Table des matières

Première partie

Leçons

Leçon 1

Exemples d'algorithmes utilisant la méthode ≪ diviser pour régner ≫

Auteur·e·s: Emile Martinez
Références :

1 Introduction

Définition 1.1 (Paradigme Diviser pour Régner) *Un algorithme de type Diviser pour Régner s'effectue en 3 étapes :*

1. *Division du problème en sous-problèmes indépendants*
2. *Résolution récursive des sous-problèmes*
3. *Construction d'une solution du problème global à partir des solutions des sous-problèmes*

Principe 1.1 (Calcul de la compléxité d'un tel algorithme.) *Trouver une fonction donnant la taille du problème (comme le nombre d'éléments pour une liste). Définir C la complexité maximale des instances de cette taille, puis trouver une relation de réccurence sur C, pour tenter de la résoudre.*

Commentaire 1.1 *La méthode de résolution sera inculqué par l'exemple, et progressivement (d'abord le cas de base, où l'on découpe en seulement 2 avec l'exponentiation rapide, puis on complexifie)*

2 Applications au calcul formel

I L'exponentiation rapide

Problème : Étant donné un entier a et un entier positif n, calculer a^n.
Solution naïve : n multiplications

Algorithme 1.1 (Méthode D&R)

Algorithme 1.1 : $exponentiation\_rapide(a, n)$

si $n = 0$ **alors**
  retourner 1
$m \leftarrow m/2$; // étape 1
$x \leftarrow exponentiation\_rapide(a, m)$; // étape 2
si n est pair **alors** ; // étape 3
  retourner $x \times x$
sinon
  retourner $x \times x \times a$

Proposition 1.1 (Complexité) *La complexité en nombre de multiplication par rapport à l'entier positif (noté $C(n)$) vaut $C(n) = O(\log(n))$*

Démonstration.

1. $C(n) = C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(1)$
2. C est majoré par $D(n) = D\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + K$
3. D est croissante
4. $D\left(2^k\right)$ est une suite arithmétique donc $D\left(2^k\right) = O(k)$
5. $C(n) \underset{\uparrow 2}{\leq} D(n) \underset{\uparrow 3}{\leq} D\left(2^{\lfloor \log(n) \rfloor + 1}\right) \underset{\uparrow 4}{=} O(\log(n))$

□

II Multiplication matricielle

Problème : Étant donné $A = (a_{i,j})$ et B $= (b_{i,j})$ deux matrices de taille n, on cherche à calculer $A_\times B$
Solution naïve : $O(n^3)$

Algorithme 1.2 (Méthode D&R : Algorithme de Strassen) *1. Rajouter des 0 pour que A et B soient de tailles paires. Diviser alors A et B en matrices de taille $\frac{n}{2}$*

$$A = \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right) \text{ et } B = \left(\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}\right)$$

2. Calculer récursivement

$M_1 = (A_{1,1} + A_{2,2}) \times (B_{1,1} + B_{2,2})$
$M_2 = (A_{2,1} + A_{2,2}) \times B_{1,1}$
$M_3 = A_{1,1} \times (B_{1,2} - B_{2,1})$
$M_4 = A_{2,2} \times (B_{2,1} - B_{1,1})$
$M_5 = (A_{1,1} + A_{1,2}) \times B_{2,2}$
$M_6 = (A_{2,1} - A_{1,1}) \times (B_{1,1} + B_{1,2})$
$M_7 = (A_{1,2} - A_{2,2}) \times (B_{2,1} + B_{2,2})$

3. Calculer $A \times B = \left(\begin{array}{c|c} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{array}\right)$

Exercice 1.1 *Prouver la correction de l'algorithme de Strassen.*

Proposition 1.2 *Cette algorithme a une complexité en* $O\left(n^{\log_2(3)}\right)$

Exercice 1.2 *Prouver cela en reprenant la preuve pour l'exponentiation en considérant* $\frac{D(2^k)}{7^k}$ *à l'étape 4 de la preuve I*

3 Application aux listes

I Recherche dichotomique

Problème : Rechercher un élément a dans une liste L d'éléments triés selon un ordre $\leq$

Algorithme 1.2 : $recherche\_dichotomique(l, a, debut, fin)$

Entrées : L une liste triée, a un élément
Sorties : i tel que $L[i] = a$ s'il existe, -1 sinon
si $fin < debut$ **alors**
  retourner -1
$m \leftarrow \left\lfloor \frac{debut + fin}{2} \right\rfloor$
si $l[m] = -a$ **alors**
  retourner m
si $l[m] < a$ **alors**
  retourner $recherche\_dichotomique(l, a, m+1, fin)$
sinon
  retourner $recherche\_dichotomique(l, a, debut, m+1)$

Exercice 1.3 *Ecrire le code en plus de lignes et faire apparaître les 3 étapes de D&R*

Proposition 1.3 *La recherche dichotomique est en* $O(\log |L|)$

Exercice 1.4 *Écrire une version itérative de cet algorithme*

II Tri fusion

Algorithme 1.3 : $fusion(L_1, L_2)$

Entrées : L_1, L_2 deux listes triées d'éléments comparables
Sorties : La fusion des listes L_1 et L_2 triée
$res \leftarrow []$
$i, j \leftarrow 0$
tant que $i < |L_1|$ et $j < |L_2|$ **faire**
  si $L_1[i] < L_2[j]$ **alors**
    res.ajouter($L_1[i]$)
    $i \leftarrow i + 1$
  sinon
    res.ajouter($L_2[j]$)
    $j \leftarrow j + 1$
Ajouter le reste de L_1 et de L_2 à res
retourner res

Algorithme 1.4 : $tri\_fusion(L)$

Entrées : Une liste L d'éléments comparables
Sorties : La liste L triée
$n \leftarrow |L|$
si $n \leq 1$ **alors**
  retourner L
$L_1, L_2 \leftarrow partionner(L)$; // Étape 1
$A \leftarrow tri\_fusion(L_1)$; // Étape 2
$B \leftarrow tri\_fusion(L_2)$ **retourner** $fusion(A, B)$

Proposition 1.4 *Le tri fusion est $O(n \log n)$ avec $n = |L|$*

Démonstration. Exercice □

Développement : Correction et terminaison du tri fusion

Application 1.1 *Si l'on veut chercher si K entiers sont présents parmi N, on peut avec I et II faire cela en $O\big((N+K)\log N\big)$ au lieu de $O(k \times N)$*

III Tri rapide

Objectif : Faire un tri D&R en place

Commentaire 1.2 *Ici, on peut avoir des questions sur le tri fusion en place, ce qui existe plus ou moins mais est assez pénible. Donc ca peut valoir le coup de se renseigner sur le sujet.*

Idée 1.1 *Choisir un élément appelé pivot et mettre à gauche tous les éléments plus petit, à droite tous les plus grands, puis à trier cette partie à droite et à gauche.*

Algorithme 1.5 : $tri\_rapide(L, debut, fin)$

si $debut \geq fin - 1$ **alors**
  retourner L
$pivot \leftarrow L[0]$
$i \leftarrow debut$
$j \leftarrow fin$
tant que $i < j$ **faire**
  si $L[i+1] \leq pivot$ **alors**
    échanger $L[i+1]$ et $L[i]$
    $i \leftarrow i + 1$
  sinon
    échanger $L[i+1]$ et $L[j]$
    $j \leftarrow j - 1$
  $tri\_rapide(L, debut, i-1)$
  $tri\_rapide(L, i+1, fin)$

Commentaire 1.3 *Par manque de place, on peut remplacer l'écriture de cet algorithme par un dessin expliquant l'idée, et mettre son écriture en exercice.*

Proposition 1.5 *Ce tri est dans le pire des cas en $O(n^2)$, dans le cas moyen en $O(n \log n)$. Il est néanmoins en place (en autorisant un espace non pas constant mais logarithmique).*

Commentaire 1.4 *L'occasion de briller sur le fait que le tri rapide écrit comme ça n'est pas en place avec la définition première, mais l'est avec la définition élargie où on autorise $O(\log n)$ espace (utilisé par les appels récursifs)*

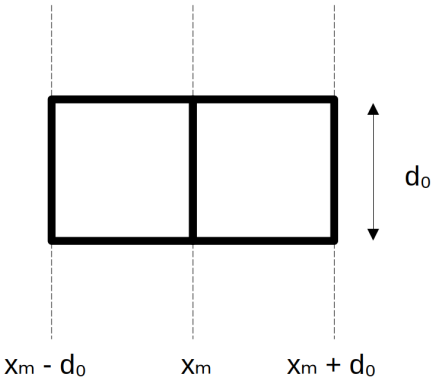
4 Application Géométrique

I Plus petite distance dans le plan

Problème : Étant donné n points de $\mathbb{R}^2$, donner la plus petite distance entre deux points
Solution naïve : n multiplications

Algorithme 1.3 (Solution D&R)
1. *Choisir une abscisse x_m séparant les points en deux sous-ensembles P_1 et P_2 de taille égale (à + ou - 1)*
2. *Trouver les plus petites distances d_1 et d_2 de P_1 et P_2*
3. *Rechercher la plus petite distance d_3 entre deux points dans la bande d'abscisse $[x_m - d_0, x_m + d_0]$ pour $d_0 = \min d_1, d_2$*

L'étape 3 peut se faire en ne regardant que les 7 points suivants (suivants au sens de l'ordonnée).



On a au plus 8 éléments dans ce rectangle, car dans chaque domaine, les points sont au moins à distance d_0. On a donc au plus 4 points par carré, d'où le résultat en mettant notre point sur le bas du rectangle.

Proposition 1.6 *On a une complexité en $O\left(n(\log n)^2\right)$*

II Arbre K-dimensionnel

Problème : Trouver les k plus proches voisins d'un point $y \in \mathbb{R}^K$ parmi un ensemble de n points $x_1, \ldots, x_n \in \mathbb{R}^K$
Solution initiale : Stocker nos k valeurs en cours dans une file de priorité et parcourir les n points, en mettant à jour la file de priorité. $\rightarrow O(n \log k)$

Algorithme 1.4 (Solution D&R) *Faire un pré traitement où l'on stockera nos valeurs dans un arbre binaire de recherche, où l'on partitionnera récursivement les données alternativement sur chaque dimension. La recherche se fait alors en ne cherchant que d'un côté si le deuxième n'est pas nécessaire.*

Développement : Présentation de la structure d'arbre K-dimensionnel

Proposition 1.7 (Complexité)

— *prétraitement :* $O(n\log(n))$

— *Recherche des* k *plus proches voisins :* $O(\log n \times k)$ *en moyenne,* $O(N \times K)$ *dans le pire des cas*

Remarque 1.1 *On y perd pour une seule recherche, mais si on cherche les* k*-plus proches voisins de* N *points, on obtient du* $O(n\log n + N \times k \times \log n)$ *au lieu de* $O(N \times n \times \log k)$.

Commentaire 1.5 *Si il reste beaucoup de places ou si on veut enlever un exemple au dessus (comme le tri rapide faisant doublons avec le tri fusion, même si il est très classique), on peut rajouter une section supplémentaire ici, et parler de l'additionneur à retenue anticipée par méthode D&R du développement *à compléter**

Leçon 2

Exemples d'algorithmes d'apprentissage supervisés et non supervisés

Auteur·e·s: Emile Martinez
Références :

1 Introduction

I Définition

Définition 2.1 *On dit que un algorithme apprend via un entraînement pour un ensemble de tâches et une mesure de performance si sa performance sur les tâches mesuré par la mesure de performance s'améliore après l'entraînement.*

Définition 2.2 *On considère deux familles de l'apprentissage machine :*

1. *l'apprentissage supervisé : on dispose d'espaces X d'entrée et Y de sortie, et d'un ensemble E d'exemples $(x_i, y_i) \in X \times Y$, représentant une fonction $f : X \to Y$. Le but étant alors de construire une fonction $\hat{h} : X \to Y$ approximant f.*
2. *l'apprentissage non supervisé : on dispose seulement de l'espace X dont on veut alors découvrir les structures sous-jacente*

Exemple 2.1 *Sur un ensemble de données sur des animaux, on peut :*

1. *Savoir lesquels sont des chiens, des chats, etc... (apprentissage supervisé)*
2. *Savoir quels animaux sont de la même espèce (apprentissage non supervisé)*

Remarque 2.1 *Il existe d'autres familles, comme l'apprentissage par renforcement qui consiste à maximiser un critère d'utilité par expériences successives.*

II Évaluation d'un algorithme d'apprentissage

On cherche à évaluer la performance d'un algorithme d'apprentissage.

Définition 2.3 *On découpe nos données d'entrées E en deux :*
- *Les données d'apprentissage, servant à entraîner l'algorithme*
- *Les données de validation, servant à imiter la prédiction, mais en comparant le résultat obtenu à celui attendu.*

Remarque 2.2 *Si l'on a trop peu de données, on peut également faire une validation croisée, consistant utiliser alternativement des données comme apprentissage et comme validation*

Remarque 2.3 *Cette phase est également utile pour calibrer les paramêtres de l'algorithme.*

2 Apprentissage supervisé

On reprend les notations de la définition 2.2

Commentaire 2.1 *Dans un vrai cours, on réécrirait les définitions pour plus de clarté*

Définition 2.4
Si Y est un ensemble fini de classes, on parle de problème de classification.
Si $Y = \mathbb{R}$, on parle de problèmes de régression.

Exemple 2.2 *Sur un ensemble de données sur des animaux, on peut avoir :*
- $Y = \{'chien','chat'\}$ *(classification)*
- $Y = \mathbb{R}$ *représentant le poids de l'animal (régression)*

Concentrons nous sur le problème de classification. On cherche à inférer de E, la classe de $\alpha \in X$. Notons C la partition de X représentant les différentes classes.

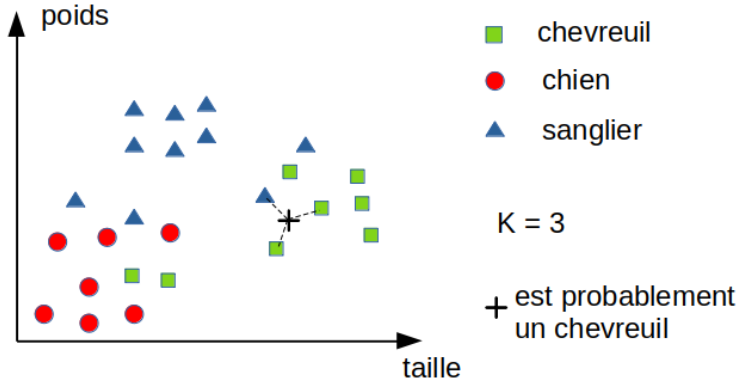
I K plus proches voisins

On s'intéresse ici au cas où $X = \mathbb{R}^d$

Algorithme 2.1 *K plus proches voisins de α*
1. *Déterminer les K plus proches voisins de α dans E*
2. *Choisir la classe majoritaire parmi ces voisins.*

Développement : Présentation de l'algorithme illustrant différents aspects de l'apprentissage machine.

Exemple 2.3



Remarque 2.4 *Ici pour choisir k, on utilise la méthode de mesure de performance du II en essayant plusieurs paramètres.*

Exercice 2.1 *Appliquer l'algorithme sur le jeu de données MINST.*

Remarque 2.5 *Pour un problème de régression, on ne choisirait pas la classe majoritaire à l'étape 2, mais on agrégerait les données (par exemple en faisant une moyenne).*

Implémentation 2.1

Solution initiale : *Stocker nos K valeurs en cours dans une file de priorité (implémentée par un tas max) et parcourir les n points en mettant à jour la file de priorité*

Complexité : $O(n \log k)$

Solution diviser pour régner : *Faire un pré traitement où l'on stockera nos valeurs dans un arbre binaire de recherche (arbre d-dimensionnel), où l'on partitionnera récursivement les données alternativement sur chaque dimension. La recherche se fait alors en ne cherchant que d'un côté si le deuxième n'est pas nécessaire.*

Complexité :

Prétraitement : $O(n \log n)$

Recherche des k voisins : $O(k \log n)$ *en moyenne et* $O(nk)$ *dans le pire des cas.*

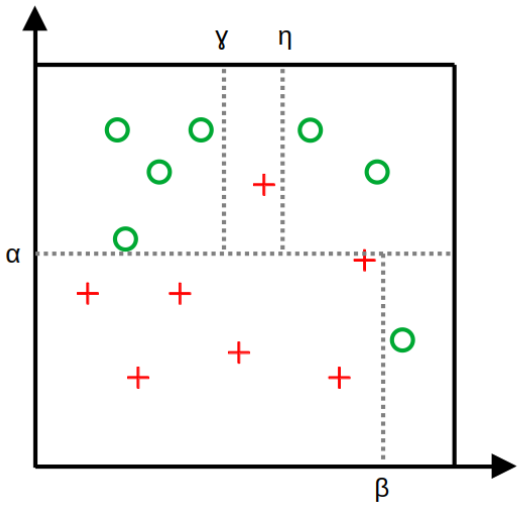
Développement : Présentation de la structure d'arbre d-dimensionnel.

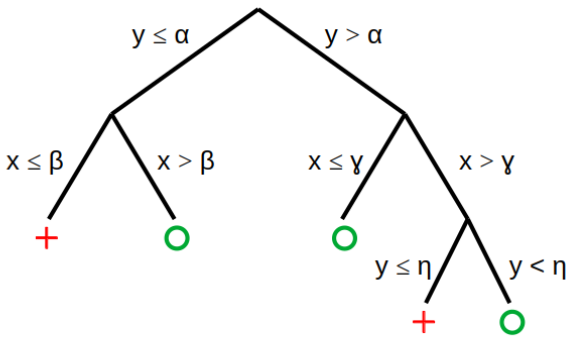
Remarque 2.6 *On parle souvent d'arbre k-dimensionnel, mais on prend ici d pour éviter la confusion avec K.*

II Arbre de décision

On s'intéresse ici au cas $X = \mathbb{R}^d$ (ou $\{O, 1\}^d$)

Idée 2.1 *Partitionner récursivement X grâce à un arbre de décision où chaque feuille a une classe.*





Définition 2.5 *Définissons l'entropie d'une partie S de E :* $H(S) = \sum_{c \in C} \frac{|S \cap c|}{|S|} \times \log\left(\frac{|S \cap c|}{|S|}\right)$

Remarque 2.7 *Si tous les éléments ont la même classe,* $H(S) = 0$

Définition 2.6 *Le gain d'une partition S_1, S_2 de S est :*

$$H(S) - \left(\frac{|S_1|}{|S|} \times H(S_1) + \frac{|S_2|}{|S|} \times H(S_2)\right)$$

Algorithme 2.2 *On construit récursivement notre arbre de décision sur notre ensemble S de données restantes :*

- *Si S est vide : Choisir la classe la plus représentée du nœud parent*
- *Si toutes les données de S ont la même classe : en faire une feuille avec cette classe.*
- *Sinon, on choisit la coordonnée i et la valeur m tel que la partition*

$$S_1 = \{(x_1, \ldots, x_n) \in S / x_i \leq m\} \text{ et } S_2 = \{(x_1, \ldots, x_n) \in S / x_i > m\}$$

maximise le gain

Remarque 2.8 *On atteint très vite du sur apprentissage. Pour éviter cela, on peut élaguer le bas de l'arbre (algorithme de Cart)*

Exercice 2.2 *Application à détecter la langue d'une page wikipedia à partir de la matrice de fréquence de facteurs de 2 lettres.*

Commentaire 2.2 *Ici pour ce TD, on peut mentionner à l'oral le fait qu'on peut utiliser ce truc pour de vrais, et surtout que on peut éventuellement faire réfléchir les élèves à comment modéliser le problème. Fréquence des mots, KNN avec distance de levenstein, fréquence des facteurs de 1, 2, 5 lettres ? Ce qui rend tout ca très intéressants selon moi. (suivant le nombre de pages en exemples, on peut prendre les facteurs d'un certain nombres de lettres (on aura un tableau de taille $27^{\text{taille des facteurs}}$) et ensuite faire du KNN, de l'arbre de décision, etc...) (Avec 4000 pages par langues pour 13 langues, et les facteurs de 2 mots, on classifie très bien).*

Remarque 2.9 *Dans le cas d'une régression, on peut prendre comme mesure d'impureté la variance.*

III Représentation de la qualité des classes

Quand on mesure la performance de notre algorithme on peut chercher à représenter la qualité de nos prédictions (pour classification).

Définition 2.7 (matrice de confusion) *On associe Y à $\{1, \dots, n\}$. La matrice de confusion est alors la matrice carrée M de taille n tel que $M_{i,j}$ est le nombre d'éléments de la classe i qui ont été classés à j*

Proposition 2.1 *Plus notre algorithme est correcte, plus la diagonale est dominante.*

3 Apprentissage non supervisé

Remarque 2.10 *Il existe plusieurs types d'apprentissage non supervisé. On peut par exemple penser à la réduction de dimension : On a $X \subset \mathbb{R}^n$ et on cherche $f : X \to \mathbb{R}^m$ avec $n < m$ et tel que $f(x)$ et $f(y)$ sont proches si x et y le sont aussi.*

On se concentrera sur ce qu'on appellera regroupement, (clustering en anglais, ou classification non supervisée) qui consiste à trouver $f : X \to \{1, ..., k\}$ essayant de regrouper les éléments les plus proches.

Exemple 2.4 *On a un manuscrit dans un alphabet inconnu, et on cherche à savoir quels lettres sont les mêmes.*

Exercice 2.3 *Dans l'exemple au dessus, quel X prendre ?*

I Classification hiérarchique ascendante

Idée 2.2 *Chacun est seul dans sa classe au début, et tant qu'on a k classes, on fusionne les classes les plus proches.*

Remarque 2.11 *C'est un algorithme glouton*

Remarque 2.12 *Pour définir la distance entre deux classes, on prendre prendre :*

- $d(S_1, S_2) = \min_{x \in S_1, y \in S_2} (d(x, y))$
- $d(S_1, S_2) = \min_{x \in S_1, y \in S_2} (d(x, y))$
- $d(S_1, S_2) = \dfrac{1}{|S_1| \times |S_2|} \sum_{x \in S_1, y \in S_2} d(x, y)$

Exercice 2.4 *Activité : Représenter l'exécution de l'algorithme sur un dendrogramme*

II ALgorithme des k-moyennes

Idée 2.3 *On cherche à trouver $S_1, \dots, S_k$ une partition de X et $z_1, \dots, z_k \in \mathbb{R}^d$ minimisant $\sum_{i=1}^{k} \sum_{x \in S_i} d(x, z_i)$*

Algorithme 2.1 : K-mean

Assigner à chaque valeur une classe aléatoire
répéter
  pour $x \in X$ **faire**
    assigner à x la classe de $\arg\min_{i \in \{1,\dots,k\}} d(x, z_i)$
  pour i allant de 1 à k **faire**
    $z_i \leftarrow \frac{1}{|S_i|} \sum_{x \in S_i} x$
jusqu'à stabilisation;