Algorithme des k plus proches voisins

Machine learning

*En anglais "**k nearest neighbors**" : **knn***

C'est un algorithme utilisé dans des structures d'apprentissage automatique, souvent appelé 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. Nourrie par un grand nombre d'exemples, elle va apprendre et devenir de plus en plus performante (L'algorithme de k plus proches voisins ne nécessite pas de phase d'apprentissage à proprement parler, il faut juste stocker le jeu de données d'apprentissage).

Machine learning et histoire :

L'idée d'apprentissage automatique (machine learning) est réellement née en 1936 avec le concept de machine universelle d'Alan Turing et de son article de 1950 sur le test de turing. En 1943 deux chercheurs représentent le fonctionnement des neurones avec des circuits électriques, c'est le début de la théorie sur les réseaux neuronaux. En 1952 Arthur Samuel écrit un programme capable de s'améliorer en jouant aux dames. Depuis les progrès ne font que continuer et le machine learning est probablement le secteur de l'informatique le plus en expansion (reconnaissance faciale, conduite automatique, création de vaccins, aide à la prise de décision (data scientist),...).

Il existe dans ce domaine, différentes approches qu'il est intéressant de connaitre:

***Apprentissage supervisé**: on a des exemples que l'on sait classer et qui sont déjà classées. L'ordinateur apprend avec les exemples et leur réponse, puis teste. Par exemple pour distinguer si l'on a une photo de fourmi ou de cassoulet, l'ordinateur va analyser des centaines de photos dont il a la réponse.*

***Apprentissage non supervisé**: l'ordinateur a des exemples mais ne connait pas la réponse. On ne lui a pas dit « voila des fourmis et voila des cassoulets ». Pas impossible du coup qu'il crée une troisième classe.*

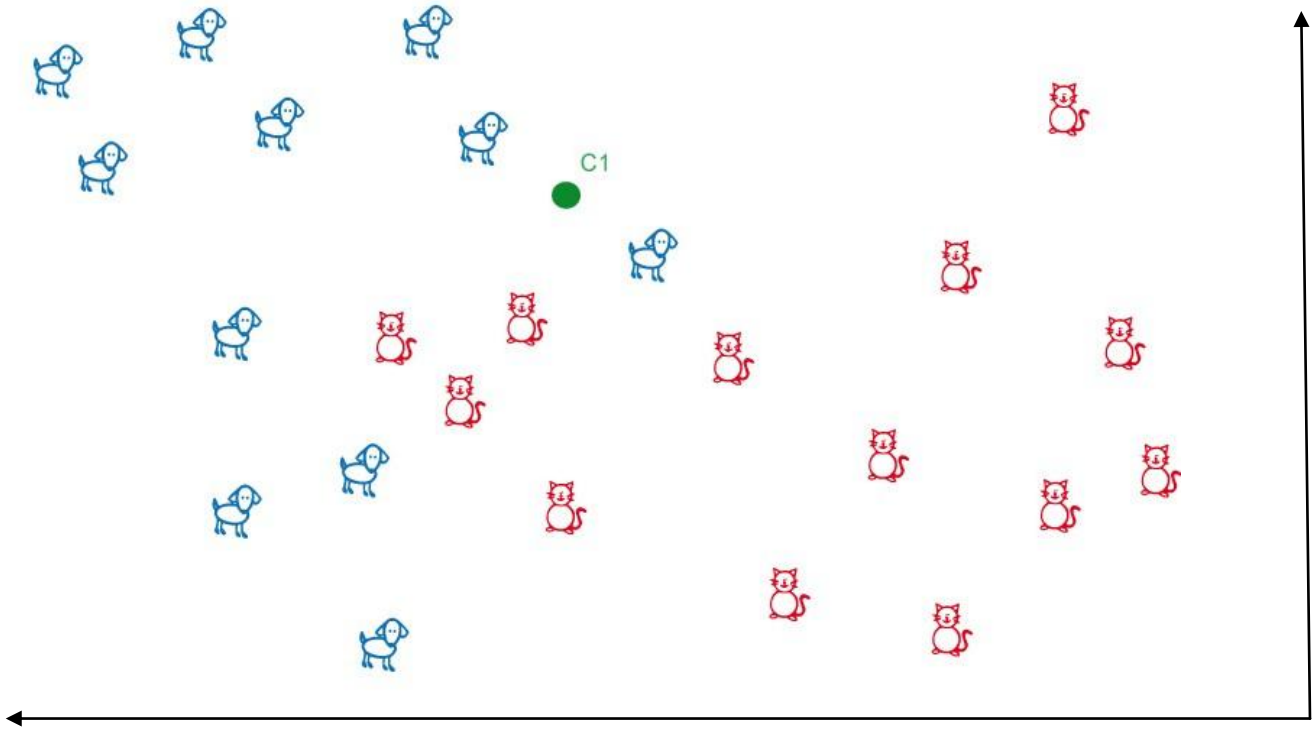
***Apprentissage par renforcememnt**: l'algorithme apprend en observant ce qu'il essaye. Un système de récompense/punition lui permet d'améliorer ses choix. L'ordinateur joue au hasard au début et plus il apprend moins il joue au hasard et utilise ce qu'il a trouvé comme 'bonne méthode'. C'est avec ce genre de méthode que l'ordinateur peut apprendre à jouer à différents jeux. Le plus difficile est de choisir ce qui va être une récompense et ce qui sera une sanction.*

***Apprentissage par transfert:** utiliser des compétences déjà apprises sur des nouvelles taches qui ont des points communs.*

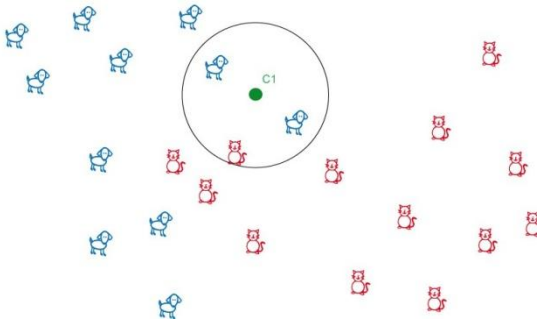
Description du problème :

Notre problème est assez simple: on relève sur des objets de différentes classes des paramètres. On sait donc que pour tel objet de telle classe, on a tels paramètres. L'objectif est de pouvoir prévoir à quelle classe appartient un nouvel objet uniquement à l'aide de ses paramètres (il s'agit clairement d'un apprentissage supervisé).

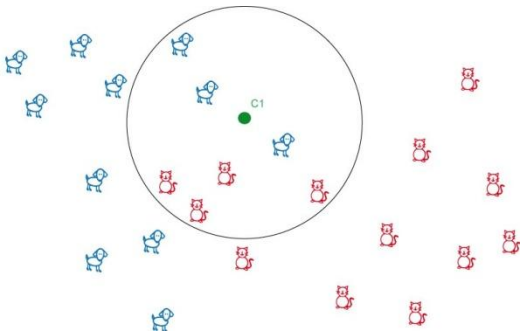
Imaginons qu'on ait quelques milliers de données disponibles sur un cheptel de chiens et de chats et que ces données relatent la taille et le poids des animaux. Ceci donnerait le graphe ci-dessous :



On donne le poids et la taille d'un nouvel animal au programme (le point vert C1) et celui-ci doit deviner si c'est un chat ou un chien. Pour ce, le programme va regarder ce qu'il y a de plus proche et va en faire sa déduction. La question a se poser est : dans quel rayon regarde-t-on ses voisins (que prendre comme k ?).



Pour k = 3, on en déduit que C1 est un chien



Pour k = 7, on en déduit que c'est un chat !

Mais, comment choisit-on ce paramètre k alors ?

- On fait varier k
- Pour chaque valeur de k, on calcule le taux d'erreur de l'ensemble de test
- On garde le paramètre k qui minimise ce taux d'erreur test.

On peut en déduire au passage que la valeur de k est essentielle, que ce type d'algorithme n'émet qu'une probabilité (mais après tout, si elle est de 99,9999%, ça peut suffire pour certaines applications), et qu'enfin le choix des paramètres est essentiel (si vous choisissez la couleur pour différencier un lapin d'un éléphant, vous risquez d'avoir quelques rongeurs qualifiés de pachyderme…)

On a illustré le principe avec deux paramètres, mais on peut utiliser cette méthode avec un seul paramètre, ou avec 3,4 ,25 paramètres (bref avec autant de paramètres qu'on veut)

TP – les Iris d'Anderson

Afin de travailler sur un exemple, nous allons utiliser un jeu de données relativement connu dans le monde du machine learning : le jeu de données "iris".

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 versicolor" ou "iris virginica")
Vous trouverez 50 de ces mesures dans le fichier ***iris.csv***(disponible sur http://ressource.elec.free.fr)i
En résumé, vous trouverez dans ce fichier :

	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

extrait du ieu de données "iris"

- 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 versicolor" et 2 pour "iris virginica")

Avant d'entrer dans le vif du sujet (algorithme knn), nous allons chercher à obtenir une représentation graphique des données contenues dans le fichier ***iris.csv***

Expérimentation :

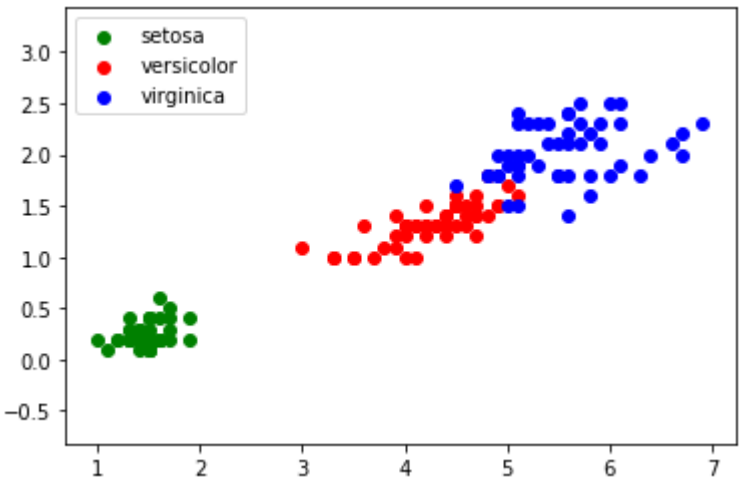
Après avoir placé le fichier ***iris.csv*** dans le même répertoire que votre fichier Python, étudiez et testez le code suivant (knn-prog1.py) :

```
import pandas
import matplotlib.pyplot as plt
iris=pandas.read_csv("iris.csv")
x=iris.loc[:,"petal_length"]
y=iris.loc[:,"petal_width"]
lab=iris.loc[:,"species"]
plt.axis('equal')
plt.scatter(x[lab == 0], y[lab == 0], color='g', label='setosa')
plt.scatter(x[lab == 1], y[lab == 1], color='r', label='versicolor')
plt.scatter(x[lab == 2], y[lab == 2], color='b', label='virginica')
plt.legend()
plt.show()
```

Quelques mots sur le programme ci-dessus :

- La partie "Pandas": 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.axis('equal')" permet d'obtenir un repère orthonormé. "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 versicolor", ces points seront rouge (color='r'), enfin le troisième "plt.scatter" permet de tracer les points correspondant à l'espèce "iris virginica", ces points seront bleu (color='b'). Nous aurons en abscisse la longueur du pétale et en ordonnée la largeur du pétale.

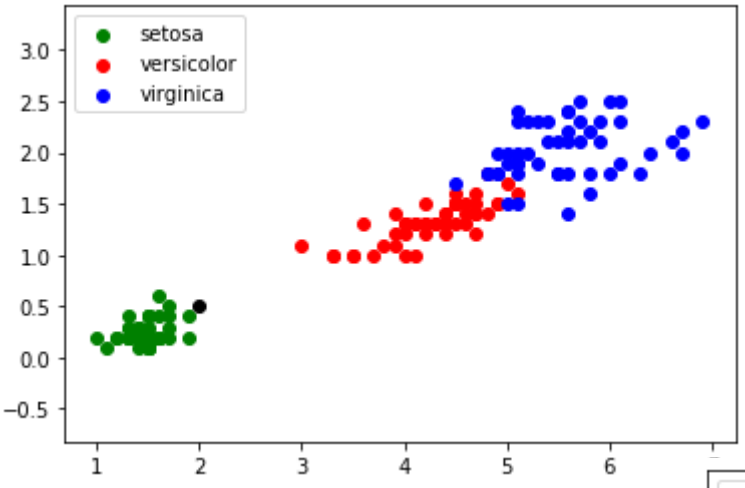
Vous devriez normalement obtenir ceci :



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

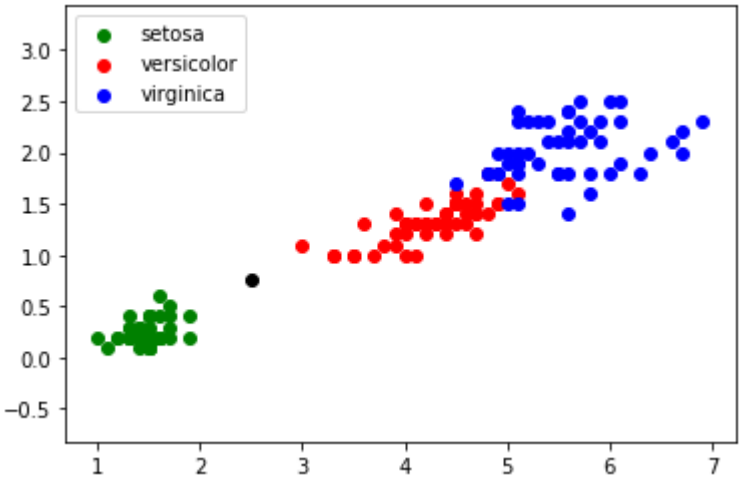
Imaginez maintenant qu'au cours d'une promenade vous trouviez un iris, n'étant pas un spécialiste, il ne vous est pas vraiment possible de déterminer l'espèce. En revanche, vous êtes capables de mesurer la longueur et la largeur des pétales de cet iris. Partons du principe qu'un pétale fasse 0,5 cm de 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')) :



Le résultat est sans appel : il y a de fortes chances que votre iris soit de l'espèce "iris setosa".

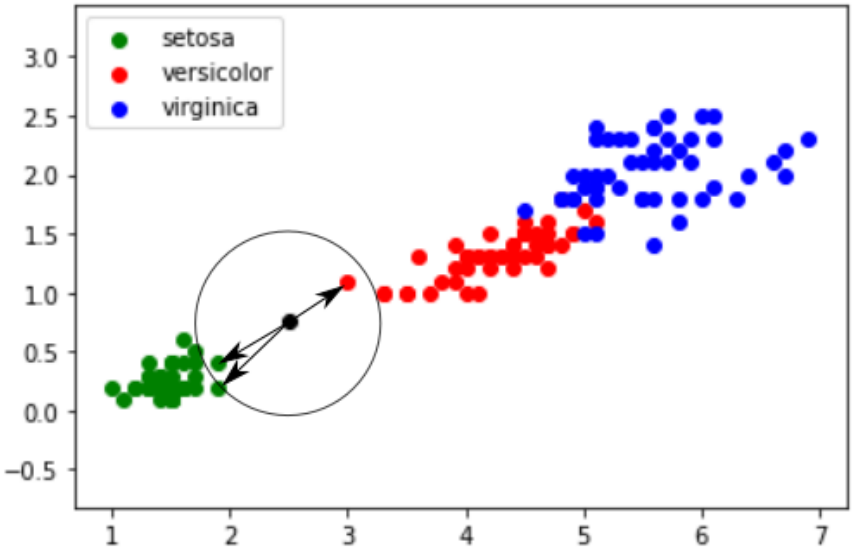
Il est possible de rencontrer des cas plus difficiles, par exemple : largeur du pétale = 0,75 cm ; longueur du pétale = 2,5 cm :



Dans ce genre de cas, il peut être intéressant d'utiliser l'algorithme des "k plus proches voisins » :

- on calcule la distance entre notre point (largeur du pétale = 0,75 cm ; longueur du pétale = 2,5 cm) et chaque point issu du jeu de données "iris" (à chaque fois c'est un calcul de distance entre 2 points tout ce qu'il y a de plus classique)
- on sélectionne uniquement les k distances les plus petites (les k plus proches voisins)
- parmi les k plus proches voisins, on détermine quelle est l'espèce majoritaire. On associe à notre "iris mystère" cette "espèce majoritaire parmi les k plus proches voisins"

Prenons k = 3 :



Les 3 plus proches voisins sont signalés ci-dessus avec des flèches : nous avons deux "iris setosa" (point vert) et un "iris versicolor" (point rouge). D'après l'algorithme des "k plus proches voisins", notre "iris mystère" appartient à l'espèce "setosa".

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.

Expérimentation : Après avoir placé le fichier ***iris.csv*** dans le même répertoire que votre fichier Python, étudiez et testez le code suivant (knn-prog2.py) :

```
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"]

#valeurs
longueur=2.5
largeur=0.75
k=3

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

#algo knn
d=list(zip(x,y))
model = KNeighborsClassifier(n_neighbors=k)
model.fit(d,lab)
prediction= model.predict([[longueur,largeur]])

#Affichage résultats
txt="Résultat : "
if prediction[0]==0:
  txt=txt+"setosa"
if prediction[0]==1:
  txt=txt+"versicolor"
if prediction[0]==2:
  txt=txt+"virginica"
plt.text(3,0.5, f"largeur : {largeur} cm longueur : {longueur} cm", fontsize=12)
plt.text(3,0.3, f"k : {k}", fontsize=12)
plt.text(3,0.1, txt, fontsize=12)

plt.show()
```

Nous allons nous attarder sur la partie "knn" :

```
d=list(zip(x,y))
model = KNeighborsClassifier(n_neighbors=k)
model.fit(d,lab)
prediction= model.predict([[longueur,largeur]])
```

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, ...]
y = [0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.4,....]
```

à une liste de tuples d :

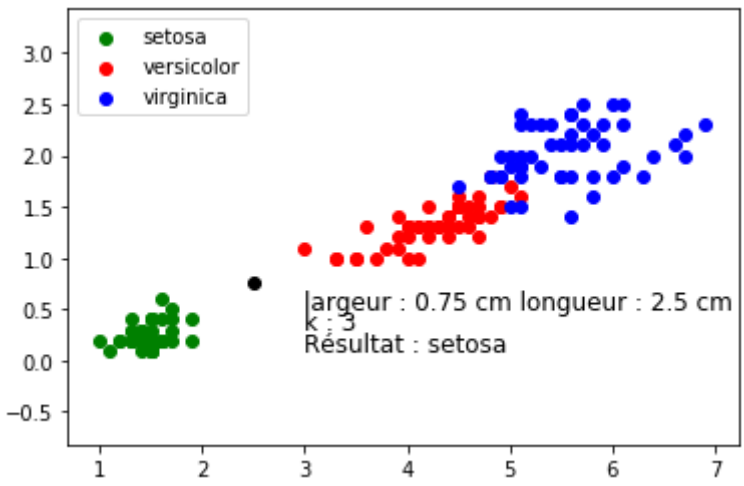
```
d = [(1.4, 0.2), (1.4, 0.2), (1.3, 0.2) (1.5, 0.2), (1.4, 0.2), (1.7, 0.2), (1.4, 0.4), ...]
```

les éléments des tableaux 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. 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 versicolor" ou 2 : "iris virginica"). 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, "prediction" 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 :

Expérimentation 2

Modifiez le programme 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 ?

Expérimentation 3 Testez l'algorithme knn avec un iris de votre choix.