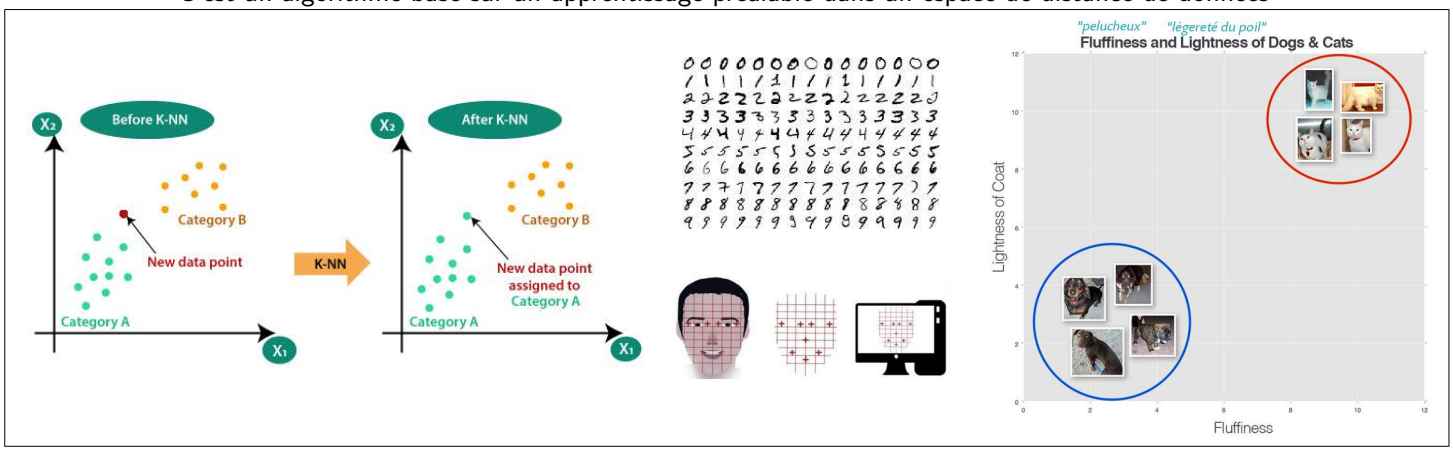
Algorithme des « k plus proches voisins »

Contenus	Capacités attendues
Algorithme des k plus proches voisins	Écrire un algorithme qui prédit la classe d'un élément en fonction de la classe majoritaire de ses k plus proches voisins.

Introduction

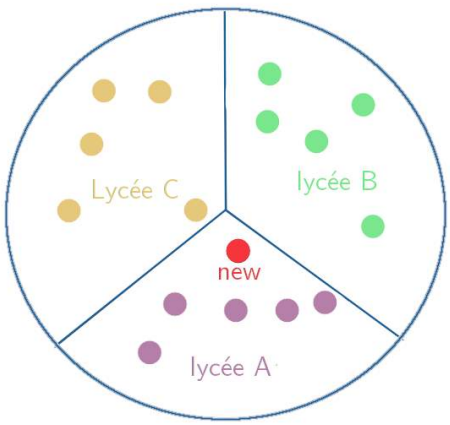
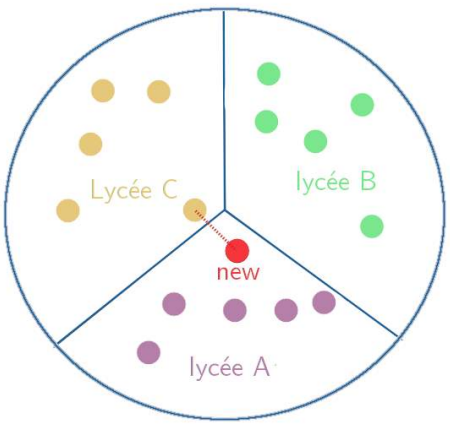
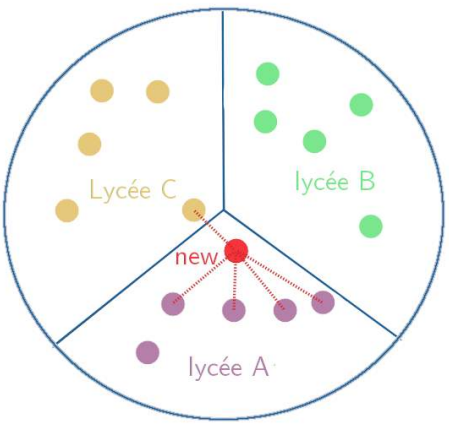
En abrégé k-NN ou KNN, de l'anglais k-nearest neighbors pour les « k plus proches voisins »
(nearest : plus proches) (neighbors(US) ou neighbours(UK) : voisins)

C'est un algorithme basé sur un apprentissage préalable dans un espace de distance de données



Les algorithmes vus jusqu'à présent apportaient une réponse exacte à un problème à résoudre. Parfois il n'est pas possible d'apporter une réponse « exacte » à un problème, on peut simplement apporter une réponse « plausible ».
Dans cette catégorie d'algorithmes, on peut citer ces algorithmes par apprentissage qui sont utilisés en traduction automatique ou encore en reconnaissance automatique diverse (caractères, visages, voix, etc.).
L'idée est de fournir à l'algorithme une grande quantité de données dont on connaît la réponse exacte, lorsque qu'on soumet une donnée similaire, l'algorithme utilise sa base connue puis applique des calculs divers pour donner la réponse la plus probable.

La notion de carte scolaire vous est sans doute connue. En fonction de votre adresse vous avez un lycée de secteur affecté. On peut imaginer un algorithme qui à partir d'une localisation d'adresse en entrée puisse vous donner quel est votre lycée de secteur le plus probable. L'idée sera de calculer la distance du lieu donné en entrée par rapport à d'autres adresses peu distantes dont on connaît le lycée affecté et d'en déduire le lycée le plus probable.

	 k = 1	 k = 5
répartition carte scolaire on place une nouvelle adresse new	1 voisin : résultat : Lycée C	5 voisins : résultat : Lycée A

1) Influence de k

Un trop petit nombre de voisins peut être évidemment trompeur. En prenant exagérément ici k = 1, avec un voisin immédiat proche dans la mauvaise zone, l'algorithme ne renvoie pas le lycée plausible.
Avec k = 5, on a ici un cas moyen cohérent, qui renverrait ici le bon lycée d'affectation
Une trop grande valeur de k a des défauts aussi : on prend en compte des éléments qui peuvent être trop éloignés et en fait peu significatifs. On augmente la durée des calculs et sans améliorer le choix final.

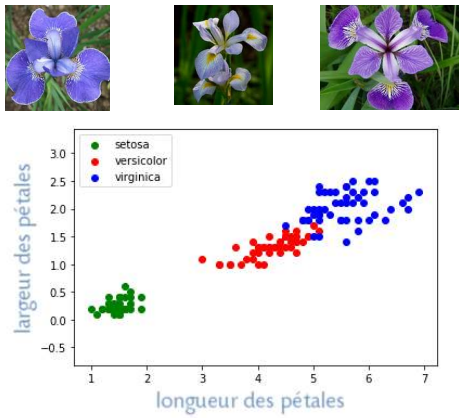
Voir Visualisations complémentaires sur une animation : http://vision.stanford.edu/teaching/cs231n-demos/knn/

2) Définitions autour de l'algorithme

Sur des représentations graphiques XY

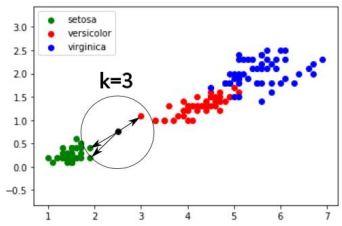
descripteurs	étiquette
Chaque enregistrement contient 2 descripteurs. Ils donnent la **position** du point	une étiquette qui donne la **classe** du point.

Ressource libre Iris DataSet
https://gist.github.com/curran/a08a1080b88344b0c8a7



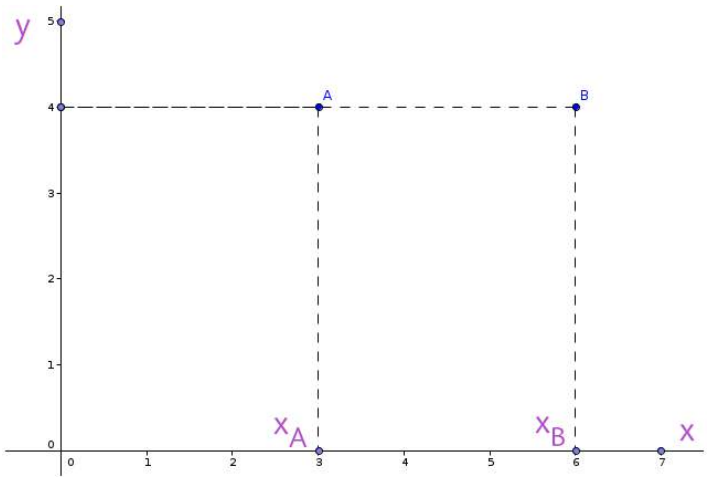
Exemple sur une classification commune d'iris

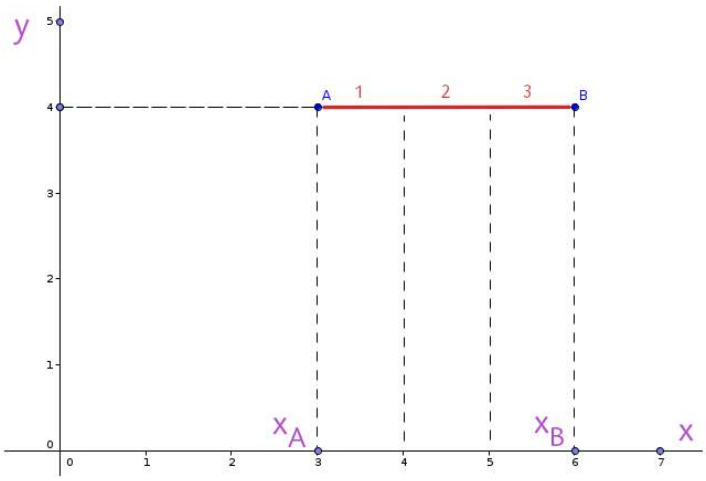
descripteurs	La **variété** (classe) de l'iris indique 3 étiquettes :
largeur des pétales (en Y) **longueur** des pétales (en X)	*setosa*, *versicolor*, *virginica*



3) Évaluation de distances en XY : comment les mesurer, comment les programmer

cas de points **alignés** : distance Euclidienne **AB**	cas de points **alignés** : distance Manhattan **AB**
distance **géométrique** dans le repère x,y	on se déplace ici comme dans les rues de l'île de Manhattan (répartition américaine en « **quadrillage** » des rues)





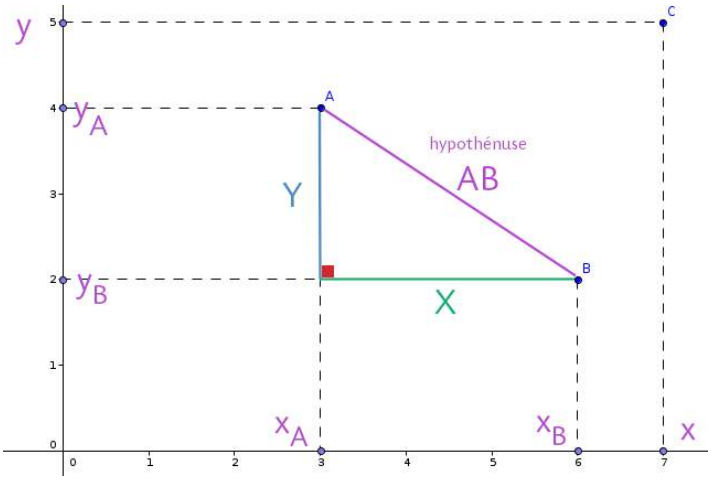
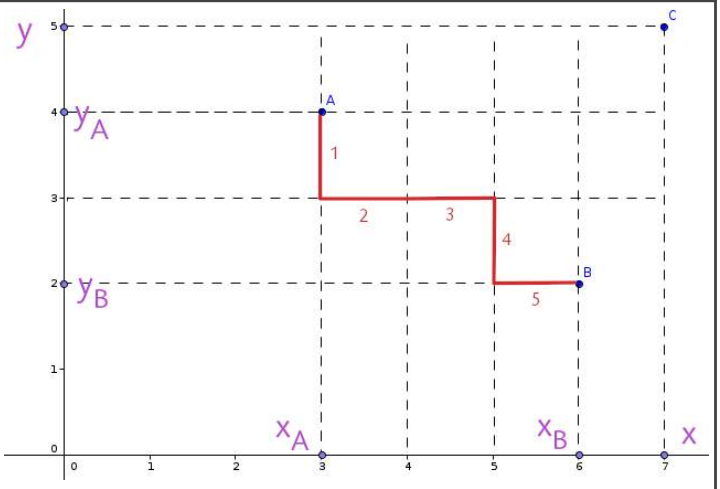
ici les deux distances sont équivalentes : on a le calcul sur l'axe x suivant : $AB = |x_B - x_A| = |6 - 3| = 3$

Programmation

```
A = (3,4)          #coordonnées sous forme de tuple (x,y)
B = (6,4)
d = abs(B[0]-A[0])
```

```
print("distance AB : ", d)
```

```
distance AB :  3
```

cas des points **non alignés** : distance Euclidienne **AB**	cas des points **non alignés** : distance Manhattan **AB**
	
Rappel : vu en 2nde distance euclidienne dans un repère orthonormé du plan ainsi on a : $AB^2 = (x_B - x_A)^2 + (y_B - y_A)^2$ ou : $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$ $AB = \sqrt{(3)^2 + (-2)^2} = \sqrt{9+4}$ $AB = \sqrt{13} = 3,6$ $\quad AB = 3,6$	On doit alors considérer les valeurs absolues des distances selon x et y correspondant ainsi au comptage $AB = \lvert x_B - x_A\rvert + \lvert y_B - y_A\rvert = \lvert 6-3\rvert + \lvert 2-4\rvert$ $AB = \lvert 3\rvert + \lvert -2\rvert = 3+2 = 5$ $\quad AB = 5$

Programmation
généralisation $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
`import math`
`A=(3,4)`
`B=(6,2)`
`d=math.sqrt((B[0]-A[0])**2 + (B[1]-A[1])**2)`
`print("distance AB : ", d)`
`distance AB : 3.605551275463989`
	Programmation
	généralisation $AB = \lvert x_B - x_A\rvert + \lvert y_B - y_A\rvert$
	`A=(3,4)`
	`B=(6,2)`
	`d=abs(B[0]-A[0])+abs(B[1]-A[1])`
	`print("distance AB : ", d)`
	`distance AB : 5`

Pour vous entraîner : retrouver les valeurs pour la distance BC

distance Euclidienne **BC** $réponse : euclidienne : BC = 3,16$	distance Manhattan **BC** $réponse : Manhattan : BC = 4$

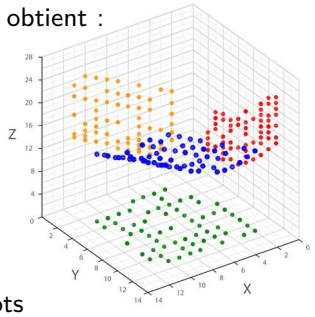
Généralisation à N descripteurs de la distance euclidienne

La distance entre deux points dont les positions sont $(x_A ; y_A)$ et $(x_B ; y_B)$ est donnée par la formule :

$$dist(A,B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Si on a **trois** descripteurs numériques pour chaque point : $(x_A ; y_A ; z_A)$ et $(x_B ; y_B ; z_B)$ on obtient :

$$dist(A,B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$



Cela se généralise ainsi à quatre ,cinq ... N descripteurs.

3) Évaluation de « distances sur un texte »

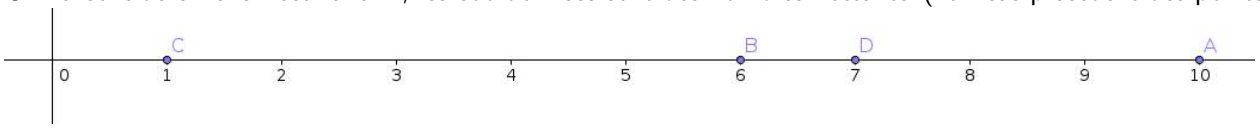
En classification de texte, on peut utiliser la distance de **Hamming** pour différencier deux mots

mot 1	chameau	Auzon	canard
mot 2	chapeau	Auront	cannard
distance de Hamming	1	2	4

On voit que cette distance de Hamming est peu représentative au niveau syntaxique car on voit une distance élevée pour une simple lettre ajoutée pour cannard qui ressemble à canard , mais une distance courte pour chameau et chapeau qui sont deux racines différentes

Un premier exemple simple de programmation en python

On va considérer ici un seul axe X, les coordonnées sont des nombres flottants (voir cas précédent des points alignés)



On met les points connus dans une liste L de nombres flottants : L = [1.0, 6.0, 7.0, 10.0]

On va placer un nouveau point et définir une fonction qui renverra le plus proche voisin

La fonction **lePlusProchevoisin(L,x)** prend en paramètres :

- une liste L des voisins
- un flottant , un point x quelconque dont on cherche le voisin le plus proche.

La fonction renvoie le voisin le plus proche sous la forme de la valeur du flottant correspondant

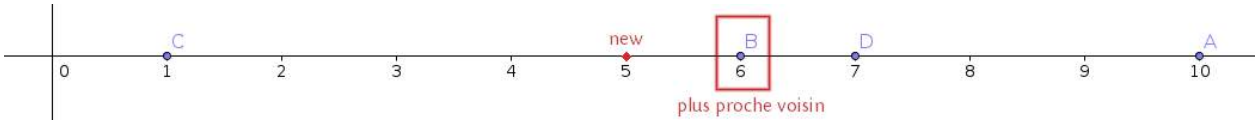
```
L = [ 1.0, 6.0, 7.0, 10.0]

def lePlusProchevoisin(L:list,x:float)->float:
    lepluspres= 0
    distanceMin = abs(L[0]- x)
    for indice in range (len (L)):
        distance = abs( L[indice]- x)
        if distance != 0 and distance < distanceMin:
            lepluspres = indice
            distanceMin = distance
    return L[lepluspres]


print(lePlusProchevoisin(L,5.0))
```

dans le parcours de la boucle for, la variable `distanceMin` prend successivement les valeurs :
(distance avec C 4.0) `distanceMin` 4.0
(distance avec B 1.0) `distanceMin` 1.0
(distance avec A 5.0) `distanceMin` reste à 1.0 avec B
À ce stade la variable `lepluspres` est à l'indice « 1 »

La fonction `lePlusProchevoisin(L,x)` renvoie **6.0** :
la valeur de L[1]) car c'est l'indice « 1 » de B qui est stocké dans la variable `lepluspres` et qui correspond à la valeur `distanceMin` de 1.0 (non retournée ici)



Affichera : 6.0

Limites de l'algorithme des plus proches voisins

Le principe de l'algorithme peut s'appliquer dans beaucoup de situations, cependant il ne donne pas toujours des résultats pertinents. Basé sur la notion de distance, il faut que cette distance soit vraiment représentative.