L'apprentissage automatique (ou *machine learning*) est un domaine de l'intelligence artificielle qui consiste à utiliser des modèles capables de s'adapter et de s'améliorer avec le temps en analysant de grandes quantités de données. L'apprentissage peut être utilisé pour faire de la classification (identifier la classe d'un objet donné), de la régression (faire une prédiction numérique) ou du partitionnement (créer des groupes de données similaires) et est utilisé dans de nombreux domaines, allant de la reconnaissance de parole et d'image, la détection de spam ou la prédiction de tendances sur des réseaux sociaux.

L'apprentissage automatique se découpe principalement en deux types d'algorithmes : l'apprentissage supervisé et l'apprentissage non supervisé. Dans tous les cas, l'apprentissage se ramène à l'analyse d'un jeu de données qui seront modélisées par des éléments de $\mathbb{R}^d$, $d \in \mathbb{N}^*$ étant la dimension de l'espace des données. Il existe d'autres catégories d'apprentissage, comme l'apprentissage par renforcement, mais ces catégories ne sont pas au programme.

1 Apprentissage supervisé

L'apprentissage supervisé permet de faire principalement de la classification ou de la régression : en considérant une nouvelle donnée, on souhaite lui donner une étiquette pouvant correspondre à une classe (classification) ou une valeur numérique (régression). Un exemple concret pourrait être la création d'un algorithme qui prend en argument une photographie d'animal et prédit à quel animal elle correspond (parmi chat, chien, ...). Le caractère « supervisé » vient du fait qu'avant l'utilisation de l'algorithme, on dispose d'un ensemble de données **étiquetées** qui servent de modèle de référence pour les données non connues.

Formellement, on dispose de $(x_1, \dots, x_N) \in (\mathbb{R}^d)^N$ et $(y_1, \dots, y_N) \in \mathbb{R}^N$, les x_i correspondant aux **données** et les y_i aux **étiquettes**, et on cherche à construire une fonction $f : \mathbb{R}^d \to \mathbb{R}$ qui minimise l'erreur entre $f(x_i)$ et y_i (selon une mesure qui peut varier selon les problèmes). On notera x_{ij} la composante j du vecteur x_i. Selon le problème, on utilisera également le terme d'**attribut** : si a est le j-ème des d attributs, alors $a(x_i) = x_{ij}$.

1.1 K plus proches voisins

L'algorithme des K plus proches voisins, ou *KNN* (pour K *nearest neighbors*), est un algorithme de classification. Pour cet algorithme, les $y_i \in [\![1, m]\!]$ correspondent à des numéros de classes parmi m classes au total. Pour x_i une donnée étiquetée, on notera $c(x_i) = y_i$ la classe associée. Pour une donnée inconnue x, l'algorithme se présente sous la forme suivante :

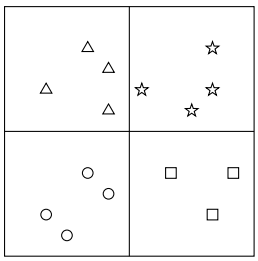
- identifier $z_1, z_2, \dots, z_K$ des éléments distincts dans $\{x_1, \dots, x_N\}$ minimisant $\delta(x, z_i)$, δ étant une fonction de distance ;
- renvoyer l'élément majoritaire parmi $c(z_1), c(z_2), \dots, c(z_K)$.

Ici, la fonction δ est une fonction de distance dans $\mathbb{R}^d$. On peut envisager différentes fonctions et obtenir des résultats différents pour chacune. En voici quelques exemples :

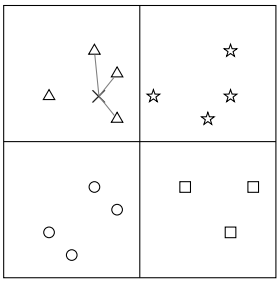
- distance euclidienne : $\delta(x, z_i) = \|x - z_i\|_2$;
- distance Manhattan : $\delta(x, z_i) = \|x - z_i\|_1$.

Exemple 1.1

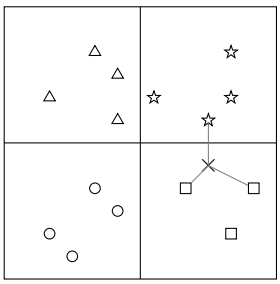
On souhaite classifier des données en quatre classes (ici modélisées par chaque quadrant). On dispose d'un jeu initial de 15 données étiquetées (par des étoiles, cercles, carrés ou triangles) réparties de la manière suivante :



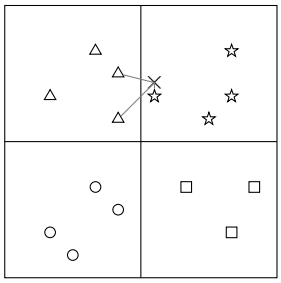
On suppose de plus que $K = 3$. On montre ici trois classifications possibles d'un point :



Le point est correctement classifié comme un triangle.



Le point est correctement classifié comme un carré.



Le point est incorrectement classifié comme un triangle.

Remarque 1.2

- Dans l'exemple précédent, on peut voir que l'algorithme peut se tromper dans la classification, même pour un point déjà étiqueté. Si on avait considéré l'étoile la plus proche du dernier point mal classé, le résultat aurait également été incorrect.
- Le choix de la valeur de K peut changer les résultats obtenus. Il n'y a pas de méthode générale, mais on peut remarquer les faits suivants : si K est trop petit, le choix peut être trop sensible au bruit statistique ; si $K = N$, alors le résultat dépend uniquement de la répartition des classes des données étiquetées. Le choix de K peut être fait empiriquement, en utilisant les données étiquetées elles-mêmes.

Exercice 1

On souhaite classifier des points de $\mathbb{R}^2$ par l'algorithme KNN. On suppose disposer d'un jeu de données étiquetées sous la forme de :

- `donnees`, de type `(float * float) array` contenant des couples de coordonnées des x_i ;
- `etiquettes`, de type `int array` contenant les numéros de classes y_i.

1. Écrire une fonction `delta : (float * float) -> (float * float) -> float` qui prend en arguments deux points x et z et renvoie $\delta(x, z)$. On utilisera la distance euclidienne.
2. Écrire une fonction `majoritaire : int -> (float * int) array -> int` qui prend en argument un entier K et un tableau de couples (δ, j) et renvoie l'élément j qui apparait le plus parmi les K premiers éléments.
3. En déduire une fonction :

   ```
   knn : int -> (float * float) array -> int array -> (float * float) -> int
   ```

   qui prend en argument un entier K, les tableaux `donnees` et `etiquettes` et un couple de flottants correspondant aux coordonnées du point à classifier et renvoie un entier correspondant à la classe prédite. On rappelle que `Array.sort compare tab` trie le tableau `tab` par ordre croissant.

L'algorithme KNN peut également servir pour faire de la régression :

- identifier $z_1, z_2, \ldots, z_K$ des éléments distincts dans $\{x_1, \ldots, x_N\}$ minimisant $\delta(x, z_i)$;
- renvoyer la moyenne $\frac{1}{K}\sum_{i=1}^{K} c(z_i)$ ou la moyenne pondérée $\dfrac{\sum_{i=1}^{K} \frac{c(z_i)}{\delta(x,z_i)}}{\sum_{i=1}^{K} \frac{1}{\delta(x,z_i)}}$.

Pour la moyenne pondérée, on utilise l'inverse de la distance pour donner plus de poids aux sommets les plus proches. On fait l'abus que si $x = z_j$, alors $f(x) = c(z_j)$, où f est la fonction calculée par cet algorithme.

1.1.1 Matrice de confusion

Une matrice de confusion est un moyen de quantifier la correction d'un algorithme de classification.

Définition 1.3

Soit $f : \mathbb{R}^d \to [\![1, m]\!]$ une fonction correspondant à un algorithme de classification et $c : \mathbb{R}^d \to [\![1, m]\!]$ la fonction exacte de classification. Pour un jeu de n données inconnues X et $i, j \in [\![1, m]\!]^2$, on note m_{ij} le cardinal de l'ensemble $\{x \in X \mid c(x) = i \text{ et } f(x) = j\}$. On appelle **matrice de confusion de** X la matrice $(m_{ij})_{i,j\in[\![1,m]\!]^2}$.

Exemple 1.4

On considère un jeu de 1000 points choisis aléatoirement dans l'ensemble du carré de l'exemple précédent auquel on applique l'algorithme KNN avec $K = 3$. On obtient la matrice de confusion suivante :

$$\begin{pmatrix} 224 & 13 & 0 & 0 \\ 1 & 248 & 0 & 0 \\ 26 & 3 & 191 & 20 \\ 0 & 40 & 0 & 234 \end{pmatrix}$$

en supposant que les numéros corrects de classe sont organisés comme : $\begin{array}{|c|c|}\hline 2 & 4 \\ \hline 1 & 3 \\ \hline \end{array}$.

On peut interpréter de cette matrice que les éléments de la classe 2 sont presque toujours correctement bien identifiés (une seule erreur sur 249), contrairement à ceux des classes 3 et 4. Le taux d'erreurs total est $\frac{13+1+26+3+20+40}{1000} = 10,3\%$.

Remarque 1.5

La matrice de confusion et le taux d'erreurs peuvent être un moyen de déterminer empiriquement la meilleure valeur de K et la distance à utiliser. En pratique, on sépare les données étiquetées en deux morceaux :

- un jeu de données utilisées pour l'algorithme KNN ;
- un jeu de données de test, utilisées uniquement pour déterminer la matrice de confusion et évaluer la pertinence des paramètres.

Cet ajustement peut se faire de manière itérative pendant une phase d'apprentissage.

1.1.2 Arbre k-d

Dans l'exercice d'implémentation de KNN, on a choisi de faire naïvement un tri des données pour déterminer les K plus proches voisins. La complexité de cette recherche est alors $\mathcal{O}(N(\log N + d))$, N étant le cardinal du jeu de données étiquetées et d la dimension (le facteur d vient du calcul de la distance). N étant en pratique très grand, cette approche n'est pas satisfaisante. Une approche utilisant l'algorithme de sélection rapide aléatoire (étudié dans un précédent chapitre) peut améliorer le calcul en $\mathcal{O}(N)$ en moyenne, mais cela reste trop grand. On propose dans cette partie d'étudier une structure de données adaptée à la recherche des K plus proches voisins dans un espace $\mathbb{R}^d$.

Définition 1.6

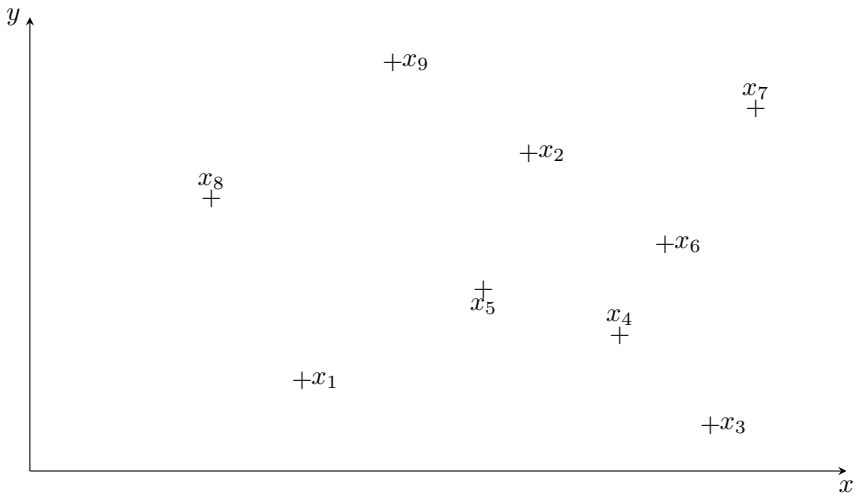
Étant donnés N points $(x_1, \ldots, x_N)$ de $\mathbb{R}^d$, un **arbre d-dimensionnel** (appelé couramment arbre k-d dans la littérature, le k faisant ici référence à la dimension) est un arbre binaire de taille N dont les nœuds sont des éléments x_i, vérifiant les propriétés suivantes :

- pour chaque nœud x de l'arbre, les points dans le fils gauche de x et dans le fils droit de x sont de part et d'autre d'un hyperplan passant par x ;

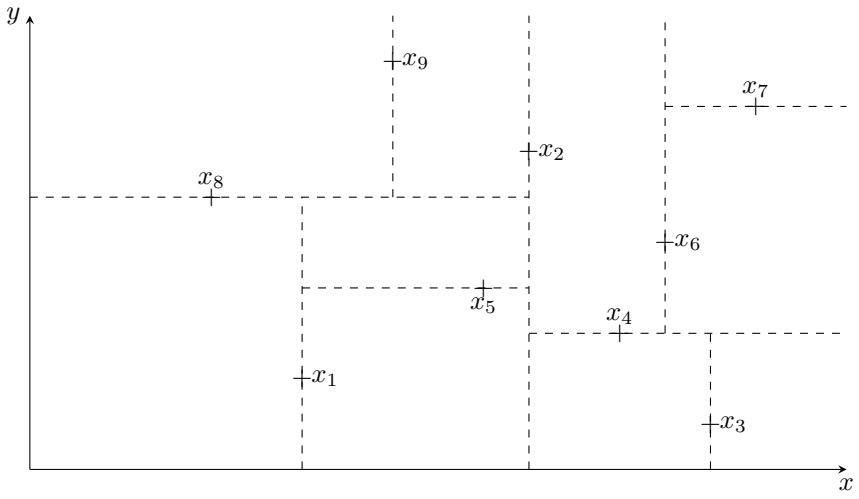
– si le nœud x est de profondeur congrue à p modulo d, alors l'hyperplan passant par x est normal au vecteur e_{p+1} (le $p+1$-ème vecteur de la base canonique de $\mathbb{R}^d$).

Exemple 1.7

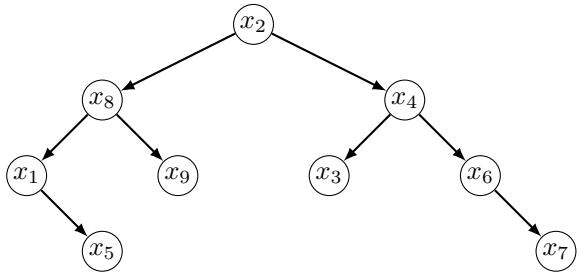
On suppose que $d = 2$ et on cherche à créer un arbre d-dimensionnel pour le jeu de données suivant :



On obtient alors le découpage spatial suivant, en alternant découpage vertical et découpage horizontal :



Ce qui donne l'arbre d-dimensionnel :



Remarque 1.8

Pour que l'arbre soit le plus équilibré possible, la racine de chaque sous-arbre est l'élément qui a la valeur médiane, parmi les points du sous-arbre, pour la dimension pour laquelle on effectue le découpage. Cette valeur médiane peut être déterminée avec un algorithme de sélection rapide par exemple.

En prenant en compte la recherche de médiane, la construction de l'arbre d-dimensionnel est en complexité $\mathcal{O}(N \log N)$ (en moyenne, mais on peut garantir dans le pire cas). Cependant, cette construction n'est exécutée qu'une seule fois.

La recherche des K plus proches voisins dans un arbre d-dimensionnel A se fait de la manière suivante :

Entrée : $x = (a_1, \ldots, a_d) \in \mathbb{R}^d$
Début algorithme
  $FP \leftarrow$ file de priorité vide.
  $\ell \leftarrow 0$.
  Fonction `Traiter(`z`)`
    $\delta_z \leftarrow \delta(x, z)$.
    $p_{\max} \leftarrow$ Priorité_max(FP).
    Si $\ell < K$ ou $\delta_z < p_{\max}$ **Alors**
      Inserer dans FP l'élément z avec priorité δ_z.
      $\ell \leftarrow \ell + 1$.
    Si $\ell > K$ **Alors**
      Extraire l'élément de priorité maximale dans FP.
      $\ell \leftarrow \ell - 1$.
  Fonction `Explore(`A, p`)`
    Si A non vide **Alors**
      `Traiter(`*racine de* A`)`.
      $j \leftarrow (p \mod d) + 1$.
      $b_j \leftarrow$ valeur de la j-ème composante de z.
      Si $a_j \leqslant b_j$ **Alors**
        $A_1 \leftarrow$ fils gauche de A.
        $A_2 \leftarrow$ fils droit de A.
      Sinon
        $A_1 \leftarrow$ fils droit de A.
        $A_2 \leftarrow$ fils gauche de A.
      `Explore(`A_1, $p + 1$`)`.
      $p_{\max} \leftarrow$ Priorité_max(FP).
      Si $\ell < K$ ou $p_{\max} > |a_j - b_j|$ **Alors**
        `Explore(`A_2, $p + 1$`)`.
  `Explore(`A, 0`)`.
  Renvoyer les éléments de FP.

Expliquée en français, l'idée est la suivante :

- On garde en mémoire une file de priorité contenant les plus proches voisins de x parmi les points étudiés, dans une limite de K au maximum.
- Chaque fois qu'on découvre un nouveau point a, on l'ajoute à la file de priorité si elle en contient moins que K, ou si ce nouveau point est plus proche de x que le voisin le plus loin. On extrait ce voisin le plus loin si nécessaire.
- On explore une partie de l'arbre d-dimensionnel selon l'idée suivante : pour chaque nœud a de l'arbre, on explore le fils gauche ou droit en premier selon où se trouve x par rapport à l'hyperplan de découpage en a. Après cette exploration, deux cas sont possibles :
  * soit on a déjà trouvé K voisins, et le plus loin d'entre eux est plus proche de x que ne l'est l'hyperplan de découpage, auquel cas on n'explore pas l'autre fils (car tous les points qui s'y trouvent seront trop loin) ;

* sinon, on explore l'autre fils.

On peut montrer que la complexité d'une recherche des K plus proches voisins dans un arbre d-dimensionnel de taille N est en moyenne en $\mathcal{O}(K(\log K + d) + d \log N)$ (même si elle peut atteindre $\mathcal{O}(N(\log N + d))$ dans le pire cas).

Exercice 2

On dispose de $N = 10$ points dans le plan, de coordonnées :

$$x_1(11, 0.5)\,;\ x_2(8.5, 6)\,;\ x_3(7, 8.5)\,;\ x_4(9, 4)\,;\ x_5(15, 1.5)$$

$$x_6(14.5, 8)\,;\ x_7(3.5, 7)\,;\ x_8(3, 2.5)\,;\ x_9(14, 5.5)\,;\ x_{10}(2, 5)$$

1. Représenter les points dans le plan et faire le découpage de construction de l'arbre 2-dimensionnel correspondant, puis dessiner cet arbre.
2. Détailler la recherche des 2 plus proches voisins de $x = (4, 5)$ et $y = (8, 1)$.

1.2 Arbres de décision

Définition 1.9

Un **arbre de décision** est un arbre binaire permettant de prendre une décision :

- les feuilles sont étiquetées par des numéros de classes ;
- les nœuds internes sont étiquetés par des questions pouvant avoir plusieurs réponses, chaque sous-arbre correspondant à une réponse possible.

Exemple 1.10

- Un algorithme de tri par comparaison peut être modélisé par un arbre de décision : chaque classe correspond à une permutation, et les questions correspondent à une comparaison entre deux éléments. Dans ce type d'arbre, les réponses aux questions sont binaires (« vrai » ou « faux »).
- Un arbre de décision peut être utilisé par exemple pour déterminer les questions à poser lors d'un démarchage téléphonique.
- Un arbre de décision peut être utilisé pour jouer au Mastermind, chaque « question » pouvant avoir 15 réponses possibles (pour le jeu classique avec une combinaison de taille 4).

L'utilisation d'un arbre de décision est plutôt naturelle et sa simplicité en fait un outil intéressant. Pour que cet outil soit efficace, il convient que la hauteur de cet arbre soit la plus petite possible.

Dans le contexte de l'apprentissage, on dispose d'un jeu de données étiquetées sous la forme de N valeurs $(x_1, \dots, x_N)$, chaque x_i étant un vecteur dans $\mathbb{R}^d$, et de N étiquettes $(y_1, \dots, y_N)$ à valeurs dans $[\![1, m]\!]$. Ici, les composantes des x_i, ou attributs, représentent chacune une question. Dans certains cas, un attribut ne peut prendre qu'un nombre fini de valeurs, auquel cas le nombre de sous-arbres peut correspondre au nombre de valeurs possibles. Si le nombre de valeurs possibles est très grand ou infini, on peut également envisager d'avoir des réponses sous formes d'intervalles, pour n'avoir qu'un nombre fini de fils dans l'arbre.

Une approche naïve consiste à « poser les questions dans l'ordre », c'est-à-dire faire un découpage selon la première composante, puis la deuxième composante, ... Cela peut cependant mener à un arbre très déséquilibré, par exemple avec le jeu de données $x_1 = (0, \dots, 0)$, $x_2 = (1, 0, \dots, 0)$, $x_3 = (1, 1, 0, \dots, 0)$, ..., $x_N = (1, \dots, 1)$, si $c(x_1) = c(x_2) = \dots = c(x_{N-1}) = 1$ et $c(x_N) = 2$. Pour éviter cela, on va plutôt chercher à mettre à une faible profondeur les composantes qui créent un découpage le plus équilibré possible.

1.2.1 Entropie

Pour un numéro de classe $j \in [\![1, m]\!]$, on note $C_j = \{x_i \mid y_i = j\}$ la j-ème classe parmi les données étiquetées.

Définition 1.11

On définit l'**entropie de Shannon** de l'ensemble E des données étiquetées par :

$$H(E) = -\sum_{j=1}^{m} \frac{|C_j|}{N} \log\left(\frac{|C_j|}{N}\right)$$

Remarque 1.12

Intuitivement, cette notion représente la quantité d'information contenue dans l'ensemble de données étiquetées. L'entropie est nulle lorsque tous les éléments sont dans la même classe (l'un des C_j est de cardinal N). Elle est maximale lorsque tous les éléments sont dans des classes distinctes. Elle vaut alors $\log N$.

Lors de la construction de l'arbre de décision, l'idée est alors de choisir la composante qui permet de gagner le maximum d'information.

Soit a un attribut dans $[\![1, d]\!]$, dont les valeurs possibles (éventuellement après regroupement en intervalles) sont $a_1, \ldots, a_p$. On note $E_{a=a_k} = \{x_i \in E \mid a(x_i) = a_k\}$

Définition 1.13

Soit $a \in [\![1, d]\!]$. On définit le **gain d'entropie selon** a comme :

$$G(E, a) = H(E) - \sum_{k=1}^{p} \frac{|E_{a=a_k}|}{N} H(E_{a=a_k})$$

1.2.2 Algorithme ID3

L'algorithme ID3, ou *Iterative Dichotomiser 3*, est un algorithme glouton de construction d'arbre de décision. Il se présente sous la forme suivante :

- si tous les éléments de E ont la même classe, créer une feuille correspondant à cette classe ;
- si $E = \emptyset$, créer une feuille correspondant à la classe majoritaire du nœud parent ;
- si tous les éléments ont les mêmes attributs, créer une feuille correspondant à la classe majoritaire ;
- sinon, soit a l'attribut qui maximise le gain $G(E, a)$. Construire un nœud a ayant pour fils les arbres construits récursivement sur les $E_{a=a_k}$, pour $k \in [\![1, p]\!]$.

Exercice 3

Deux joueurs de tennis sont en vacances dans un hôtel de luxe. Après avoir observé leur routine pendant deux semaines où les joueurs ont parfois joué ou non, le personnel de l'hôtel qui garantit un service impeccable souhaite faire une prévision pour les jours à venir, pour anticiper quels jours vont jouer les joueurs en fonction des conditions météorologiques.

Les jours de jeu en fonction de la météo sont les suivants :

Ciel	Température	Humidité	Vent	Ont-ils joué au tennis ?
Soleil	Chaud	Élevée	Faible	Non
Soleil	Chaud	Élevée	Fort	Non
Nuages	Chaud	Élevée	Faible	Oui
Pluie	Moyen	Élevée	Faible	Oui
Pluie	Frais	Normale	Faible	Oui
Pluie	Frais	Normale	Fort	Non
Nuages	Frais	Normale	Fort	Oui
Soleil	Moyen	Élevée	Faible	Non
Soleil	Frais	Normale	Faible	Oui
Pluie	Moyen	Normale	Faible	Oui
Soleil	Moyen	Normale	Fort	Oui
Nuages	Moyen	Élevée	Fort	Oui
Nuages	Chaud	Normale	Faible	Oui
Pluie	Moyen	Élevée	Fort	Non

1. Construire un arbre de décision selon l'algorithme ID3. On généralisera à des attributs pouvant prendre plus que deux valeurs.
2. Est-il prévu de jouer au tennis pour un jour Ensoleillé, frais, avec une humidité élevée et un vent faible selon ce modèle ?

Corrigé

1. Pour simplifier, on utilise les notations suivantes :
   - l'attribut « Ciel » sera nommé a_1, et les valeurs « Pluie », « Nuages » et « Soleil » vaudront respectivement 0, 1 et 2 ;
   - l'attribut « Température » sera nommé a_2 et les valeurs « Frais », « Moyen » et « Chaud » vaudront respectivement 0, 1 et 2 ;
   - l'attribut « Humidité » sera nommé a_3 et les valeurs « Normale » et « Élevée » vaudront 0 et 1 ;
   - l'attribut « Vent » sera nommé a_4 et les valeurs « Faible » et « Fort » vaudront 0 et 1.

   Pour la racine, on calcule :
   - $H(E) = -\frac{5}{14}\log\frac{5}{14} - \frac{9}{14}\log\frac{9}{14} \simeq 0,6518$;
   - pour l'attribut a_1 :
     * $H(E_{a_1=0}) = -\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5} \simeq 0,673$;
     * $H(E_{a_1=1}) = 0$;
     * $H(E_{a_1=2}) = H(E_{a_1=0}) \simeq 0,673$;
     * soit $G(E,a_1) = H(E) - \frac{5}{14}H(E_{a_1=0}) - \frac{5}{14}H(E_{a_1=2}) \simeq 0,171$.
   - pour l'attribut a_2 :
     * $H(E_{a_2=0}) = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log\frac{3}{4} \simeq 0,5623$;
     * $H(E_{a_2=1}) = -\frac{2}{6}\log\frac{2}{6} - \frac{4}{6}\log\frac{4}{6} \simeq 0,6365$;
     * $H(E_{a_2=2}) = \log 2 \simeq 0,6931$;
     * soit $G(E,a_2) = H(E) - \frac{4}{14}H(E_{a_2=0}) - \frac{6}{14}H(E_{a_2=1}) - \frac{4}{14}H(E_{a_2=2}) \simeq 0,0203$.
   - pour l'attribut a_3 :
     * $H(E_{a_3=0}) = -\frac{1}{7}\log\frac{1}{7} - \frac{6}{7}\log\frac{6}{7} \simeq 0,4101$;
     * $H(E_{a_3=1}) = -\frac{3}{7}\log\frac{3}{7} - \frac{4}{7}\log\frac{4}{7} \simeq 0,6829$;
     * soit $G(E,a_3) = H(E) - \frac{1}{2}H(E_{a_3=0}) - \frac{1}{2}H(E_{a_3=1}) \simeq 0.1052$.
   - pour l'attribut a_4 :
     * $H(E_{a_4=0}) = -\frac{2}{8}\log\frac{2}{8} - \frac{6}{8}\log\frac{6}{8} \simeq 0,5623$;
     * $H(E_{a_4=1}) = \log 2 \simeq 0,6931$;

* soit $G(E, a_4) = H(E) - \frac{8}{14}H(E_{a_4=0}) - \frac{6}{14}H(E_{a_4=1}) \simeq 0,0334$.

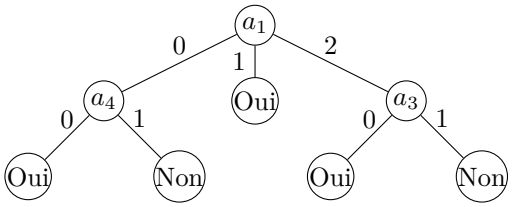
On choisit donc l'attribut a_1 pour la racine.

- Pour le sous-arbre $a_1 = 0$, on repart de $E = E_{a_1=0}$, soit $H(E) = H(E_{a_1=0}) \simeq 0,673$. On calcule de même :
  * $G(E, a_2) \simeq 0,0138$;
  * $G(E, a_3) \simeq 0,0138$;
  * $G(E, a_4) \simeq 0,673$.

  On choisit donc l'attribut a_4 comme racine du sous-arbre. Ce sous-arbre aura pour fils $a_4 = 0$ une feuille « Oui » et pour fils $a_4 = 1$ une feuille « Non ».
- Pour le sous-arbre $a_1 = 1$, on crée simplement une feuille « Oui ».
- Pour le sous-arbre $a_1 = 2$, on repart de $E = E_{a_1=2}$. On calcule :
  * $G(E, a_2) \simeq 0,3958$;
  * $G(E, a_3) \simeq 0,673$;
  * $G(E, a_4) \simeq 0,0138$.

  On choisit donc l'attribut a_3 comme racine du sous-arbre. Ce sous-arbre aura pour fils $a_3 = 0$ une feuille « Oui » et pour fils $a_3 = 1$ une feuille « Non ».

Finalement, l'arbre obtenu est :



2. Comme il y a du soleil et une humidité élevée, on choisira de ne pas jouer au tennis.

1.3 Surapprentissage

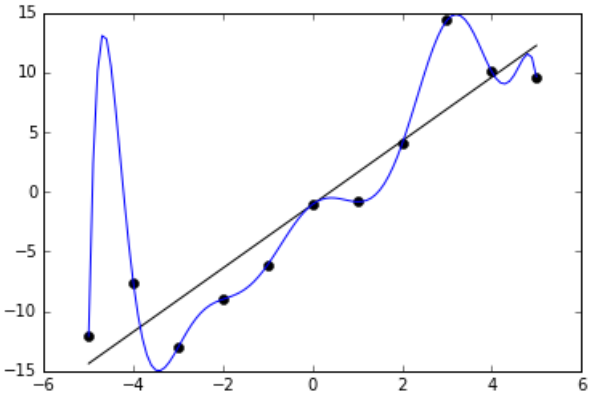
Les données contiennent plusieurs informations :

- le signal, qui correspond à ce que le modèle d'apprentissage cherche à reproduire ;
- le bruit, qui peut être dû au choix aléatoire des données ou à la variance des informations qu'on cherche à mesurer.

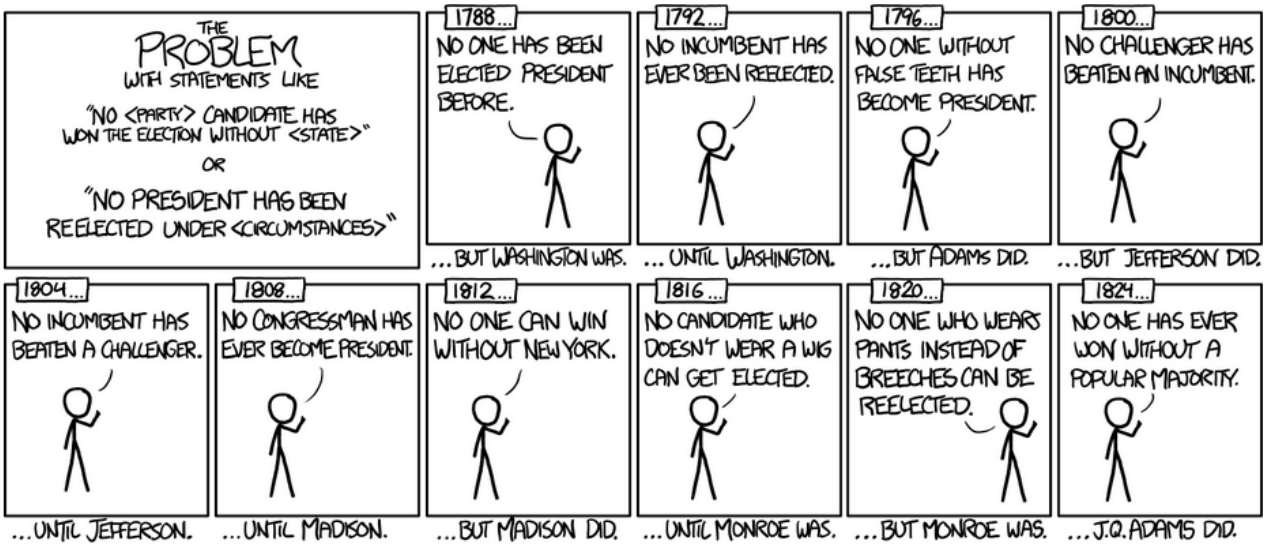
Si le modèle essaie d'être trop proche des données, le bruit peut prendre le pas sur le signal dans ce qu'on cherche à déterminer. C'est ce qu'on appelle le **surapprentissage**.

Exemple 1.14

Dans certaines situations, une régression linéaire est une meilleure interprétation qu'une interpolation, même si l'interpolation colle mieux aux données fournies.



Une situation de surapprentissage peut se produire si on considère un trop grand nombre d'attributs pour les données à classer. À nouveau, l'utilisation d'un jeu d'apprentissage et d'un jeu de test peut permettre de détecter la création d'un biais dû au bruit, et éventuellement de réduire le nombre d'attributs pertinents.



la suite sur `xkcd.com`...

2 Apprentissage non supervisé

Dans l'apprentissage non supervisé, on dispose d'un jeu de données non étiquetées et on cherche à dégager les similitudes pouvant exister entre certaines de ces données en les regroupant en différentes catégories : on cherche à les partitionner. On appelle cela le *clustering*, parfois (mal ?) traduit en classification (mais à ne pas confondre avec la classification effectuée dans la partie précédente).

L'objectif d'un clustering est de créer des classes d'équivalence où on maximise l'homogénéité au sein d'une classe, et on maximise l'hétérogénéité entre différentes classes. Comme pour la classification de la partie précédente, on utilisera une fonction de distance qui pourra varier selon les données utilisées.

2.1 Clustering hiérarchique ascendant

Le clustering hiérarchique ascendant (ou CHA) consiste à regrouper les données petit à petit en fonction de leur proximité avec d'autres données. L'approche est relativement simple et peut se résumer en :

Entrée : ensemble de données $x_1, \dots, x_N$.
Début algorithme
  Créer N classes $\{x_1\}, \dots, \{x_n\}$.
  Tant que nécessaire **Faire**
    Fusionner les deux classes les plus proches.

Dans l'algorithme précédent, certaines choses sont implicites :

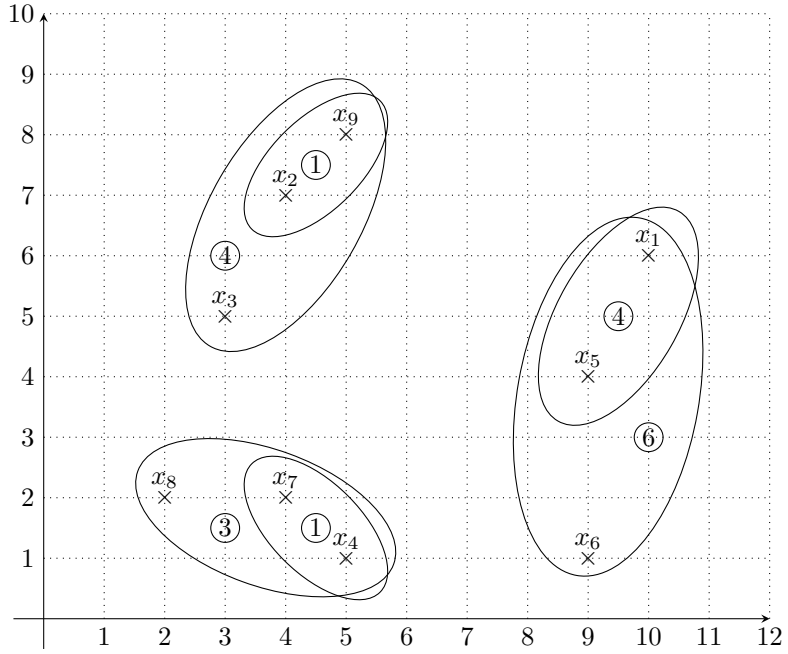
- La condition « nécessaire » de la boucle `Tant que` peut avoir différentes interprétations, on peut choisir d'arrêter les regroupements lorsque :
  * un nombre de classes déterminé à l'avance a été atteint ;
  * les deux classes les plus proches sont suffisamment éloignées l'une de l'autre ;
  * le diamètre d'une des classes devient trop important.
- Par ailleurs, la notion de « plus proches » est volontairement vague. On peut obtenir des résultats différents selon la distance $\Delta(A, B)$ choisie entre deux classes A et B :
  * liaison simple : $\min\{\delta(x, y) \mid x \in A, y \in B\}$;
  * liaison complète : $\max\{\delta(x, y) \mid x \in A, y \in B\}$;
  * liaison moyenne : $\dfrac{1}{|A||B|} \displaystyle\sum_{x \in A, y \in B} \delta(x, y)$;
  * liaison barycentrique : $\delta(b_A, b_B)$ où b_A et b_B sont les barycentres de A et B ;
  * liaison de Ward : $\sqrt{\dfrac{|A||B|}{|A| + |B|}}\delta(b_A, b_B)$, correspondant à une liaison barycentrique en donnant moins de poids aux classes composées de données isolées ;
  * ...

Exemple 2.1

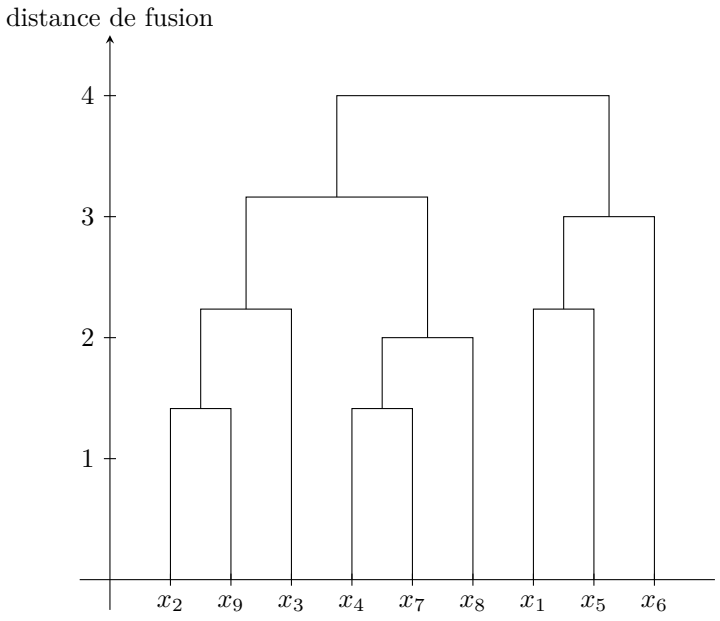
On considère l'ensemble de points suivant :

$$x_1(10, 6), x_2(4, 7), x_3(3, 5), x_4(5, 1), x_5(9, 4), x_6(9, 1), x_7(4, 2), x_8(2, 2), x_9(5, 8)$$

On souhaite appliquer l'algorithme CHA par liaison simple, avec une distance maximale avant fusion de deux clusters de 3. On obtient les fusions successives suivantes (les cas d'égalité sont simultanés) :



On peut représenter l'ordre des fusions, ainsi que les distances au moment de la fusion par un **dendrogramme** (qu'on a continué ici jusqu'à la fusion de l'ensemble des classes) :



Remarque 2.2

Bien que simple à appréhender, le CHA possède deux inconvénients :

- il n'y a pas de « remise en question » : deux classes fusionnées ne seront jamais séparées ;
- la complexité peut être élevée : la recherche des deux classes les plus proches peut prendre un temps $\mathcal{O}(N^2)$, ce qui donne une complexité totale en $\mathcal{O}(N^3)$ (et encore, seulement si les structures de données sont optimisées).

Exercice 4

Reprendre l'exemple précédent et déterminer le dendrogramme correspondant à un CHA avec liaison complète.

Exercice 5

On représente une classe de points dans le plan par une liste simplement chaînée des points, chaque point étant représenté par un couple de flottants correspondant aux coordonnées.

1. Écrire une fonction

```
delta_simple : (float * float) list -> (float * float) list -> float
```

   qui calcule la distance $\Delta(a, b)$ de liaison simple entre deux classes a et b.
2. Écrire une fonction

```
proximite : (float * float) list array -> float array array
```

   qui prend en argument un tableau de m classes et renvoie une matrice de proximité `mp` de dimensions $m \times m$ telle que `mp.(a).(b)` est égal à $\Delta(a, b)$.
3. En déduire une fonction

```
classes_plus_proches : (float * float) list array -> int * int * float
```

qui prend en argument un tableau de classes et renvoie un triplet $(a, b, \Delta(a, b))$ où a et b sont les indices des classes les plus proches.

4. Écrire une fonction

```
cha : (float * float) list -> float -> (float * float) list list
```

qui prend en argument une liste de points et un seuil et renvoie une partition en classes selon l'algorithme de clustering hiérarchique ascendant, en arrêtant les fusions dès que la distance minimale entre deux classes dépasse le seuil.

2.2 k-moyennes

L'algorithme des k-moyennes est un algorithme qui cherche à construire un nombre de clusters défini à l'avance (appelé k dans la littérature, mais qu'on notera m en cohérence avec les parties précédentes). L'objectif est de résoudre le problème d'optimisation `CLUSTERING` :

* **Instance** : un ensemble de données $E = \{x_1, \dots, x_N\}$ et un entier m.
* **Solution** : une partition de E en m classes $C_1, \dots, C_m$.
* **Optimisation** : minimiser $\sum_{j=1}^{m} \sum_{x \in C_j} \delta(x, b_j)^2$, où b_j est le barycentre de la classe C_j.

Cependant, ce problème d'optimisation est NP-difficile donc l'algorithme présenté ici est une heuristique considérée comme satisfaisante mais qui peut renvoyer un résultat non optimal. L'idée de l'algorithme des k-moyennes est la suivante :

– créer m points $b_1, \dots, b_m$, soit au hasard dans l'espace considéré, soit choisis au hasard parmi les x_i ;
– tant que les b_j changent :
  * affecter à chaque point x_i la classe j correspondant au barycentre b_j qui lui est le plus proche ;
  * modifier chaque b_j en le barycentre de la nouvelle classe j.

Proposition 2.3

Chaque itération de la boucle `Tant que` de l'algorithme précédent a une complexité $\mathcal{O}(m \times N)$.

Preuve

Proposition 2.4

L'algorithme des k-moyennes termine.

Preuve

Admis. L'idée est de montrer qu'à chaque itération, $\sum_{x \in C_j} \delta(x, b_j)^2$ diminue pour tous les C_j, et strictement pour au moins l'un d'entre eux (en supposant que les cas d'égalité sont toujours tranchés de la même manière).

Remarque 2.5

L'algorithme des k-moyennes, bien que généralement plus rapide que l'algorithme de clustering hiérarchique ascendant présente certains défauts :

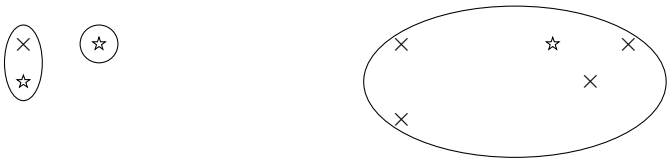
– Le choix initial des b_j est un point critique de l'algorithme. Un mauvais choix peut entraîner un grand déséquilibre par rapport à une solution optimale (si les points choisis sont éloignés des

données réelles, ou si deux points sont initialement choisis très proches).
- Il peut converger vers un minimum local qui n'est pas un minimum global.
- En pratique, on fixe un nombre limite d'itération plutôt que d'attendre la convergence.

Exemple 2.6

On cherche à partitionner l'ensemble de points suivant en 3 classes. La figure suivante représente un choix possible des trois points initiaux (les étoiles) avec une convergence immédiate en une solution optimale.

La figure suivante représente une convergence vers un minimum local.



Exercice 6

On représente une partition d'un ensemble E en m classes comme un tableau `classes` de taille N telle que `classes[i]` vaut l'entier j tel que $x_i \in C_j$. On se place dans le cas où $d = 2$. On représente un point de $\mathbb{R}^2$ par le type :

```
struct Point {
    double abs;
    double ord;
};

typedef struct Point point;
```

tel qu'un point $x = (a, b) \in \mathbb{R}^2$ est représenté par un objet `x` de type `point` tel que `x.abs` vaut a et `x.ord` vaut b.

1. Écrire une fonction `point* echantillon(point* donnees, int N, int m)` qui prend en argument un tableau `donnees` contenant N points et un entier m et renvoie un tableau correspondant à un échantillon de taille m dans le tableau `donnees`. On autorisera la modification du tableau et on rappelle qu'un appel à `rand() % n` renvoie un entier choisi aléatoirement et uniformément dans $[\![0, n-1]\!]$.
2. Écrire une fonction `int plus_proche(point x, point* bary, int m)` qui prend en arguments un point x et un tableau contenant m points et renvoie l'indice j du point du tableau le plus proche de x.
3. En déduire une fonction

   `bool modif_classes(point* donnees, point* bary, int* classes, int N, int m)`

   qui prend en arguments l'ensemble E (un tableau de N points), un tableau de m barycentres `bary` et un tableau de classes de taille N et modifie `classes` de telle sorte qu'après modification, `classes[i]` correspond à l'indice j du barycentre le plus proche de `donnees[i]`. La fonction renverra `true` s'il y a eu une modification de `classes` et `false` sinon.

4. Écrire une fonction

```
void modif_bary(point* donnees, point* bary, int* classes, int N, int m)
```

qui prend les mêmes arguments que la fonction précédente et modifie le tableau `bary` de telle sorte qu'après modification `bary[j]` correspond au barycentre de la classe j.

5. En déduire une fonction `int* k_moyennes(point* donnees, int N, int m, int itermax)` qui prend en arguments un ensemble E de données de taille N, un entier m et un entier `itermax` et renvoie un pointeur vers un tableau `classes` correspondant à une partition de E de taille m selon l'algorithme des k-moyennes. On arrêtera l'algorithme soit lorsqu'il y a convergence, soit lorsqu'on a dépassé le nombre maximal d'itérations.