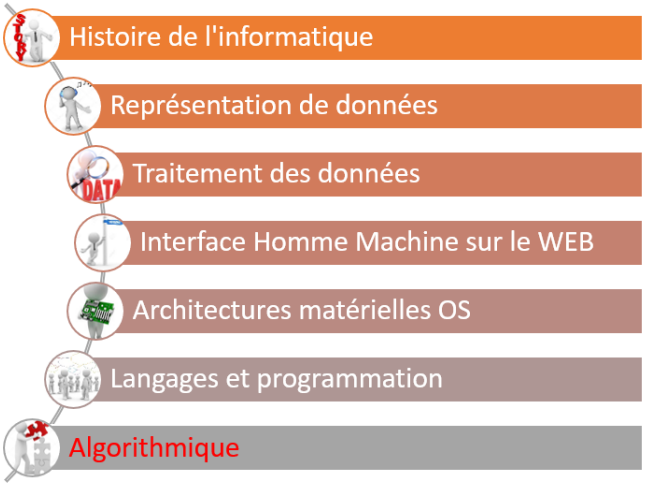
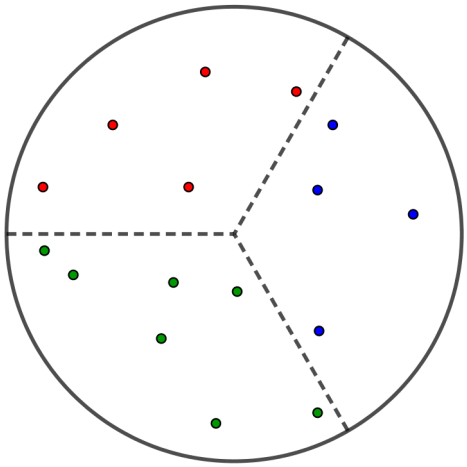


1 Situation déclenchante : deviner le lycée de rattachement d'un élève.

En France, les élèves de troisième, lorsqu'ils choisissent de poursuivre leurs études dans un lycée général, sont affectés dans un lycée dit "de rattachement" suivant une carte scolaire.

Imaginons une ville (représentée par le disque ci-dessous) comptant 3 lycées (le lycée rouge, le lycée vert et le lycée bleu).
L'affectation d'un élève dans un lycée de la ville est basée sur le secteur dans lequel il habite.
Sur la figure ci-dessous, on a fait apparaitre la ville et le lycée de rattachement de 16 élèves.
Ne connaissant pas les différents secteurs, on peut prévoir le lycée de rattachement d'un nouvel élève a de cette ville en sondant ses plus proches voisins.



La question que l'on se pose : "Peut-on prévoir le lycée de rattachement d'un nouvel élève a de cette ville ?" a pour but de déterminer à quelle *catégorie*, à quelle *classe* appartient ce nouvel élément, c'est un *problème de classification*.

Nous allons maintenant travailler sur un algorithme d'apprentissage automatique, souvent appelé, même en français, algorithme de machine learning. L'idée est d'utiliser un grand nombre de données afin "d'apprendre à la machine" à résoudre un certain type de problème (nous verrons un exemple dans le troisième paragraphe).
Cette idée d'apprentissage automatique et le terme de machine learning ont été utilisés pour la première fois par l'informaticien américain Arthur Samuel en 1959, il s'agit fournir à la machine en ensemble de données de qualité et en quantité suffisante pour qu'elles apprennent à résoudre un problème.
Le machine learning est un pan de l'informatique très utilisé puisqu'aujourd'hui sur internet les ressources de données disponibles sont gigantesques. Par exemple de nombreux sites commerciaux traquent nos données pour afficher des publicités susceptibles de nous intéresser, pour nous proposer de nouveaux articles à acheter etc grâce à des algorithmes de machine learning.
L'algorithme d'apprentissage que nous allons découvrir dans ce chapitre est : l'algorithme des "k plus proches voisins", le $k$ppv (en anglais "k nearest neighbors" : knn)

2 Généralités

2.1 Classification avec l'algorithme des k plus proches voisins

Un problème de classification est caractérisé par :

- un **ensemble $\mathcal{C}$ de classes** : dans notre exemple il s'agit des 3 lycées (vert, rouge et bleu) ;
- un ensemble $\mathcal{B}$ d'éléments, appelé **base d'apprentissage**, tel que pour tout élément e de $\mathcal{B}$ on connait sa classe $c(e)$: dans notre exemple, il s'agit des 16 premiers lycéens ;
- **un ensemble $\mathcal{E}$** d'éléments dont on ne connait pas la classe : il s'agit dans notre exemple (voir ci dessous) des 3 lycéens a, b et c dont on ne connait pas le lycée de rattachement.

L'objectif est alors de produire un algorithme qui, pour chaque élément e de $\mathcal{E}$ détermine sa classe $c(e)$.

Cet algorithme repose sur une **notion de distance** dans $\mathcal{E}$, c'est à dire d'une fonction *dist* qui à une paire d'éléments de $\mathcal{E}$ associe un nombre réel positif ou nul mesurant la différence entre deux éléments (des valeurs proches de 0 si les éléments sont proches, de plus en plus grandes si les éléments sont très différents).

L'algorithme dépend également d'**un paramètre k**, entier naturel strictement positif indiquant le nombre de voisins à prendre en compte.

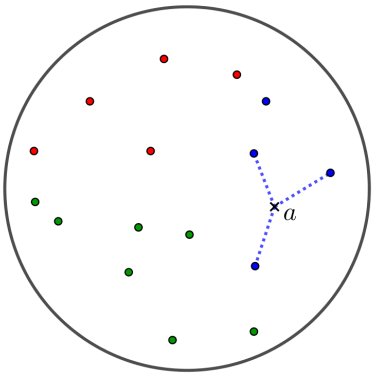
Tous ces points étant fixés, **l'algorithme des k plus proches voisins prend en entrée un élément e de $\mathcal{E}$ et renvoie une classe c de $\mathcal{C}$ pour laquelle il est plausible au vu des associations connues dans $\mathcal{B}$ que c soit égale à $c(e)$.**

L'algorithme procède ainsi :

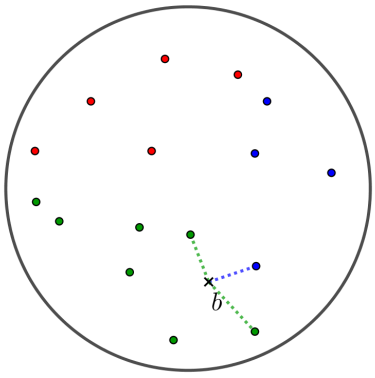
1. chercher parmi les éléments de la base d'apprentissage $\mathcal{B}$ les k éléments ayant les plus faibles distances à e ;
2. collecter les classes associées à ces k éléments ;
3. déterminer la classe la plus souvent présente parmi les classes retenues.

2.2 Retour à notre exemple

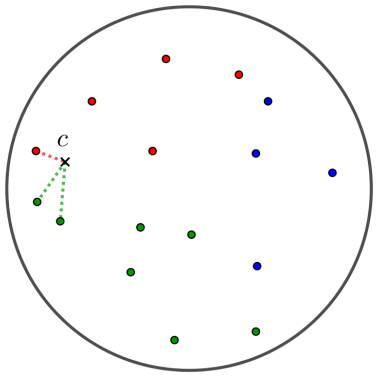
Pour appliquer l'algorithme des k plus proches voisins, on utilise la fonction *dist* correspondant à la distance géographique usuelle.
On considère $\mathcal{E} = \{a\ ;\ b\ ;\ c\}$, et alors pour $k = 3$, les résultats affichés sont :



3 *voisins* : ●●●
résultat : ●



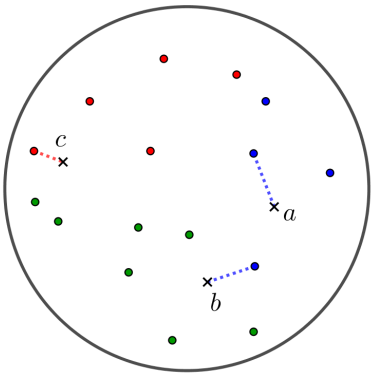
3 *voisins* : ●●●
résultat : ●



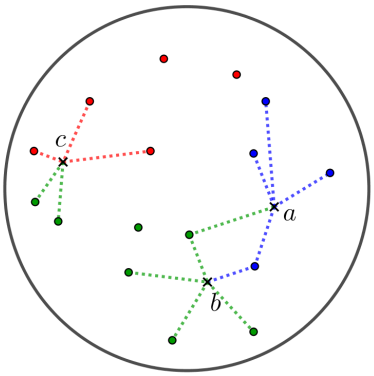
3 *voisins* : ●●●
résultat : ●

2.3 Influence du paramètre k

La réponse apportée par l'algorithme est susceptible de changer si l'on utilise d'autres valeurs de k.
Ainsi pour $k = 1$ on associerait à b le lycée bleu alors que pour $k = 5$ on lui associerait le lycée vert :



$k = 1$
a :
b :
c :



$k = 5$
a :
b :
c :

Avec une petite valeur de k on effectue la classification avec un petit nombre de valeurs, cela augmente l'impact d'une perturbation venant d'un point très proche. A l'inverse, avec une grande valeur de k, on tient compte d'un plus grand nombre de données, cela diminue l'influence de chaque point individuel mais on prend en compte des points qui peuvent être éloignés et donc peu significatifs.

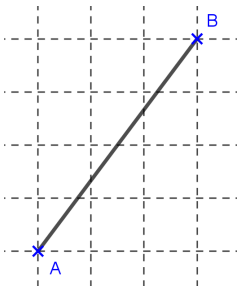
2.4 Notion de distance

L'algorithme des k plus proches voisins repose sur la notion de distance et sur la fonction *dist* évoquée plus haut. Cette fonction peut-être différente selon les objets manipulés.

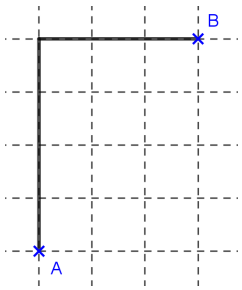
2.4.1 Distances géométriques

Lorsqu'on évalue la distance entre deux points A et B, on peut utiliser la distance naturelle que l'on pourrait mesurer avec une règle.
Si les deux points A et B ont pour coordonnées respectives $A\,(x_A\ ;\ y_A)$ et $B\,(x_B\ ;\ y_B)$, cette *distance euclidienne* est donnée par la formule : $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.

Si la distance euclidienne est la distance à "vol d'oiseau", dans la *distance Manhattan* on se déplacera comme dans les rues d'une ville.
Si les deux points A et B ont pour coordonnées respectives $A\,(x_A\ ;\ y_A)$ et $B\,(x_B\ ;\ y_B)$, cette *distance Manhattan* est donnée par la formule : $AB = |x_B - x_A| + |y_B - y_A|$.



distance ecuclidienne $AB = 5$



distance Manhattan : $AB = 7$

2.4.2 Distances entre chaines de caractères

La notion de distance entre deux chaines de carcatères peut également être évaluée de différentes manières.

La *distance de Hamming* associe à deux chaines de caractères (c'est à dire deux suites de symboles), le nombre de positions où les deux chaines diffèrent.

Chaines	Dist
ramer tarer	2
ramer rager	1
sœur frêre	5

La distance de Hammming n'est pas pas la plus pertinente si l'on veut identifier deux chaines qui sont les mêmes à une faute de frappe près :

Chaines	Dist
ramer amer	5
ramer rramer	5

Dans ce cadre la *distance d'édition*, qui donne le nombre minimum d'ajouts, de suppressions et de modifications permettant de transformer l'une en l'autre.

Chaines	Dist
rater amer	2
ramer rramer	1

2.5 Limites de l'approche

L'algorithme des k plus proches voisins, peut-être appliqué à de situations diverses (reconnaissances de chaines de caractères, reconnaissances de formes dans une image de sons etc...). D'une situation à l'autre la qualité de ses prédictions peut beaucoup varier en particulier la qualité de la base d'apprentissage est essentiel (données "correctes", suffisamment nombreuses, relativement proches de l'objet à classifier, réparties de manière égale dans les différentes classes...)

Travail à faire :
Exercices 1 à 5

3 Les iris de Fischer

Cette partie a été intégralement reprise sur le site Pixees (Ressources pour les sciences du numérique) :
`https://pixees.fr/informatiquelycee/n_site/nsi_prem_knn.html`

En 1936, Edgar Anderson a collecté des données sur 3 espèces d'iris : "iris setosa", "iris virginica" et "iris versicolor"

iris setosa

iris virginica

iris versicolor

Pour chaque iris étudié, Anderson a mesuré (en cm) :

- la largeur des sépales
- la longueur des sépales
- la largeur des pétales
- la longueur des pétales

Par souci de simplification, nous nous intéresserons uniquement à la largeur et à la longueur des pétales.
Pour chaque iris mesuré, Anderson a aussi noté l'espèce ("iris setosa", "iris virginica" ou "iris versicolor")
Vous trouverez 50 de ces mesures dans le fichier iris.csv

	A	B	C
1	petal_length	petal_width	species
2	1.4	0.2	0
3	1.4	0.2	0
4	1.3	0.2	0
5	1.5	0.2	0
6	1.4	0.2	0
7	1.7	0.4	0
8	1.4	0.3	0
9	1.5	0.2	0
10	1.4	0.2	0
11	1.5	0.1	0

En résumé, vous trouverez dans ce fichier :

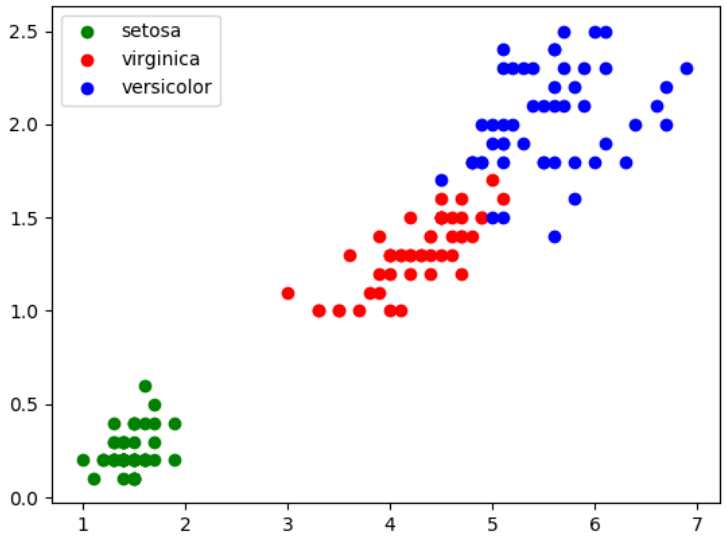
- la longueur des pétales
- la largeur des pétales
- l'espèce de l'iris (au lieu d'utiliser les noms des espèces, on utilisera des chiffres : 0 pour "iris setosa", 1 pour "iris virginica" et 2 pour "iris versicolor")

Travail à faire :
Après avoir placé le fichier iris.csv dans le même répertoire que votre fichier Python, étudiez et testez le code suivant :

```
1  import pandas
2  import matplotlib.pyplot as plt
3  iris=pandas.read_csv("iris.csv")
4  x=iris.loc[:,"petal_length"]
5  y=iris.loc[:,"petal_width"]
6  lab=iris.loc[:,"species"]
7  plt.scatter(x[lab == 0], y[lab == 0], color='g', label='setosa')
8  plt.scatter(x[lab == 1], y[lab == 1], color='r', label='virginica')
9  plt.scatter(x[lab == 2], y[lab == 2], color='b', label='versicolor')
10 plt.legend()
11 plt.show()
```

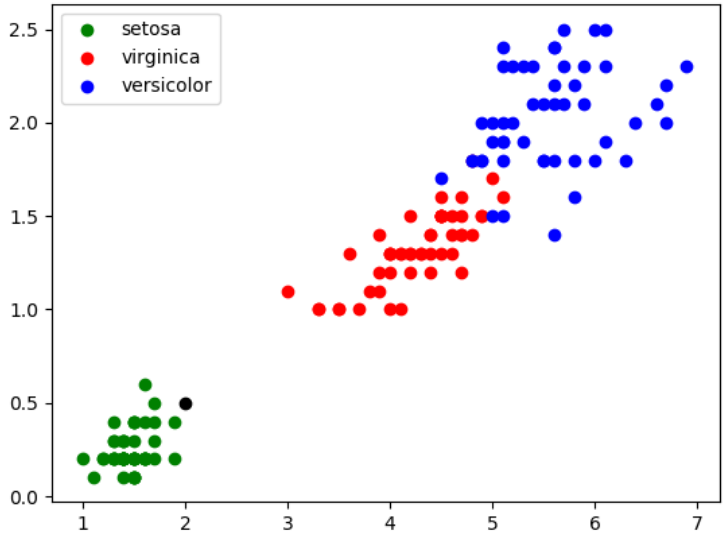
Quelques mots sur le script précédent :
Il faut installer la librairie pandas.
La partie "Pandas" ne devrait pas vous poser de problèmes : x correspond à la longueur des pétales, y correspond à la largeur des pétales et lab correspond à l'espèce d'iris (0,1 ou 2).

Nous utilisons ensuite la bibliothèque *matplotlib* qui permet de tracer des graphiques très facilement.
"*plt.scatter*" permet de tracer des points, le "*x[lab == 0]*" permet de considérer uniquement l'espèce "iris setosa" (*lab==0*).
Le premier "*plt.scatter*" permet de tracer les points correspondant à l'espèce "iris setosa", ces points seront vert (*color='g'*), le deuxième "*plt.scatter*" permet de tracer les points correspondant à l'espèce "iris virginica", ces points seront rouge (*color='r'*), enfin le troisième "*plt.scatter*" permet de tracer les points correspondant à l'espèce "iris versicolor", ces points seront bleu (*color='b'*). Nous aurons en abscisse la longueur du pétale et en ordonnée la largeur du pétale.
Vous devriez normalement obtenir ceci :



Nous obtenons des "nuages" de points, on remarque ces points sont regroupés par espèces d'iris (sauf pour "iris virginica" et "iris versicolor", les points ont un peu tendance à se mélanger).

Imaginez maintenant qu'au cours d'une promenade vous trouviez un iris, n'étant pas un spécialiste, il ne vous est pas vraiment possible de déterminer l'espèce. En revanche, vous êtes capables de mesurer la longueur et la largeur des pétales de cet iris. Partons du principe qu'un pétale fasse 0,5 cm de large et 2 cm de long. Plaçons cette nouvelle donnée sur notre graphique (il nous suffit d'ajouter la ligne "*plt.scatter(2.0, 0.5, color='k')*", le nouveau point va apparaitre en noir (*color='k'*)) :



Je pense que le résultat est sans appel : il y a de fortes chances que votre iris soit de l'espèce "iris setosa".
Il est possible de rencontrer des cas plus difficiles, par exemple : largeur du pétale = 0,75 cm ; longueur du pétale = 2,5 cm.
Dans ce genre de cas, il peut être intéressant d'utiliser l'algorithme des "k plus proches voisins".
La bibliothèque Python *Scikit Learn* propose un grand nombre d'algorithmes lié au machine learning (c'est sans aucun doute la bibliothèque la plus utilisée en machine learning). Parmi tous ces algorithmes, *Scikit Learn* propose l'algorithme des k plus proches voisins.

Travail à faire :
Après avoir placé le fichier iris.csv dans le même répertoire que votre fichier Python, étudiez et testez le code suivant :

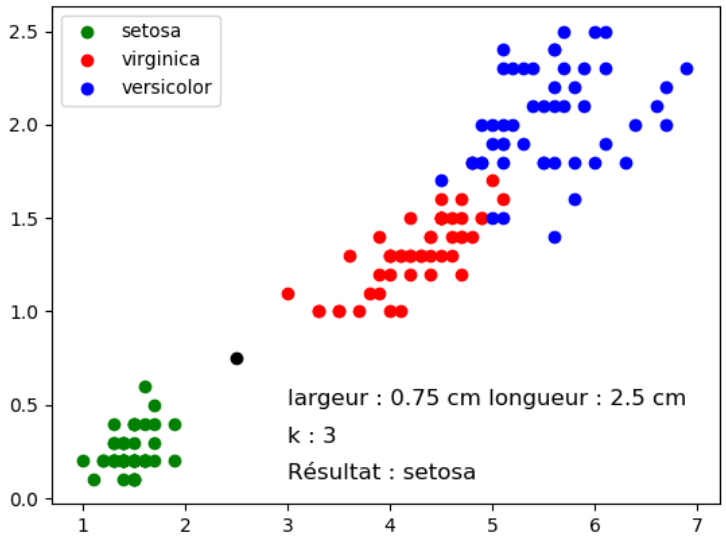
```
import pandas
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

#traitement CSV
iris=pandas.read_csv("iris.csv")
x=iris.loc[:,"petal_length"]
y=iris.loc[:,"petal_width"]
lab=iris.loc[:,"species"]
#fin traitement CSV

#valeurs
longueur=2.5
largeur=0.75
k=3
#fin valeurs

#graphique
plt.scatter(x[lab == 0], y[lab == 0], color='g', label='setosa')
plt.scatter(x[lab == 1], y[lab == 1], color='r', label='virginica')
plt.scatter(x[lab == 2], y[lab == 2], color='b', label='versicolor')
plt.scatter(longueur, largeur, color='k')
plt.legend()
#fin graphique
```

```
25
26  #algo knn
27  d=list(zip(x,y))
28  model = KNeighborsClassifier(n_neighbors=k)
29  model.fit(d,lab)
30  prediction= model.predict([[longueur,largeur]])
31  #fin algo knn
32
33  #Affichage résultats
34  txt="Résultat : "
35  if prediction[0]==0:
36      txt=txt+"setosa"
37  if prediction[0]==1:
38      txt=txt+"virginica"
39  if prediction[0]==2:
40      txt=txt+"versicolor"
41  plt.text(3,0.5, f"largeur : {largeur} cm longueur : {longueur} cm", fontsi
42  plt.text(3,0.3, f"k : {k}", fontsize=12)
43  plt.text(3,0.1, txt, fontsize=12)
44  #fin affichage résultats
45
46  plt.show()
```

Le programme ci-dessus n'est pas très complexe à comprendre, nous allons tout de même nous attarder sur la partie "knn" :
La première ligne "*d=list(zip(x,y))*" permet de passer des 2 listes x et y : $x = [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, ...]$ et $y = [0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, ...]$ à une liste de tuples d : $d = [(1.4, 0.2), (1.4, 0.2), (1.3, 0.2)(1.5, 0.2), (1.4, 0.2), (1.7, 0.2), (1.4, 0.4), ...]$
Les éléments des listes x et y ayant le même indice sont regroupés dans un tuple, nous obtenons bien une liste de tuples.

"$KNeighborsClassifier$" est une méthode issue de la bibliothèque scikit-learn (voir plus haut le "$fromsklearn.neighborsimportKN$ cette méthode prend ici en paramètre le nombre de "plus proches voisins" ($model = KNeighborsClassifier(n_neighbors = k)$)
"$model.fit(d, lab)$" permet d'associer les tuples présents dans la liste "d" avec les labels (0 : "iris setosa", 1 : "iris virginica" ou 2 : "iris versicolor"). Par exemple le premier tuple de la liste "d", (1.4, 0.2) est associé au premier label de la liste lab (0), et ainsi de suite...

La ligne "*prediction= model.predict([[longueur,largeur]])*" permet d'effectuer une prédiction pour un couple *[longueur, largeur]* (dans l'exemple ci-dessus "longueur=2.5" et "largeur=0.75"). La variable "*prediction*" contient alors le label trouvé par l'algorithme knn. Attention, "*predection*" est une liste Python qui contient un seul élément (le label), il est donc nécessaire d'écrire "*predection[0]*" afin d'obtenir le label.
Vous devriez normalement obtenir ceci :



Travail à faire :
Modifiez le programme précédent afin de tester l'algorithme knn avec un nombre "de plus proches voisins" différent (en gardant un iris ayant une longueur de pétale égale à 2,5 cm et une largeur de pétale égale à 0,75 cm). Que se passe-t-il pour k = 5 ?

Travail à faire :
Testez l'algorithme knn (toujours à l'aide du programme précédent avec un iris de votre choix (si vous ne trouvez pas d'iris, vous pouvez toujours inventer des valeurs ;-))

4 Exercices

Exercice n° 1.

Écrire une fonction calculant la distance de Hamming entre deux chaines de caractères.

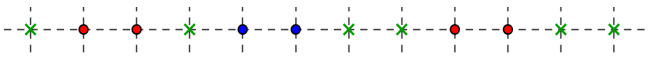
Exercice n° 2.

On se donne un tableau *lexique* contenant des mots.

1. Écrire un script python qui demande un mot m et une distance d à l'utilisateur et qui affiche l'ensemble des mots du *lexique* ayant une distance de Hamming à m inférieure ou égale à d.
2. Écrire un script python qui demande un mot m à l'utilisateur et qui affiche le mot du *lexique* ayant avec m la plus faible distance de Hamming.

Exercice n° 3.

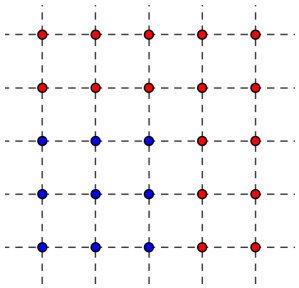
On sépare les points d'un segment en deux classes (les bleus et les rouges).
On dispose de l'échantillon suivant :



1. Indiquer pour chaque point vert si l'algorithme des k plus proches voisins, pour $k = 3$, classe le point correspondant en rouge ou en bleu.
2. En déduire les zones du segment dont les zones sont classés en bleu ou en rouge.

Exercice n° 4.

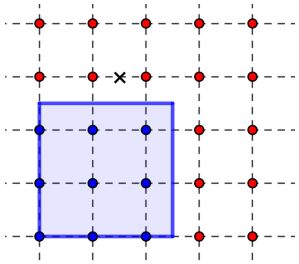
On sépare les points d'une partie du plan en 2 classes (bleu ou rouge). On dispose comme échantillon des points suivants :



1. Déterminer les régions du plan dont les points sont classés en bleu par l'algorithme des k plus proches voisins pour $k = 1$ avec la distance euclidienne.
2. Reprendre la question précédente pour $k = 3$, $k = 5$ et $k = 9$.

Exercice n° 5.

On cherche à perturber l'algorithme des k plus proches voisins en ajoutant des points bleus dans la zone bleutée.



1. Quel point bleu peut-on ajouter à l'échantillon précédent pour que le point désigné par la croix soit classé bleu par l'algorithme des k plus proches voisins avec $k = 5$ et la distance euclidienne ?
2. Supposons que l'on puisse ajouter autant de points bleus que l'on souhaite dans la zone bleutée. Si on considère à nouveau les 5 plus proches voisins, quels point rouges pourraient alors être classifiés en bleu ?