Chap 18. Algorithmes – 2ème partie

Livre p 153 – Chap 11 Recherche dichotomique dans un tableau trié
Livre p 161 – Chap 12 Algorithmes gloutons
Livre p 175 – Chap 13 Apprentissage et algorithme des plus proches voisins

1. Recherche dichotomique

a. Introduction

Lorsque l'on recherche une valeur dans une liste, le plus simple est de parcourir toute la liste en regardant chaque élément l'un après l'autre pour le comparer à la valeur cherchée… Ce n'est pas très rapide, existe-t-il une méthode plus efficace ? Oui, à condition que la liste ait été trié précédemment !

b. La dichotomie

La recherche dichotomique (*binary search* en anglais) s'effectue sur une liste triée dans l'ordre croissant, il repose sur la méthode « diviser pour régner » qui sera plus approfondie en Terminale.
On compare la valeur cherchée avec la valeur de l'élément central situé au milieu de la liste :

- Soit la valeur cherchée est égale à l'élément central : l'algorithme s'arrête !
- Soit la valeur cherchée est plus petite que l'élément central : on continue alors la recherche sur la moitié de la liste située à gauche, qui est alors deux fois plus petite.
- Soit la valeur cherchée est plus grande que l'élément central : on continue alors la recherche sur la moitié de la liste située à droite, qui est alors deux fois plus petite.

Remarque : Si, à un moment donné, on se retrouve à chercher la valeur dans une liste vide, c'est que la valeur cherchée n'était pas présente dans la liste initiale !
Exemples classiques : Le jeu du « juste prix », la recherche d'un mot dans un dictionnaire, …
Attention : Cette méthode ne fonctionne que si la liste de départ est triée !

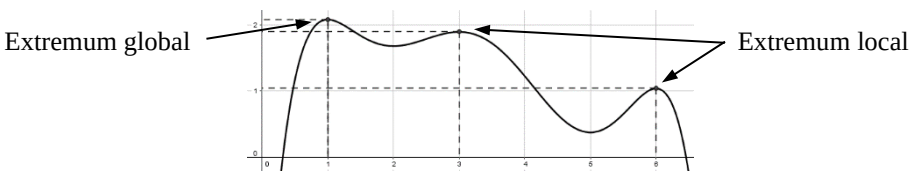
2. Algorithmes gloutons

a. Problème

Considérons un problème possédant de nombreuses solutions où chaque solution peut être évaluée par un critère numérique (un nombre de valeur, un coût, une distance, une durée, …)
Le but est de déterminer, parmi toutes les solutions possibles, quelle est celle qui est optimale pour le critère numérique choisi (le moins de valeurs, le moindre coût, la plus petite distance, le plus court, …)

b. Résolution

Pour certains problèmes, le nombre de solutions est trop grand pour les examiner toutes dans un temps raisonnable et il n'existe pas d'algorithmes simples et efficaces pour obtenir la solution optimale par une autre voie (comme la dichotomie vue précédemment) …
On peut alors rechercher une solution « convenable » mais pas nécessairement « optimale » au problème en utilisant un algorithme glouton (*greedy algorithm* en anglais) : On va utiliser une stratégie permettant d'optimiser localement la solution en avançant pas à pas sans avoir à parcourir toutes les solutions du problème et sans avoir à revenir en arrière.
Attention : Avec un algorithme glouton, on obtient une solution au problème qui est **un extremum local** mais pas nécessairement **l'extremum global** désiré !



3. Algorithme des k plus proches voisins

a. Introduction

En intelligence artificiel, on a de plus en plus recours à ce que l'on appelle l'apprentissage automatique (*machine learning* en anglais) en utilisant une grande quantité de données commentées. Ces données sont alors traitées par la machine afin d'établir des règles qui vont permettre de s'adapter à une situation nouvelle.

b. Données

On utilise un grand jeu de données comprenant les caractéristiques de nombreux individus.
Chaque individu A appartient à ce qu'on appelle une classe c(A), et se caractérise par un ou plusieurs paramètres (x_A ; y_A ; …) que l'on nomme vecteur.

c. Distance

On a besoin d'une distance pour comparer les vecteurs entre eux, souvent les paramètres sont numériques,

- S'il n'y a qu'un paramètre, on utilise habituellement la valeur absolue :
  Si on a : A(x_A) et B(x_B), alors AB = $|x_B - x_A|$
- S'il y a deux paramètres, on utilise habituellement la distance euclidienne :
  Si on a : A(x_A ; y_A) et B(x_B ; y_B), alors : AB = $\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
- On peut rencontrer plus de 2 paramètres, on utilise souvent la distance euclidienne…

Remarque : On peut aussi rencontrer des paramètres non numériques comme des mots, des couleurs, …

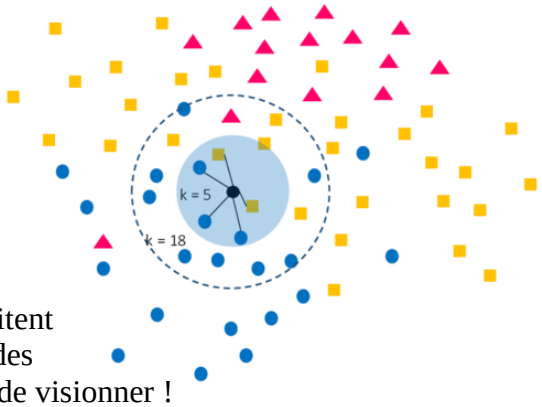
d. Problème

Supposons qu'on ait un grand jeu de données, on est face à un nouvel individu M, dont on connait ses caractéristiques sous la forme d'un vecteur (x_M ; y_M ; …) mais dont on ignore la classe c(M).
Le but est d'essayer de « deviner » la classe du nouvel individu M, en comparant ses caractéristiques avec les caractéristiques des individus contenus dans le jeu de données connues.

e. Résolution

L'algorithme des k plus proches voisins (*k nearest neighborgs* en anglais, ou *k-NN*) est l'un des algorithmes utilisés en apprentissage supervisé.
Pour déterminer la classe d'un nouvel individu M,

- On choisit une distance pour comparer deux vecteurs.
- On choisit une valeur de *k*.
- On cherche les *k* plus proches voisins de M.
- On compte le nombre de ces voisins proches appartenant à chaque classe.
- On attribue à M la classe majoritaire.



Attention : C'est pour cela que les géants du numériques exploitent un maximum de données sur vous pour ensuite vous proposer des publicités ciblées, ou pour deviner quel film vous auriez envie de visionner !
Remarque : Le choix de la valeur de *k* se fait généralement par essais successifs sur un jeu de valeurs tests, en essayant de minimiser le nombre d'erreurs commises.

f. Bibliothèque Python

Il existe une bibliothèque nommée *scikit-learn* qui contient la fonction *KNeighborsClassifier*() parmi d'autres fonctions associées à l'apprentissage automatique…