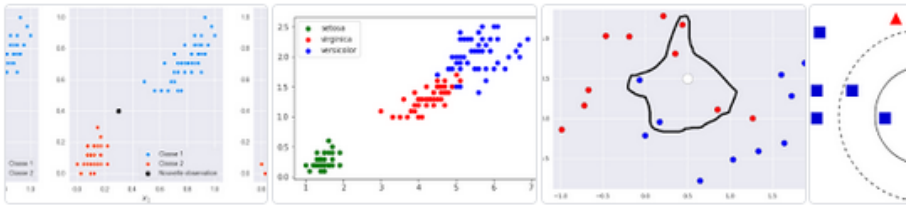
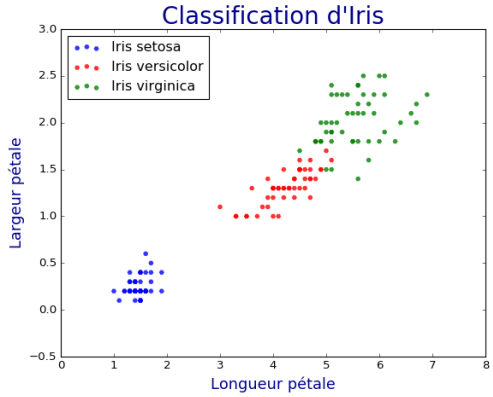
Algorithme avancé : classification

Résumé :



Les 'k plus proches voisins' ou **k**-nearest neighbors en anglais (d'où l'appellation knn) est une méthode non paramétrique dans laquelle le modèle mémorise les observations de l'ensemble d'apprentissage pour la classification des données de l'ensemble de test. 21 sept. 2019



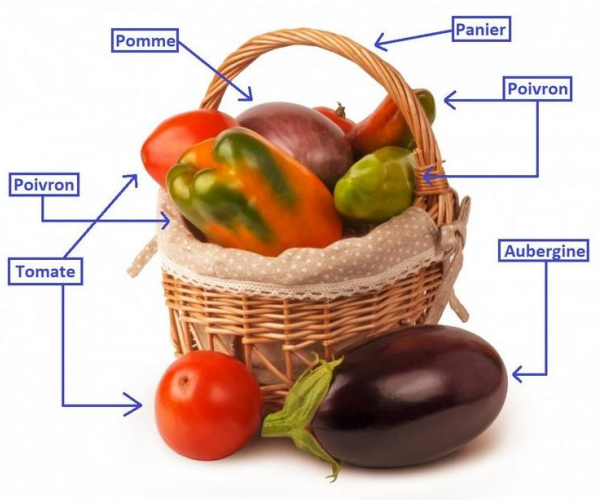
Les réponses aux questions posées seront données dans un document de travail joint.

Sommaire :

1 La classification

1.1 Introduction

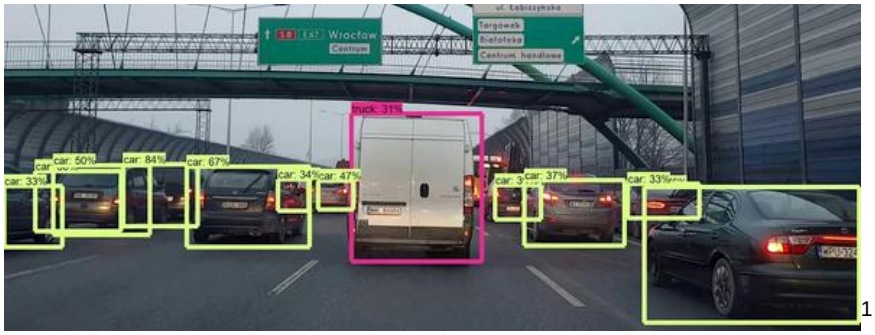
Pourquoi classifier ? Sans nous en rendre compte nous classifions toute la journée. Par exemple en regardant le contenu de ce panier nous identifions instantanément les différents fruits présents.



Faire réaliser ce travail par une machine consiste à utiliser un algorithme de classification.

Différents algorithmes existent depuis des technologiques 'classiques' jusqu'aux technologies d'IA : à l'intelligence artificielle.

Ces technologies sont intensivement développées actuellement. En effet elles sont utilisées dans des domaines aussi divers tels que l'analyse d'imagerie médicale, la reconnaissance faciale, le développement de la voiture autonome ….

 [1]

1.2 Qu'allons-nous découvrir ?

Nous travaillerons à un algorithme appelé : l'algorithme des k plus proches voisins, puis nous découvrirons les réseaux de neurones. En effet ces deux algorithmes font de la classification. Nous découvrirons également que ces deux algorithmes nécessitent de pouvoir avoir à disposition des données déjà classifiées pour pouvoir en classifier de nouvelles.

En effet c'est à partir de données dûment répertoriées et identifiées par l'homme que l'on peut proposer un algorithme qui réalise une classification automatique. Cet algorithme fonctionnant sur une machine pourra alors être plus rapide et performant qu'un opérateur humain. Nous verrons également que la classification n'est pas forcément juste à 100% il subsiste une part d'incertitude quand aux résultats.

[1] https://www.framboise314.fr/i-a-realisez-un-systeme-de-reconnaissance-dobjets-avec-raspberry-pi/

2 Algorithme des k plus proches voisins présentation

2.1 La problématique

L'algorithme des k plus proches voisins ou k-nearest neighbors (kNN) est un algorithme d'apprentissage automatique (Machine Learning) supervisé simple.

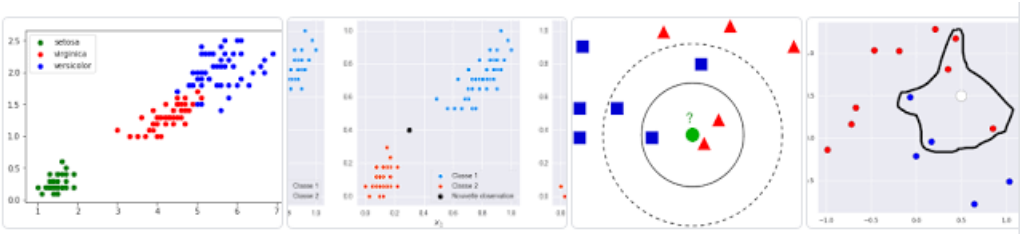
Apprentissage automatique : *« L'apprentissage automatique est la discipline donnant aux ordinateurs la capacité d'apprendre sans qu'ils soient explicitement programmés » Arthur Samuel, 1959*

Machine learning : apprentissage automatique en anglais.

Supervisé : l'algorithme fonctionne à partir d'un jeu de données d'entrées-sorties connu.

Nous observons sur l'image ci-dessous que les données sont ici représentées avec deux paramètres et donc elles sont représentables dans un plan XOY. La grandeur de sortie est représentée par la couleur du point désignant une entrée du jeu de données, ou sa forme, …

On dira que le jeu de données de test est réparti en classes.



L'algorithme utilisera les points déjà connus pour identifier la classe d'un nouveau point inconnu. Il utilisera pour cela une notion, à définir, de distance, et un hyper-paramètre k le nombre de voisins pris en compte dans l'analyse.

2.2 Un exemple académique la classification d'Iris

Un jeu de données d'entrainement pour la classification est disponible[2] pour déterminer à quelle espèce appartient un iris parmi trois espèces possibles :

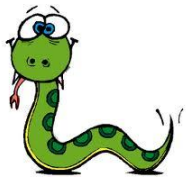
Iris Setosa

Iris Versicolor

Iris Virginica

[2] https://gist.github.com/curran/a08a1080b88344b0c8a7



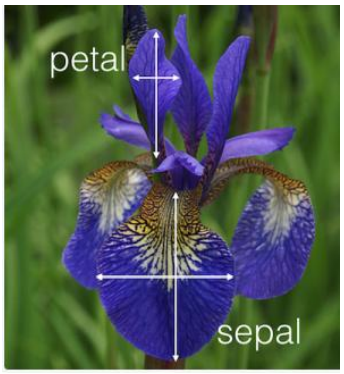



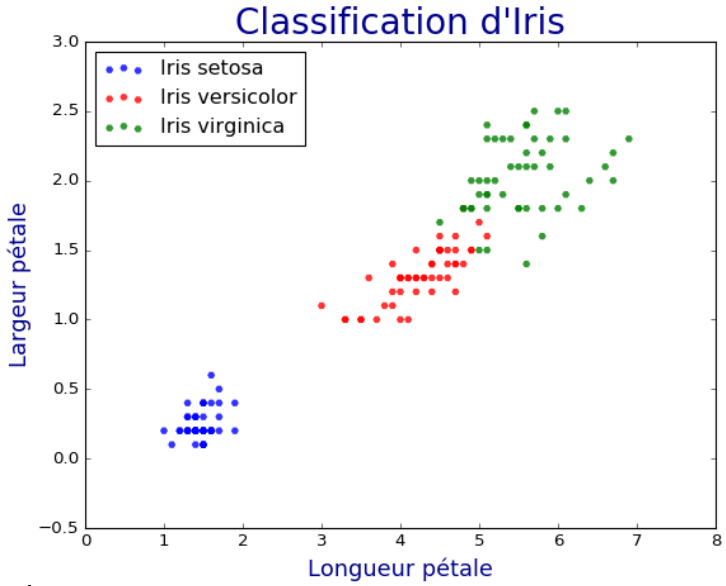

Le jeu de données contient 150 échantillons classés des trois espèces avec pour chacune d'entre elles les dimensions longueur - largeur des pétales et longueur - largeur des sépales en centimètres.

	sepal_length	sepal_width	petal_length	petal_width	species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3.0	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5.0	3.6	1.4	0.2	setosa

Ce dataset est disponible sous la forme d'un fichier csv. Voilà la classification obtenue à partir des dimensions des pétales :

iris_dataset.csv





Vous travaillerez sur cet exemple en exercices.

3 Un premier dataset sur les pokémon

3.1 Présentation du jeu de données

Nous utiliserons dans cette présentation un jeu de données proposé autour des caractéristiques de Pokémon[3].

pokemon.csv

```
1  Nom;Points de vie;Points d'attaque;Type
2  Ecayon;49;49;Eau
3  Tiplouf;53;51;Eau
4  Carabaffe;59;63;Eau
5  Prinplouf;64;66;Eau
6  Gobou;50;70;Eau

33 Mewtwo;106;150;Psy
34 Mewtwo;106;190;Psy
35 Symbios;110;65;Psy
```

Les caractéristiques de nos Pokemon :

Points de vie, points d'attaque et type. Il y a deux types Eau et Psy.

Le dataset contient 34 échantillons.

[3] Origine de l'idée et des tables csv PrépaBac Hatier NSI, les scripts python et l'exploitation sont de l'auteur.

3.2 Représentation du jeu de données : Matplotlib.

La représentation des données est cruciale pour comprendre le monde. Un outil très performant existe avec Python c'est la librairie Matplotlib. Vous ne perdrez pas votre temps si vous profitez d'un peu de votre temps libre pour découvrir et utiliser cette bibliothèque en prévision de vos travaux futurs quelque soit votre domaine[4].

« Je suis convaincu que la visualisation est l'un des moyens les plus puissants pour atteindre ses objectifs personnels » Harvey Mackay.

La visualisation des données répond à deux besoins principaux :

- explorer les données, les comprendre
- communiquer les données.

Nous allons donc représenter le jeu de données Pokémon pour cela il nous faut :

1. Lire le fichier csv
2. Préparer les données à répartir dans les deux classes
3. Utilisation de Matplotlib pour l'affichage

Dans les exemples ci-dessous les différentes étapes permettant l'affichage sont indiquées et les codes sont à compléter, cela sera fait en exercices

a) Lire le fichier csv

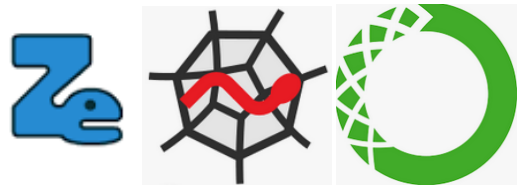
```
# Lecture de la table csv, le résultat est dans une
# liste de liste
with open('xxxxxxxxxx', encoding="utf8") as myFile:
    reader = csv.reader(myFile,delimiter='x')
    for row in reader:
        table.append(row)
```

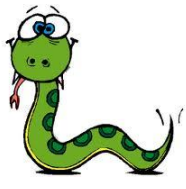
Voilà le résultat dans table :

```
>>> table
[['Nom', 'Points de vie', "Points d'attaque", 'Type'], ['Ecayon', '49', '49', 'E
au'], ['Tiplouf', '53', '51', 'Eau'], ['Carabaffe', '59', '63', 'Eau'], ['Prinpl
ouf', '64', '66', 'Eau'], ['Gobou', '50', '70', 'Eau'], ['Gamblast', '71', '73',

 'Psy'], ['Okeoke', '95', '23', 'Psy'], ['Mew', '100', '100', 'Psy'], ['Mewtwo',
 '106', '110', 'Psy'], ['Mewtwo', '106', '150', 'Psy'], ['Mewtwo', '106', '190',
 'Psy'], ['Symbios', '110', '65', 'Psy']]
```

Les données de nos Pokémons sont dans le type construit table.

[4] Voir la documentation officielle qui contient un très grand nombre d'exemples : https://matplotlib.org/






Deux remarques :

- Vous devez maintenant reconnaître la structure de données en table utilisée.
- Notez le type des valeurs points d'attaque et points de vies.

b) Préparation des données

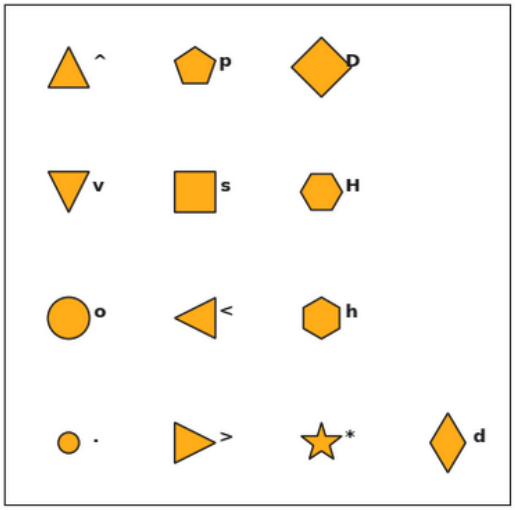
Nous allons préparer les données conformément au tutoriel de Matplotlib sur les nuages de points[5].

```
## Préparation des données à tracer
# Préparation des données à répartir dans deux classes
liste_Pokemon_g1 = [ [ int(x[1])  for x in table if x[3]=='Eau'] , \
                     [ int(x[2])  for x in table if x[3]=='Eau'] ]
liste_Pokemon_g2 = [ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  , \
                     [ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  ]

# Préparation des options pour le tracé matplotlib
data = [liste_Pokemon_g1, liste_Pokemon_g2]
colors = ("blue", "red")
groups = ("Eau", "Psy")
forme_marqueur = ("s","p")
```

c) Tracé du dataset

Voilà le code du tracé :

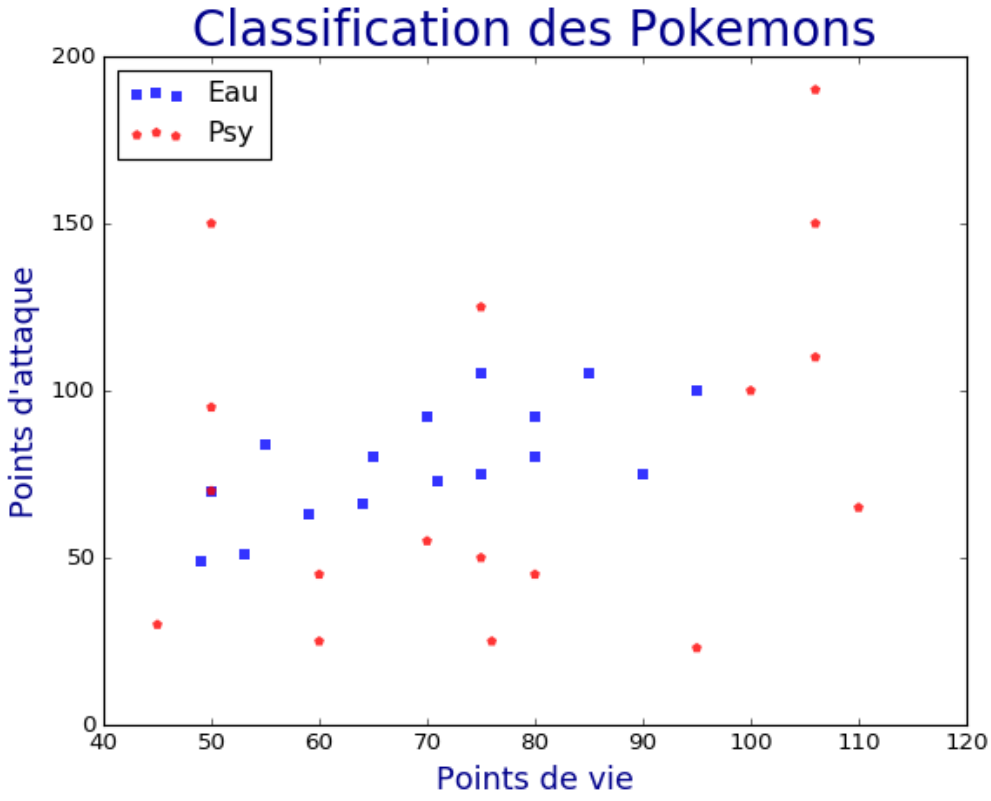
```
## Creation du tracé
# https://pythonspot.com/matplotlib-scatterplot/

# Création de l'objet figure pour matplotlib
fig = plt.figure()

# Tracé du nuage de points, scatter plot
# On utilise l'itération parallèle avec la fonction zip
for coordonnees, color, group, marque in zip(data, colors, groups, forme_marqueur):
    x, y = coordonnees
    plt.scatter(x, y, alpha=0.8, c=color, marker=marque, edgecolors='none', s=30, label=group)

# Préparation du titre
plt.title("Classification des Pokemons")
# Choix de la position de la légende dans le graphique
plt.legend(loc=2)
# Préparation des labels en abscisses et ordonnées
plt.xlabel("Points de vie")
plt.ylabel("Points d'attaque")
# Tout est prêt On trace le graphique
plt.show()
```



⇦

[5] https://pythonspot.com/matplotlib-scatterplot/

Et le résultat :



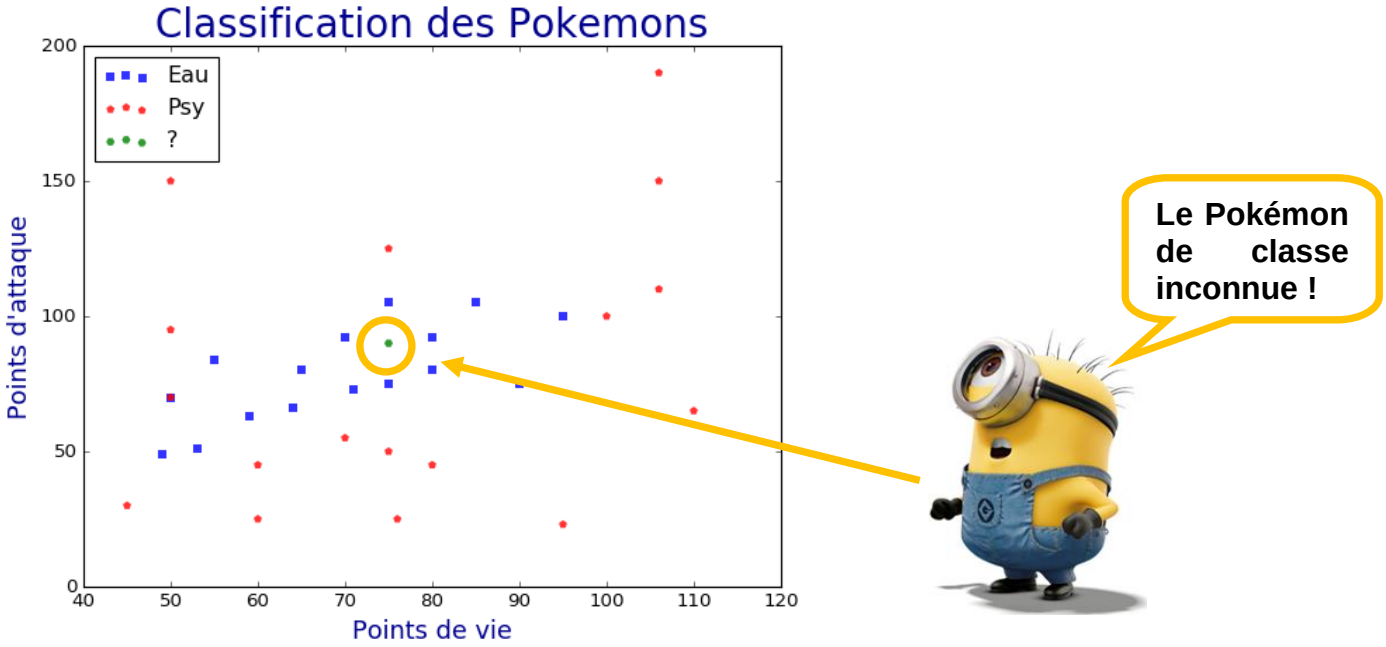
Nous observons bien que les données ont été classifiées. Nous allons pouvoir exploiter cette connaissance c'est-à-dire la classification à partir du jeu de données de tests pour identifier un Pokémon inconnu.

4 Algorithme des k plus proches voisins mise en oeuvre

4.1 Le problème à résoudre

Nous avons un nouveau Pokemon qui possède 75 de points de vie et 90 de points d'attaque de quelle classe est-il ?

Pour répondre à cette question nous allons mettre en œuvre l'algorithme des k plus proches voisins.



Intuitivement nous pouvons dire : « il est entouré de bleu donc il doit être bleu ! »

Quelques remarques :

- Il faut définir quelle notion de distance nous allons utiliser pour pouvoir dire : « *qui est le voisin de qui ?* ».
- Le nombre de voisins k ne permet pas toujours de d'identifier la classe. En effet si nous utilisons ici k=34 ! alors tous sont voisins dans ce cas comment conclure ?
- Le choix de l'hyperparamètre k est donc crucial mais comment le choisir ? On voit ici l'importance de la compétence 'métier' pour définir les bons algorithmes et les bons 'réglages' de ces algorithmes.

La Data science : est au croisement de trois métiers : l'informatique, les statistiques et l'expertise fonctionnelle (compétences pointues dans les données du métier cible de l'analyse Data).

4.2 Mise en œuvre

a) Comment définir la distance ?

Il existe plusieurs fonctions de calcul de distance[6] on choisit la fonction de distance en fonction des types de données qu'on manipule.

Pour les données quantitatives (exemple : poids, salaires, taille, etc.…) et du même type, la distance euclidienne est un bon candidat. Quant à la distance de Manhattan, elle est une bonne mesure à utiliser quand les données ne sont pas du même type (exemple : âge, sexe, longueur, poids etc.…).

Pour deux points d1(x1,y1) et d2(x2,y2) on peut utiliser :

☐ La distance géométrique naturelle ou distance euclidienne :

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

☐ La distance de Manhattan :

|x1-x2| + |y1-y2|

☐ La distance selon un seul axe :

|x1-x2| pour l'axe des x l'axe y n'intervient pas.

b) Trie de la table en fonction de la distance choisie[7]

Pour réaliser ce tri on utilise une fonction de tri sorted() qui permet de conserver la donnée itérable d'origine et qui en créée une nouvelle copie.

```
# On tri la table avec une fonction définie spécifiquement par key
# Le tri avec la fonction sorted construit une nouvelle liste la liste
# d'origine n'est pas modifiée
table_triee = sorted(table, key = distance_cible)
```

Nous utilisons la possibilité d'ajouter notre propre fonction pour réaliser le tri :

> `list.sort()` et `sorted()` ont un paramètre nommé *key* afin de spécifier une fonction qui peut être appelée sur chaque élément de la liste afin d'effectuer des comparaisons.

[6] https://mrmint.fr/introduction-k-nearest-neighbors
[7] https://docs.python.org/fr/3/howto/sorting.html

Il reste à définir la fonction distance_cible nous utiliserons la distance géométrique :

```
from math import sqrt

def distance_cible(donnee):
    distance = sqrt( ( int(donnee[1])-cible[0] )**2 + \
                     xxxxxxxxxxxxxxxxxxxxxx          \
                   )
    return distance
```

Et le résultat avec k = 12 :

```
Les  12  proche_voisins
['Bargantua', '70', '92', 'Eau']
['Poissoroy', '80', '92', 'Eau']
['Phione', '80', '80', 'Eau']
['Crocrodil', '65', '80', 'Eau']
['Mateloutre', '75', '75', 'Eau']
['Octillery', '75', '105', 'Eau']
['Gamblast', '71', '73', 'Eau']
['Aligatueur', '85', '105', 'Eau']
['Rosabyss', '55', '84', 'Eau']
['Tarpaud', '90', '75', 'Eau']
['Clamiral', '95', '100', 'Eau']
['Deoxys', '50', '95', 'Psy']
```

c) <u>Déduire le résultat final et répondre à la question : à quelle classe appartient le Pokémon inconnu</u>

Il nous reste à trouver quelle est le type de Pokémon le plus représenté. L'œil humain le voit tout de suite mais il faut automatiser le processus.

Cela vous revient, le travail devra être décomposé en deux étapes :

1) Faire le bilan des plus proches voisins en regroupant tous les types présents dans la liste finale. Le résultat pourra alors être mis sous la forme d'un dictionnaire :

```
{'Eau': 11, 'Psy': 1}
```

2) Trouver dans ce dictionnaire le type de Pokémon, correspondant ici à la clé, le plus représenté donc au final :

```
Classe du pokemon inconnu :  Eau
```

3) traité le cas particulier où plusieurs classes sont possibles.

5 Les limites de l'algorithme l'hyperparamètre k

Nous voyons que les résultats de notre analyse peuvent dépendre de la répartition des échantillons connus par rapport à notre élément à classer.

Cela dépendra également du nombre k pris pour effectuer le classement.

Nous voyons donc que la mise en œuvre de ces algorithmes demande une compétence Data et Métier pour pouvoir conduire correctement les analyses et aboutir aux résultats.

6 Ressources

6.1 Cours NSI P.G Vaucanson

NSI_TYPES_CONSTRUITS.pdf

6.2 Sur openclassroom

https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning/4022441-entrainez-votre-premier-k-nn

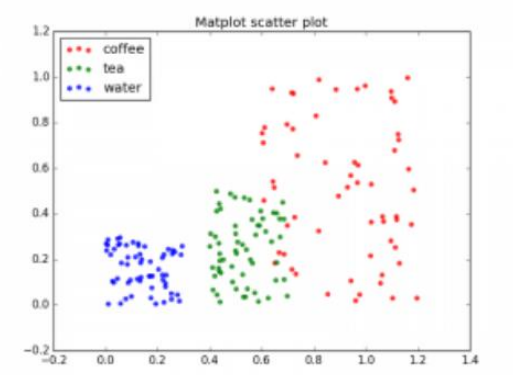
https://www.w3schools.com/python/ref_func_zip.asp

6.3 Matplotlib

a) Utilisation de scatter plot

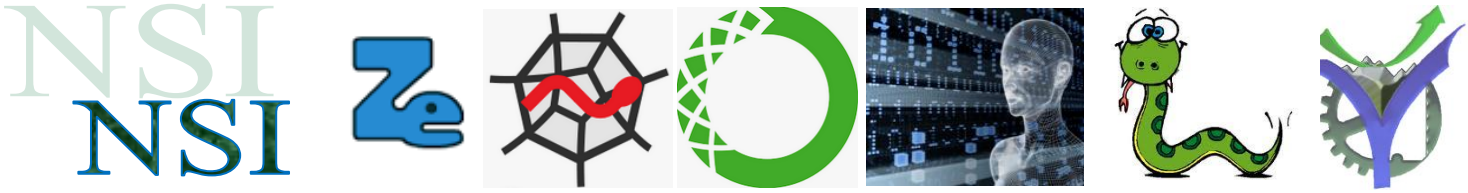
https://pythonspot.com/matplotlib-scatterplot/

```
import numpy as np
import matplotlib.pyplot as plt

# Create data
N = 60
g1 = (0.6 + 0.6 * np.random.rand(N), np.random.rand(N))
g2 = (0.4+0.3 * np.random.rand(N), 0.5*np.random.rand(N))
g3 = (0.3*np.random.rand(N),0.3*np.random.rand(N))

data = (g1, g2, g3)
colors = ("red", "green", "blue")
groups = ("coffee", "tea", "water")

# Create plot
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, axisbg="1.0")

for data, color, group in zip(data, colors, groups):
x, y = data
ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none', s=30, label=group)

plt.title('Matplot scatter plot')
plt.legend(loc=2)
plt.show()
```



b) Autres exemples d'utilisation

http://python-graph-gallery.com/scatter-plot/



https://sites.google.com/view/aide-python/graphiques/les-graphiques-courbes-et-nuages-de-points-scatter-plot

https://python.doctor/page-creer-graphiques-scientifiques-python-apprendre

https://python-graph-gallery.com/131-custom-a-matplotlib-scatterplot/

c) Analyse de données

https://www.datacorner.fr/