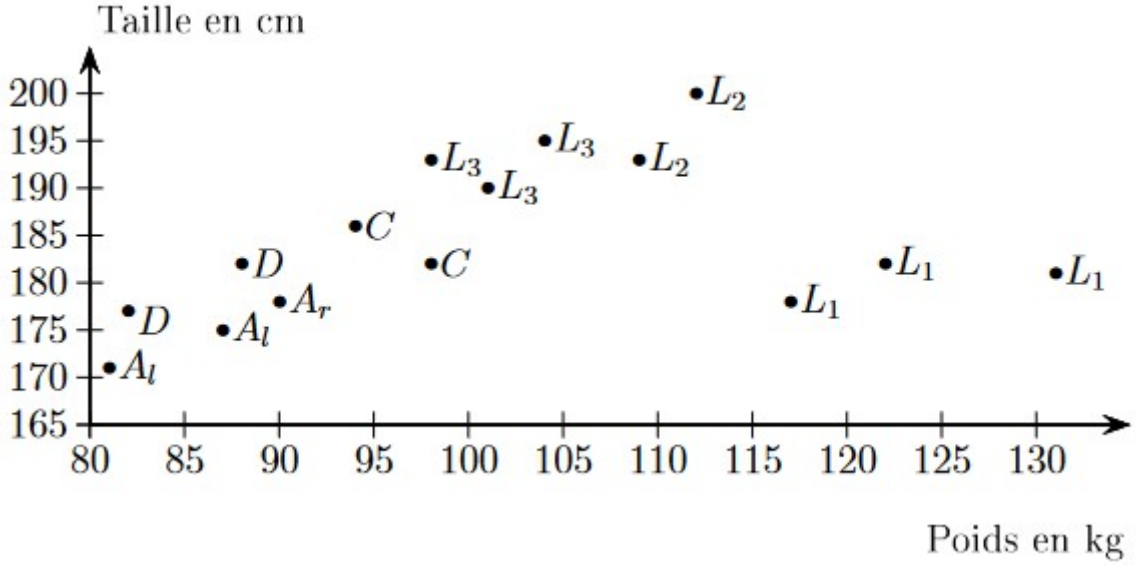
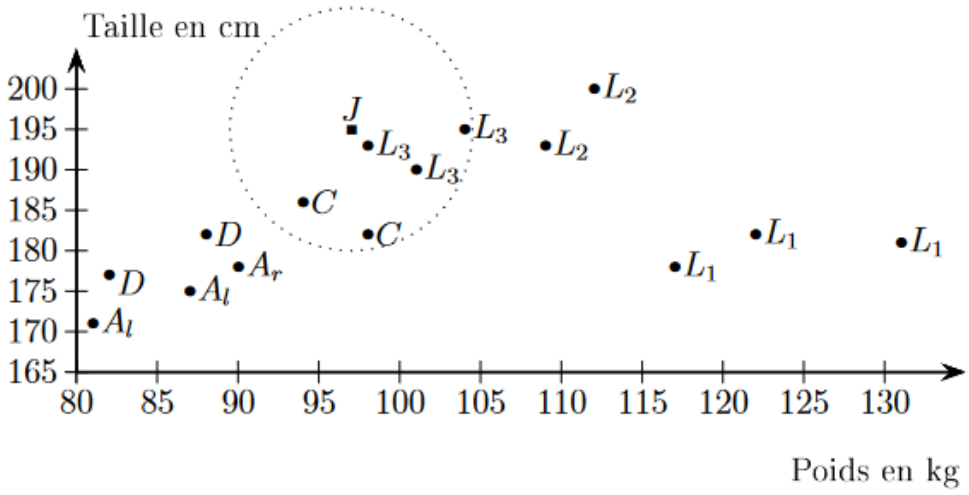
Algorithme des k plus proches voisins

1.1 Heuristique

On considère une équipe de rugby professionnelle. Celle-ci est composée de quinze joueurs habilement numéroté de un à quinze que l'on subdivise en plusieurs catégories :
1res lignes : numéros 1 à 3 ; notés L1.
2es lignes : numéros 4 et 5 ; notés L2.
3es lignes : numéros 6 à 8 ; notés L3.
Les demis de mêlée et d'ouverture : numéros 9 et 10 ; notés D.
Les centres : numéros 12 et 13 ; notés C.
Les ailiers : numéros 11 et 14 ; notés Al.
L'arrière : numéro 15 ; noté Ar.
Pour attribuer un poste à un joueur (outre la technique, la vitesse, l'intelligence de jeu...), deux critères a priori sont le poids et la taille du joueur. On considère que l'on a la répartition suivantes en fonction des postes :



On considère alors un nouveau joueur n'ayant pas encore de poste au sein de l'équipe et on cherche à lui en donner un. En première approche, on va lui affecter un poste correspondant sa taille et son poids. Le joueur mesurant 1m93 et pesant 97kg, on le positionne à l'aide du point J dans notre graphe des répartitions.



1.2 Histoire et utilisation

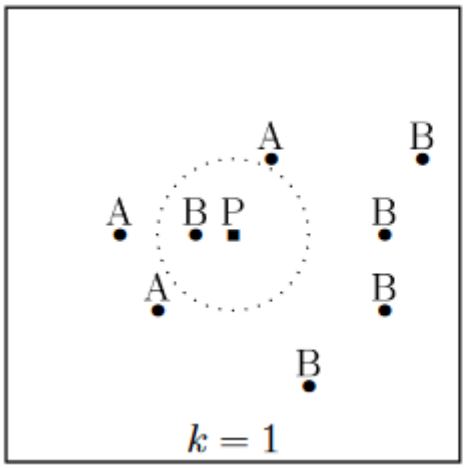
Dans un rapport de la faculté de médecine aéronautique de la US Air Force publié en 1951, Fix et Hodges introduisirent une méthode pour la classification des motifs, connue depuis sous la règle des k plus proches voisins. Il s'écrit en abrégé k-NN ou KNN, de l'anglais k-nearest neightbors. Il s'agit d'un des algorithmes de machine learning essentiel dans le milieu de l'intelligence artificielle.
Le principe peut être résumé par « Dis-moi qui sont tes amis, et je te dirai qui tu es ! ».
Ce genre d'algorithme permet par exemple de prédire le comportement d'une personne en s'intéressant à son milieu. Il peut par exemple être utilisé par des géants de la vente afin de prévoir si vous seriez ou non intéressé par un produit. En effet, en disposant de vos données et en les comparant à celle d'un client qui a acheté un produit, un algorithme peut tâcher de prédire si vous seriez intéressé ou non par le produit. Cela vaut aussi pour les opinions, les centres d'intérêts, etc qui sont ensuite exploités par certains réseaux sociaux ou services de vidéos à la demande à des fins de personnalisations de contenus.
Bien évidemment, le champ d'application des KNN ne s'arrête aux exemple décrits ci-dessus, il en existe de nombreux autres, notamment en sciences.

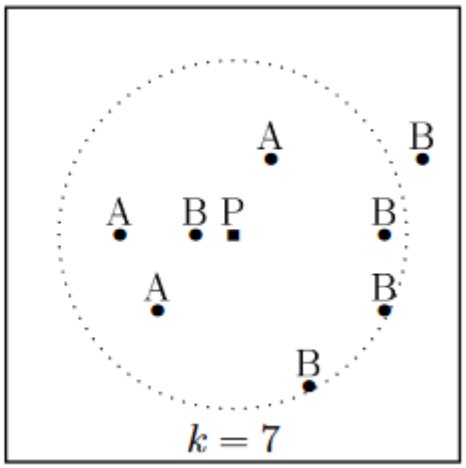
De manière générale, cet algorithme peut être utilisé selon deux objectifs :
Les régressions : on calcule la moyenne des valeurs des k plus proches voisins d'un élément et on attribue cette moyenne à l'élément étudié.
Les classifications : on cherche le résultat majoritaire des classes d'appartenance des k plus proches voisins d'un élément. On attribue la classe de l'élément suivant le résultat obtenu.
Les algorithmes à base d'apprentissage apportent une réponse plausible, mais pas nécessairement exacte, à un problème auquel il est difficile d'appliquer un algorithme traditionnel.

1.3 Paramétrage

1.3.1 Le nombre de voisins k

Le choix du nombre de voisin k n'est pas anecdotique, il fait radicalement changer les résultats que procurent cette méthode :
— Si k est trop petit, on utilise peu de valeurs donc il suffit d'un intrus (le bruit) pour modifier la classification.
— Si k est trop grand, l'influence de chaque point devient faible et on prend en compte des éléments peu significatifs car trop éloignés.





Dans les deux cas ci-dessus, le nombre de voisins considérés nous donne en réponse B alors que A semblerait être la réponse la plus naturelle. Cette méthode n'a donc rien d'absolu mais fournit de bonnes premières approches.

1.3.2 Distances

Les distances sont des fonctions mathématiques prenant en argument deux objets et donnant en sortie un réel représentant le degré de similarité entre eux. Il existe de nombreuses distances et le choix de celle-ci n'est pas anodin. Certaines prennent en compte tous les paramètres équitablement, d'autres donnent plus de poids à certaines données. En voici quelques exemples classiques.

Distance euclidienne : en dimension deux avec A (xA ; yA) et B (xB ; yB), on a :

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

Distance de Manhattan : en dimension deux :

$$d(A,B) = |x_A - x_B| + |y_A - y_B|,$$

Aussi appelée taxi-distance, c'est la distance entre deux points parcourue par un taxi lorsqu'il se déplace dans une ville où les rues sont agencées selon un réseau ou quadrillage comme à Manhattan.

Distance de Tchebychev : en dimension deux :

$$d(A,B) = \max(|x_A - x_B|, |y_A - y_B|),$$

C'est la distance entre deux points donnée par la différence maximale entre leurs coordonnées sur une dimension.

Distance de Hamming : notion de distance entre deux chaînes de caractères

Exercice

[Des mots] On se propose d'utiliser l'algorithme du plus proche voisin pour corriger un mot faux donné par un utilisateur. La distance entre deux mots, la distance de Hamming, consiste à comptabiliser le nombre de différences de caractères entre les deux mots :

— d(and,end)=1 car ces deux mots font la même longueur et ils diffèrent en un caractère.

— d(assert,finally)=7 car le plus long fait 7 caractères et ils diffèrent pour tous les caractères.

On donne ci-dessous la liste des 29 mots réservés de Python :

and assert break class continue def del elif else except
exec finally for form global if import in is lambda
not or pass print raise return try while yield

1. Calculs de distances

(a) Calculer la distance d(assert,except).

(b) Citer les mots de la liste à distance 1 de « if ».

2. Recherche du plus proche voisin

(a) Quel est le plus proche voisin du mot « ou » ? Justifier la réponse.

(b) Existe-t-il des mots qui ont plusieurs « plus proche voisin » ?

(c) Quel est le plus proche voisin de « form » ? Qu'en penser ?

Illustration des k plus proches voisins : Crocodiles vs Alligators ?
Cet exemple complet vous présente l'implémentation des k plus proches voisins pour classifier des animaux.
On utilise seulement deux critères très simples :

- le longueur de l'animal
- la taille de sa gueule

L'exemple est volontairement limité à deux caractéristiques afin de pouvoir les présenter graphiquement dans un *plan*. Au delà de deux caractéristiques, il faut utiliser plusieurs graphiques et c'est nettement moins lisible.
Sur une figure, c'est simple, on voit nettement la démarcation.
Avec les données chiffrées aussi.
Nous devrions donc parvenir à distinguer facilement et repérer nos erreurs.

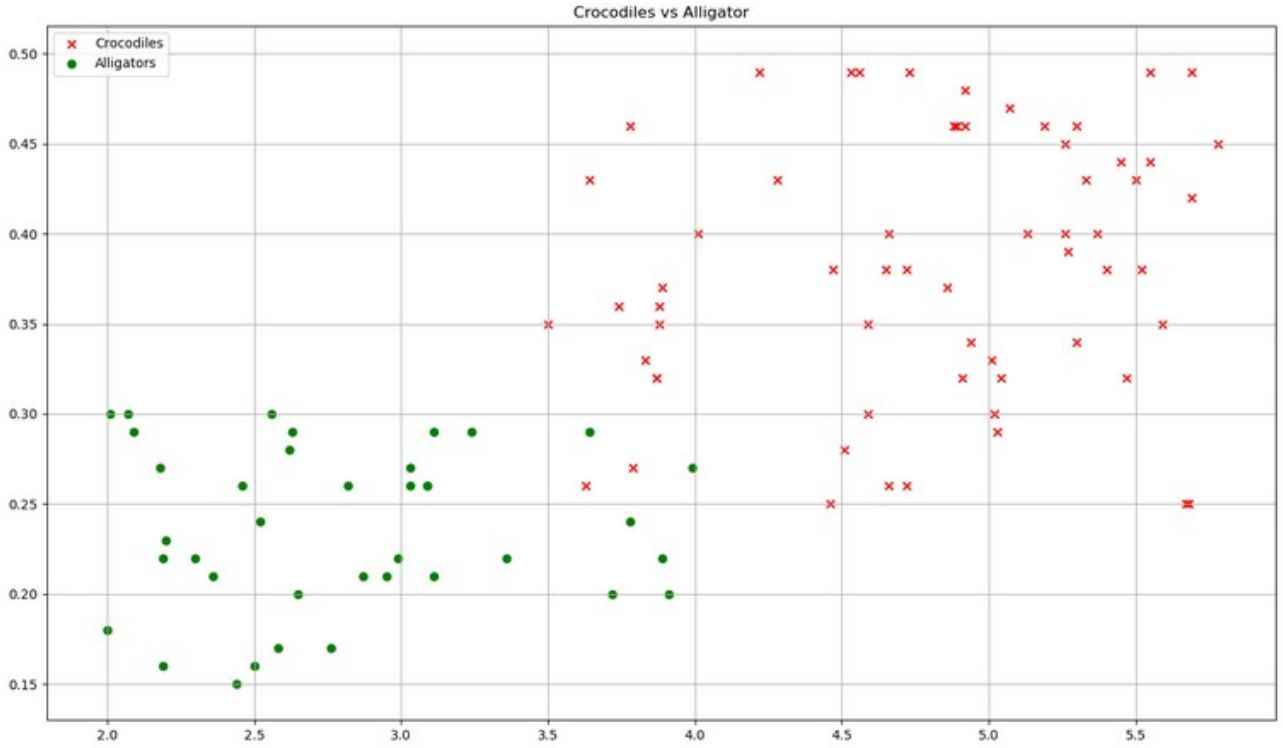
Données
Toutes les données utilisées ici sont fictives. Je vous joint d'ailleurs le programme qui me sert à les générer.
Il illustre le cours "donnée en table : création d'une table".

Examen des données

crocos.csv

```
taille,gueule,espece
3.78,0.46,crocodile
2.18,0.27,alligator
5.04,0.32,crocodile
2.2,0.23,alligator
3.24,0.29,alligator
5.55,0.49,crocodile
5.4,0.38,crocodile
3.03,0.27,alligator
...
```

Ce fichier (généré aléatoirement par generer_donnees.py) contient une table csv de données concernant les reptiles.Les champs son t donc taille, gueule et espece. Une fois dessiné, on obtient ce graphique :

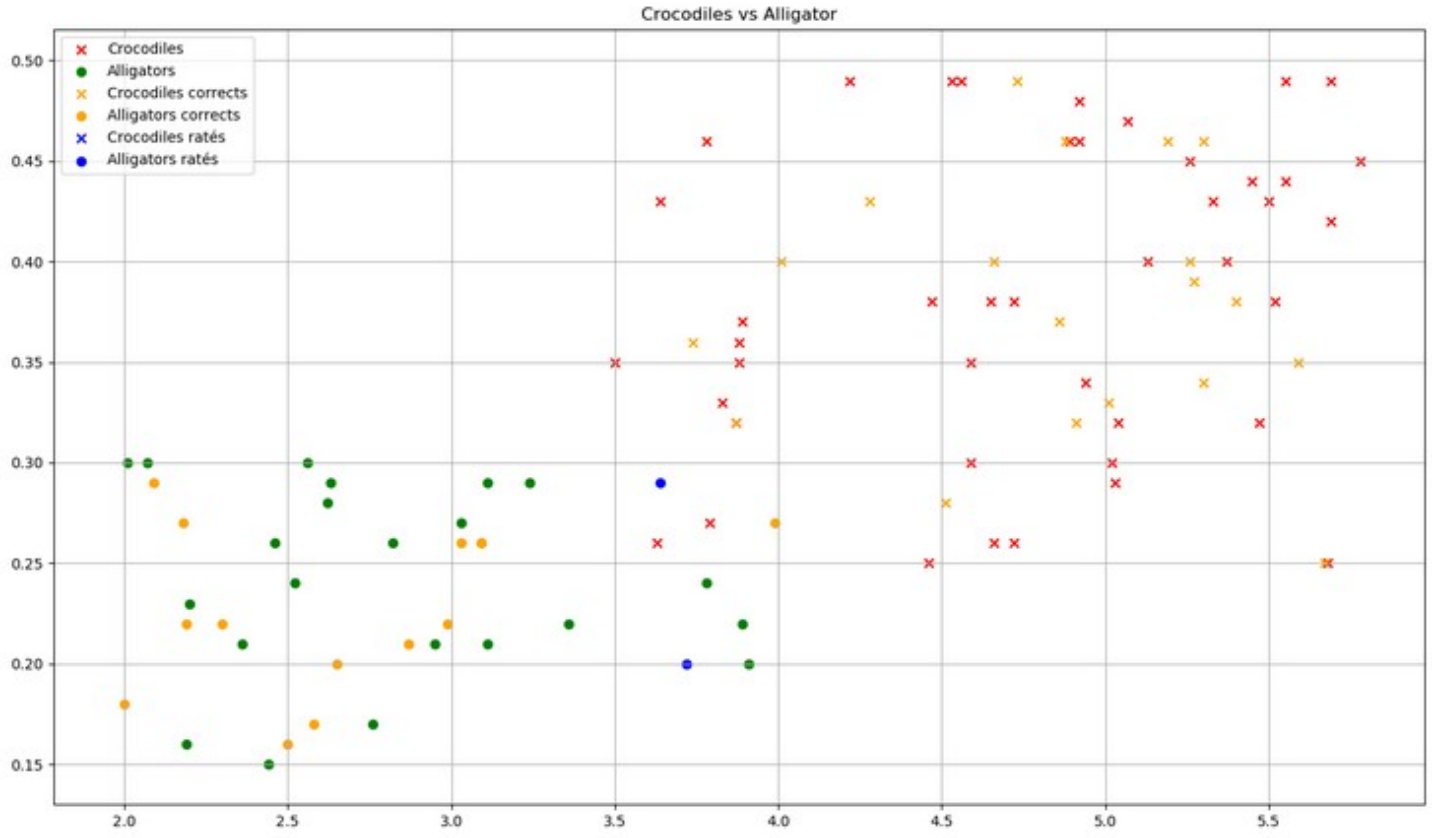


Ainsi qu'on peut le voir, il y a un léger chevauchement des données.

Utiliser le programme

Dans l'ordre, voici les étapes à exécuter :

1. récupérer tous le fichier crocos.zip et tout extraire dans un même dossier (depuis github, cliquer dessus, download)
2. (optionnel) exécuter le fichier generer_donnees
   Il va modifier le contenu du fichier csv.
3. exécuter le fichier presenter_donnees
   Ce fichier va vous dessiner le graphique présentant les données. Fermer la fenêtre ou arrêter le script.
4. exécuter le fichier knn
   Ce script applique l'algorithme des k plus proches voisins avec k=3 au jeu de données précédent.
   Une fois les calculs effectués, il affiche dans la console la précision (entre 0 et 1).
   Espérons qu'elle soit proche de 1 !
   Ensuite il dessine les résultats. Vous devriez obtenir un graphique similaire à :



4. On y distingue :
   - en rouge et vert les données ayant servi à l'entraînement,
   - en orange, les données testées qui sont correctes (rond alligators, croix crocodiles)
   - en bleu, les données testées qui ont échoué (mauvaises prédictions par kNN)

   Comme on pouvait s'y attendre, les reptiles mal classifiés sont ceux qui sont au centre de la figure, là où les zones "rouges" et "vertes" se rencontrent.
   Attention, il y a plusieurs étapes aléatoires (génération des données, séparation des jeux de tests et d'entraînement) qui font que vous pouvez parfaitement n'avoir aucun reptile mal classifié. Vous ne verriez alors aucun point bleu.
   Recommencez la simulation pour changer vos données.

Remarque : le dernier fichier presenter_resultats n'a pas grand intérêt. C'est un copier coller sauvage pour créer rapidement la figure précédente.

Consignes

- Lire soigneusement tout le code (sauf presenter_resultats)

Remarquez que j'utilise surtout les outils de base et qu'on passe en revue presque toute la partie programmation (liste, dictionnaire, fonctions, charger un csv, écrire dans un csv etc.)

- Compléter la documentation des fonction dont la documentation ne contient rien.

Il y en a quatre.

- Créer un script python classifier_nouveaux.py

Importer le script knn.py et créer une fonction classifier
Elle prend en paramètre un reptile sous la forme :

```
reptile = {
  "largeur": 5.2,
  "gueule": 0.7
}
```

- Les données numériques sont libres mais doivent rester aux alentours des celles du jeu initial.

Votre fonction renvoie la classe du reptile en question.

- Améliorer votre fonction pour qu'elle ajoute une clé "espece"

dont la valeur est le vote obtenu.

- Copier ce qui est fait dans charger_donnees.py générer des reptiles dont on connait les dimensions mais pas l'espèce.

Englober ce travail dans une fonction creer_nouveaux(nombre)
dont le paramètre nombre est... le nombre de reptiles à créer.
Retourner une liste de nouveaux reptiles.

- Présenter vos données.

On doit voir, la figure avec les reptiles du jeu de données et les nouveaux en bleu.
Inspirez-vous de ce qui est fait dans presenter_donnees.py