

ET LA MACHINE APPRIT

L'intelligence artificielle n'existe pas.

Luc Julia [julia_intelligence_2019]

À la fin de ce chapitre, je sais :

- ☒ définir les termes intelligence artificielle et apprentissage automatique
- ☒ expliquer et utiliser l'algorithme d'apprentissage supervisé des k plus proches voisins (KNN)
- ☒ expliquer et utiliser l'algorithme d'apprentissage non supervisé des k moyennes (Kmeans)

A Principles

■ **Définition 1 — Intelligence artificielle.** L'intelligence artificielle est un mot valise qui désigne des systèmes électroniques et informatiques capables de percevoir, de comprendre, d'apprendre et d'agir pour atteindre un but précis.

Cette définition pose de nombreux problèmes car elle repose sur le concept d'intelligence qu'il est bien difficile de définir. D'après le Larousse, l'intelligence est un *ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et rationnelle*. La machine aurait-elle des fonctions mentales ? C'est à dire des fonctions cognitives, exécutives et langagières ? De nombreuses personnes [julia_intelligence_2019] insistent sur l'absurdité du mot intelligence artificielle qui a fait naître de nombreux fantasmes depuis son invention en 1955 par John McCarthy[mccarthy_proposal_2006]. C'est pourquoi, dans ce chapitre, on s'intéressera davantage aux algorithmes d'apprentissage automatique, terme plus précis, plus pragmatique, qui pose moins de problèmes métaphysiques.

🇬🇧 **Vocabulary 1 — Machine Learning ↵** Apprentissage automatique

■ **Définition 2 — Apprentissage automatique.** L'apprentissage automatique est un ensemble de techniques qui a pour but d'automatiser les prises de décision en créant des systèmes

capables, à partir d'un jeu de données, de découvrir, d'identifier et de classifier des motifs récurrents. Ces systèmes peuvent en plus être capables d'améliorer leurs performances au fur et à mesure de leur utilisation.

Les données d'entrée peuvent être de nature très différente : langage, nombres, images, sons, signaux... Les motifs récurrents identifiés par la machine ne sont généralement pas perceptibles par l'être humain ce qui donne à ces techniques une apparence de boîte noire. Cependant ces motifs identifiés le sont grâce à une analyse rationnelle du problème de décision considéré : rien d'ésotérique là dedans.

L'apprentissage automatique est considéré comme un sous-ensemble de l'intelligence artificielle.

B Objectifs et algorithmes

Il existe deux grands objectifs à l'apprentissage automatique :

1. la régression,
2. et la classification.

■ **Définition 3 — Régession.** Un algorithme d'apprentissage automatique dont la sortie est une régression génère un résultat sous la forme d'une valeur continue. Il s'agit de prédire par exemple un prix, une température ou le cours d'une action.

■ **Définition 4 — Classification.** Un algorithme d'apprentissage automatique dont la sortie est une classification génère un résultat sous la forme d'une valeur discrète qui permet de trier un ensemble de données selon plusieurs catégories.

Par exemple masculin ou féminin, vrai ou faux, spam ou pas spam, carotte, navet ou pomme de terre si on considère des légumes, pierre, sable, végétal ou eau si l'on considère la nature d'un sol à identifier.

 Parfois un même algorithme peut permettre d'effectuer une régression ou une classification.

On distingue plusieurs types d'algorithmes d'apprentissage automatique : l'apprentissage supervisé, non supervisé et l'apprentissage par renforcement. Seules deux approches sont au programme : une supervisée (KNN), l'autre non supervisée (K-means).

■ **Définition 5 — Apprentissage supervisé.** Un algorithme d'apprentissage supervisé s'appuie sur un jeu de données étiquetées pour s'entraîner à produire des résultats corrects. Au fur et à mesure que les données sont lues, il ajuste les poids du modèle afin de coller au mieux aux étiquettes (phase d'entraînement). On réserve généralement une partie du jeu de données pour vérifier la validité du modèle ainsi obtenu (phase de test).

R Il existe de nombreux algorithmes pour la régression et/ou la classification en apprentissage automatique. Le TP associé à ce cours est l'occasion d'illustrer les performances de plusieurs algorithmes confrontés à des jeux de données différents. Le choix d'un algorithme plutôt qu'un autre en apprentissage automatique est un compromis issu de l'expérience.

Parmi les algorithmes les plus utilisés on note :

1. l'analyse en composantes principales (Principal Component Analysis) pour réduire les dimensions d'un jeu de données,
2. les k-moyennes et les k plus proches voisins,
3. les arbres de décisions (Decision Trees),
4. les séparateurs à vaste marge (Support Vector Machine),
5. les réseaux de neurones (ANN),
6. les réseaux de neurones convolutionnels (CNN) pour l'apprentissage profond.

C Données

Les données jouent un rôle central dans l'apprentissage automatique. Avant de se lancer dans l'apprentissage automatique, il est nécessaire d'extraire, de sélectionner et de transformer les données pour constituer un jeu de données utilisable. C'est ce jeu de données qui constitue l'**entrée** d'un algorithme.

Les données peuvent être **étiquetées** : on peut indiquer ainsi au système les caractéristiques qu'il va devoir apprendre à identifier. Elles peuvent aussi être **non étiquetées** et le système devra alors repérer et extraire les motifs récurrents sans indications.

a Normalisation

Les algorithmes KNN et K-moyennes reposent sur des calculs de distances qui sont dépendantes de l'échelle des variables. Si les échelles relatives des variables sont d'ordre très différent, par exemple [10, 1000] contre [0.01, 0.1], la plus grande domine le calcul des distances, écrase de son ordre le résultat de la distance. Les paramètres de petite échelle sont donc invisibilisés. Cela peut entraîner :

- des groupes biaisés vers les dimensions avec des échelles plus grandes, même si ces paramètres n'ont pas plus d'importance,
- une convergence plus lente.

Il existe plusieurs types de normalisation des données. Parmi les plus courantes, on trouve :

- Le Z-score : $e = \frac{e - \mu}{\sigma}$ où μ est la moyenne et σ l'écart type du paramètre. Cette normalisation aboutit à des paramètres de moyenne nulle et d'écart-type 1. Ce Z-score permet d'identifier les données aberrantes, en utilisant la loi normale.
- la mise à l'échelle par rapport au minimum et au maximum : $e = \frac{e - v_{\min}}{v_{\max} - v_{\min}}$. Les paramètres sont alors tous à valeur dans [0, 1].

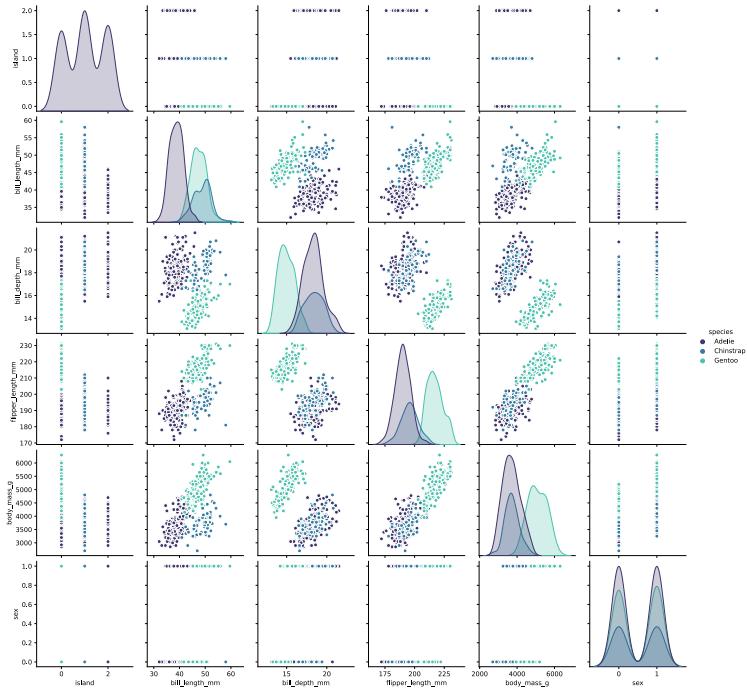


FIGURE 1 – Visualisation de la dispersion des paramètres sur le jeu de données des manchots répartis sur trois îles et en trois espèces

Après normalisation, la contribution de tous les paramètres devient similaire.

b Visualisation

La visualisation des caractéristiques d'un jeu de données et des résultats de l'apprentissage est un élément déterminant pour l'analyse des performances des algorithmes. Un exemple de visualisation des données brutes produit par les bibliothèques Scikit-Learn et Seaborn Python est donné sur la figure 1 : il s'agit des paramètres d'un jeu de données sur les manchots. Un exemple de projection des résultats de l'algorithme KNN sur le jeu de données iris selon toutes les dimensions est donné sur la figure 2.

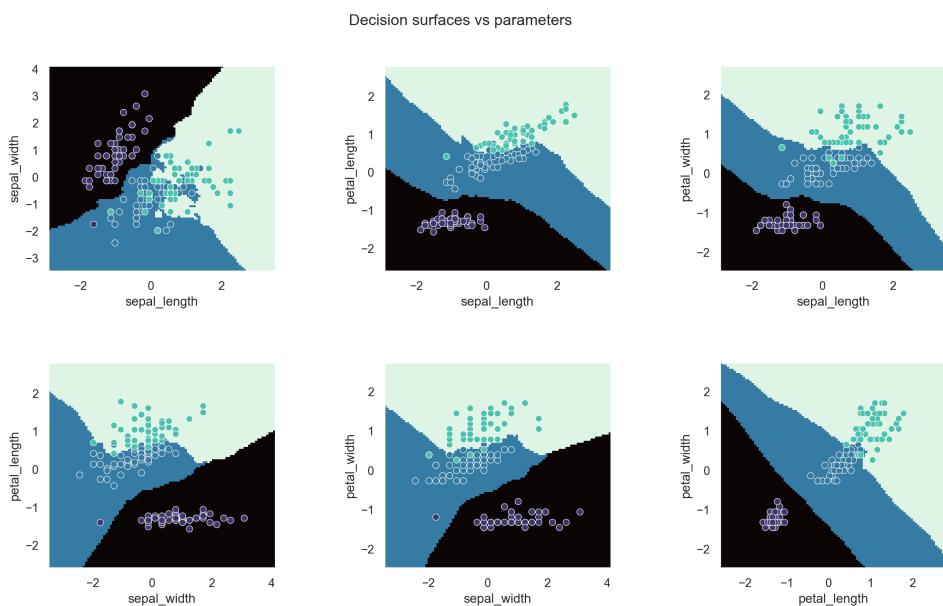


FIGURE 2 – Résultats de l'apprentissage de l'algorithme KNN sur le jeu de données de l'iris. Les régions de décisions du classificateur sont superposées au valeurs vraies des espèces d'iris.

c Évaluation de la performance

■ **Définition 6 — Matrice de confusion.** La matrice de confusion est un outil qui permet de visualiser les performances d'un algorithme de classification. Chaque case contient le nombre^a de cas d'une classe *donnée* qui a été **prédite** par l'algorithme comme appartenant à une certaine classe.

La matrice de confusion est donc organisée comme suit :

- les lignes représentent les classes données (par le jeu de données étiquetées),
- les colonnes représentent les classes prédites par l'algorithme.

a. ou le pourcentage

■ **Exemple 1 — Matrice de confusion.** On considère un algorithme de traitement automatique des messages instantanés : on cherche à classer les messages en différentes catégories (peur (P), joie (J), amour (A), tristesse (T), violence (V)) en fonction de leur contenu. On dispose d'un jeu de données étiquetées dont on utilise une partie pour l'apprentissage de l'algorithme et une partie pour le test de l'algorithme^a. La matrice ci-dessous représente les résultats obtenus par l'algorithme sur le jeu de données de test.

Donnée	P	J	A	T	V
	55	6	4	30	5
J	10	75	2	11	2
A	3	1	91	5	1
T	25	15	1	58	1
V	4	1	2	0	93

Prédiction

On peut lire la première ligne de la matrice ainsi : sur la centaine de messages de test étiquetés P (peur), 55 ont été effectivement prédits comme tels par l'algorithme. Mais, 6 ont été prédits joie, 4 amour, 30 tristesse et 5 violence. Notre algorithme engendre donc une certaine confusion entre la peur et la tristesse^b, confusion qui est explicite lorsqu'on affiche cette matrice! Lorsqu'un algorithme fonctionne correctement, les valeurs sur la diagonale sont maximales et les éléments non diagonaux minimaux.

a. Généralement on sélectionne 70% des données pour l'apprentissage et 30% pour le test.

b. à moins que ce ne soit notre jeu de données et les classes choisies qui ne sont pas pertinentes!

R Dans le cas où il y a deux classes, la diagonale représente les vrais positifs et les vrais négatifs.

D Apprentissage supervisé : k plus proches voisins

 **Vocabulary 2 — K-Nearest Neighbours (KNN) Algorithm** ⇔ Algorithme des k plus proches voisins

L'algorithme des k plus proches voisins [fix_discriminatory_1952] est un algorithme relativement simple, supervisé et encore très utilisé aujourd'hui pour effectuer des régressions ou des classifications. Il ne présente pas à proprement parler de phase d'entraînement qui est confondue avec la phase de prédiction¹ : il utilise toutes les données à chaque calcul. Il est cependant couramment utilisé pour les systèmes simples de recommandation, la reconnaissance de modèle, la fouille de données, les prévisions des marchés financiers ou la détection d'intrusion.

Pour les problèmes de **classification**, l'étiquette de classe est affectée à l'échantillon analysé sur la base d'un **vote à la majorité** : on utilise l'étiquette de la classe la plus fréquemment représentée autour de l'échantillon.

Pour les problèmes de **régression** et une certaine fonction f à valeur continue, on procède de la même manière mais on calcule la prédiction sur la **valeur de la moyenne** de f sur les k plus proches voisins.

a Description de l'algorithme KNN

Algorithme 1 k plus proches voisins (KNN)

```

1: Fonction KNN( $D, x, k, \delta$ )
2:    $n \leftarrow |D|$ 
3:    $\Delta \leftarrow \emptyset$                                 ▷ Distances à calculer
4:   pour chaque échantillon étiqueté  $e \in \mathcal{E}$  répéter
5:     Ajouter  $\delta(x, e)$  à  $\Delta$ 
6:   Sélectionner les  $k$  voisins les plus proches de  $x$  en utilisant  $\Delta$ 
7:   Compter le nombre d'occurrences de chaque classe des  $k$  voisins de  $x$ 
8:   renvoyer la classe  $c$  la plus représentée parmi les  $k$  plus proches voisins

```

On considère un problème de classification dans \mathbb{R}^d . On dispose d'un ensemble d'échantillons étiquetés \mathcal{E} dont les éléments sont des couples composés d'un vecteur de \mathbb{R}^d et d'une étiquette désignant la classe à laquelle appartient l'échantillon. On note l'ensemble des classes possibles \mathcal{C} . On cherche donc à déterminer la classe d'autres échantillons non étiquetés. On note $\mathcal{E} = \{(e_i, c_j), i \in \llbracket 1, n \rrbracket, j \in \llbracket 1, m \rrbracket, e_i \in \mathbb{R}^d, c_i \in \mathcal{C}\}$, l'ensemble des données étiquetées dont

1. ce n'est pas un avantage, mais bien un inconvénient

on dispose. On cherche à trouver la classe c de e , un élément de \mathbb{R}^d . On dispose d'une distance δ sur \mathbb{R}^d .

La complexité temporelle de l'algorithme KNN (cf. algorithme 1) pour un jeu de n données de dimension d est $O(1)$ à l'entraînement car cette phase n'existe pas², mais $O(n \times d)$ à la prédiction. Plus le jeu de données grandit, plus KNN devient inefficace : ce problème est souvent désigné en IA par la malédiction de la dimension d'un problème.

b Distances

Pour savoir si un voisin est proche, il faut être capable de mesurer sa distance. L'algorithme KNN utilise donc une distance dont la définition varie selon le problème considéré : distance euclidienne, de Manhattan, de Tchebychev. La figure 3 illustre les distances les plus utilisées en pratique. Mais, selon le contexte on peut faire appel à la distance de Levenshtein (pour la comparaison de mots) ou de Hamming (pour des séquences de même longueur).

R L'analyse du jeu de données est parfois nécessaire pour choisir la distance la plus appropriée.

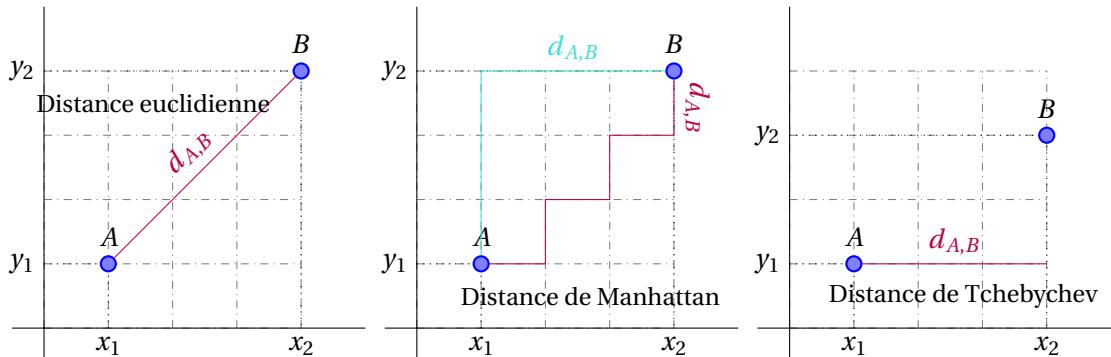


FIGURE 3 – Illustration de différentes distances dans le plan

c Comment choisir k ?

Si n est la dimension du problème, c'est-à-dire que le jeu de données est constitué de n échantillons, alors il faut nécessairement choisir k dans $\llbracket 1, n \rrbracket$. Choisir un k trop grand amène à classer la majorité des échantillons dans la classe dominante. Choisir un k trop petit ne permet pas de conclure précisément : la réponse est trop sensible au bruit. Par ailleurs, on choisit souvent k impair pour éviter les égalités lors du vote majoritaire.

Le choix le plus commun et empirique est de prendre \sqrt{n} , la racine carrée du nombre de points ou bien $\sqrt{n/m}$, la racine du nombre de points divisé par le nombre de classes dans le cas d'une classification. Mais tester les différentes valeurs de k peut s'avérer nécessaire.

2. on parle alors de *lazy learning*

d Comment sélectionner les k plus proches voisins?

Les étapes de l'algorithme 1 des lignes 4 à 6 nécessitent une forme de tri des indices des échantillons d'après l'ensemble Δ des distances à x pour faire émerger les k plus proches voisins. On peut imaginer effectuer un tri en ligne au fur et à mesure par insertion, un tri fusion, un tri rapide ou un tri par tas min.

R Certaines structures de données comme les arbres k-d, , constituent des solutions très adaptées à la recherche des voisins : les noeuds de ces arbres de recherches sont construits à partir d'hyperplans qui divisent l'espace \mathbb{R}^d des échantillons : tous les points situés en dessous d'un noeud hyperplan sont stockés dans le sous-arbre gauche, les autres dans sous-arbre droit. Cette structure «*dichotomique*» permet de trouver rapidement les voisins en éliminant des parties de l'espace de recherche très rapidement --> HORS PROGRAMME

 **Vocabulary 3 — k-dimensional trees, k-d trees** ↔ les arbres k-d

e Comment détecter la classe majoritaire?

Pour élire la classe majoritaire d'un échantillon, il est possible de procéder de différentes manières :

- à la majorité simple des classes des k voisins : dans ce cas, la classe la plus représentée parmi les k voisins est la classe prédictive pour l'échantillon.
- à la majorité des classes des k voisins pondérées par la distance du voisin : dans ce cas, les voisins les plus proches contribuent davantage à la classe prédictive.

P Pour implémenter en Python ce choix, l'option la plus simple est de créer une représentation des occurrences des classes (histogramme), par exemple sous la forme d'un tableau. Ensuite, il suffit de sélectionner l'indice du maximum de ce tableau pour trouver la classe majoritaire, avec ou sans pondération.

R Il convient de noter que le choix d'une valeur appropriée de k (le nombre de voisins les plus proches) et la méthode de pondération des votes (le cas échéant) peuvent avoir un impact sur les performances de l'algorithme KNN. La validation croisée et la recherche en grille sont des techniques courantes pour ajuster ces hyperparamètres : on cherche le paramètre k optimal en testant des valeurs.

f KNN pour la régression

On cherche à effectuer une régression sur une série de données expérimentales sur un modèle à une seule dimension. La figure 4 représente les données expérimentales par des points. Aucun axe ne possède de titre ni d'unités car ces données sont simulées à partir d'une équation de type : $f(x) = e^{-\alpha x} \cos(x)$. Les données sont rendues «expérimentales» par l'ajout d'un bruit additif.

La figure 4 montre l'importance du choix de k lorsque KNN est utilisé : lorsque k est trop petit, la régression est sensible au bruit; lorsque k est trop grand, la régression ne suit plus les variations expérimentales.

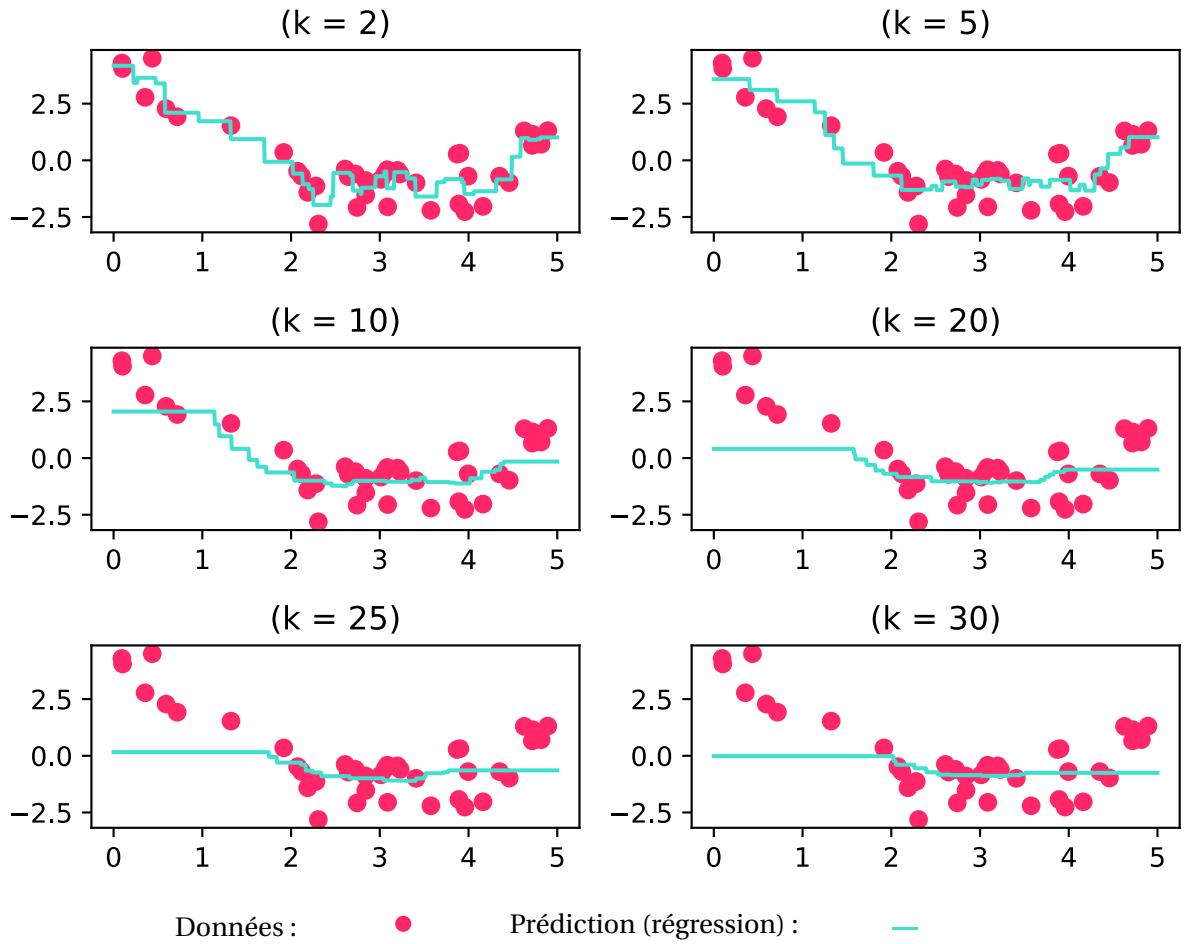


FIGURE 4 – Illustration de l'influence du nombre de voisins k sur une régression de modèle non linéaire dans le cadre de l'algorithme KNN

g KNN pour la classification

Les figures 2 et 5 illustrent le résultat de l'algorithme 1 sur le très célèbre jeu de données de l'iris [fisher_use_1936]. Les résultats montrent clairement que KNN est capable de discriminer plusieurs variétés d'iris en connaissant uniquement les paramètres morphologiques des pétales et des sépales. On remarque essentiellement une légère confusion entre Versicolor et Virginica. Certaines Virginica sont prises pour des Versicolor. La matrice de confusion démontre la performance de l'algorithme.

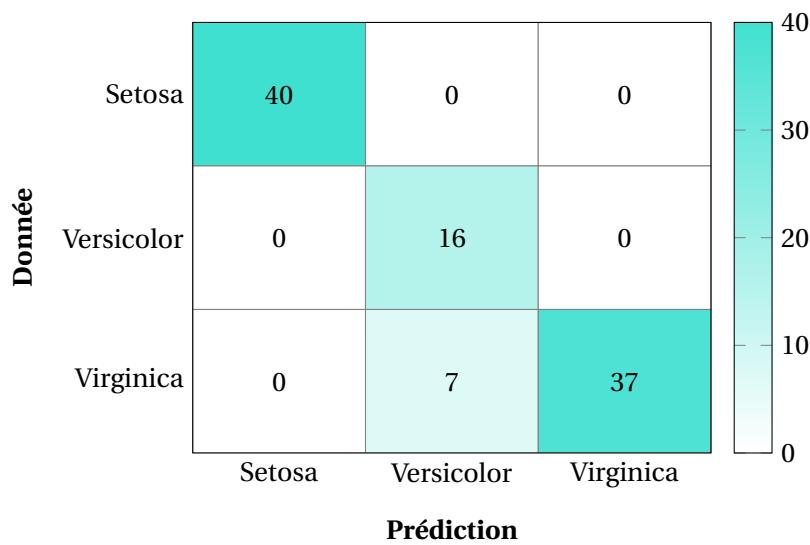


FIGURE 5 – Matrice de confusion dans le cadre de la classification des variétés d'Iris : Setosa, Versicolor et Virginica. Les paramètres d'entrées sont les largeurs et longueurs des pétales et sépales. La prédiction est correcte dans plus de 90% des cas comme le montre la matrice de confusion.

E Apprentissage non supervisé : k-moyennes

a Description de l'algorithme des K-moyennes



Vocabulary 4 — K-means ⇔ Algorithmes des k-moyennes

On ne dispose pas toujours d'un jeu de données étiquetées, c'est-à-dire des échantillons dont on connaît la classe. C'est pourquoi certains algorithmes d'apprentissage automatique développent une approche non supervisée. L'absence d'échantillons de vérification fait qu'il est parfois plus difficile d'évaluer la performance de ces algorithmes.

On considère de nouveau un problème de **classification**. On suppose qu'on dispose d'un ensemble d'échantillons $\mathcal{E} = \{e_i, i \in \llbracket 1, n \rrbracket, e_i \in \mathbb{R}^d\}$. Il s'agit de créer une partition de \mathcal{E} selon k classes.

Algorithme 2 K-moyennes

```

1: Fonction KMEANS( $\mathcal{E}$ ,  $k$ )
2:    $\mathcal{P} \leftarrow \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k\}$  une partition quelconque de  $\mathcal{E}$  en  $k$  classes
3:   tant que des échantillons changent de classe répéter
4:      $(b_1, b_2, \dots, b_k) \leftarrow \text{BARYCENTRE}(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k)$ 
5:     pour chaque échantillon  $e$  de  $\mathcal{E}$  répéter
6:       Trouver la classe  $\mathcal{P}_i$  la plus proche de  $e$ ,  $\|e - b_i\|^2$  est minimale sur  $\mathcal{P}$ 
7:       Ajouter  $e$  à la classe  $\mathcal{P}_i$  trouvée
8:   renvoyer  $(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k)$ 

```

Le cœur de cet algorithme est la construction des partitions de l'ensemble du jeu de données. L'algorithme crée donc des catégories, autant qu'on lui en demande. Si l'utilisateur n'a pas d'idée précise du nombre de catégories à créer, il est nécessaire de tâtonner. Par ailleurs, le lien sémantique entre les données et le résultat est à la charge de l'utilisateur de l'algorithme : K-moyennes se contente de créer des partitions, il ne leur attribue aucun sens particulier. On est loin de l'intelligence artificielle qui fait fantasmer les foules! La figure 6 montre que K-moyennes trouve bien k classes (cluster ou sous-groupe) même si ce n'est pas la réalité.



L'algorithme des K-moyennes minimise le coût défini par la fonction :

$$f(\mathcal{E}, k) = \sum_{i=1}^k \sum_{e \in \mathcal{P}_i} \|e - m_i\|^2 \quad (1)$$

L'algorithme consiste à minimiser la variance des partitions (clusters). La fonction BARYCENTRE utilise donc la norme **euclidienne**.



--> HORS PROGRAMME On peut démontrer que cet algorithme termine en considérant

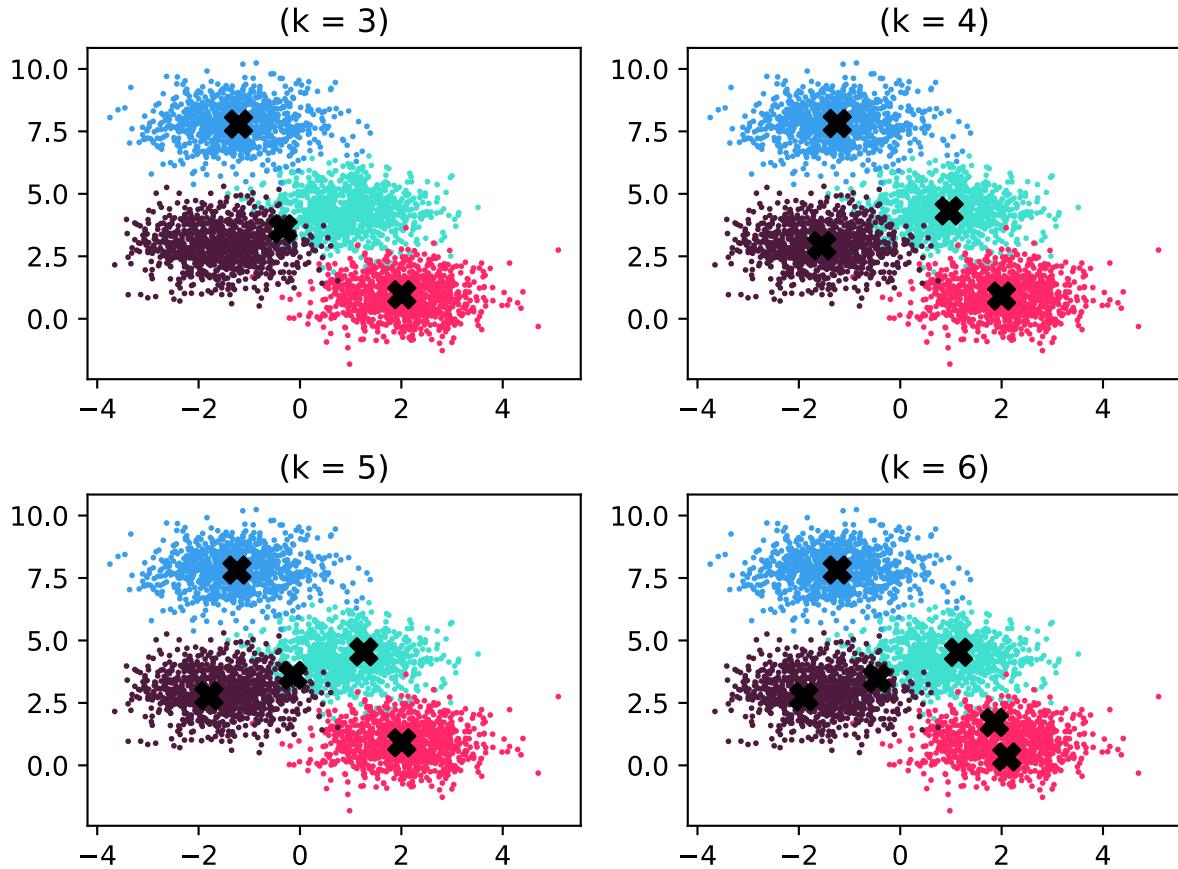


FIGURE 6 – Illustration des conséquences du choix du paramètre K dans K-moyennes. Sur cet exemple, seules quatre classes existent dans les données. K varie entre 3 et 6. Les barycentres des classes sont représentés par des croix noires.

la fonction de coût $f(\mathcal{E}, k) = \sum_{i=1}^k \sum_{e \in \mathcal{P}_i} \|e - m_i\|^2$.

Comme il y a k^n façons de répartir n points en k partitions, l'ensemble des partitions possible est un ensemble fini. L'algorithme itère sur cet ensemble fini. S'il ne termine pas, c'est donc qu'il opère un cycle dans cet ensemble fini. Si c'était le cas, comme la fonction f est strictement décroissante d'après notre choix des partitions à chaque itération, cela signifierait qu'une même partition pourrait présenter un coût plus faible que son propre coût, ce qui est absurde. Donc, l'algorithme n'itère pas sur un cycle et nécessairement termine de parcourir l'ensemble des partitions en minimisant la fonction de coût.

En résumé : la fonction de coût décroît strictement tant que la partition change. Si la partition ne change pas, l'algorithme s'arrête.

b Points forts et points faibles de l'algorithme des K-moyennes

L'algorithme des K-moyennes est :

1. n'est pas nécessairement optimal. La fonction f converge, mais pas nécessairement vers l'optimum global.
2. est sensible à l'initialisation : selon la partition de départ, le résultat peut ne pas être le même, c'est-à-dire f peut atteindre un minimum local.

Ces points faibles sont dépassables en pratique :

- dans le cas où l'optimal global n'a pas été atteint, il est possible de redémarrer l'algorithme à partir d'autres partitions initiales et de trouver l'optimal global.
- il est également possible d'optimiser la partition de départ afin d'augmenter les chances de trouver l'optimal global.

F Classification hiérarchique ascendante

→ HORS PROGRAMME

■ **Définition 7 — Classification Hiérarchique Ascendante (CHA)** . La Classification Hiérarchique Ascendante (CHA) est une méthode gloutonne de classification non supervisée dont le principe est d'**agréger des échantillons dans des classes** de plus en plus grandes sur la base de leurs similitudes. À la fin de ce processus *bottom-up*, c'est-à-dire qui part des échantillons isolés pour les regrouper, une classe unique contient tous les échantillons.

M **Méthode 1 —** Le principe de fonctionnement de la CHA est le suivant :

1. Créer les classes : au démarrage, chaque échantillon est considéré comme une classe à part entière. Avec n échantillons, l'algorithme commence avec n classes.
2. Calculer des distances inter-classes pour mesurer leur ressemblance.
3. Regrouper les deux classes les plus proches pour former une nouvelle classe.
4. Répéter les points 2 et 3 jusqu'à ce que tous les individus soient réunis dans la même classe.

R La distance choisir n'est pas nécessairement la distance euclidienne pour la CHA.

a Critère d'agrégation (linkage)

Le critère d'agrégation définit comment mesurer la distance entre deux classes. Plusieurs critères d'agrégation sont possibles :

- le saut minimum : choisir distance entre les deux échantillons les plus proches des deux classes.
- le saut maximum : choisir la distance entre les deux échantillons les plus éloignés.

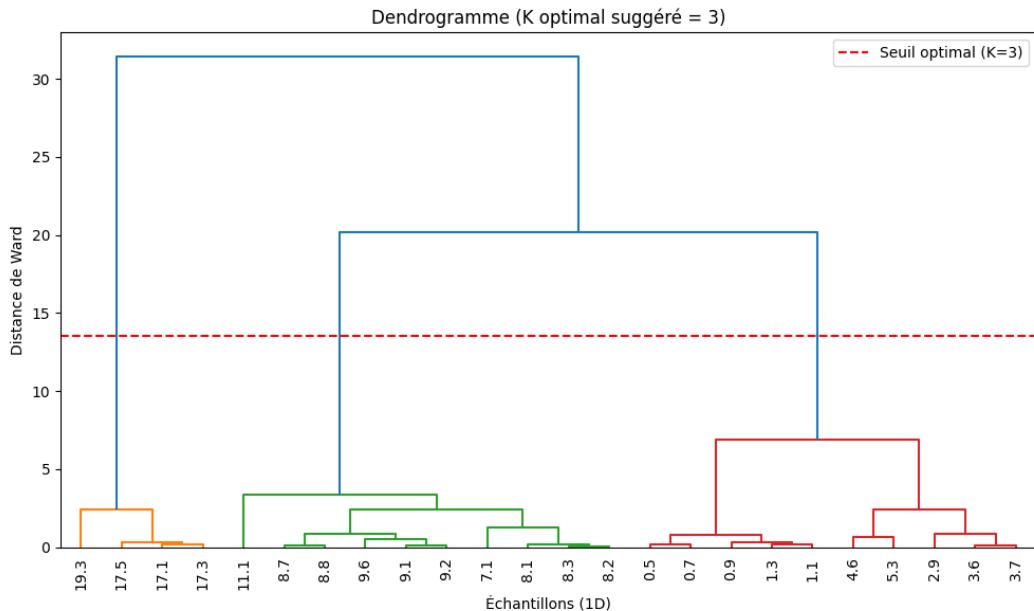


FIGURE 7 – Dendrogramme d'une CHA utilisant la distance de Ward comme critère d'agrégation. Les échantillons sont des notes entre 0 et 20. On souhaite faire des groupes de niveau.

- la moyenne : choisir la moyenne de toutes les distances entre les échantillons des deux classes.
- la méthode de Ward : choisir le minimum de l'augmentation de l'inertie intra-classe pour créer des classes les plus compactes possibles.

b Dendrogramme

Le résultat d'une CHA est généralement représentée sur un dendrogramme comme sur la figure 7.

Le résultat d'une CHA est représenté graphiquement par un dendrogramme ou *arbre de classification*. Sur cette figure, l'axe vertical représente la distance. Plus la branche qui relie deux classes est haute, plus ces classes sont différentes. Pour obtenir une partition finale, c'est-à-dire choisir le nombre de classes, l'utilisateur doit tracer une ligne horizontale à une certaine hauteur de l'arbre. Les branches coupées déterminent alors les classes finales.

R Contrairement à la méthode des K-moyennes, il n'est pas nécessaire de choisir le nombre de classes à l'avance. C'est le calcul ou l'observation du dendrogramme qui aide à décider du nombre optimal de classes.

G Arbres de décision --> HORS PROGRAMME

a Définition

Un **arbre de décision** est le résultat d'un algorithme d'apprentissage automatique qui permet de faire de la classification et de la régression. L'arbre est un système prédictif.

Soit un ensemble de données d'apprentissage $\mathcal{E} = \{(e_i, c_i)\}_{i=1}^N$, où $e_i \in \mathbb{R}^d$ est le vecteur de paramètres d'un échantillon et c_i est l'étiquette cible (valeur ou classe).

Un arbre de décision partitionne l'espace des caractéristiques \mathbb{R}^d en un ensemble de rectangles disjoints (ou hyper-rectangles) $\mathcal{R}_1, \dots, \mathcal{R}_M$.

- La **racine** contient l'ensemble de toutes les données.
- Un **nœud interne** applique un test sur un paramètre (par exemple $e_j < t$).
- Une **branche** et le sous-arbre sous-tendu par celle-ci représentent une partie des échantillons qui ont été séparés des autres par le test.
- Une **feuille** attribue une valeur ou une classe à la région correspondante.

Un arbre de décision peut être binaire ou n-aire selon l'algorithme de construction (cf. figure 8). On peut toujours transformer un arbre n-aire en un arbre binaire.

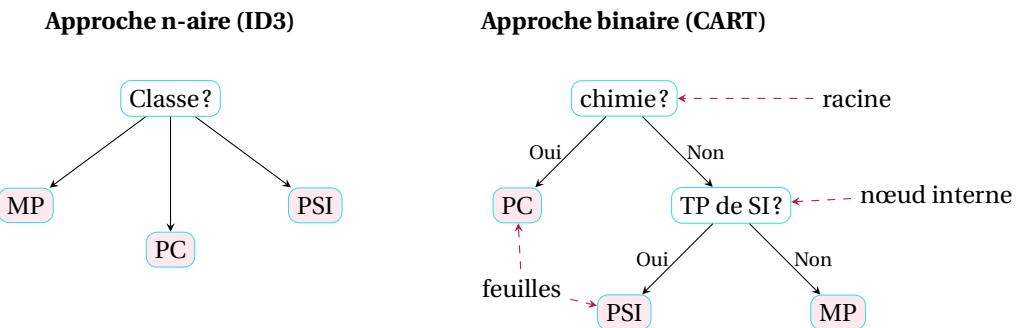


FIGURE 8 – Différents types d'arbre de décision : n-aire ou binaire

b Exemples d'application

Régression

Dans un problème de régression, la variable cible y est continue ($y \in \mathbb{R}$). L'arbre prédit une constante c_m pour chaque région \mathcal{R}_m de l'espace paramétrique. La fonction de prédiction $f(e)$ s'écrit :

$$f(e) = \sum_{m=1}^M c_m \cdot \mathbb{1}_{x \in \mathcal{R}_m}(e) \quad (2)$$

Généralement, c_m est la moyenne des valeurs cibles des données d'entraînement tombant dans \mathcal{R}_m .

Le symbole $\mathbb{1}_{x \in \mathcal{R}_m}$ désigne la **fonction indicatrice**. Elle agit comme un interrupteur logique. Pour une condition logique \mathcal{P} donnée, elle est définie par :

$$\mathbb{1}_{\mathcal{P}}(e) = \begin{cases} 1 & \text{si } \mathcal{P}(e) \text{ est vraie} \\ 0 & \text{si } \mathcal{P}(e) \text{ est fausse} \end{cases} \quad (3)$$

Le rôle de l'indicatrice est de **sélectionner** la bonne constante. Puisque les régions $\mathcal{R}_1, \dots, \mathcal{R}_M$ forment une **partition** (elles sont disjointes), un point e donné appartient à une et une seule région R_k . Par conséquent :

- pour l'indice $m = k$, la condition $e \in R_k$ est vraie, donc $\mathbb{1}_{(x \in \mathcal{R}_k)}(e) = 1$.
- pour tout autre indice $m \neq k$, la condition $e \in R_m$ est fausse, donc $\mathbb{1}_{(x \in \mathcal{R}_m)}(e) = 0$.

La somme se simplifie alors pour ne garder que le terme pertinent :

$$f(x) = 0 + \dots + c_k \cdot 1 + \dots + 0 = c_k$$

Cela permet d'écrire une fonction définie par morceaux sous la forme compacte d'une somme unique.

Régression par arbre de décision (une seule variable)

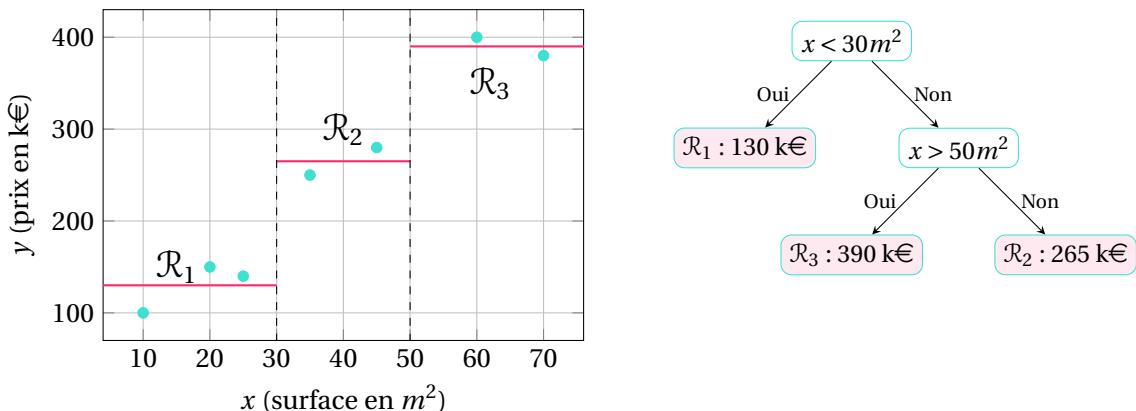


FIGURE 9 – Exemple de régression avec un arbre de décision : prédiction du prix d'un logement en fonction de la surface. L'espace est divisé en trois régions zones \mathcal{R}_1 , \mathcal{R}_2 et \mathcal{R}_3 par les seuils $x = 30$ et $x = 50$. La valeur prédite en rouge est la moyenne locale de chaque région.

Classification

Dans un problème de classification, y prend une valeur discrète $k \in \{1, \dots, K\}$. Chaque région \mathcal{R}_m est assignée à la classe la plus fréquente parmi les échantillons qu'elle contient.

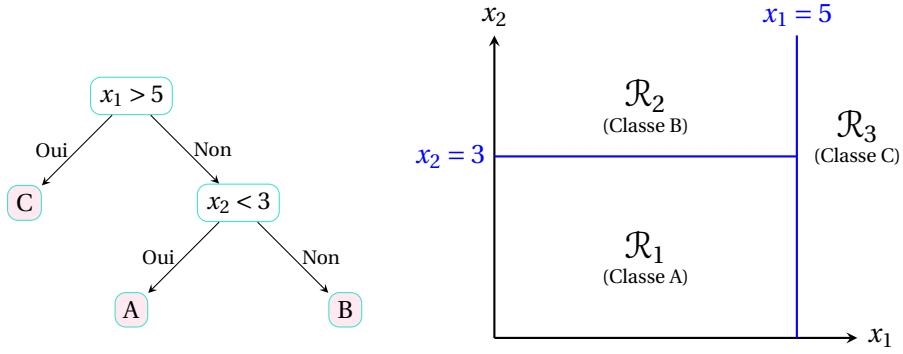


FIGURE 10 – Illustration de la structure d'un arbre de décision et de la partition de l'espace paramétrique associée

c Construction d'un arbre de décision

La construction d'un arbre optimal est un problème complexe. En pratique, on utilise une approche gloutonne et récursive. À chaque étape, pour un nœud \mathcal{N} donné contenant un sous-ensemble de données S , on cherche le couple (paramètre a , seuil t) qui *divise au mieux* S en deux sous-ensembles S_g et S_d .

Comment définir mathématiquement ce *au mieux*, c'est-à-dire comment maximiser l'homogénéité ou l'ordre dans les nœuds et les feuilles. Il existe plusieurs métriques utilisables pour diviser au mieux parmi lesquelles on trouve l'indice de Gini et le gain d'information.

d Métriques pour la division *au mieux*

Soit p_k la proportion d'éléments de la classe k dans un nœud donné \mathcal{N} contenant un sous-ensemble des données S .

$$p_k = \frac{\text{Nombre d'éléments de la classe } k}{\text{Nombre total d'éléments dans } S} = \frac{1}{|S|} \sum_{(e,c) \in S} \mathbb{1}_{x=k}(c) \quad (4)$$

Indice de Gini

L'indice de Gini mesure la probabilité qu'un élément choisi aléatoirement dans le nœud soit mal classifié s'il était étiqueté selon la distribution des classes du nœud. Si on suppose qu'il existe K classes dans l'ensemble des données :

$$G(S) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2 \quad (5)$$

- Si $G(S) = 0$ alors le nœud est pur : tous les échantillons sont de la même classe.
- Si $G(S)$ est maximal, alors les classes sont équitablement réparties : l'impureté est maximale.

Gain d'information

L'entropie $H(S)$ est l'espérance mathématique de l'information portée par les échantillons :

$$H(S) = - \sum_{k=1}^K p_k \log_2(p_k) = \sum_{k=1}^K p_k \underbrace{(-\log_2 p_k)}_{\text{Information de la classe } k} \quad (6)$$

R

Le logarithme porte l'information d'une classe $(-\log_2 p_k)$ et ce n'est pas arbitraire :

- l'information est inversement proportionnelle à la probabilité.
 - Un événement certain ($p = 1$) n'apporte **aucune information**. On veut donc une fonction qui vaut 0 en 1 : $\log(1) = 0$ convient.
 - Un événement rare ($p \rightarrow 0$) apporte une **énorme information**, c'est-à-dire une grande surprise. On veut donc une fonction qui tend vers $+\infty$: $-\log(p)$ convient.
- L'information est additive :
 - Si on observe deux événements **indépendants** A et B de probabilités p_A et p_B , l'information totale reçue devrait être la somme des informations individuelles. Or l'indépendance des événements se traduit par un produit : $P(A \cap B) = p_A \times p_B$.
 - On cherche donc une fonction f telle que $f(p_A \times p_B) = f(p_A) + f(p_B)$. La seule fonction continue (à un facteur multiplicatif près) qui transforme un produit en somme est le **logarithme**.
- L'unité de l'information est le bit, car, en informatique, on code l'information en binaire.
 - Une distribution uniforme avec N possibilités (par exemple $N = 2$ pour une pièce, $N = 8$ pour un octet) est telle que la probabilité de chaque événement est $p = 1/N$. Le nombre de bits nécessaires pour encoder ces N états est $\log_2(N)$.
 - Or, $\log_2(N) = \log_2(1/p) = -\log_2(p)$. L'entropie représente donc le **nombre de bits** nécessaires pour encoder la classe d'un échantillon tiré au hasard dans S .

Ces éléments sont illustrés par la figure 11.

Pour choisir la meilleure séparation des échantillons, on calcule donc le **gain d'information** $IG(S, a)$:

$$IG(S, a) = H(S) - H(S | a) \quad (7)$$

où

$$H(S | a) = \sum_{v \in \mathcal{V}(a)} \frac{|T_a(v)|}{|S|} \cdot H(T_a(v)) \quad (8)$$

désigne l'entropie conditionnelle de S connaissant le paramètre a . $T_a(v) = \{\mathbf{e} \in S \mid e_a = v\}$ désigne l'ensemble des échantillons pour lesquels le paramètre a vaut v . $\mathcal{V}(a)$ est l'ensemble des valeurs possibles du paramètre a .

Un algorithme qui construit un arbre de décision choisit l'attribut a qui maximise ce gain d'information.

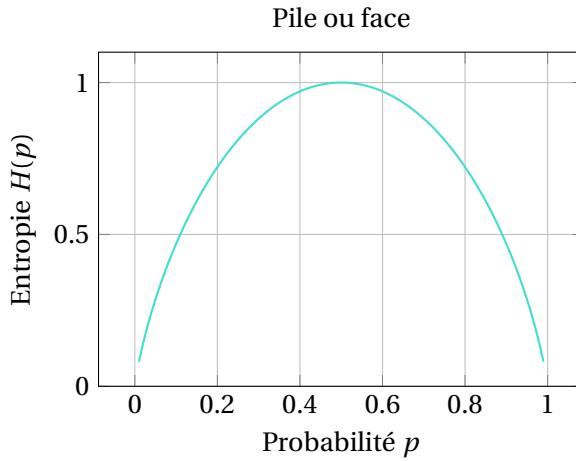


FIGURE 11 – L'entropie pour une classification pile ou face de probabilité pile p et face $1 - p$: $H(S) = -p \log_2 p - (1 - p) \log_2 (1 - p)$. L'entropie est maximale lorsque les classes sont parfaitement équilibrées ($p = 0,5$) et nulle lorsque le groupe est pur ($p = 0$ ou $p = 1$)

e Algorithme ID3

ID3 (Iterative Dichotomiser 3) utilise l'**entropie** et le gain d'Information. Il est conçu principalement pour des paramètres discrets. Il est glouton non optimal et sa complexité est en $O(Pnd)$, où P est la profondeur de l'arbre, n le nombre d'échantillons et d la dimension de l'espace paramétrique.

Algorithme 3 Iterative Dichotomiser 3

```

1: Fonction ID3( $S, A$ ) ▷  $A$  est l'ensemble des paramètres des échantillons
2:   si tous les éléments de  $S$  sont de la même classe alors
3:     renvoyer une feuille avec cette classe
4:   si  $A$  est vide alors
5:     renvoyer une feuille avec la classe majoritaire de  $S$ 
6:   Choisir l'attribut  $a \in A$  qui maximise  $IG(S, a)$ 
7:   Créer un nœud  $\mathcal{N}_a$  étiqueté  $a$ 
8:   pour chaque valeur possible  $v$  de  $a$  répéter
9:      $T_v \leftarrow \{e \in S \mid e_a = v\}$ 
10:    si  $T_v$  est vide alors
11:      Ajouter une feuille avec la classe majoritaire de  $S$  à  $\mathcal{N}_a$ 
12:    sinon
13:      Ajouter le fils  $ID3(S_v, A \setminus \{a\})$  à  $\mathcal{N}_a$ 
renvoyer  $\mathcal{N}_a$ 

```

f Algorithme CART

CART (Classification And Regression Trees) génère uniquement des **arbres binaires** mais s'applique à des paramètres discrets et continus. CART utilise :

- l'indice de Gini comme métrique pour la classification et l'erreur quadratique moyenne pour la régression.
- un seuil t de séparation pour une variable continue e_a , créant $e_a \leq t$ et $e_a > t$.
- une fonction de coût d'un nœud est la somme pondérée des impuretés des fils gauche et droit :

$$C(a) = \frac{|S_g|}{|S|} G(S_g) + \frac{|S_d|}{|S|} G(S_d)$$

CART cherche à minimiser la fonction de coût pour construire l'arbre de décision. Il est glouton non optimal et sa complexité est en $O(Pdn \log n)$, où P est la profondeur de l'arbre, n le nombre d'échantillons et d la dimension de l'espace paramétrique.

Algorithme 4 Algorithme CART (Construction d'arbre binaire)

```

1: Fonction CART( $S, P$ )
2:   Entrées :  $S$  (données),  $P$  (profondeur), MaxP (prof. limite), MinS (cardinal min de  $S$ )
3:   si  $G(S) = 0$  ou  $P \geq \text{MaxP}$  ou  $|S| < \text{MinS}$  alors                                 $\triangleright$  Limites à la récursivité
4:     renvoyer Feuille avec la classe majoritaire de  $S$ 
5:   MeilleurScore  $\leftarrow \infty$ 
6:   MeilleureSep  $\leftarrow \emptyset$ 
7:   pour chaque paramètre  $a$  répéter
8:     pour chaque seuil possible  $t$  pour  $a$  répéter
9:       Séparer  $S$  en  $S_g$  ( $e_a \leq t$ ) et  $S_d$  ( $e_a > t$ )
10:      Coût  $\leftarrow C(a)$ 
11:      si Coût < MeilleurScore alors
12:        MeilleurScore  $\leftarrow$  Coût
13:        MeilleureSep  $\leftarrow (a, t, S_g, S_d)$ 
14:      si MeilleureSep n'est pas vide alors
15:         $F_g \leftarrow \text{CART}(S_g, P + 1)$ 
16:         $F_d \leftarrow \text{CART}(S_d, P + 1)$ 
17:        renvoyer Nœud( $e_a \leq t, F_g, F_d$ )
18:      sinon
19:        renvoyer Feuille avec la classe majoritaire de  $S$ 

```

g Avantages et inconvénients des arbres de décision

Les arbres de décisions sont intéressants car :

- ils sont interprétables par un être humain,
- la phase de prédiction est rapide,

- les données n'ont pas besoin d'être normalisées.

Par contre,

- la structure de l'arbre peut varier radicalement en cas de modification des données,
- le sur-apprentissage est possible s'il est très profond.

h Des arbres dans la forêt

Pour pallier la forte variance des arbres de décision, on utilise les forêts d'arbres décisionnels (Random Forest). Le principe de cette approche ensembliste est le Bagging (Bootstrap Aggregating) :

1. on entraîne B arbres de décision indépendants sur des sous-échantillons aléatoires des données,
2. lors de la construction de chaque noeud, on ne considère qu'un sous-ensemble aléatoire des caractéristiques,
3. la prédiction finale est la moyenne (pour la régression) ou le vote majoritaire (pour la classification) des B arbres.

Cela permet de réduire drastiquement la variance et le risque de sur-apprentissage.