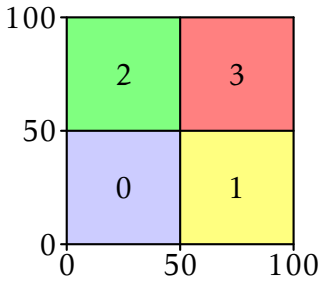
Python – Algorithme des k plus proches voisins

Représentation des éléments

Pour cette feuille, nous considérerons majoritairement des éléments représentés par un couple de deux nombres, comme `(3.45, 5.67)`. Un exemple sera un couple contenant un élément et sa catégorie, comme `((3.45, 5.67), 'avion')`.

Afin de tester nos fonctions, nous allons générer des exemples en partageant un carré de côté 100 en 4 carrés numérotés de 0 à 3. La catégorie d'un point sera le nombre associé au sous-carré. Le coin inférieur gauche servira d'origine du repère.
Afin de rendre le programme plus modulaire, vous pouvez rajouter les constantes suivantes en début de votre fichier :




```
xmin = 0
xmax = 100
ymin = 0
ymax = 100
xmoy = (xmax+xmin)/2
ymoy = (ymax+ymin)/2
```

Un échantillon sera un dictionnaire où chaque catégorie sera associée à une liste de points.

Exercice 1 : Écrire une fonction `generation_echantillon(n)` qui renvoie un échantillon contenant les catégories 0 à 3, avec `n` points pour chacune. Vous pouvez utiliser la fonction `uniform(a, b)` du module `random` qui tire un réel au hasard compris entre `a` et `b` inclus.

```
>>> generation_echantillon(2)
{0: [(27.762650624230396, 25.20359285195414),
     (3.605324504341562, 33.4065781204959)],
 1: [(69.81709501025465, 38.9218695667356),
     (63.79188810839042, 24.611569414080282)],
 2: [(9.432136187628027, 57.67143576793811),
     (27.89754878233496, 72.72498286968712)],
 3: [(69.49185092272312, 57.480508827597795),
     (74.63735833708166, 77.51133144667148)]}
```

Afin d'afficher les point obtenus, vous pouvez utiliser la fonction suivante :

```
import matplotlib.pyplot as plt

def affichage_echantillon(echantillon):
    #plt.axis('equal')
    for cat in echantillon:
        liste = echantillon[cat]
        x, y = list(zip(*liste))
        plt.scatter(x, y, label=str(cat))
    plt.legend()
    plt.show()
```

Si vous utilisez cette fonction, vous remarquerez que les unités ne sont pas les mêmes sur les deux axes. Il suffit de décommenter la première ligne pour obtenir cette égalité, mais le rendu est un peu moins joli et cela pourrait poser problème pour certains exemples.

Distances

Afin de pouvoir chercher les k plus proches voisins, il faut avoir une notion de distance. Nous allons considérer la distance euclidienne et la distance de Manhattan.
La fonction `sqrt(n)` du module `math` permet de calculer la racine carrée de `n` et `abs(n)` calcule la valeur absolue de `n`.

```
>>> import math
>>> math.sqrt(398)
19.949937343260004
>>> abs(-5)
5
>>> abs(5)
5
```

Les points étant des couples de valeurs, il faut les décomposer pour calculer les distances :

```
>>> x, y = (20, 50)
```

Exercice 2 : Écrire une fonction `distance_euclidienne(pt1, pt2)` qui renvoie la distance euclidienne entre les deux points.

```
>>> distance_euclidienne((0,0), (4,0))
4.0
>>> distance_euclidienne((1,-3), (2,7))
10.04987562112089
```

Exercice 3 : Écrire une fonction `distance_manhattan(pt1, pt2)` qui renvoie la distance de Manhattan entre les deux points.

```
>>> distance_manhattan((0,0), (4,0))
4
>>> distance_manhattan((1,-3), (2,7))
11
```

Recherche des k plus proches voisins

Afin de pouvoir trouver les k plus proches voisins, nous allons construire une liste de triplets de la forme `(dist, pt, cat)`, où `dist` est la distance du point `pt`, de catégorie `cat`, avec l'élément considéré.
Pour trier la liste, vous pouvez soit utiliser `liste.sort()` ou `l = liste.sorted()`. Pour ne garder que les `k` premières valeurs, vous pouvez utiliser :

```
>>> l = [1, 5, 90, 7680, 99920]
>>> l[:2]
[1, 5]
>>> k = 3
>>> l[:k]
[1, 5, 90]
```

Exercice 4 : Écrire la fonction `k_plus_proches_voisins(cible, echantillon, k, distance)` qui renvoie la liste des `k` plus proches voisins de `cible` selon la distance `distance` dans l'échantillon `echantillon`. Vous pouvez construire la liste des triplets `(dist, pt, cat)`, la trier, puis ne garder que les `k` premiers résultats.

```
>>> k_plus_proches_voisins((20,20), echantillon, 3, distance_euclidienne)
[(7.1846487991039165, (22.130632728353085, 13.138543700981836), 0),
 (15.397111123887077, (35.01027454088991, 23.42967770620497), 0),
 (17.92707320889055, (29.4015044708685, 35.2640645806142), 0)]
```

Catégorisation

Une fois la liste des k plus proches voisins obtenue, il faut déterminer la catégorie majoritaire dans cette liste. Pour cela, nous allons parcourir la liste en comptant le nombre d'apparitions de chaque catégorie. Pour compter ces apparitions, nous utiliserons un dictionnaire qui à chque catégorie associe le nombre d'éléments vus.
Pour déplier les éléments de la liste, il faut utiliser des affectations multiples :

```
>>> dist, pt, cat = (7.18, (22.13, 13.13), 0)
>>> dist
7.18
>>> pt
(22.13, 13.13)
>>> cat
0
```

Exercice 5 : Écrire une fonction `decision(liste_voisins)` qui renvoie la catégorie majoritaire. En cas d'égalité, la première catégorie arrivée au maximum sera considérée comme la catégorie majoritaire. Vous pourrez faire une recherche de la catégorie majoritaire pendant l'analyse de la liste, en utilisant un algorithme de recherche de maximum.
Dans l'exemple suivant, les valeurs sont totalement artificielles pour simplifier l'écriture.

```
>>> decision([(0.2,(2,2),0),(0.4,(3,1.5),1),(0.7,(4,6),0)])
0
>>> decision([(0.2,(2,2),0),(0.4,(3,1.5),1),(0.7,(4,6),2)])
0
>>> decision([(0.2,(2,2),0),(0.4,(3,1.5),1),(0.7,(4,6),1),(1.9,(9,-7),0)])
1
```

Application au iris

En 1936, Edgar Anderson, un botaniste américain, a collecté les mesures de nombreuses iris de 3 espèces : setoca, viginica et versicolor.


iris setosa


iris versicolor

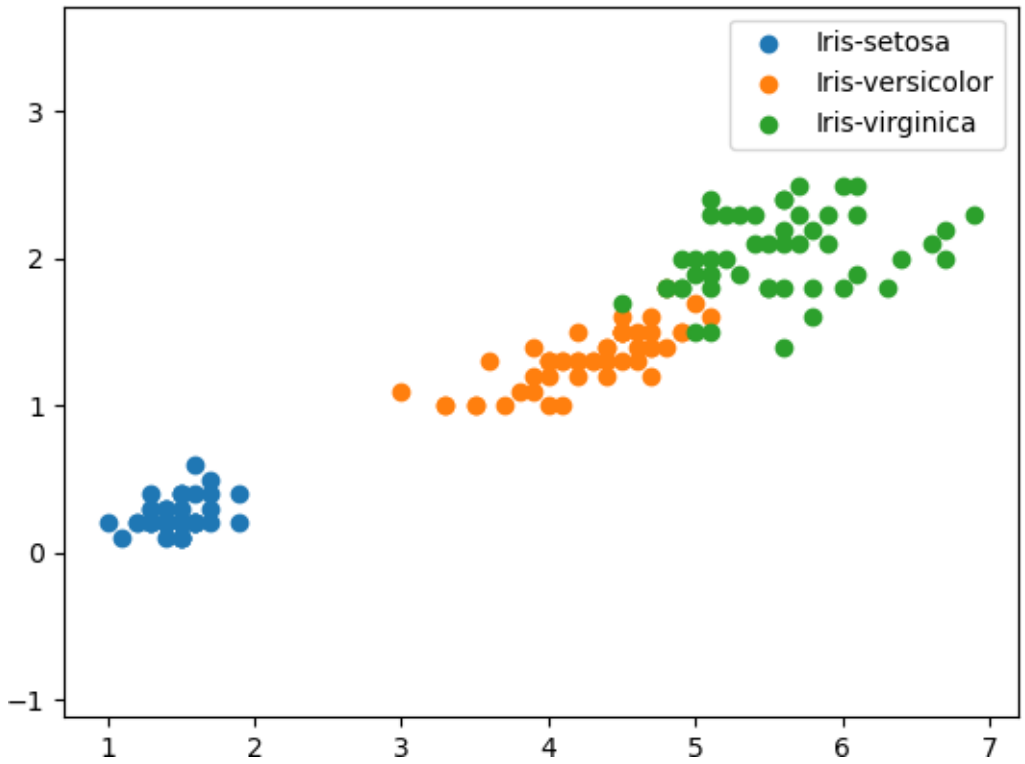

iris virginica

Pour chacune, il a mesuré la largeur et la longueur des sépales et des pétales. Ces mesures peremettent de déteminer l'espèce d'une iris. Les données sur les pétales sont stockées dans le fichier `iris.csv`.
Pour importer ces données, vous pouvez utiliser la fonction suivante :

```
def importation_iris():
    with open('iris.csv', 'r', encoding='utf-8') as fichier:
        echantillon = dict()
        for iris in csv.DictReader(fichier, delimiter = ','):
            cat = iris["espece"]
            x = float(iris["longueur_petale"])
            y = float(iris["largeur_petale"])
            if cat in echantillon:
                echantillon[cat].append((x, y))
            else:
                echantillon[cat] =[(x, y)]
    return echantillon
```

Vous pouvez afficher visualiser ces exemples en utilisant les fonctions précédemment définies :

```
>>> iris = importation_iris()
>>> affichage_echantillon(iris)
```



Exercice 6 : Pour chacune des iris proposées et du nombre de voisins à considérer, déterminez l'espèce en utilisant la distance euclidienne.

1) Pétales de 2 cm de long et 0,5 cm de large et les 3 plus proches voisins.
2) Pétales de 2,5 cm de long et 0,75 cm de large et les 3 plus proches voisins.
3) Pétales de 5,1 cm de long et 1,7 cm de large et les 3 plus proches voisins.
4) Pétales de 5,1 cm de long et 1,7 cm de large et les 5 plus proches voisins.

Recherche de la valeur de k

Comme nous l'avons vu, le résultat obtenu peut différer selon la valeur de k utilisée. Afin de trouver la "meilleure" valeur de k utiliser, il est possible d'en tester plusieurs pour trouver celle qui donne les meilleurs résultats. Pour cela, il faut un nouvel échantillon que l'on peut classifier de façon exacte.
Nous allons reprendre notre carré avec 4 zones et générer de nouvelles valeurs, mais cette fois en nous concentrant sur les frontières entre les zones pour tester les valeurs limites. La fonction suivante permet de générer un échantillon de test :

```
def generation_echantillon_test(n):
    liste = []
    for i in range(n):
        if i%2 == 0:
            x = random.gauss(xmoy, (xmax-xmin)/10)
            y = random.uniform(ymin, ymax)
        else:
            x = random.uniform(xmin, xmax)
            y = random.gauss(ymoy, (ymax-ymin)/10)
        liste.append((x, y))
    return liste
```

La fonction `gauss(mu, sigma)` génère une valeur aléatoire selon une loi normale d'espérance `mu` et d'écart-type `sigma`. C'est à dire que les valeurs obtenues sont proches de `mu` et plus `sigma` est grand, plus elles seront proches de cette valeur moyenne.
Les valeurs générées ont donc une chance sur deux d'être proches de la frontière horizontale ou de la frontière verticale. Afin de déterminer de façon exacte la catégorie à laquelle appartient chacun des points de l'échantillon de test, nous allons utiliser la fonction suivante :

```
def determiner_categorie(pt):
    x, y = pt
    reponse = 0
    if x > xmoy:
        reponse += 1
    if y > ymoy:
        reponse += 2
    return reponse
```

Exercice 7 : Écrire une fonction `taux_reussite(echant_test, echantillon, k, distance)` qui renvoie le taux de réussite obtenu en comparant les valeurs attendues avec les valeurs obtenues pour toutes les valeurs de `echant_test` en utilisant l'algorithme des k plus proches voisins sur l'échantillon et la distance donnés.

```
>>> echantillon = generation_echantillon(10)
>>> echant_test = generation_echantillon_test(100)
>>> taux_reussite(echant_test, echantillon, 3, distance_euclidienne)
0.67
>>> taux_reussite(echant_test, echantillon, 3, distance_manhattan)
0.65
>>> taux_reussite(echant_test, echantillon, 7, distance_euclidienne)
0.7
>>> taux_reussite(echant_test, echantillon, 7, distance_manhattan)
0.69
```

Vous pouvez remarquer que lorsque la taille de l'échantillon initial augmente, le taux de réussite augmente également.

```
>>> echantillon = generation_echantillon(100)
>>> echant_test = generation_echantillon_test(100)
>>> taux_reussite(echant_test, echantillon, 7, distance_euclidienne)
0.93
>>> echantillon = generation_echantillon(1000)
>>> echant_test = generation_echantillon_test(100)
>>> taux_reussite(echant_test, echantillon, 7, distance_euclidienne)
0.98
```

Exercice 8 : Écrire une fonction `choix_k(echant_test, echantillon, k_max, distance)` qui renvoie la valeur de k comprise entre 1 et `k_max` qui renvoie le meilleur taux de réussite pour les échantillons donnés.

```
>>> echantillon = generation_echantillon(10)
>>> echant_test = generation_echantillon_test(100)
>>> choix_k(echant_test, echantillon, 10, distance_euclidienne)
9
>>> choix_k(echant_test, echantillon, 10, distance_manhattan)
8
```

La meilleur valeur de k peut être différente ou la même selon les distances et les échantillons. Lorsque la taille de l'échantillon augmente, le temps de calcul nécessaire augmente aussi. Pour limiter cette augmentation, il est possible de ne pas rechercher les k plus proches voisins pour chaque valeur de k, mais plutôt de le faire pour `k_max` et ensuite, lors de l'étude de la liste de plus proches voisins, il suffit de noter à chaque étape si la catégorie trouvée est la bonne ou pas.

Pour aller plus loin

Pour la recherche des k plus proches voisins, nous avons adopté une approche simple qui consiste à calculer la distance avec chacun des éléments et ensuite à les classer. Il est bien plus efficace de maintenir une liste de k résultats et d'insérer au bon endroit tout nouvel élément qui serait plus proche que le k-ième plus proche. Si la taille de l'échantillon est très grand, cela permet de ne pas avoir à trier 1 000 ou 10 000 valeurs pour ne garder que les 3 premières. Pour cela il faut utiliser l'algorithme suivant :

- Construire une liste de k valeurs avec des distances très grandes.
  Par exemple `(10**10, (0, 0), 'nul')`.
- Pour chaque élément de l'échantillon, calculer sa distance à la cible et si cette distance est inférieure au dernier de la liste, le mettre en dernière position, puis le faire remonter en l'échangeant avec l'élément précédent jusqu'à ce qu'il ait trouvé sa place.

On obtient ainsi la liste des k plus proches voisins une fois que tous les éléments ont été examinés.

Exercice 9 : Modifiez `k_plus_proches_voisins(cible, echantillon, k, distance)` pour qu'elle utilise cet algorithme.

Dans la base des iris, il y a également les informations sur les sépales. Cela fait donc 4 mesures par iris. Toutes les données se trouvent dans le fichier `iris-complet.csv`.

Exercice 10 : Modifiez la fonction `importation_iris()` en `importation_iris(abscisse="longueur_petale", ordonnee="largeur_petale")` afin de pouvoir choisir quelles me-

sures iront sur chacun des axes. Les quatres valeurs possibles sont `"longueur_petale"`, `"largeur_petale"`, `"longueur_sepale"` et `"largeur_sepale"`.

Exercice 11 : Déterminez l'espèce d'une iris ayant des sépales de 5,9 cm de long et 3 cm de large avec :

1) les 3 plus proches voisins
2) les 5 plus proches voisins

Il est également possible de prendre en compte les 4 paramètres en même temps. Pour cela, il faut modifier les distances pour qu'elles puissent calculer l'écart entre deux points à n coordonnées.

Exercice 12 : Modifiez la fonction `distance_euclidienne(pt1, pt2)` pour qu'elle accepte des n-uplets de longueur quelconque et calcule la distance entre les deux. Les deux n-uplets doivent être de même longueur.

```
>>> distance_euclidienne((1,2,3), (3,4,5))
3.4641016151377544
>>> distance_euclidienne((1, 2, 3, 4), (3, 4, 5, 6))
4.0
```

Exercice 13 : Écrire une fonction `importation_iris_complet()` qui renvoie l'échantillon obtenu à partir du fichier `iris-complet.csv`. Chacun des éléments de l'échantillon est représenté par un 4-uplet composé de la longueur et la largeur d'un pétale et la longueur et la largeur d'un sépale.
Par exemple, une iris dont les pétales mesurent 5,1 cm de long et 1,4 cm de large, et les sépales mesurent 5,9 cm de long et 3 cm de large sera représentée par `(5.1, 1.4, 5.9, 3)`.

Exercice 14 : Déterminer l'espèce de l'iris de l'exemple précédent en utilisant :

1) les 3 plus proches voisins
2) les 5 plus proches voisins