Chapitre 6

Algorithmique avancée

6.1 Algorithme des K plus proches voisins

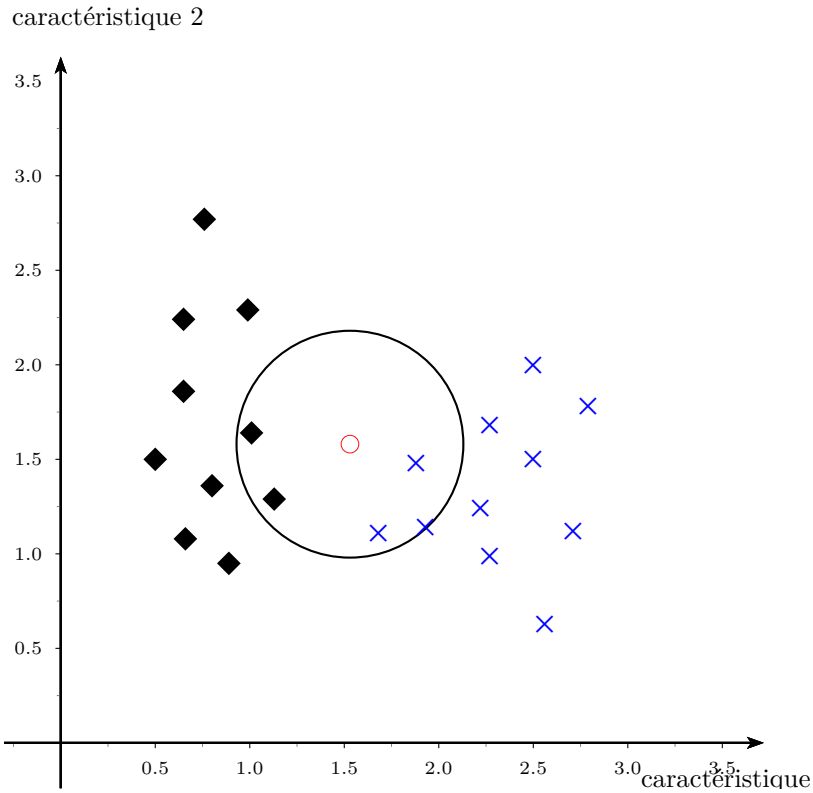
L'algorithme KNN (K Nearest Neighbors en anglais) est un algorithme essentiel en apprenstissage automatique ou «Machine Learning ».

a. Application 1 : classification

Comme son nom l'indique, cet algorithme permet de déterminer les K plus proches voisins pour ensuite, par exemple, classifier une donnée dans une catégorie ou une autre.

Considérons une image d'un animal on souhaite déterminer algorithmiquement si c'est un chat ou un chien.
Pour cela on dispose d'un grand nombre de données sur les chats et les chiens. On compare l'image de l'animal aux données de chats et de chiens. On cherche parmi les échantillons les k plus proches voisins de l'image d'animal. Si dans les k plus proches voisins, il y a plus de chats alors on affirme que c'est un chat sinon on affirme que c'est un chien.

Graphiquement, on peut illustrer le principe comme suit. On a deux classes d'objets les losanges et les croix pour lesquels on possède des données chiffrées de deux caractéristiques.



On cherche à déterminer à quelle classe d'objet appartient le nouvel objet dont on connaît les deux caractéristiques. On choisit arbitrairement $K = 5$ on cherche donc ses cinq plus proches voisins. Parmi les cinq plus proches voisins trois sont de classe croix. On classe donc le nouvel objet comme croix.

b. Application 2 : algorithme de suggestion

L'algorithme de suggestion consiste à suggérer à un utilisateur qui aurait sélectionné ou acheté un item d'autres articles similaires.
La technique est simple : il suffit d'avoir des statistiques sur un grand nombre d'utilisateurs et de
chercher parmi les statistiques les k plus proches voisins de l'utilisateur puis proposer le comportement majoritaire chez les k plus proches voisins.

c. Implémentation :

• On aura besoin d'écrire une fonction qui retourne la distance euclidienne entre deux points
`distance_euclidienne(point1, point2)` avec `point1` et `point2` deux listes de nombres de même dimension n
Rappel la formule mathématique pour calculer la distance entre deux points dans un repère orthornormé est
$d = \sqrt{(x'_1 - x_1)^2 + (x'_2 - x_2)^2 + (x'_3 - x_3)^2 + \ldots + (x'_n - x_n)^2}$

```
def distance_euclidienne(point1, point2):
    distance_carre = 0
    for i in range(len(point1)):
        distance_carre .....................
    return math.sqrt(distance_carre)
```

• On aura besoin d'une fonction qui parmi des `donnees` retourne les `k` plus proche voisins de `point`.
`knn(donnees, point, k)`. Proposez un algorithme pour cette fonction `knn`.
• Test et applications :
Ci-dessous on donne un tableau de statisiques : première colonne taille en inches et seconde colonne poids en pounds.
Déterminer les 3 plus proches voisins de `point`.

```
data = [
        [65.75, 112.99],
        [71.52, 136.49],
        [69.40, 153.03],
        [68.22, 142.34],
        [67.79, 144.30],
        [68.70, 123.30],
        [69.80, 141.49],
        [70.01, 136.46],
        [67.90, 112.37],
        [66.49, 127.45],
       ]

point = [70, 140]
```

d. Mini-Projet

Les élèves de seconde se demandent souvent quels enseignements de spécialités ils devraient choisir. On dispose des moyennes de seconde de 70 élèves de premières générales ainsi que leurs spécialités de première `spe_premiere.csv`.
Écrire un programme qui permette à un élève de seconde de saisir ses moyennes de secondes et lui proposera les choix les plus fréquemment pris par ses $k = 7$ plus proches voisins.

6.2 Programmation gloutonne

a. Introduction

Explications et commentaires en vidéo partie 1/3 : `https://www.youtube.com/watch?v=icdQJ8uV-_M`
L'optimisation est un domaine très vaste et essentielle dans de nombreux secteurs.
En informatique on cherche à trouver une solution optimale, ou approchée, parmi un ensemble de solutions possibles dépendantes de paramètres d'entrée d'où sa dimension combinatoire.

Définition 1
Soit n un entier naturel et E un ensemble de n élément distinct.
Pour tout un entier naturel k inférieur ou égale à n une combinaison de k éléments parmi n est un sous ensemble de E de k éléments distincts.

➤ **Exemple 1 :**
Considérons les lettres du mots `TRUC`, l'ensemble `E = {'T', 'R', 'U', 'C'}`. Les combinaisons de 3 lettres de E sont :

- `{'T', 'R', 'U'}`
- `{'T', 'R', 'C'}`
- `{'T', 'U', 'C'}`
- `{'R', 'U', 'C'}`

En Python, la bibliothèque `itertools` contient un grand nombre de fonctions utiles en combinatoire. La fonction `combinaitons` par exemple permet d'obtenir les combinaisons d'une liste.
Testez cet exemple dans une console.

```
>>> from  itertools import combinations
>>> data  = [ 'T', 'R', 'U', 'C']
>>> for comb in combinations(data, 3) :
        comb
```

il doit donner les combinaisons de trois lettres de la liste `data`

Le problème auquel nous souhaitons apporter une réponse est le suivant. Déterminer parmi toutes les combinaisons d'un ensemble celles qui sont optimisées selon un ou plusieurs paramètres donnés.

b. Présentation d'un problème

Un photographe de presse doit déposer des photos sur un serveur, l'espace qui lui accordé est très limité 15 Mo. Il estime le prix qu'il pourra vendre ses photos dans le tableau ci-dessous.

Type	photo chat	photo paysag	photo sport	photo manif	photo ville	photo astronomie
Poids	4.5 Mo	7.5 Mo	2.5 Mo	10 M0	8 Mo	12 Mo
Prix	8 €	3 €	7 €	5€	12 €	10 €

Question quelles photos doit-il déposer pour que cela lui rapporte le plus ?
Une première démarche pour répondre à cette question est ce qu'on appelle **la force brute**. L'algorithme serait le suivant :

- Trouver toutes les combinaisons de 1, 2, 3, 4, 5 et 6 photos.
- Calculer le poids et le taille de chaque combinaison.
- Garder uniquement celles qui ont un poids inférieur ou égale à 15.
- Retourner celle qui rapportte le plus (en tout cas au moins une).

Dans la suite la structure de données choisie sera une liste de tuples. Ou chaque tuple est constitué respectivement du poids, du prix et de l'intitulé. Comme indiqué ci-dessous :

```
photos = [(4.5, 8, 'chat'), (7.5 , 3, 'paysage'), (2.5, 7, 'sport'),
          (10, 5, 'manif'), (8, 12, 'ville'), (12, 10, 'astronomie')]
```

▶ Exercice 1

1. Écrire une fonction `gain` qui prend en argument une liste, du même type que la liste `photos` ci-dessus, et qui retourne le gain en euros du photographe pour cette liste.

```
>>> exemple =  [(4.5, 8, 'chat'), (7.5 , 3, 'paysage'), (2.5, 7, 'sport')]
>>> gain(exemple)
18
```

2. Même question pour une fonction `poids` qui prend une liste, du même type que la liste `photos` ci-dessus, et qui retourne le poids en Mo de ses élèments.

```
>>> poids(exemple)
14.5
```

3. Rédiger une fonction qui parmi toutes les combinaisons de photos possibles pesant moins de 15 Mo retourne celle qui rapportera le plus au photographe.

Indication utilisez `combinations` de la bibliothèque `itertools`.

Cherchez l'exercice puis consultez les explications et corrections en vidéo `https://www.youtube.com/watch?v=qAhJm3qwsPs`

La méthode force brute n'est pas un choix pérenne elle demande d'examiner toutes les combinaisons et le nombre de combinaisons possibles devient rapidement très grand.
La méthode dite gloutonne consiste à faire à chaque étape un choix optimal localement en espérant qu'au final cela donne une solution globalement optimale .
Il faut savoir que ce n'est pas toujours le cas.

c. Algorithme glouton

Appliquons la méthode glouton, à chaque étape on va tenter de choisir la photo qui rapporte le plus si c'est possible.

- Choix glouton N°1 : la photo de ville raporte 12 € et pèse 8 Mo
- Choix glouton N°2 : la photo d'astronomie rapporte 10 € mais elle pèse 12 Mo c'est trop $8+12>15$. On se rabat sur la photo de chat 8 € et pèse 4.5 Mo
- Choix gourmand N°3 : la photo de sport rapporte 7 € et pèse 2.5 Mo, on a $8+4.5+2.5=15$ Mo

Fin de l'algorithme on a choisi les photos de ville, astronomie, sport pour $12+8+7=27$ € et 15 Mo.

▶ Exercice 2 À votre tour

À votre tour, écrire un programme glouton permettant de déterminer un solution optimale au problème du photographe. On réutilisera les fonctions `poids` et `gain` de l'exercice 1 pour afficher les résultats de manière plus parlante.