi qui sont tes amis et je te dirai qui tu es.*

Cet algorithme d'apprentissage automatique est par exemple utilisé par des entreprises d'Internet comme Amazon, Netflix, Spotify ou iTunes afin de prévoir si vous seriez ou non intéressés par un produit donné en utilisant vos données et en les comparant à celles des clients ayant acheté ce produit particulier.

Cet algorithme a été introduit en 1951 par Fix et Hodges dans un rapport de la faculté de médecine aéronautique de la US Air Force.

Exemple introductif

Dans cette introduction, nous considérons un jeu de données constitué de la façon suivante :

- les données sont réparties suivant deux types : le type 1 et le type 2,
- les données n'ont que deux caractéristiques : caractéristique 1 et caractéristique 2,

Imaginez la situation suivante dans un jeu :

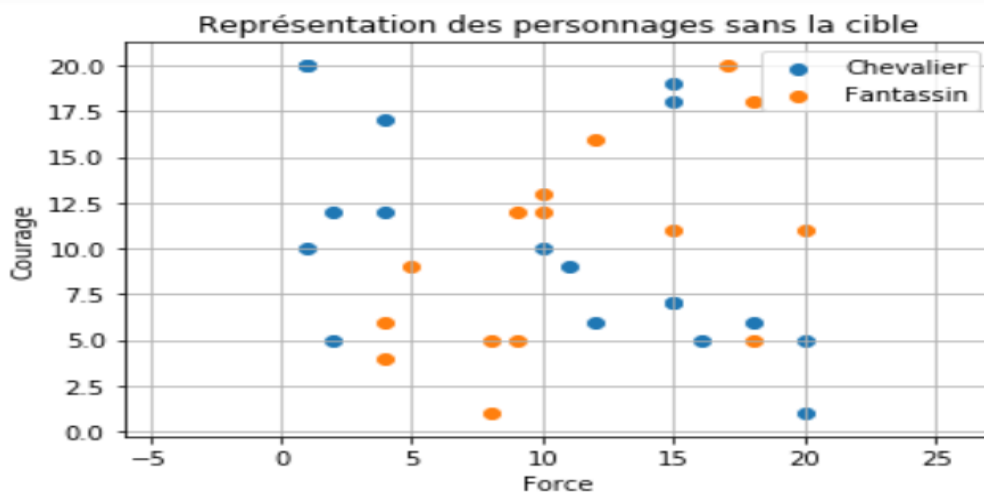
- Vous avez deux types de personnages : les fantassins (type 1 : "fantassin") et les chevaliers (type 2 : "chevalier").
- Vous avez deux types de caractéristiques : la force (caractéristique 1 : nombre entre 0 et 20) et le courage (Caractéristique 2 : nombre entre 0 et 20).
- Vous avez une collection de personnages dont vous connaissez les caractéristiques et le type.

Vous introduisez un nouveau personnage dont vous ne connaissez pas le type. Vous possédez les caractéristiques de ce nouveau personnage. Le but de l'algorithme KNN (k Nearest Neighbors = k plus proches voisins en français) est de déterminer le type de ce nouveau personnage.

Nom	Force	Courage	Type
Ario	20	1	Chevalier
Axal	10	10	Chevalier
Cargo	20	11	Fantassin
Clark	2	12	Chevalier
Fancy	9	5	Fantassin
Fanks	16	5	Chevalier
Faq	15	11	Fantassin
Fool	10	12	Fantassin
Helen	8	1	Fantassin
Karl	11	9	Chevalier
Korg	1	20	Chevalier
Lis	18	18	Fantassin
Lomo	17	20	Fantassin
Iouli	20	5	Chevalier
Louly	15	18	Chevalier
Loumi	4	12	Chevalier

Voici un aperçu des données :

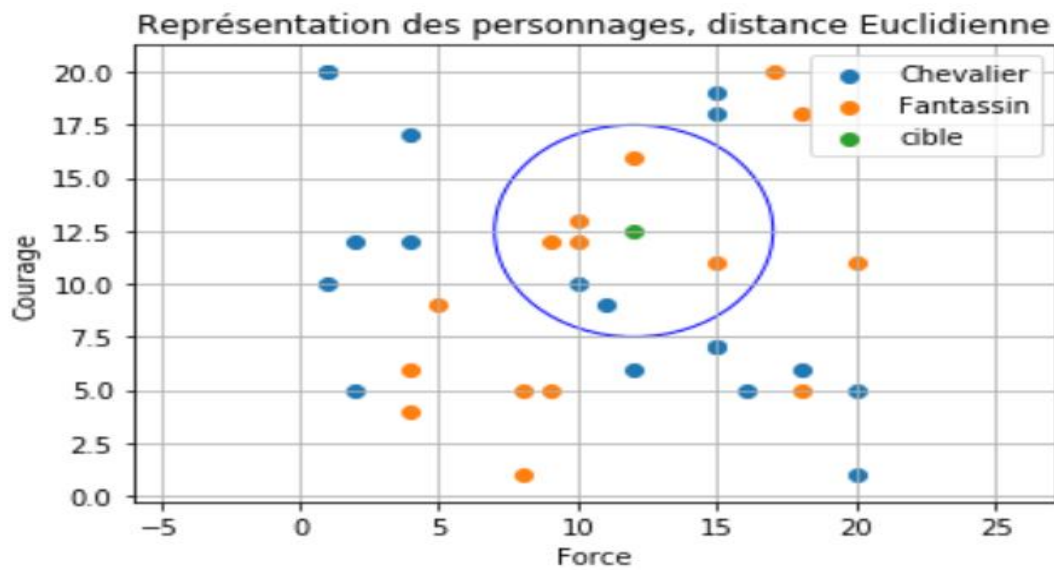
Voici une représentation de ces données :



Nous introduisons une nouvelle donnée (appelée cible dans notre exemple) avec ses deux caractéristiques : une force de 12 et un courage de 12,5 . Le but de l'algorithme KNN des k plus proches voisins est de déterminer le type de cette nouvelle donnée.

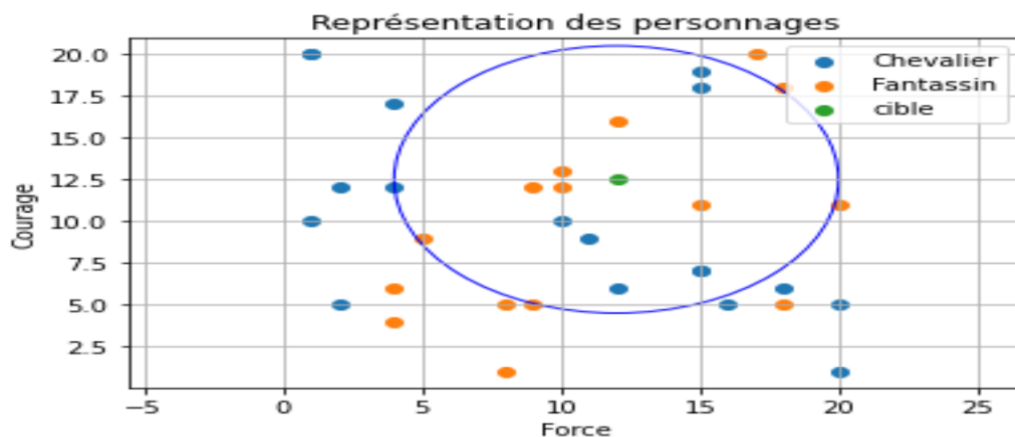
1. Dans un premier, il faut fixer le nombre de voisins. Nous allons choisir $k=7$.

Voici une nouvelle représentation avec la cible et la recherche des 7 voisins les plus proches, ceux qui se trouvent dans le cercle bleu :



En utilisant les voisins de type "chevalier" et de type "fantassin", déterminer le type le plus probable de notre cible.

2. On considère désormais la valeur $k=13$. Voici une nouvelle représentation avec la cible et la recherche des 13 voisins les plus proches, ceux qui se trouvent dans le cercle bleu :



3. Déterminer le type le plus probable de notre cible dans ce cas ?

L'algorithme

Pour prédire la classe d'un nouvel élément, il faut des données :

- Un échantillon de données ;
- Un nouvel élément dont on connaît les caractéristiques et dont on veut prédire le type ;
- La valeur de k , le nombre de voisins étudiés.

Une fois ces données modélisées, nous pouvons formaliser l'algorithme de la façon suivante :

1. Trouver, dans l'échantillon, les k plus proches voisins de l'élément à déterminer.
2. Parmi ces proches_voisins, trouver la classification majoritaire.
3. Renvoyer la classification_majoritaire comme type cherché de l'élément.

Ce qui nous donne l'algorithme naïf suivant :

- **Données :**

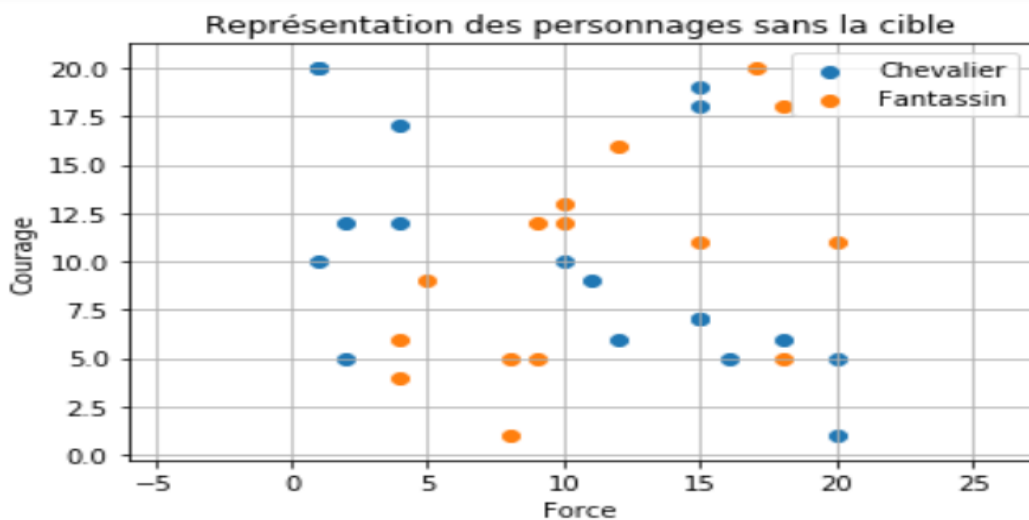
- o une **table** de données de taille n ;
- o une donnée cible ;
- o un entier k plus petit que n ;
- o une règle permettant de calculer la **distance** entre deux données.

• **Algorithme :**

1. Trier les données de la table selon la distance croissante avec la donnée cible.
2. Créer la liste des k premières données de la table triée.
3. Renvoyer cette liste.

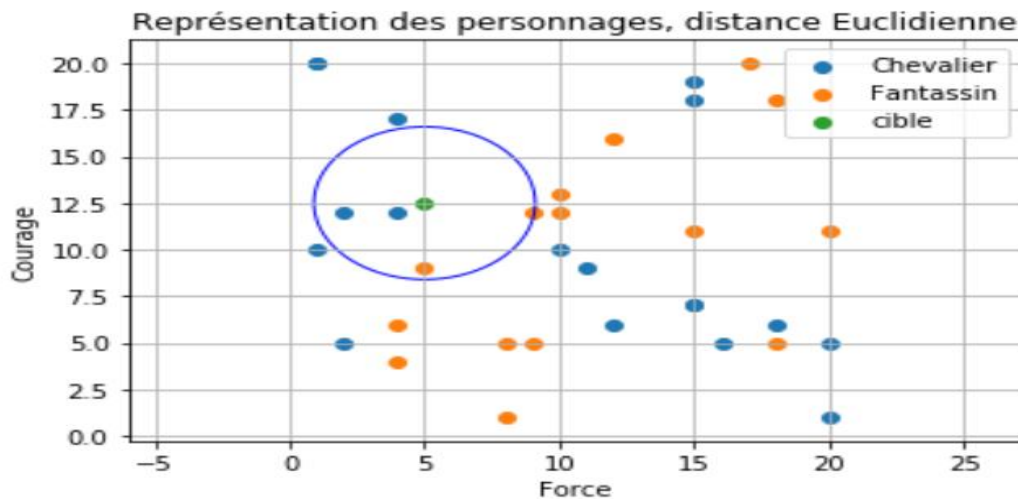
Un premier exercice

Nous reprenons le même ensemble de données :

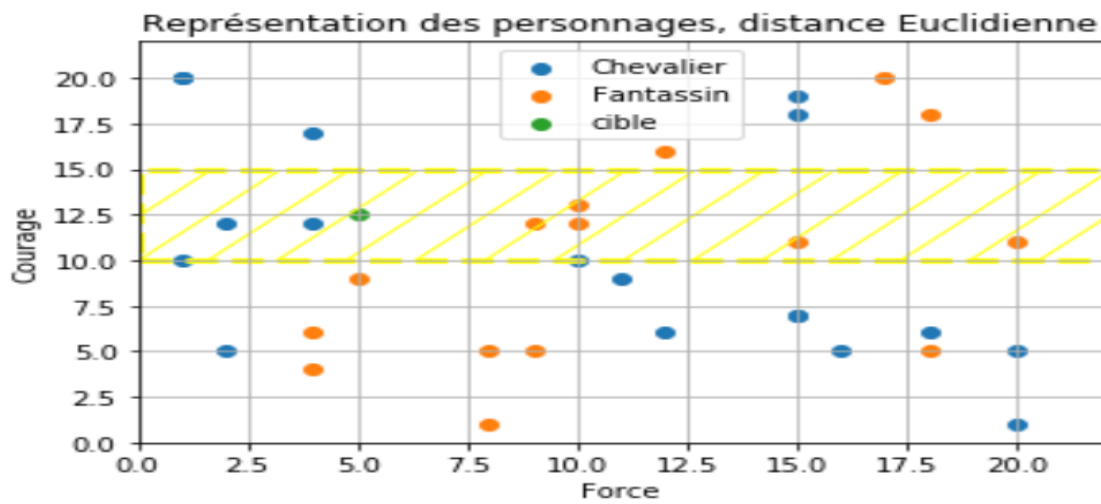


Nous introduisons une cible : force = 5 et courage = 12,5

1. On choisit $k=4$ et la distance schématisée par un disque .

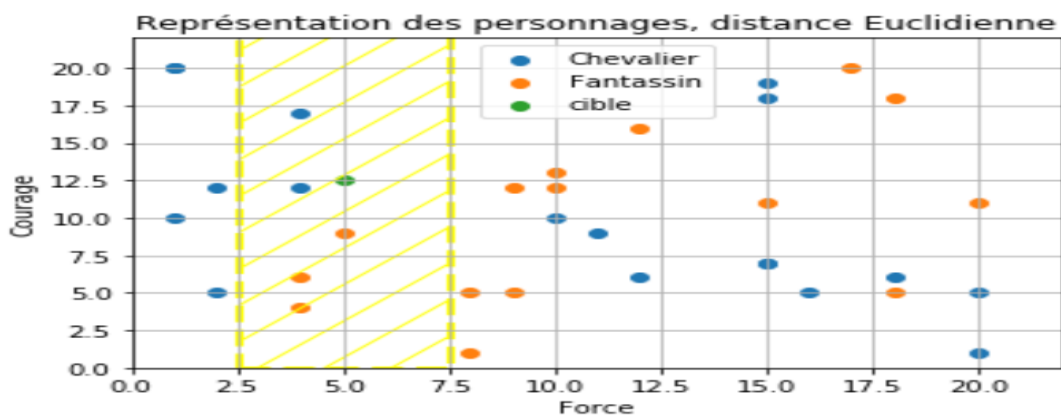


- a. Quel est le type de notre donnée cible ? Quel est le problème ?
 - b. Donner une valeur de k qui permette de décider du type de notre donnée cible ?
 - c. Donner pour cette valeur de k le type de la donnée cible.
2. On choisit $k=9$. Pour la distance, on décide que les valeurs de la force n'ont pas d'importance. La distance dépend du courage.



Quel est le type de notre donnée cible ?

3. On choisit k=5. Pour la distance, on décide que les valeurs du courage n'ont pas d'importance. La distance dépend de la force.



Quel est le type de notre donnée cible ?

Comment représenter ce type de données en Python ?

1. a. Tester le script Python suivant :

```

b. from math import *
c. import matplotlib.pyplot as plt
d. # Données de type 1
e. liste_x_1=[1,3,8,13]
f. liste_y_1=[28,27.2,37.6,40.7]
g. # Données de type 2
h. liste_x_2=[2,3,10,15]
i. liste_y_2=[30,26,39,35.5]
j. plt.axis([0,15, 0, 50]) # Attention [x1,x2,y1,y2]
k. plt.xlabel('Caractéristique 1') # premier commentaire attendu
l. plt.ylabel('Caractéristique 2')

```

- m. `plt.title('Représentation des deux types')`
- n. `plt.axis('equal')` # Pour avoir un repère orthonormé
- o. `plt.grid()`
- p. `plt.scatter(liste_x_1,liste_y_1, label='type 1')` # second commentaire attendu
- q. `plt.scatter(liste_x_2,liste_y_2, label='type 2')`
- r. `plt.legend()`
- s. `plt.show()`

t. Remplacer les deux commentaires attendus par un commentaire expliquant le rôle des fonctions `plt.label` et `plt.scatter`.

2.

- a. Vous pouvez ajouter des formes (rectangle, ellipse, etc.).

Exécuter le script suivant :

```
from math import *

import matplotlib.pyplot as plt

# Données de type 1
liste_x_1=[1,3,8,13]
liste_y_1=[28,27.2,37.6,40.7]

# Données de type 2
liste_x_2=[2,3,10,15]
liste_y_2=[30,26,39,35.5]

fig, ax = plt.subplots()

plt.axis([0, 15, 0, 50]) # Attention [x1,x2,y1,y2]

plt.xlabel('Caractéristique 1')
plt.ylabel('Caractéristique 2')

plt.title('Représentation des deux types')

plt.axis('equal') # pour avoir un repère orthonormé. Faire des tests.

plt.grid()

plt.scatter(liste_x_1,liste_y_1, label='type 1')
plt.scatter(liste_x_2,liste_y_2, label='type 2')

plt.legend()

# Ajout d'un cercle (ou d'une ellipse)
ax.add_artist(plt.Circle((6, 30), 4, edgecolor='b', facecolor='none'))

# Ajout d'un rectangle
ax.add_artist(
```

```
plt.Rectangle((6,0), 2, 50,
             edgecolor = 'black', facecolor = 'none',
             fill = True, hatch = '/', linestyle = 'dashed',
             linewidth = 3, zorder = 1))

plt.show()
```

- Quel est le rôle du premier argument (6, 30) de la fonction `plt.Circle` ?
- Quel est le rôle des trois premiers arguments (6,0), 2 et 50 de la fonction `plt.Rectangle` ?

La distance euclidienne (dans un repère orthonormé)

Soit deux données `donnée1` et `donnée2` de coordonnées respectives (x_1, y_1) et (x_2, y_2) .

$distance(donnée1, donnée2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Complexité/coût de l'algorithme KNN

La **complexité** de l'algorithme KNN repose sur la complexité du tri de la table.

D'après la documentation du langage Python, le tri de la table dans `sorted` selon la **key distance** est en $O(n \log(n))$. Cela veut dire que son coût est comparable à $n \log(n)$ (n est la taille de la table à trier). Dans ce cas on parle de complexité **quasi-linéaire**.

```
• from math import *
• # from random import *
• import matplotlib.pyplot as plt
• # import numpy as np
• # from scipy.stats import linregress
• # Données de type 1
• liste_x_1=[1,3,8,13]
• liste_y_1=[28,27.2,37.6,40.7]
• # Données de type 2
• liste_x_2=[2,3,10,15]
• liste_y_2=[30,26,39,35.5]
• plt.axis([0, 15, 0, 50]) # Attention [x1,x2,y1,y2]
• plt.axis('equal')
• plt.xlabel('Caractéristique 1')
• plt.ylabel('Caractéristique 2')
• plt.title('Représentation des deux types')
• plt.grid()
• plt.scatter(liste_x_1, liste_y_1, label='type 1')
• plt.scatter(liste_x_2, liste_y_2, label='type 2')
• plt.scatter(7, 28.4, label='cible')
• plt.legend()
• plt.show()
• table = [['t1', 1, 28], ['t1', 3, 27.2], ['t1', 8, 37.6], ['t1', 13, 40.7], ['t2', 2, 30], ['t2', 3, 26], ['t2', 10, 39], ['t2', 15, 35.5]]
• cible = [7, 28.4]
```

- k=3
- def k_plus_proches_voisins(table, cible, k) :
- """Revoie la liste des k plus proches voisins de la cible"""
- def distance_cible(donnee) :
- """ renvoie la distance entre la donnée et la cible """
-
- distance = (abs(donnee[1]-cible[0])**2+abs(donnee[2]-cible[1])**2)**(1/2)
- return distance
- table_triee = sorted(table, key = distance_cible)
-
- proches_voisins = []
-
- for i in range(k) :
- proches_voisins.append(table_triee[i])
-
- return proches_voisins
-

print("La liste des ",k," plus proches voisins de la cible : ",k_plus_proches_voisins(table,cible,k))
