Chapitre 3 : Intelligence artificielle

1 Vocabulaire des bases de données

Dans ce chapitre, nous utiliserons des bases des données, aussi on rappelle, dans cette première section le vocabulaire associé.

Vocabulaire 1.1

*On appelle **signature de données** un tuple de couples : nom, ensemble. On la typographie alors : (nom$_1$: S_1, nom$_2$: S_2, ..., nom$_n$: S_n).*

Exemple 1.2

Par exemple $\mathbb{S}_1$: (titre : `string`, longueur : $\mathbb{N}$, date : $\mathbb{N}$) est une signature de données, mais aussi $\mathbb{S}_2$: (x : $\mathbb{R}$, y : $\mathbb{R}$), mais aussi $\mathbb{S}_3$: (R : $[\![0, 255]\!]$, G : $[\![0, 255]\!]$, B : $[\![0, 255]\!]$).

Vocabulaire 1.3

*Étant donné une signature de données $\mathbb{S} = (\text{nom}_1 : S_1, \text{nom}_2 : S_2, \ldots, \text{nom}_n : S_n)$, on appelle **donnée**, ou **enregistrement** un vecteur $\overline{v} = (v_1, v_2, \ldots, v_n) \in S_1 \times S_2 \times \ldots \times S_n$ potentiellement hétérogène. Nous noterons $\mathbb{D} = S_1 \times S_2 \times \ldots \times S_n$ l'ensemble des données possibles pour la signature $\mathbb{S}$.*

Remarque 1.4

Il n'est pas rare de ne pas donner de noms aux coordonnées des vecteurs et de se contenter de leur indexation naturelle.

Exemple 1.5

Par exemple $(\texttt{"2001, a space odyssey"}, 139, 1968)$ est une donnée de signature $\mathbb{S}_1$. $(\pi, \sqrt{2})$ est une donnée de signature $\mathbb{S}_2$.

Vocabulaire 1.6

*On appelle **base de données** (ou parfois **jeu de données**) un ensemble fini de données de même signature.*

L'objectif de ce chapitre est de construire des fonctions dites "de classification", ces fonctions prennent en arguments des données et leur associent une **classe**. L'ensemble des **classes** possible sera ici un ensemble fini, généralement noté $\mathcal{C}$.

Vocabulaire 1.7

*Étant donné une signature de données $\mathbb{S}$, et un ensemble de classes $\mathcal{C}$, on appelle **fonction de classification** (ou parfois **classifieur**), une fonction de l'ensemble des données de signatures $\mathbb{S}$ dans l'ensemble $\mathcal{C}$.*

Vocabulaire 1.8

Soient $\mathbb{S}$ une signature de données, $\mathbb{D}$ l'ensemble correspondant et $\mathcal{C}$ un ensemble de classes.
*On appelle **jeu de données classifiées** un ensemble fini $\mathcal{D} \subseteq \mathbb{D} \times c$ tel que π_1 la projection sur $\mathbb{D}$ est injective. Autrement dit $\mathcal{D}$ est un ensemble de couples de la forme (x, c) dans lequel chaque élément $x \in \mathbb{D}$ apparaît au plus une fois.*
*Si un couple (x, c) apparaît dans le jeu de donnée D, on dit que **la classe de** x est c.*

2 Apprentissage supervisé

L'apprentissage supervisé consiste à construire des **fonctions de classification**, aussi appelées **classifieurs**, à partir d'un jeu de données classifiées. Dans cette section on suppose donc fournie une base de données dans laquelle chaque donnée est classifiée (une classe lui est associée), on se pose alors le problème de chercher une fonction de classification qui coïncide avec les données.

Remarque 2.1

Le problème de l'apprentissage dans le cas supervisé est en fait le problème de prolongement d'une fonction : le jeu de données classifiées fournit une fonction partielle de $\mathbb{D}$ dans $\mathcal{C}$, la fonction de classification doit en être un prolongement.

2.1 Algorithme des k plus proches voisins (k-NN)

L'abréviation k-NN vient du nom anglais de cet algorithme : *k-nearest neighbours*.

On considère dans cette section que l'espace des données possibles $\mathbb{D}$ est muni d'une distance d♣. Cette distance peut être, par exemple, dérivée d'une norme.

L'idée de l'algorithme est la suivante. Étant donné un jeu de données classifiées $\mathcal{D} \subseteq \mathbb{D} \times \mathcal{C}$, et une donnée $x \in \mathbb{D}$ à classifier, on cherche k points de $\mathcal{D}$ les plus proches de la donnée x et on les fait voter sur la classe de x, c'est-à-dire qu'on associe à x la classe majoritaire parmi celles de ses k plus proches voisins.

Algorithme 1 : Algorithme k-NN

Entrée : Un jeu de données classifiées $\mathcal{D} \subseteq \mathbb{D} \times \mathcal{C}$, une donnée à classifier $x \in \mathbb{D}$, un paramètre $k \in \mathbb{N}^*$
Sortie : Une classification de x (une classe de $\mathcal{C}$ donc)

```
On extrait de D les k points les plus proches ♡ de x : (y_1, c_1), (y_2, c_2), ..., (y_k, c_k) ;
compteurs ← dictionnaire de C vers ℕ, initialisé à 0 ;
pour i = 1 à k faire
    compteurs[c_i] = compteurs[c_i] + 1 ;
retourner arg max_{c∈C}(compteurs[c]) ;
```

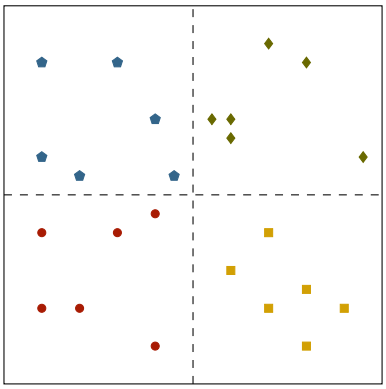
♣. $d : \mathbb{D} \times \mathbb{D} \to \mathbb{R}^+$ est une distance dès lors que
- $\forall (a, b) \in \mathbb{D} \times \mathbb{D}, d(a, b) = 0 \Leftrightarrow a = b$;
- $\forall (a, b) \in \mathbb{D} \times \mathbb{D}, d(a, b) = d(b, a)$;
- $\forall (a, b, c) \in \mathbb{D}^3, d(a, b) + d(b, c) \geqslant d(a, c)$.

♡. Implicitement ici la distance d'une donnée classifiée (y, c) à x est $d(y, x)$.

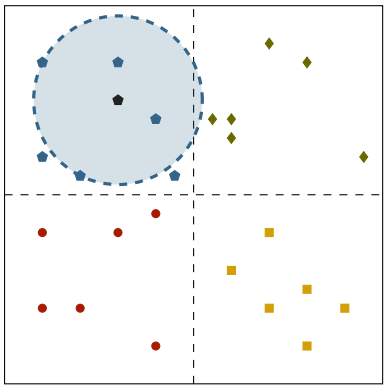
Remarque 2.2

La ligne 1 de l'algorithme 1 peut être modifiée. En effet il n'est pas nécessaire pour la suite de l'algorithme d'avoir trié toutes les données, il suffit d'avoir identifié $(y_1, c_1), (y_2, c_2), \ldots, (y_k, c_k)$ les k plus proches voisins du point à classifier. On verra à la section suivante comment on peut, dans certains cas, réaliser cette étape plus efficacement.

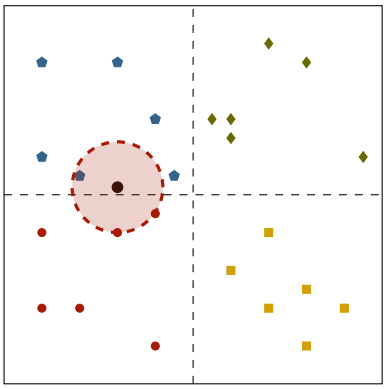
On illustre cet algorithme en essayant de classifier les points du carré unité selon le quadrant auquel ils appartiennent. Ainsi on fournit à l'algorithme une liste de points déjà classifiés, c'est le jeu de données ci-dessous à gauche♣.



Base de données classifiées



Classification "correcte"



Classification "incorrecte"

Plus précisément chaque point correspond à une donnée, la donnée elle-même fixe les coordonnées du point tandis que sa classe fixe la forme et la couleur du point. L'algorithme de classification doit alors "deviner" dans quel quadrant se trouve un point à partir de ses coordonnées. Ci-dessus on illustre le résultat d'une classification correcte, et le résultat d'une classification incorrecte. On a choisi la distance euclidienne pour l'algorithme des k plus proches voisins (d'où les boules rondes) et le paramètre $k = 3$.

Exercice de cours 2.3

Donner un exemple de problème de classification (un jeu de données et une donnée à classifier) pour lequel :

- l'augmentation de la valeur de k change le résultat de la classification ;
- le choix de la distance influe sur le résultat de la classification.

On pourra s'inspirer de l'exemple de la classification en quadrants.

Exercice de cours 2.4

On considère le jeu de données classifiées ci-contre.
Les données sont représentées par 3 coordonnées entières : x, y et z, et l'ensemble des classes est $\mathcal{C} = \{A, B\}$. On munit l'espace des données, ici $\mathbb{D} = \mathbb{R}^3$ de la norme infinie :

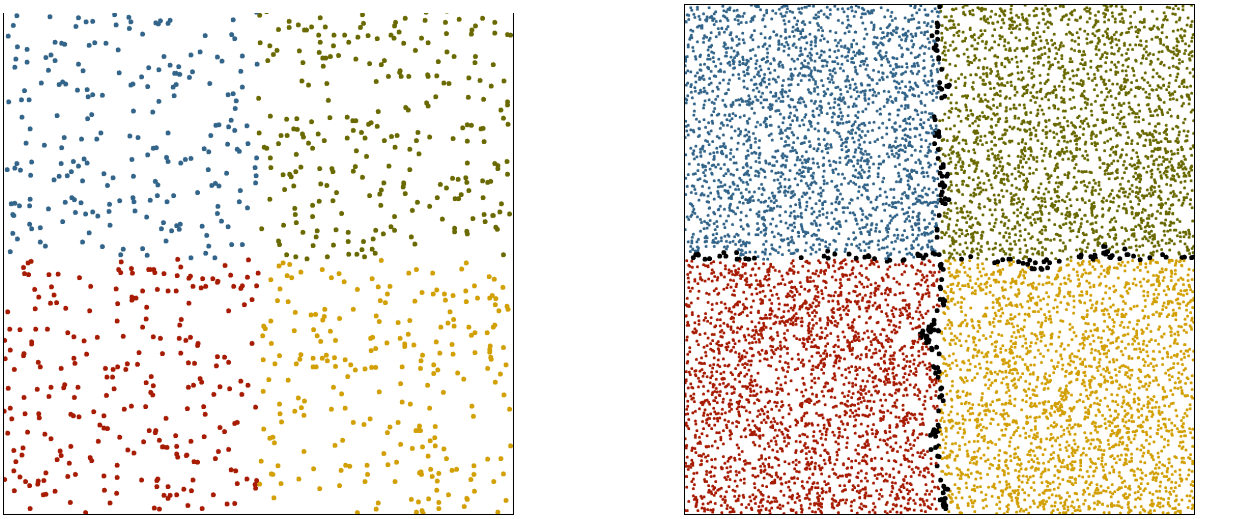
$$d((x_1, y_1, z_1), (x_2, y_2, z_2)) = \max\{|x_1 - y_1|, |x_2 - y_2|, |x_3 - y_3|\}$$

Exécuter l'algorithme des 3 plus proches voisins pour classifier les points $(2, 1, 1)$ et $(3, 0, 0)$.

x	y	z	classe
0	1	0	A
0	3	1	B
0	2	1	A
3	2	1	B
1	3	1	B
2	1	0	A
1	0	2	B
3	2	3	A

♣. ⚠ **Attention :** Sur les figures on représente la délimitation des quadrants à l'aide de pointillés, mais cette information n'est pas contenue dans le jeu de données classifiées.

Influence du paramètre k. Le paramètre k a une influence sur la qualité des résultats fournis par l'algorithme k-NN. Pour évaluer ces résultats, il faut avoir un moyen de connaître la "vraie" classe d'une donnée, ce qui n'est pas toujours possible. On reprend l'exemple de classification en quadrants, pour lequel on sait classifier de manière sûre (on compare x et y à 0.5). Dans la Figure 1 ci-dessous on représente d'une part le jeu de données classifiées (Figure 1a), et d'autre part les résultats de 10000 classifications (Figure 1b). Ainsi, dans la Figure 1b, chaque point correspond à une donnée qu'on a classifiée, la donnée elle-même fixe sa position, tandis que la classe qu'on lui a attribué fixe l'apparence du point : ● indique un échec de classification (on a attribué la mauvaise classe à cette donnée).



(a) Jeu de 1000 points classifiés

(b) Résultats de 10000 classifications, avec $k = 5$

FIGURE 1 – Erreurs de classification de k-NN pour les quadrants

Matrice de confusion. Afin de représenter les résultats empiriques d'une classification, on évite de simplement différencier réussite et échec. On préfère montrer quelles classes ont été confondues avec quelles classes, au moyen d'une **matrice de confusion**.

Définition 2.5

On assimile $\mathcal{C}$ l'ensemble des classes considérées à $[\![1, n_{\mathcal{C}}]\!]$. Pour un jeu de données à classifier et un classifieur fixés, on appelle ***matrice de confusion*** *la matrice dont le coefficient d'indice (i, j) est le nombre d'éléments de la classe i qui ont été classifiés comme étant de la classe j.*

Remarque 2.6

Dans le cas d'un classifieur parfait, on obtiendrait une matrice diagonale pour n'importe quel jeu de données à classifier.

La somme des coefficient de cette matrice est le nombre total de données classifiées prises en compte, ce qui n'a rien à voir avec le nombre de données dans le jeu de données classifiées utilisé par le classifieur.

On représente ci-dessous les matrices de confusion de deux classifieurs classifiant les mêmes 10000 points tirés uniformément dans le carré unité. Le classifieur de gauche est l'algorithme k-NN (pour $k = 5$), construit sur un jeu de données classifiées de taille 1000, tandis que le classifieur de droite est aussi l'algorithme k-NN (pour $k = 5$) mais construit sur un jeu de données classifiées de taille 10000. On observe que cette multiplication par 10 de la taille du jeu de données d'apprentissage entraîne une réduction importante du nombre d'éléments mal classifiés.

$$\begin{array}{c} \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array}\begin{array}{c} \begin{array}{cccc} \bullet & \bullet & \bullet & \bullet \end{array} \\ \begin{pmatrix} 2444 & 33 & 24 & 0 \\ 13 & 2458 & 0 & 48 \\ 26 & 1 & 2411 & 31 \\ 0 & 11 & 13 & 2487 \end{pmatrix} \end{array}$$

Pour jeu d'apprentissage de taille 1000

$$\begin{array}{c} \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array}\begin{array}{c} \begin{array}{cccc} \bullet & \bullet & \bullet & \bullet \end{array} \\ \begin{pmatrix} 2473 & 7 & 7 & 0 \\ 5 & 2512 & 0 & 4 \\ 3 & 0 & 2396 & 6 \\ 0 & 2 & 9 & 2576 \end{pmatrix} \end{array}$$

Pour jeu d'apprentissage de taille 10000

Exercice de cours 2.7

Une matrice de confusion est-elle nécessairement symétrique ?

Exercice de cours 2.8

Pour la matrice de confusion ci-contre, donner : le nombre de données utilisées pour la construction de la matrice de confusion ; le nombre de données de classe 1 ; le nombre de données que l'algorithme a classifié de classe 2 ; le taux de succès (la proportion des données qui ont été correctement classifiées).

$$\begin{pmatrix} 27 & 2 & 3 \\ 4 & 10 & 3 \\ 1 & 0 & 14 \end{pmatrix}$$

Faux positifs. Dans le cas particulier d'un classifieur dans $\mathcal{C} = \{\text{vrai}, \text{faux}\}$, on nomme les différents cas de classification avec un adjectif et un substantif :
- l'adjectif "vrai" ou "faux" indique si la classification est correcte ou non ;
- le substantif "positif" ou "négatif" indique si le résultat de la classification est vrai ou faux.

Par exemple dans le cas où une donnée est classifiée à faux à tort, on parle de faux négatif. La matrice ci-dessous synthétise les 4 cas possibles en suivant la présentation des matrices de confusion : la ligne indique la classe réelle de l'élément et la colonne indique la classe qu'on lui a attribuée.

$$\begin{array}{c} \\ \text{vrai} \\ \text{faux} \end{array}\begin{array}{c} \begin{array}{cc} \text{vrai} & \text{faux} \end{array} \\ \begin{pmatrix} \text{vrai positif} & \text{faux négatif} \\ \text{faux positif} & \text{vrai négatif} \end{pmatrix} \end{array}$$

Une telle nomenclature est bien sûr étendue à d'autres domaines, on peut penser par exemple à des tests Covid♣.

2.2 Arbres k-dimensionnels

Dans l'algorithme k-NN, la recherche des k plus proches voisins est une opération centrale de la classification. Une implémentation naïve de ce calcul nécessitera (au moins) un parcours de tout le jeu de données. Dans cette section on présente une structure de données pour stocker le jeu de données $\mathcal{D}$ qui permet d'améliorer la complexité des opérations de recherche de plus proche voisin. On travaille sous l'hypothèse que $\mathbb{D} \subseteq \mathbb{R}^k$ ♡, et que la distance d considérée est la distance euclidienne (que l'on note $||\bullet||_2$). De plus, afin de se concentrer sur la recherche de voisins, on omettra dans cette section la classe des données. Bien sûr, lors d'une application pratique pour l'algorithme k-NN, il faut stocker avec chaque donnée sa classe, afin d'avoir accès en temps constant aux classes des plus proches voisins.

♣. ou comme on le verra plus tard dans l'année aux algorithmes probabilistes de type Monte-Carlo

♡. cette hypothèse implique non seulement que le jeu de données $\mathcal{D}$ est inclus dans $\mathbb{R}^k$, mais aussi que les points à classifier sont dans $\mathbb{R}^k$.

⚠ **Attention :** Dans la suite de cette section k n'est plus le nombre de plus proches voisins à prendre en compte, mais la dimension de l'espace des données.

Construction arborescente. Pour ce faire on utilise une structure arborescente : les arbres binaires. Pour ce faire on utilise une structure arborescente, plus précisément des arbres binaires dont le cas de base est l'arbre vide♣. Les éléments du jeu de données $\mathcal{D}$ seront stockés dans les nœuds de l'arbre. À chaque nœud interne, on sépare géographiquement les données en deux, les unes seront alors stockées dans le sous-arbre droit, les autres dans le sous-arbre gauche, en dehors de la donnée sur laquelle s'appuie cette séparation que l'on stocke dans le nœud lui même. On se limite à des séparations du type "au dessus ou en dessous de la valeur v_i sur la coordonnée i". Plus formellement, étant donné un ensemble $\mathcal{V}$ de vecteurs de $\mathbb{R}^k$, un vecteur $v \in \mathcal{V}$ et une dimension $i \in [\![1, k]\!]$, il est possible de partitionner $\mathcal{V} \setminus \{v\}$ est deux sous-ensembles : ceux dont la i-ème composante est strictement inférieure à v_i et ceux, différents de v, dont la i-ème composante est supérieure ou égale à v_i. On note ces deux ensembles respectivement $\mathcal{V}_v^{<i}$ et $\mathcal{V}_v^{\geqslant i}$.

Définition 2.9

Soit $\mathcal{V} \subseteq \mathbb{R}^k$. Soit $i \in [\![1, k]\!]$ une dimension fixée.
*Un **arbre k-dimensionnel** pour $\mathcal{V}$ à partir de la dimension i est :*
- *l'arbre vide si $\mathcal{V} = \emptyset$,*
- *un arbre dont la racine est étiquetée par (v, i), de fils gauche g, de fils droit d où :*
  - *$v \in \mathcal{V}$;*
  - *g est un arbre k-dimensionnel pour $\mathcal{V}_v^{<i}$ à partir de la dimension $i + 1 \mod k$;*
  - *d un arbre k-dimensionnel pour $\mathcal{V}_v^{\geqslant i}$ à partir de la dimension $i + 1 \mod k$.*

Cette définition récursive mène naturellement à une construction récursive. Sur la figure 2 on représente la construction d'un tel arbre sur un ensemble $\mathcal{V}$ de $\mathbb{R}^2$ à 5 points (l'arbre résultat a donc cinq nœuds). Les nœuds qui coupent selon la première composante sont représentés par | car ils correspondent à un partitionnement vertical, et ceux sur la seconde par $-$ car ils correspondent à un partitionnement horizontal. De plus on a représenté en feuille les zones géographique sous-jacentes♡, mais ces feuilles ne sont pas présentes en machine (pour l'arbre résultat il y a des arbres vides à la place).

Choix des valeurs de découpe. Afin de limiter le nombre d'étapes lors de la recherche de plus proches voisins, on cherche à fabriquer un arbre le moins haut possible. La hauteur de l'arbre fabriqué dépend du choix du vecteur $v \in \mathcal{V}$ selon lequel le découpage est fait. Étant donnée une dimension de découpe i, on choisit un vecteur $v \in \mathcal{V}$ de sorte que v_i soit la valeur médiane des $\{w_i \mid w \in \mathcal{V}\}$. Ainsi à chaque itération l'ensemble $\mathcal{V}$ à traiter est découpé en deux sous-ensembles de même taille♠.

Exercice de cours 2.10

Donner, en fonction de n le nombre de vecteurs à ranger dans l'arbre, une majoration pertinente de la hauteur de l'arbre ainsi obtenu.

♣. par opposition aux arbres binaires non vides dont le cas de base est la feuille
♡. pour chaque feuille la zone géographique est déterminée par les contraintes le long de la branche depuis la racine
♠. À un près

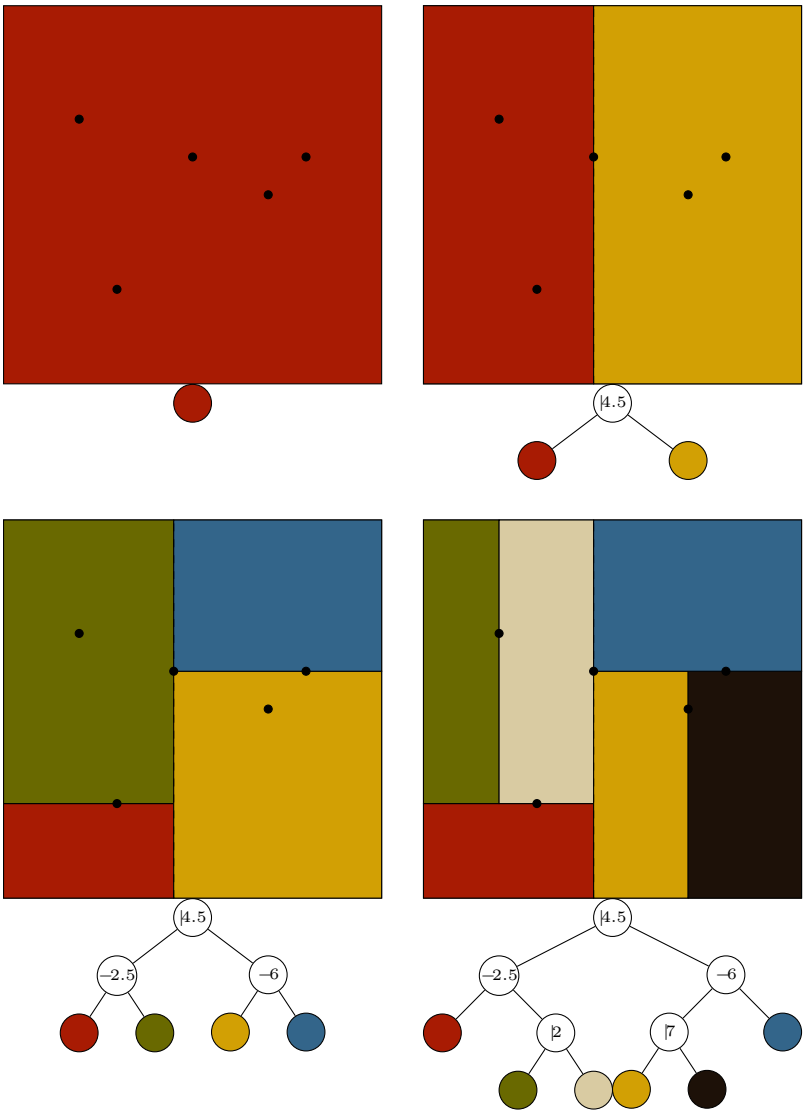


FIGURE 2 – Étapes de construction d'un arbre k-dimensionnel

Recherche d'un plus proche voisin. Une fois l'arbre k-dimensionnel représentant un ensemble de vecteurs $\mathcal{V}$ construit♣, on implémente la recherche d'un plus proche voisin de $x \in \mathbb{R}^k$ dans $\mathcal{V}$ de manière récursive (*Cf.* algorithme 2). L'intérêt de la structure d'arbre k-dimensionnel est que l'on peut éviter de parcourir tout le jeu de données, pour en profiter pleinement il faut être parcimonieux dans les appels récursifs. Ainsi on commence à chercher le plus proche voisin de x dans la zone où x se trouve (zone géographique définie par l'étiquette de la racine de l'arbre). Si l'on en trouve un, disons c_d (comme candidat), on teste si on peut être sûr qu'il est le plus proche voisin global. Pour cela on compare la distance entre c_d et x à la distance entre l'axe de découpe et x. En effet tous les points de l'autre zone ont une distance à x supérieure à celle de l'axe. Dans le cas où c est plus proche de x que la frontière, alors on s'épargne la recherche dans l'autre zone. On illustre ces différents cas de figure dans le cas où $k = 2$, la racine représente une séparation selon la première composante (donc l'axe de découpe est vertical) sur le vecteur v, et où $x_i \geqslant v_i$ (le point x est donc à droite de l'axe).

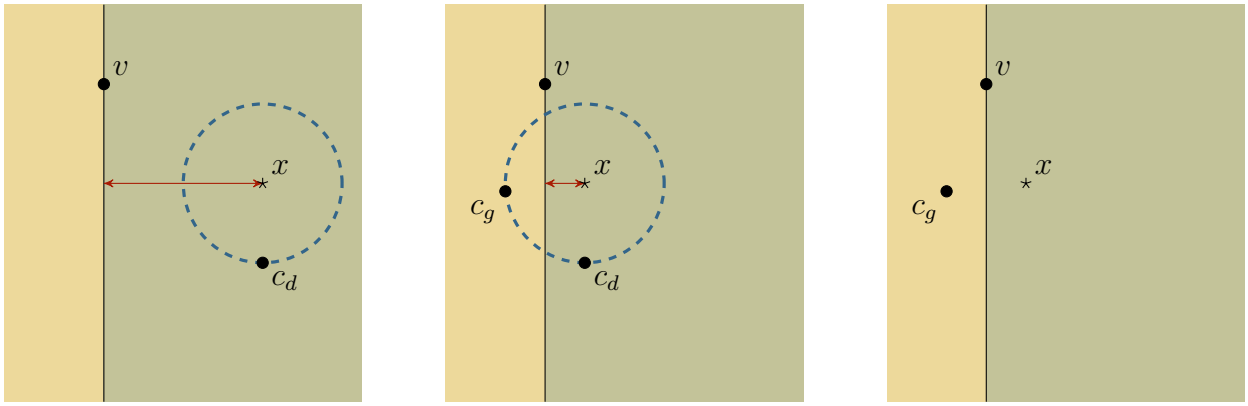


FIGURE 3 – Trois cas de figure possibles lorsque $v_i \leqslant x_i$

♣. Il est important de noter que la construction de l'arbre n'est faite qu'une seule fois, en pré-traitement, et non pas lors de chaque classification.

Algorithme 2 : Recherche du point le plus proche (PPV)

Entrée : Un arbre k-dimensionnel T représentant $\mathcal{V} \subseteq \mathbb{R}^k$, un point $x \in \mathbb{R}^k$
Sortie : $\arg\min_{w \in \mathcal{V}} \|x - w\|_2$ si $\mathcal{V} \neq \emptyset$, None sinon

```
si T est vide alors
    retourner None
sinon
    (v, i, g, d) ← T ;                    // vecteur de découpe, dimension, ss-arbres g. et d.
    si x_i ⩽ v_i alors
        c_g ← PPV(g, x) ;
        si c_g ≠ None et ‖c_g − x‖_2 ⩽ x_i − v_i alors
            retourner c_g ;
        sinon
            c_d ← PPV(d, x) ;
            Candidats ← {v, c_g, c_d} \ {None} ;
            retourner le point de Candidats qui est le plus proche de x ;
    sinon
        c_d ← PPV(d, x) ;
        si c_d ≠ None et ‖c_d − x‖_2 ⩽ x_i − v_i alors
            retourner c_d ;
        sinon
            c_g ← PPV(g, x) ;
            Candidats ← {v, c_g, c_d} \ {None} ;
            retourner le point de Candidats qui est le plus proche de x ;
```

Recherche de k' plus proche voisin. Il est bien sûr possible d'étendre cet algorithme pour calculer non plus le plus proche voisin, mais les k' plus proches voisins. En ce cas il faut paramétrer l'algorithme par k' le nombre de voisins recherchés. Lorsque la recherche du premier côté remonte des points candidats, on teste lesquels parmi eux sont "sûrs" (*i.e.* plus proche de x que de la frontière), disons qu'il y en a s, et on ne lance la recherche de l'autre côté que pour $k' - s$ voisins, ce qui revient à ne pas chercher dans le cas où $k' - s = 0$, c'est-à-dire dans le cas où l'on est sûr que la recherche du premier côté a fourni les k plus proches voisins.

De l'intérêt des arbres k-dimensionnels. On ne fera pas ici d'étude de complexité précise de l'algorithme de recherche de plus proche voisin. On peut cependant retenir qu'en pratique on passe d'une complexité linéaire à une complexité logarithmique puisque lors d'une recherche, la plupart du temps on ne fait qu'un seul appel récursif sur un ensemble de données deux fois plus petit. La figure 4 illustre le fait qu'on ne parcourt qu'une très petite partie des données sur un exemple dans $\mathbb{R}^2$ ($k = 2$), pour la recherche des 10 plus proches voisins ($k' = 10$). Sur cette figure :

- les points ● représentent les points du jeu de données non visités lors la recherche ;
- les points ● sont les points ayant été considéré par l'algorithme comme étant des voisins potentiels ;
- les points ● sont les 10 plus proches voisins du point se trouvant au centre du cercle.

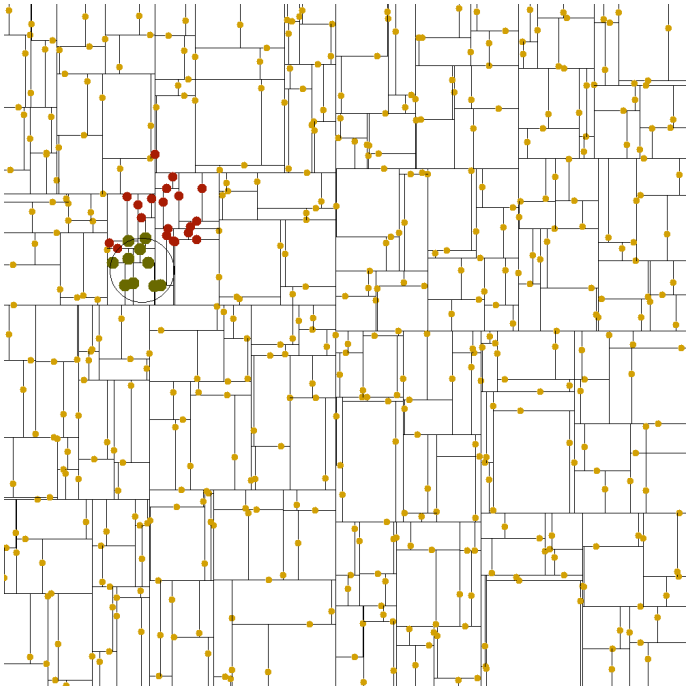


FIGURE 4 – Recherche dans un arbre k-dimensionnel.

2.3 Algorithme ID3

Dans les sections précédentes nous avons pris l'exemple simpliste de l'apprentissage de la fonction de classification des points du carré unité en 4 quadrants. Sur de telles données il aurait été préférable que l'algorithme d'apprentissage consacre plus de temps à la phase apprentissage que l'algorithme des k plus proches voisins. Cette phase d'apprentissage aurait pu servir à inférer l'arbre ci-dessous.

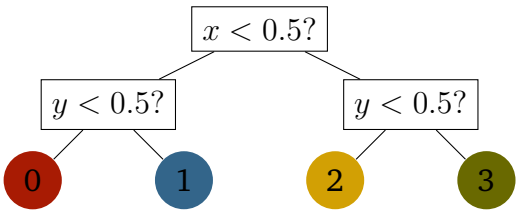


FIGURE 5 – Arbre de décision qui classifie en quadrants

Un tel arbre est appelé **arbre de décision** et correspond à un algorithme de classification. En effet, pour un arbre donné on trouve la classe d'une donnée dans la feuille dans laquelle on aboutit pour cette donnée : chaque nœud étant étiqueté par un prédicat, on descend dans l'arbre à gauche ou a droite selon que la donnée vérifie ou non ce prédicat. En général les décisions qui étiquettent les nœuds d'un arbre de décision sont des prédicats quelconques sur les différentes composantes d'une donnée, mais on se limite souvent à des prédicats sur une seule composante, comme ici "$y < 0.5?$". De plus dans ce cours on se limite aussi à des données pour lesquelles chaque composante n'a que deux valeurs possibles♣. Ainsi un nœud d'un arbre de décision sera simplement étiqueté par une des composantes, et le sous-arbre gauche correspond au cas où cette composante vaut F et le droit au cas où elle vaut V.

♣. On peut assimiler ces deux valeurs à V et F

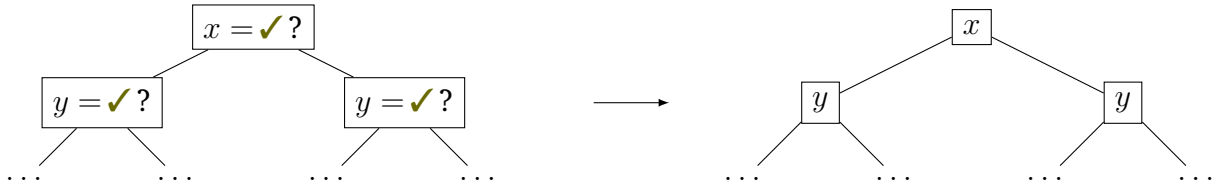


FIGURE 6 – Simplification des prédicats portant sur une unique donnée booléenne

Remarque 2.11

On peut parfois transformer le jeu de données initiales pour qu'il corresponde à cette restriction. Par exemple si une des composante est l'âge, on peut la transformer en une donnée booléenne qui indique si la personne est majeure, ce qui suppose que séparer les âges selon la valeur 18 est pertinent. On peut aussi proposer plusieurs données booléennes à partir d'une donnée numérique. Par exemple, on pourrait aussi déduire de l'âge si la personne est "senior", en supposant cette fois que séparer les âges selon la valeur 65 est probablement pertinent. L'algorithme qui suit, s'il a à sa disposition dans les données d'entrée une composante "adulte" et une composante "senior", pourra choisir laquelle l'intéresse le plus pour classifier, mais ne permet pas, à partir d'une composante numérique, de trouver sur quelle valeur il faut "brancher".

Exercice de cours 2.12

Donner un encadrement, le plus précis possible, du nombre de nœuds internes d'un arbre de décision pour un jeu de données sur $\mathbb{B}^n$ avec $n \in \mathbb{N}^\star$.

Exemple de la classification de véhicules. Dans cette section on travaille sur un jeu de données qui présente les caractéristiques de plusieurs véhicules. Plus précisément, chaque véhicule est représenté par un vecteur de $\mathbb{B}^4$ puisqu'il est décrit par quatre caractéristiques booléennes :
- Moteur qui indique si le véhicule a un moteur ;
- Rails qui indique si le véhicule avance sur des rails ;
- Souterrain qui indique si le véhicule se déplace sous terre ;
- Vitesse qui indique si le véhicule dépasse les 320km/h.

Ce jeu de données est de plus classifié : la classe d'un véhicule indique s'il s'agit ou non d'un train, et est un donc un élément de $\mathbb{B}$. Le tableau ci-dessous présente les six enregistrements de cette base de donnée (la dernière colonne donne la classe). Pour chaque véhicule on indique l'abréviation qui sera utilisée pour le représenter par la suite.

véhicule	Moteur	Rails	Souterrain	Vitesse	train ?
(a) A380	✓	✗	✗	✓	✗
(t) TGV	✓	✓	✗	✓	✓
(m) métro	✓	✓	✓	✗	✓
(w) wagonnet ♣	✗	✓	✓	✗	✗
(d) draisine ♡	✗	✓	✗	✗	✗
(r) tram	✓	✓	✗	✗	✓

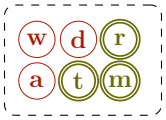
♡. image de wagonnet disponible sur https://upload.wikimedia.org/wikipedia/commons/d/d0/Wagonnet_de_mine_Regourdou.jpg?uselang=fr

♡. draisine à pompe, image disponible sur https://fr.wikipedia.org/wiki/Draisine#/media/Fichier:Pump_trolley.jpg

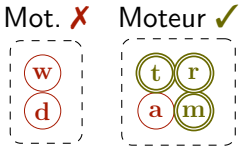
À partir de ce jeu de données d'apprentissage, on essaye de fabriquer un arbre de décision permettant de décider si un véhicule est ou non un train en fonction des quatre caractéristiques susmentionnées de ce véhicule.

Dans la suite, on représente chaque véhicule par son abréviation, et celle-ci apparaît :
- simplement encerclé de rouge si leur classe est ✗ (*i.e.* si ce n'est pas un train) ;
- doublement encerclé de vert si leur classe est ✓ (*i.e.* si c'est un train).

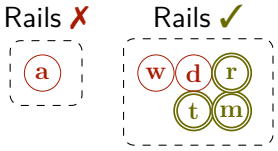
Comment découper ? Initialement, on n'a encore retenu aucune composante selon laquelle séparer les données, les données sont toutes rassemblées dans un seul groupe. Pourtant, puisque ce groupe n'est pas homogène en classe (*i.e.* il contient des véhicules classifiés comme étant des trains et d'autres classifiés comme n'étant pas des trains), il ne peut être représenté par une seule feuille dans l'arbre de décision.



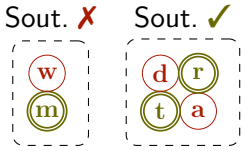
On cherche donc à séparer les données selon un prédicat. Puisque l'on s'est restreint à des prédicats portant sur une seule caractéristique, on a quatre prédicats possibles. On représente ci-dessous comment chacune des caractéristiques partitionne les données.



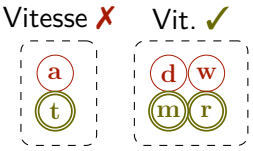
Partition selon Moteur



Partition selon Rails



Partition selon Souterrain



Partition selon Vitesse

Se pose donc la question de savoir laquelle de ces 4 partitions "range" le mieux les données au regard de leur classe. Afin de comparer les différentes partitions possibles, on aimerait utiliser une fonction numérique permettant l'évaluation du "dérangement" d'une partition, c'est l'objet de la suite de cette section.

Notation 2.13

Soit une variable aléatoire X à valeurs dans E un ensemble fini.
*On note $p_X : E \to [0,1]$ sa **loi de probabilité**, à savoir : $\forall x \in E, p_X(x) \overset{\text{déf}}{=} \mathbb{P}[X = x]$.*

Définition 2.14

Soit une variable aléatoire X à valeurs dans E un ensemble fini.
*On définit l'**entropie** (ou entropie de Shannon) de la variable aléatoire X comme étant :*

$$H(X) \overset{\text{déf}}{=} \sum_{x \in E} -p_X(x) \ln(p_X(x)) \clubsuit$$

Remarque 2.15

Il existe des définitions de l'entropie où $\ln$ est remplacé par $\log_b$ ou $b \in \mathbb{N} \setminus \{0,1\}$. Comme les différents logarithmes sont égaux à une constante multiplicative près♣, les différentes entropies que l'on peut définir sont égales à une constante près, ce qui n'a aucune influence par la suite.

Remarque 2.16

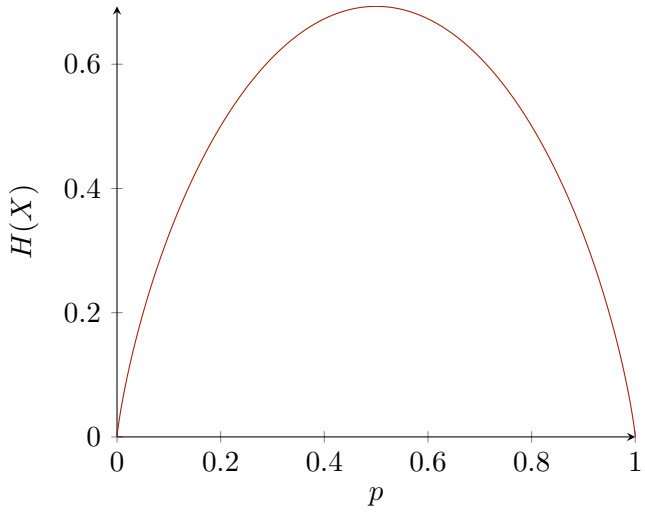
L'entropie de X est nulle dès lors qu'il existe $x_0 \in E$ tel que $p_X(x_0) = 1$. En effet pour les $x \in E$ différents de x_0, $p_X(x) = 0$, donc la somme est réduite à $-p_X(x_0)\ln(p_X(x_0)) = -1\ln(1) = 0$.

Remarque 2.17

L'entropie d'une variable aléatoire ne dépend pas des valeurs prises par cette variable mais seulement de la forme de sa distribution, c'est-à-dire sa distribution vue comme un multi-ensemble fini de réels sommant à 1.

Exemple 2.18

Considérons une variable aléatoire X représentant le résultat d'une expérience de pile ou face avec une pièce biaisée de sorte que $\mathbb{P}[\text{pile}] = p \in [0,1]$ et donc $\mathbb{P}[\text{face}] = 1 - p$. On représente ci-dessous l'entropie de la variable aléatoire X en fonction de p.



Proposition 2.19

Avec les notations précédentes, et en notant $n = \text{card}\{x \in E \,|\, \mathbb{P}[X = x] \neq 0\}$, $H(X) \leqslant \ln(n)$ *et cette borne est atteinte si* X *est uniforme sur ses images.*

Démonstration : On rappelle quelques inégalités qui reposent sur la concavité de $\ln$.

Lemme 2.20 (*Inégalité de Jensen*)

Si $g : \mathbb{R} \to \mathbb{R}$ *est une fonction concave, alors*

$$\forall n \in \mathbb{N}^*, \forall (x_i)_{i\in[\![1,n]\!]} \in \mathbb{R}^n, \forall (\alpha_i)_{i\in[\![1,n]\!]} \in [0,1]^n, \sum_{i=1}^{n} \alpha_i = 1 \Rightarrow g\left(\sum_{i=1}^{n} \alpha_i\, x_i\right) \geqslant \sum_{i=1}^{n} \alpha_i\, g(x_i)$$

♣. On prolonge $x \mapsto x\ln(x)$ par continuité comme valant 0 en 0, ainsi s'il existe $x_0 \in E$ tel que $p_X(x_0) = 0$, on considère que le terme $p_X(x_0)\ln(p_X(x_0))$ est nul.

♣. Rappel : $\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$ car $a^{\log_a(b)\times\log_b(x)} = (a^{\log_a(b)})^{\log_b(x)} = b^{\log_b(x)} = x$ soit $\log_a(b) \times \log_b(x) = \log_a(x)$.

Lemme 2.21 (*Inégalité de Gibbs*)

Soit $n \in \mathbb{N}^*$. *Soit* $(p_i, q_i)_{i\in[\![1,n]\!]} \in (]0,1] \times [0,1])^n$ *telle que* $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i = 1$.
Alors $-\sum_{i=1}^{n} p_i \ln(p_i) \leqslant -\sum_{i=1}^{n} p_i \ln(q_i)$.

Démonstration : En appliquant l'inégalité de Jensen à $g = \ln$, $x_i = \frac{q_i}{p_i}$ et $\alpha_i = p_i$:

$$\sum_{i=1}^{n} p_i \ln\left(\frac{q_i}{p_i}\right) \leqslant \ln\left(\sum_{i=1}^{n} p_i \frac{q_i}{p_i}\right) = \ln\left(\sum_{i=1}^{n} q_i\right) = \ln(1) = 0.$$

Or $\sum_{i=1}^{n} p_i \ln\left(\frac{q_i}{p_i}\right) = \sum_{i=1}^{n} p_i(\ln(q_i) - \ln(p_i))$, donc $\sum_{i=1}^{n} p_i \ln(q_i) - \sum_{i=1}^{n} p_i \ln(p_i) \leqslant 0$.
Finalement $-\sum_{i=1}^{n} p_i \ln(p_i) \leqslant -\sum_{i=1}^{n} p_i \ln(q_i)$ □

On pose $F = \{x \in E \mid \mathbb{P}[X = x] \neq 0\}$, ainsi $H(X) = -\sum_{x\in F} p_X(x) \ln(p_X(x))$. De plus, comme seule la forme de la distribution de X importe, on peut identifier F avec $[\![1, n]\!]$ où $n = \mathrm{card}(F)$ ($n \neq 0$ car $F \neq \emptyset$ par définition d'une probabilité).

Ainsi en notant $p_i = p_X(i)$ pour tout $i \in [\![1, n]\!]$, on a $H(X) = -\sum_{i=1}^{n} p_i \ln(p_i)$.

En posant de plus $q_i = \frac{1}{n}$ pour tout $i \in [\![1, n]\!]$, l'inégalité de Gibbs donne

$$H(X) \leqslant -\sum_{i=1}^{n} p_i \ln(\underbrace{q_i}_{=1/n}) = -\ln\left(\frac{1}{n}\right) \underbrace{\sum_{i=1}^{n} p_i}_{=1} = \ln(n)$$

De plus on remarque que si $Y \simeq \mathcal{U}(F) = [\![1, n]\!]$, alors $H(Y) = -\sum_{i=1}^{n} \frac{1}{n} \ln\left(\frac{1}{n}\right) = \ln(n)$. □

Définition 2.22

Soient S *et* $\mathcal{C}$ *deux ensembles finis. Soit* f *une fonction de classification de* S *dans* $\mathcal{C}$.
*L'**entropie de** S **relativement à** f, notée $H(S)$, est l'entropie de la variable aléatoire donnant la classe d'un élément choisi uniformément au hasard dans l'ensemble S, autrement dit c'est l'entropie de $f(Y)$ où $Y \simeq \mathcal{U}(S)$*[♣].

$$H(S) \overset{\text{déf}}{=} H(f(Y)) = \sum_{c\in\mathcal{C}} -\frac{\mathrm{card}(f^{-1}(c))}{\mathrm{card}(S)} \ln\left(\frac{\mathrm{card}(f^{-1}(c))}{\mathrm{card}(S)}\right)$$

Exercice de cours 2.23

Calculer l'entropie d'un ensemble dans lequel 4 éléments sont de classe A, 5 éléments de classe B et 1 élément de classe C.

Exercice de cours 2.24

Justifier que l'entropie d'un ensemble homogène (un ensemble dans lequel tous les éléments ont même classe) est nulle. Justifier que l'entropie d'un ensemble est maximisée lorsque chaque classe de $\mathcal{C}$ est représentée en même quantité.

♣. Y est une variable aléatoire de loi uniforme sur S.

Définition 2.25

Soit S un jeu de données classifiées♣*. Soit $\{S_1, S_2, \ldots, S_n\}$ une partition de l'ensemble S. L'**entropie de cette partition** est définie comme la moyenne des entropies des parties, pondérées proportionnellement à leur cardinal.*

$$H(\{S_1, S_2, \ldots, S_n\}) \overset{\text{déf}}{=} \sum_{i=1}^{n} \frac{\operatorname{card}(S_i)}{\operatorname{card} S} H(S_i)$$

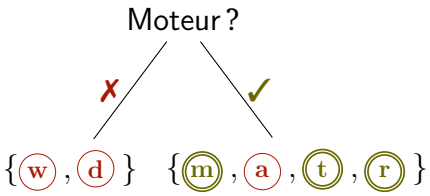
Exercice de cours 2.26

Calculer l'entropie d'une partition dont :
- le premier ensemble est celui décrit dans l'exercice de cours 2.23 ;
- le second est un ensemble contenant 3 éléments de classe A, 3 éléments de classe B ;
- le troisième est un ensemble contenant 1 élément de classe A.

Retour à l'exemple. Revenons donc à l'exemple considéré précédemment et calculons les entropies des différentes partitions possibles afin de savoir laquelle "range" le mieux au regard de leur classe (*i.e.* diminue le plus l'entropie).

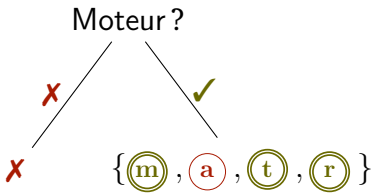
- Moteur : $H\left(\{\{\text{w},\text{d}\},\{\text{m},\text{a},\text{t},\text{r}\}\right) = \frac{2}{6}\left(-\frac{2}{2}\ln\frac{2}{2} - \frac{0}{2}\ln\frac{0}{2}\right) + \frac{4}{6}\left(-\frac{1}{4}\ln\frac{1}{4} - \frac{3}{4}\ln\frac{3}{4}\right) \simeq 0.37$
- Rails : $H\left(\{\{\text{a}\},\{\text{t},\text{m},\text{w},\text{d},\text{r}\}\}\right) = \frac{1}{6}\left(-\frac{1}{1}\ln\frac{1}{1} - \frac{0}{1}\ln\frac{0}{1}\right) + \frac{5}{6}\left(-\frac{3}{5}\ln\frac{3}{5} - \frac{2}{5}\ln\frac{2}{5}\right) \simeq 0.56$
- Sout. : $H\left(\{\{\text{w},\text{m}\},\{\text{t},\text{a},\text{d},\text{r}\}\}\right) = \frac{2}{6}\left(-\frac{1}{2}\ln\frac{1}{2} - \frac{1}{2}\ln\frac{1}{2}\right) + \frac{4}{6}\left(-\frac{2}{4}\ln\frac{2}{4} - \frac{2}{4}\ln\frac{2}{4}\right) \simeq 0.69$
- Vitesse : $H\left(\{\{\text{t},\text{a}\},\{\text{w},\text{m},\text{d},\text{r}\}\}\right) = \frac{2}{6}\left(-\frac{1}{2}\ln\frac{1}{2} - \frac{1}{2}\ln\frac{1}{2}\right) + \frac{4}{6}\left(-\frac{2}{4}\ln\frac{2}{4} - \frac{2}{4}\ln\frac{2}{4}\right) \simeq 0.69$

On peut remarquer que les critères Souterrain et Vitesse ne rangent pas du tout les éléments au regard de leur classe puisque l'entropie de la partition qui en résulte est la même que l'entropie de la partition initiale. En effet $H\left(\{\{\text{a},\text{t},\text{m},\text{w},\text{d},\text{r}\}\}\right) = -\frac{3}{6}\ln\frac{3}{6} - \frac{3}{6}\ln\frac{3}{6} \simeq 0.69$.
On remarque surtout que le critère qui offre une partition de plus petite entropie est Moteur. On choisit alors que ce critère étiquettera la racine de l'arbre de décision, et on répartit les données dans les sous-arbres gauche et droit selon ce critère. On obtient alors le *proto*-arbre de décision suivant.



On remarque que la partie $\{\text{w},\text{d}\}$ est uniforme en classe, et qu'il n'est donc pas nécessaire de la subdiviser davantage. On peut alors indiquer dans l'arbre de décision la valeur ✗ (*i.e.* n'est pas un train) donnant la classe de chacun des éléments se trouvant dans cette partie. Ce qui donne le proto-arbre de décision suivant.

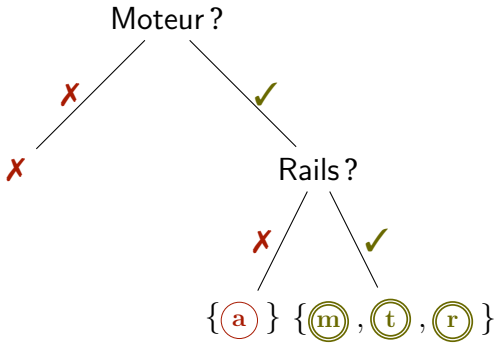
♣. La fait que l'ensemble soit classifié suppose qu'il existe un ensemble fini de classes $\mathcal{C}$ et une fonction de classification f de S dans $\mathcal{C}$. L'entropie de chaque partie S_i est implicitement définie pour cette classification.



Au contraire, la partie {m, a, t, r} n'est pas homogène en classe. Autrement dit il n'est pas encore possible de conclure pour les véhicules avec moteur. On cherche alors à nouveau comment partitionner au mieux ces données. On calcule donc les entropies pour les partitions associées aux trois critères restants (le critère Moteur est exclu car il ne séparerait pas ces données).

- Rails : $H\left(\{\{\text{a}\},\{\text{m},\text{t},\text{r}\}\}\right) = \frac{1}{4}\left(-\frac{1}{1}\ln\frac{1}{1} - \frac{0}{1}\ln\frac{0}{1}\right) + \frac{3}{4}\left(\frac{3}{3}\ln\frac{3}{3} - \frac{0}{3}\ln\frac{0}{3}\right) = 0$
- Souterrain : $H\left(\{\{\text{m}\},\{\text{a},\text{t},\text{r}\}\}\right) = \frac{1}{4}\left(-\frac{1}{1}\ln\frac{1}{1} - \frac{0}{1}\ln\frac{0}{1}\right) + \frac{3}{4}\left(-\frac{2}{3}\ln\frac{2}{3} - \frac{1}{3}\ln\frac{1}{3}\right) \simeq 0.47$
- Vitesse : $H\left(\{\{\text{a},\text{t}\},\{\text{m},\text{r}\}\}\right) = \frac{2}{4}\left(-\frac{1}{2}\ln\frac{1}{2} - \frac{1}{2}\ln\frac{1}{2}\right) + \frac{2}{4}\left(-\frac{2}{2}\ln\frac{2}{2} - \frac{0}{2}\ln\frac{0}{2}\right) \simeq 0.34$

On minimise ici l'entropie en choisissant comme critère Rails, et obtient alors le proto-arbre ci-dessous.



On remarque que ce proto-arbre correspond à une partition dont toutes les parties sont homogènes en classe. L'algorithme est alors terminé, et en remplaçant chaque partie par sa classe on obtient l'arbre de décision (complet) donné en figure 7.

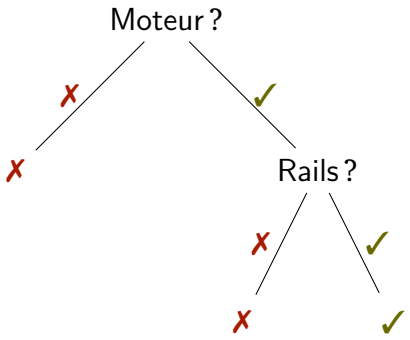


FIGURE 7 – Arbre de décision construit par l'algorithme ID3 sur l'exemple

Maintenant que nous avons fabriqué un classifieur, il est possible, et très rapide de l'évaluer sur des entrées différentes de celles des données d'apprentissage. On apprend alors que :

- un bus (Moteur : ✓, Rails : ✗, Souterrain : ✗, Vitesse : ✗) n'est pas un train,
- un TER (Moteur : ✓, Rails : ✓, Souterrain : ✗, Vitesse : ✗) est un train,

- un cheval (Moteur : ✗, Rails : ✗, Souterrain : ✗, Vitesse : ✗) n'est pas un train,
- un ascenseur spatial ♣ (Moteur : ✓, Rails : ✓, Souterrain : ✗, Vitesse : ✗) est un train.

Synthèse des données. Par opposition à l'algorithme k-NN, l'algorithme ID3 extrait de l'information des données d'apprentissage et structure cette information sous une forme arborescente. Par exemple ici on a découvert que la vitesse et le fait de rouler en souterrain ne semblent jouer aucun rôle dans la définition de ce qu'est un train.

Différence avec les arbres k-dimensionnels. L'arbre construit ici par la phase d'apprentissage n'est en rien comparable aux arbres k-dimensionnels. En effet les arbres k-dimensionnels servaient à structurer les données d'apprentissages, qui étaient d'ailleurs toutes enregistrées dans la structure. Ici l'arbre de décision obtenu ne contient pas les données initiales d'apprentissage, mais en quelque sorte une synthèse de ce qui apparaît utile pour classifier d'après ce qu'on a observé dans ce jeu d'apprentissage. De plus, dans un arbre k-dimensionnel, les sous-arbres gauches et droites ont la même "sous-structure" : ils branchent sur la même suite de composantes $1, 2, 3, \ldots, n, 1, 2, \ldots$. Ici les sous-arbres gauche et droit n'ont pas nécessairement les mêmes critères de découpe. Enfin une composante qui apparaît dans un nœud n'apparaît plus dans aucun des nœuds de ses fils.

Remarque 2.27

> L'algorithme ID3 est un algorithme glouton au sens où, à chaque étape, le choix du critère de décision est fait pour minimiser l'entropie localement : on choisit en fonction de l'entropie de la partition en deux induite par ce seul critère, et non en fonction de l'entropie de la partition finalement obtenue. Ces choix n'assurent pas la décroissance globale d'entropie la plus rapide.

Exercice de cours 2.28

> On considère le jeu de données classifiées ci-contre. Les données sont représentées par 3 coordonnées booléennes : x, y et z et ces données sont classifiées au moyen de 3 classes : A, B, C.
> Exécuter l'algorithme ID3 pour fabriquer un arbre de décision pour ce jeu de données.

x	y	z	Classe
F	F	F	A
F	F	V	B
F	V	F	B
F	V	V	C
V	F	F	B
V	F	V	C
V	V	F	C
V	V	V	C

Exercice de cours 2.29

> Soit $n \in \mathbb{N}^\star$, proposer un jeu de données de $\mathbb{B}^n$, classifiées dans $\mathbb{B}$ (donc deux classes seulement), telle que tout arbre de décision contient nécessairement $2^{n+1} - 1$ nœuds.

3 Apprentissage non supervisé

Dans cette section on s'intéresse au problème de l'apprentissage **non supervisé** qui s'oppose à l'apprentissage supervisé en ce sens qu'il opère sur des données **non classifiées**. Il s'agit alors de regrouper les données en classes d'éléments *semblables*. On suppose pour ce faire que les données sont des éléments d'un espace muni d'une distance d et que celle-ci représente bien la similarité des données.

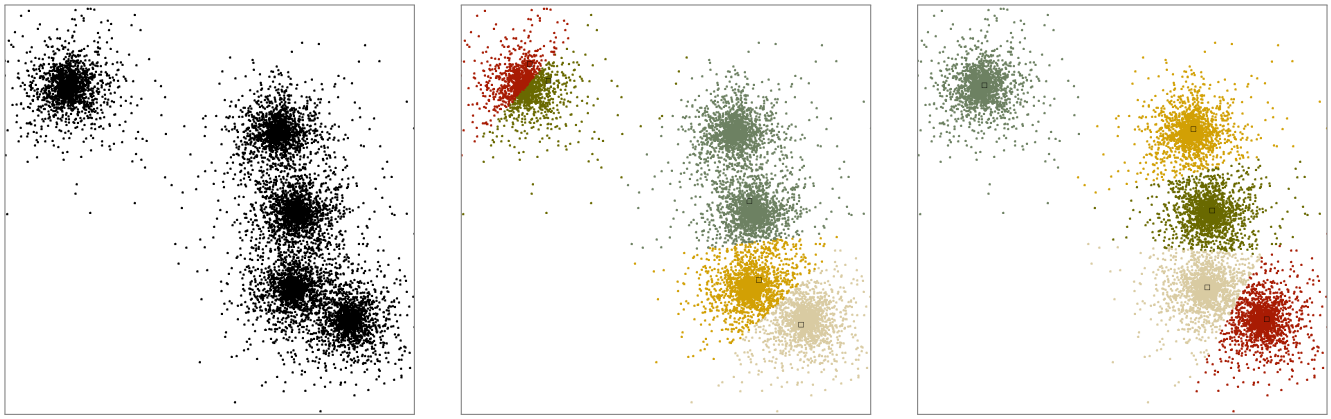
♣. https://en.wikipedia.org/wiki/Space_elevator

Dans la suite on s'intéresse à deux algorithmes qui fournissent une partition des données d'un jeu de données non classifiées. Le premier opère sur des données dans $\mathbb{R}^2$ muni de la distance euclidienne et prend en paramètre le nombre de parties souhaité dans la partition. Le second est un algorithme générique qui opère sur des données dans un espace métrique quelconque, et ne demande pas de connaître a priori le nombre de parties souhaité.

3.1 Algorithmes des k moyennes

Dans cette section on travaille sur des données $\mathcal{D} \subseteq \mathbb{R}^p$ et on suppose que la similarité entre deux données est mesurée par la distance euclidienne entre les deux vecteurs correspondants. Cette hypothèse, si elle paraît assez naturelle, requiert en réalité de soigner le jeu de données considéré. En effet cette mesure est sensible à un changement d'échelle sur une composante (dû par exemple au choix d'exprimer une longueur en centimètres ou en mètres), ce qui peut donner un poids plus important à une composante, ou au contraire un poids moins important. De plus cette mesure est sensible aux données parasites. En effet deux vecteurs qui coïncident sur les composantes pertinentes mais très différents sur une composante non pertinente sont perçus comme très différents à travers la distance euclidienne, à tort.

Illustration. Afin de se représenter le problème, on donne ci-dessous à gauche un jeu de données de $\mathbb{R}^2$ que l'on cherche à partitionner en 5 groupes de points proches. On donne deux exemples de 5 partitions de ce jeu de données, l'une qui correspond apparemment à ce que l'on attend et l'autre non.



(a) Données à regrouper (b) Données "mal" regroupées (c) Données "bien" regroupées

Partition par k centres. Dans l'algorithme des k moyennes, le nombre de groupes à trouver est fourni à l'algorithme. Nommons le k. On cherche alors une k-partition de $\mathcal{D} \subseteq \mathbb{R}^p$ parmi celles représentables par des **centres**. Pour une famille de k points qu'on appelle **centres**, on peut répartir les points de $\mathcal{D}$ en les rattachant au centre le plus proche, ce qui donne en général une k-partition♣. Formellement, pour $(m_i)_{i\in[\![1,k]\!]} \in (\mathbb{R}^p)^k$ une famille de k centres et $i \in [\![1,k]\!]$, on note $\mathcal{C}_i^m$ l'ensemble des vecteurs du jeu de données dont le centre le plus proche est celui d'indice i. Ce qui s'écrit comme suit : ♡

$$\mathcal{C}_i^m = \left\{ x \in \mathcal{D} \;\middle|\; \operatorname*{arg\,min}_{j\in[\![1,k]\!]} \|x - m_j\|_2 = i \right\}.$$

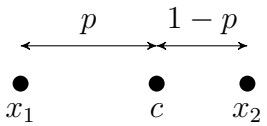
♣. il peut arriver qu'un centre ne soit le plus proche d'aucune donnée, ce qui fait qu'il correspond à une partie vide…

♡. Bien que $\arg\min$ désigne classiquement l'ensemble des points réalisant le minimum, on l'utilise ici pour désigner, par convention, le plus petit de ces minimiseurs, sachant que le minimum existe puisque l'ensemble $[\![1,k]\!]$ est fini.

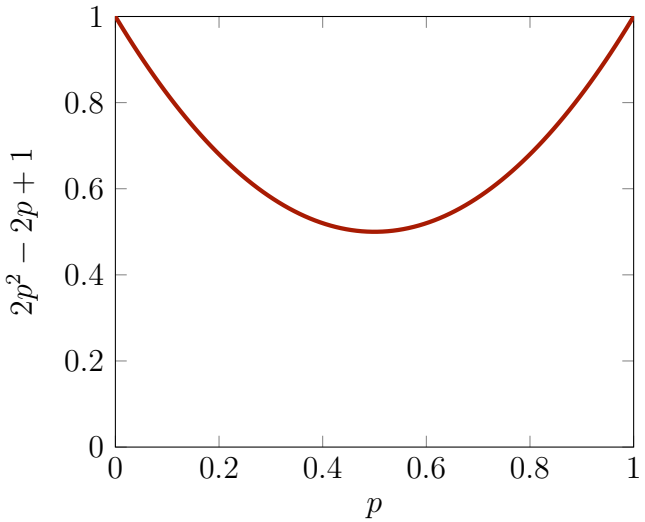
On évalue alors les partitions représentées par des centres à l'aune du "coût global de raccordement aux centres", ce qui revient à considérer la fonction suivante.

$$F(m) = \sum_{i=1}^{k} f(m_i, C_i^m) \quad \text{où} \quad f = \begin{pmatrix} \mathbb{R}^p \times \mathscr{P}(\mathbb{R}^p) & \to & \mathbb{R}^+ \\ (c, P) & \mapsto & \sum_{x \in P} \|c - x\|_2^2 \end{pmatrix}.$$

Où l'on justifie l'utilisation de $\|\cdot\|_2^2$ plutôt que $\|\cdot\|_2$. On souhaite que les centres soient proches des points qu'ils représentent, ce qui motive l'utilisation de la norme $\|c - x\|_2$. La mise au carré est justifiée par le caractère strictement convexe de la fonction $x \mapsto x^2$ qui apporte l'unicité des minimiseurs. En effet considérons l'exemple ci-dessous, où le jeu de données $\mathcal{D} \subseteq \mathbb{R}^1$ est constitué de deux données : $x_1 = (0)$ et $x_2 = (1)$.



Si on cherche à minimiser la fonction $c \mapsto \|c - x_1\|_2 + \|c - x_2\|_2$, on a une infinité de minimiseurs : tous les points de l'intervalle $[x_1, x_2]$ réalisent le minimum, à savoir $\|x_1 - x_2\|_2$. Au contraire si on cherche à minimiser la fonction $c \mapsto \|c - x_1\|_2^2 + \|c - x_2\|_2^2$, on a un seul minimiseur. En effet cette minimisation revient à minimiser la fonction $p \mapsto 2p^2 - 2p + 1$ (dont le graphe sur $[0, 1]$ est donné ci-dessous), qui admet un unique minimum en 0.5.



Définition 3.1

*Soit $P \subseteq \mathbb{R}^p$ tel que $n = \mathrm{card}(P) \in \mathbb{N}^*$. Le **barycentre** de P est alors $\mathrm{bar}(P) = \frac{1}{n} \sum_{x \in P} x$.*

Exercice de cours 3.2

Soit $P \subseteq \mathbb{R}^p$ tel que $n = \mathrm{card}(P) \in \mathbb{N}^*$.
Montrer que le barycentre de P est l'unique point $z \in R^p$ vérifiant $\sum_{x \in P}(x - z) = 0$.

Proposition 3.3

Soit $P \subseteq \mathbb{R}^p$ non vide. Le barycentre de P est l'unique minimiseur de $x \mapsto f(x, P)$.

Démonstration : Soit $y \in \mathbb{R}^p$. On note $g = \mathrm{bar}(P)$, on a alors :

$$\begin{aligned} f(y,P) &= \sum_{x\in P} \|y-x\|_2^2 \\ &= \sum_{x\in P} \|(y-g)+(g-x)\|_2^2 \\ &= \sum_{x\in P} \left(\|y-g\|_2^2 + \|g-x\|_2^2 + 2\langle y-g, g-x\rangle\right) \\ &= \mathrm{card}(P)\|y-g\|_2^2 + \underbrace{\sum_{x\in P}\|g-x\|_2^2}_{=f(g,P)} + 2\,\langle y-g, \underbrace{(\sum_{x\in P}(g-x))}_{=0}\rangle \\ &= f(g,P) + \underbrace{\mathrm{card}(P)\|y-g\|_2^2}_{\geqslant 0} \end{aligned}$$

On a donc $\forall y \in \mathbb{R}^p$, $f(y,P) \geqslant f(g,P)$, donc g est bien un minimiseur. De plus s'il existe g' un autre minimiseur, on a $f(g,P) = f(g',P) = f(g,P) + \mathrm{card}(P)\|g'-g\|_2^2$ donc $\|g'-g\|_2^2 = 0$ et $g' = g$. D'où l'unicité. □

La dernière propriété justifie qu'on décide, à chaque fois que l'on a une partition associée à des centres, de déplacer chaque centre au barycentre de sa partie afin de réduire F. Par unicité du minimum de $f(\bullet, P)$ pour toute partie P, si l'un des centres, disons m_i est déplacé par cette opération, alors F est strictement réduite par cette opération (car $f(m_i', C_i^m) < f(m_i, C_i^m)$).
Ainsi $F(m)$ admet un minimum *local* lorsque chacun des vecteurs m_i est le barycentre de sa partie, *i.e.* de C_i^m.

Exercice de cours 3.4

> Donner un exemple de partition par des centres où le déplacement des centres aux barycentre modifie la partition associée aux centres et une où, au contraire, la partition associée n'est pas modifiée par ce déplacement.

L'algorithme consiste alors à choisir d'abord k centres (au hasard dans le jeu de données pour commencer), de regarder la partition associée, puis changer les centres pour réduire les $f(m_i, C_i^m)$. Si ce déplacement donne de nouveaux centres, on a une nouvelle♣ partition associée, et passer à cette partition réduit la somme des coûts de raccordement puisqu'un point ne change de partie que si cela réduit sont coût de raccordement. Cette nouvelle partition a alors de nouveaux barycentres. On alterne donc les calculs de partition associée à des centres, et les calculs de barycentre des parties d'une partition jusqu'à atteindre une configuration stable.

Algorithme 2 : Algorithme des k moyennes

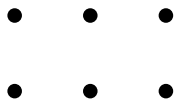
Entrée : Un jeu de données $\mathcal{D}$, un paramètre k
Sortie : Une partition de $\mathcal{D}$ en k groupes qui minimise localement la somme des carrés des distances de chaque point de $\mathcal{D}$ au barycentre de son groupe

```
(m_i)_{i∈⟦1,k⟧} ← k points parmi les données 𝒟 ;
Stable ← faux ;
tant que ¬Stable faire
    C ← (C_i^m)_{i∈⟦1,k⟧} ;
    m' ← (barycentre(C_i))_{i∈⟦1,k⟧} ;
    Stable ← m ≟ m' ;
    m ← m' ;
retourner C
```

♣. potentiellement nouvelle, *Cf.* exercice de cours 3.4

Exercice de cours 3.5

Donner l'exécution de l'algorithme des 2 moyennes pour l'instance suivante.



Les points se situent aux coordonnées $\{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1)\}$. On choisira comme centres initiaux les points $(0,0)$ et $(2,1)$.

Exercice de cours 3.6

Donner un exemple de jeu de données pour lequel le choix initial des centres influe sur le minimum local atteint.

Exercice de cours 3.7

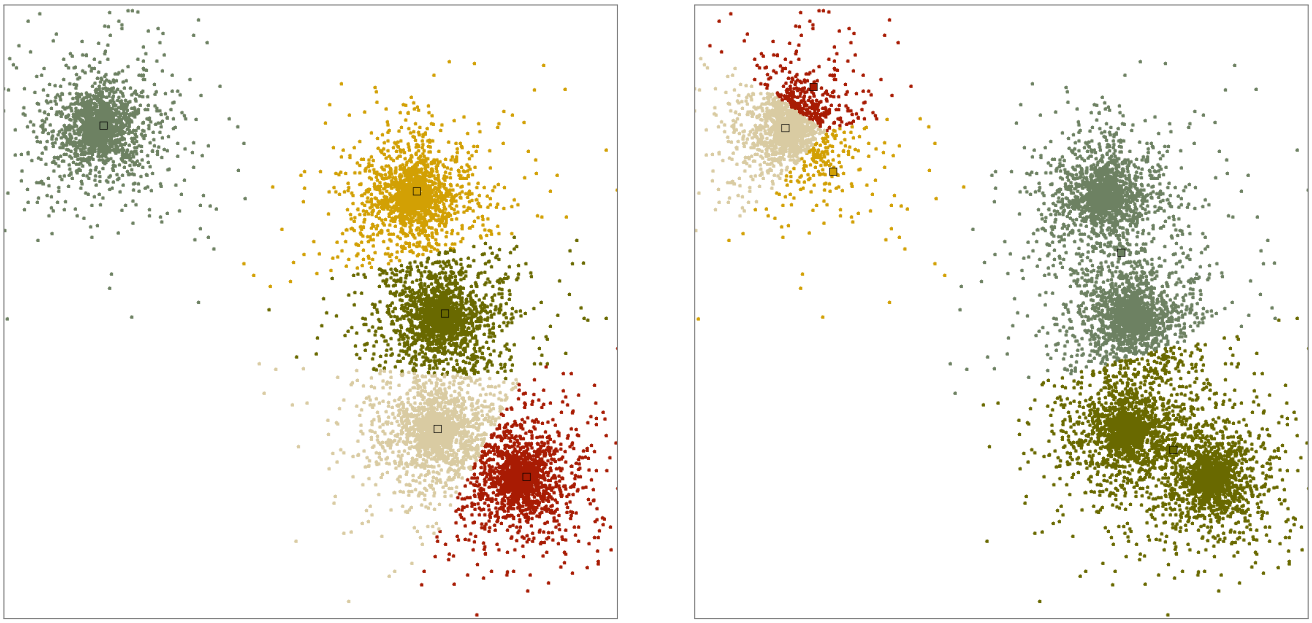
Donner un exemple de jeu de données, pour lequel le nombre d'itérations de l'algorithme des 3-moyennes est linéaire en le nombre de points du jeu de données.

Arrêt de l'algorithme. On peut montrer que l'algorithme termine♣. Pour cela on peut considérer le variant $F(m)$. En effet, on a déjà dit que le coût de raccordement aux centres est réduit à chaque étape, à moins qu'aucun centre ne soit déplacé, auquel cas l'algorithme s'arrête. S'il y avait une infinité de tours de boucle, on aurait alors une suite infinie strictement décroissante et positive. Cela n'est a priori pas absurde si l'on considère que F est à valeurs réelles. Mais bien que ces valeurs soient réelles, seul un nombre fini de valeurs sont possibles, puisqu'il y a un nombre fini de k-partitions de $\mathscr{D}$ possibles♡, et donc un nombre fini de famille de k centres obtenus comme barycentres des k parties d'une telle partition. Ainsi il est absurde que $F(m)$ décrive une suite infinie strictement décroissante au cours de l'algorithme. L'algorithme termine donc.

Correction de l'algorithme. Cet algorithme peut converger vers un minimum local et non un minimum global de la fonction F.
Considérons par exemple les deux partitions obtenues ci-dessous, obtenues après convergence sur le même jeu de données depuis des familles de centres initiales différentes. Le regroupement de gauche a un score F de $\simeq 42$, le regroupement de droite a un score F de $\simeq 106$. Les centres utilisés pour chaque groupe sont indiqués au moyen d'un carré, coloré de la couleur du groupe.

♣. Ce n'est pas au programme

♡. Même si on considère les k partitions qui ne sont pas des partitions associées à des centres, elles sont dans $\mathscr{P}(\mathscr{P}(\mathscr{D}))$ qui est aussi fini.



(a) Minimum global atteint (b) Mininimum local atteint

Comportement de convergence. La figure ci-dessus illustre l'évolution de la valeur de F sur un même jeu de données mais pour des familles de centres initiales différentes. Dans le premier cas on a une convergence en 6 itérations vers un minimum local, dans le second cas on a une convergence en 29 itérations, vers un minimum global.

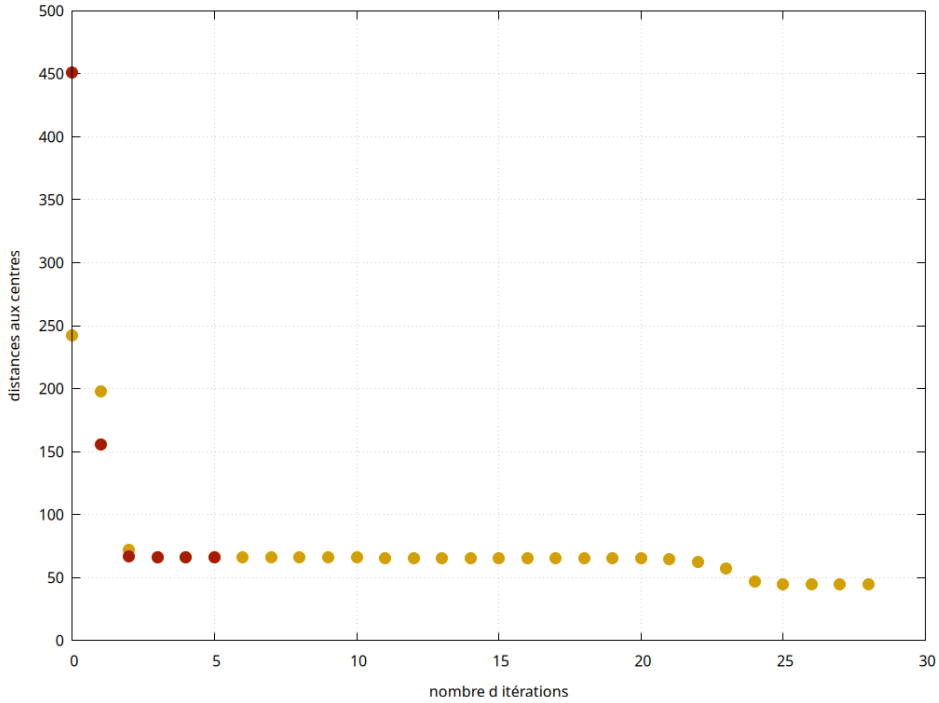


FIGURE 10 – Convergence de l'algorithme pour deux des familles de centres initiales différentes

3.2 Algorithme de classification hiérarchique ascendant

Dans cette section on s'intéresse à une famille d'algorithmes de regroupement. Cette fois le nombre de groupes n'est plus un paramètre de l'algorithme. L'algorithme fabrique une partition par fusions successives de parties, partant initialement de la partition en singletons. Il réalise des fusions entre

parties qui semblent similaires et ce tant que le critère d'arrêt n'est pas satisfait. Pour ce faire il est nécessaire de pouvoir mesurer la similarité (ou dissimilarité) entre deux parties. Pour une même distance sur les données il y a plusieurs manière de faire.

Mesure de dissimilarité. Si une distance d est définie sur l'ensemble de données $\mathcal{D}$, on définit différentes **mesures de dissimilarité** entre deux ensembles de données disjoints. Pour $A \subseteq \mathcal{D}$ et $B \subseteq \mathcal{D}$ tels que $A \cap B = \emptyset$, $A \neq \emptyset$ et $B \neq \emptyset$:

- le **saut minimum** entre A et B est $D_{\min}(A,B) = \min_{a\in A, b\in B} d(a,b)$;
- le **saut maximum** entre A et B est $D_{\max}(A,B) = \max_{a\in A, b\in B} d(a,b)$;
- le **lien moyen** entre A et B est $D_{\text{moy}}(A,B) = \frac{1}{\text{card}(A\times B)} \sum_{(a,b)\in A\times B} d(a,b)$.

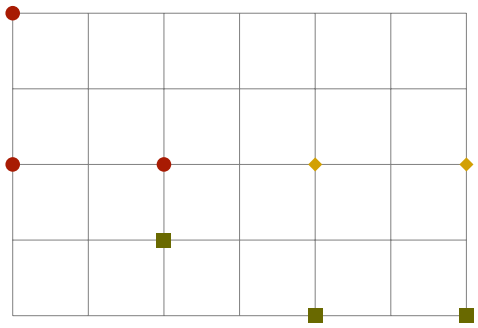
Exercice de cours 3.8

Pour l'exemple ci-contre, on travaille dans $\mathbb{R}^2$ avec d la distance euclidienne. On travaille avec la partition $\{A, B, C\}$ où :

- A est l'ensemble des points ● ;
- B est l'ensemble des points ◆ ;
- C est l'ensemble des points ■.

Donner alors :

- $D_{\min}(A,B)$, $D_{\min}(B,C)$ et $D_{\min}(A,C)$;
- $D_{\max}(A,B)$, $D_{\max}(B,C)$ et $D_{\max}(A,C)$;
- $D_{\text{moy}}(A,B)$, $D_{\text{moy}}(B,C)$ et $D_{\text{moy}}(A,C)$.



Dans la suite on note $D : \mathcal{P}(\mathcal{D}) \times \mathcal{P}(\mathcal{D}) \to \mathbb{R}^+$ la mesure de dissimilarité choisie. C'est cette mesure de dissmilarité qui va guider le choix des fusions : à chaque itération les deux parties ayant la plus petite dissimilarité sont fusionnées. L'algorithme est donc le suivant.

Algorithme 3 : Algorithme de classification hiérarchique ascendant (HAC)

Entrée : Un jeu de données $\mathcal{D}$
Sortie : Une partition de $\mathcal{D}$

```
P ← {{d} | d ∈ D} ;
n_c ← card(D) ;
tant que critère et n_c > 1 faire
    (A*, B*) ∈ arg min_{(A,B)∈P²} D(A, B) ;
    P ← (P \ {A*, B*}) ∪ {A* ∪ B*} ;
    n_c ← n_c − 1
retourner P
```

Au cours de l'algorithme ci-dessus, $\min_{(A,B)\in P^2} D(A,B)$ ne fait que croître. On enlève à chaque étape les parties réalisant ce min, mais on en ajoute une autre, leur union. La proposition ci-dessous justifie la croissance.

Proposition 3.9

Soit P une partition de $\mathcal{D}$.
Si $(A^\star, B^\star) \in \arg\min_{(A,B)\in P^2} D(A,B)$, alors $\forall C \in P,\ D(A^\star \cup B^\star, C) \geqslant D(A^\star, B^\star)$.

Exercice de cours 3.10

Prouver le résultat précédent dans le cas de la dissimilarité $D_{\min}$.

Le critère d'arrêt de l'algorithme peut par exemple être :

- la plus petite dissimilarité entre les classes est plus grande qu'un seuil ;

- le ratio de dissimilarité avec l'étape précédente dépasse un seuil ;
- le nombre de partitions est k.

Exercice de cours 3.11

Exécuter l'algorithme HAC, jusqu'à l'obtention de trois classes, puis deux classes, sur le jeu de données suivant. On utilisera la mesure de dissimilarité $D_{\min}$