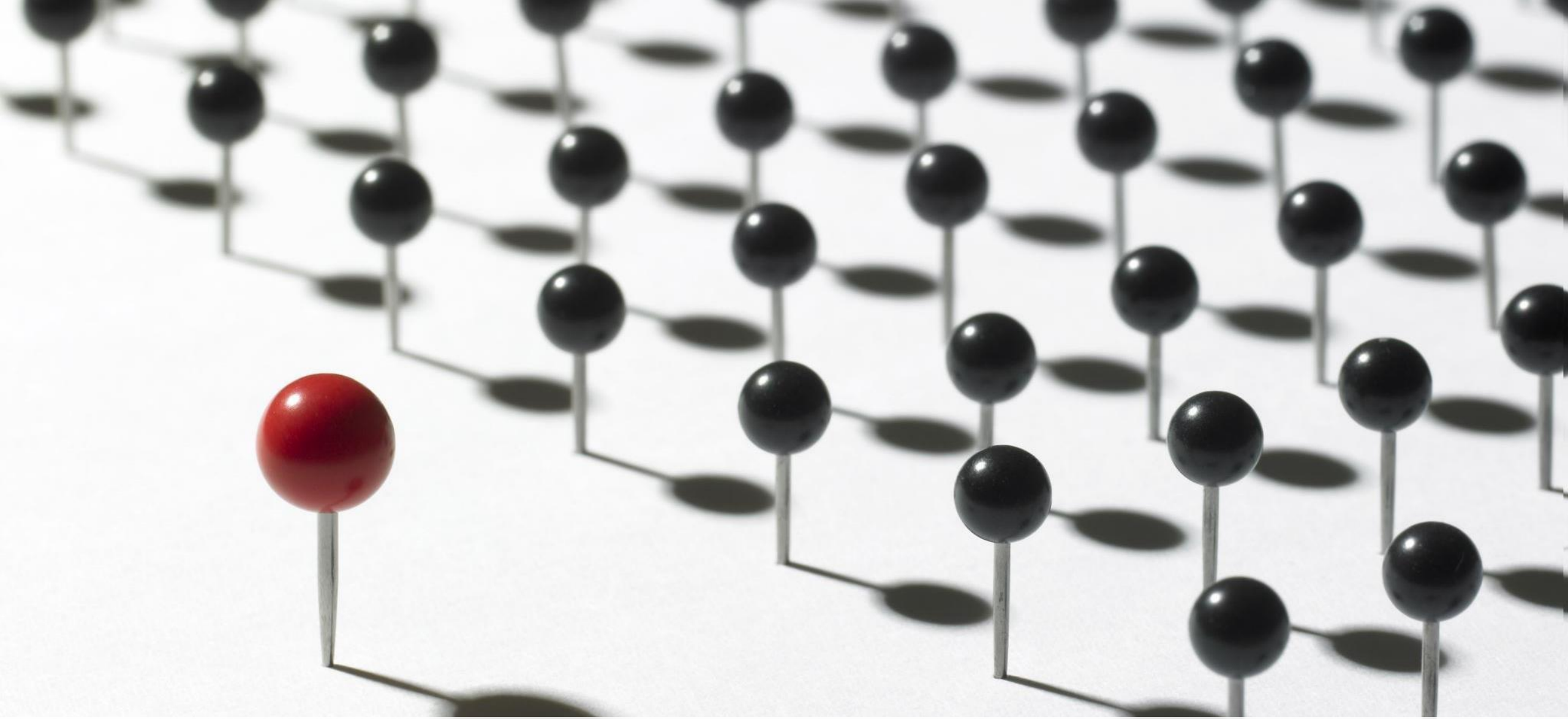


Classification

Pr.Idrissi Zouggari Nadia

Introduction

- Dans de nombreux problèmes réels, nous cherchons à attribuer automatiquement une étiquette (classe) à un nouvel objet en se basant sur des exemples déjà étiquetés.
- Exemples :
  - ❑ Identifier si une plante est toxique ou comestible
  - ❑ Déterminer si un email est spam ou non spam
  - ❑ Reconnaître un chiffre manuscrit (0–9)
  - ❑ Détecter une transaction bancaire frauduleuse ou normale

Introduction

Classification binaire vs multi classe:

1. **Classification binaire**

   Deux classes seulement : 0/1

   Exemples :

   - Toxique / Comestible
   - Frauduleuse / Normale
   - Malade / Sain

2. **Classification multi classe**

   Plusieurs classes (3, 10, 100…)

   Exemples :

   - Reconnaissance de chiffres manuscrits (10 classes)
   - Catégorisation d'images (chien, chat, oiseau…)
   - Type de plantes (arbre, herbe, fleur, etc.)

Introduction

- **Comment le modèle décide-t-il à quelle classe appartient un nouvel exemple ?**

  Il existe plusieurs stratégies décisionnelles :

  - **Basées sur la proximité (distance)**

    C'est la logique du **K-Nearest Neighbors (KNN):** Le modèle regarde les points les plus proches et vote.

  - **Basées sur la séparation géométrique**

    C'est la philosophie du **Support Vector Machine (SVM):** Le modèle cherche l'hyperplan qui sépare le mieux les classes.

  - **Basées sur la prise de décision hiérarchique**

    C'est la logique des **arbres de décision (Decision Trees):** Le modèle pose une succession de conditions (IF… THEN…) pour trier les exemples.

KNN – K Nearest Neighbors

KNN – K Nearest Neighbors

Dans le paysage des algorithmes de Machine Learning, KNN appartient à la famille :

- Apprentissage supervisé
- Méthode non paramétrique
- Lazy learning (apprentissage paresseux)

KNN ne construit aucun modèle explicite pendant l'entraînement.

KNN – K Nearest Neighbors

- Considérant une data set des plantes basée sur deux caractéristiques (features) :
  - **Acidité de la sève (pH) :** Mesurée de 0 à 14.
  - **Diamètre du fruit (cm) :** Mesuré en centimètres.

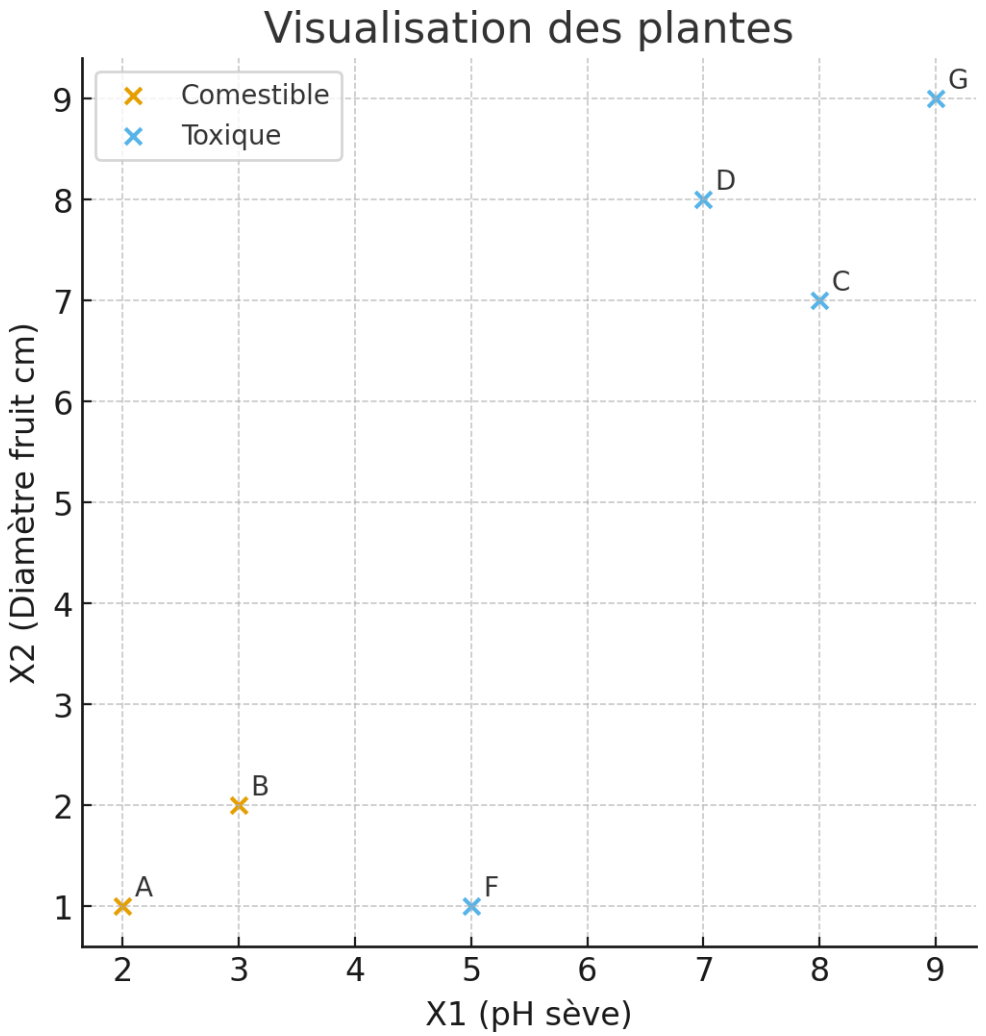
Plante (ID)	**X1 (pH sève)**	**X2 (Diam. fruit cm)**	**Y (Classe)**
A	2	1	**Comestible** (0)
B	3	2	**Comestible** (0)
C	8	7	**Toxique** (1)
D	7	8	**Toxique** (1)
F	5	1	**Toxique** (1)
G	9	9	**Toxique** (1)

Objectif : Classifier une nouvelle plante : Toxique ou Comestible.

KNN – K Nearest Neighbors

Étape 1 : Chargement et Visualisation des Données

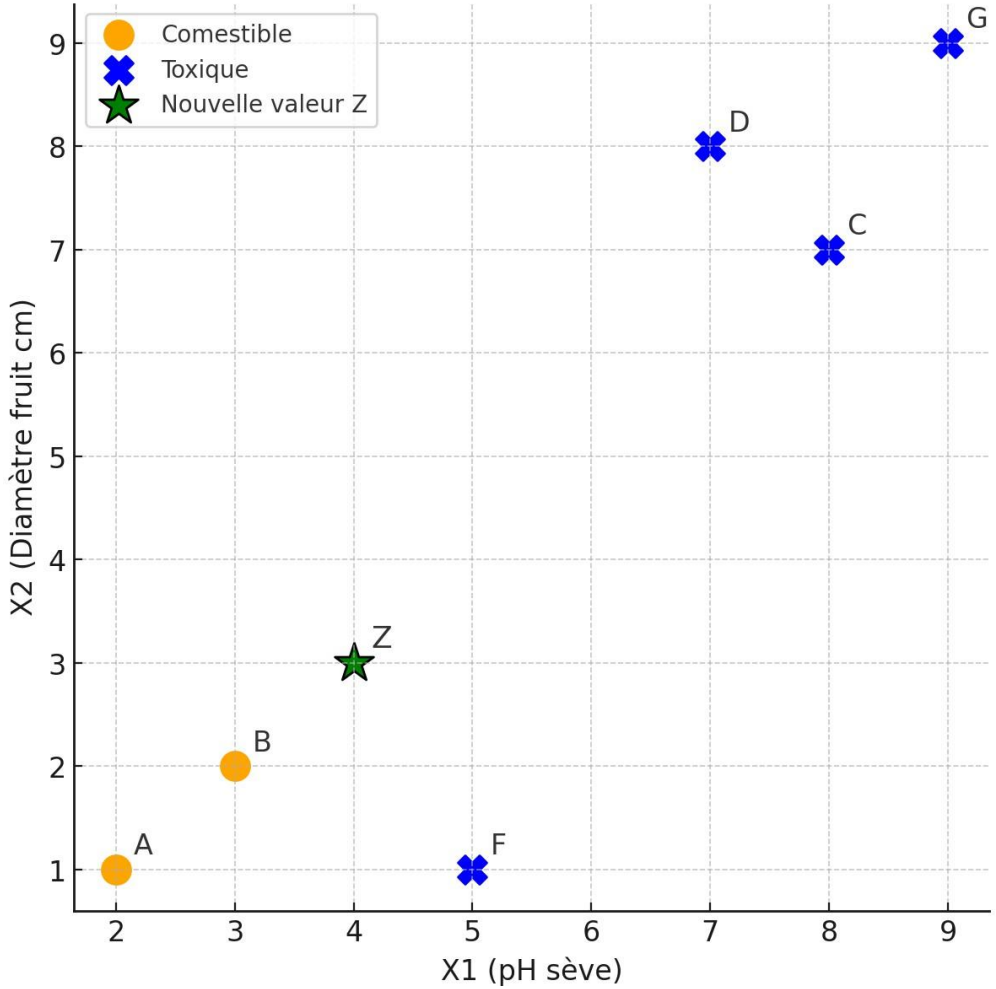
- On voit un groupe de plantes comestibles en bas à gauche (pH bas, petit diamètre).
- On voit un "groupe" de plantes toxiques en haut à droite (pH élevé, grand diamètre).
- La plante F (5, 1) est un peu bizarre, elle est proche des comestibles mais classée toxique (c'est le "bruit").



KNN – K Nearest Neighbors

Étape 2 : Nouvelle Plante

Plante Z: **pH=4, Diam=3**
Le Modèle doit décider : Comestible (0) ou Toxique (1) ?

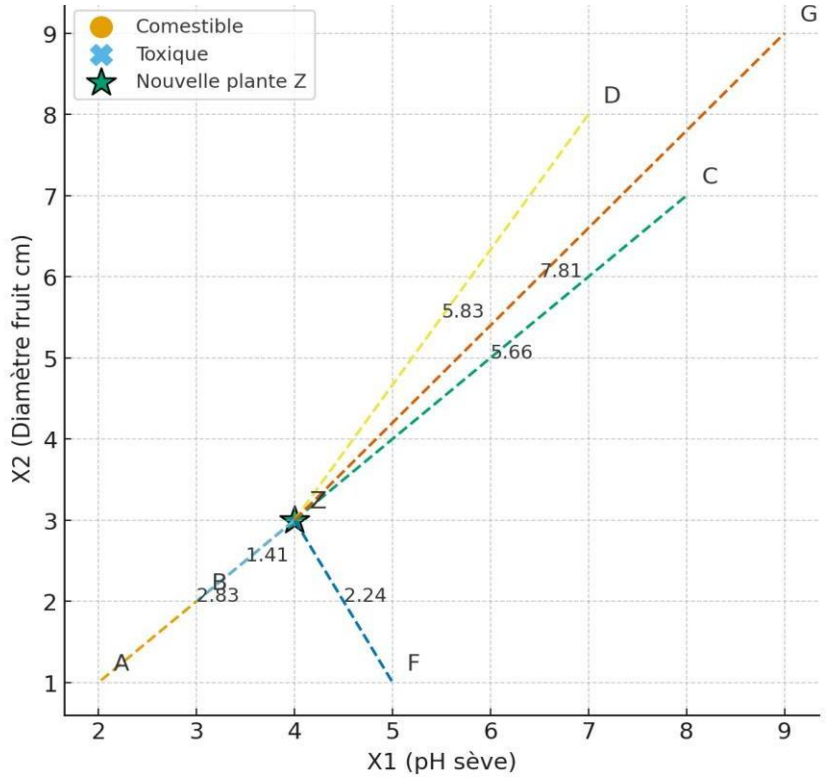


KNN – K Nearest Neighbors

Étape 2 : Calculs des Distances

Le K-NN n'apprend rien. Il va simplement "comparer" Z à *toutes* les plantes qu'il connaît. Il calcule la **distance**

Euclidienne entre Z et chaque plante (A, B, C, D, F, G). $D(Z, A) = \sqrt{(Z_{PH} - A_{PH})^2 + (Z_{Diam} - A_{Diam})^2}$

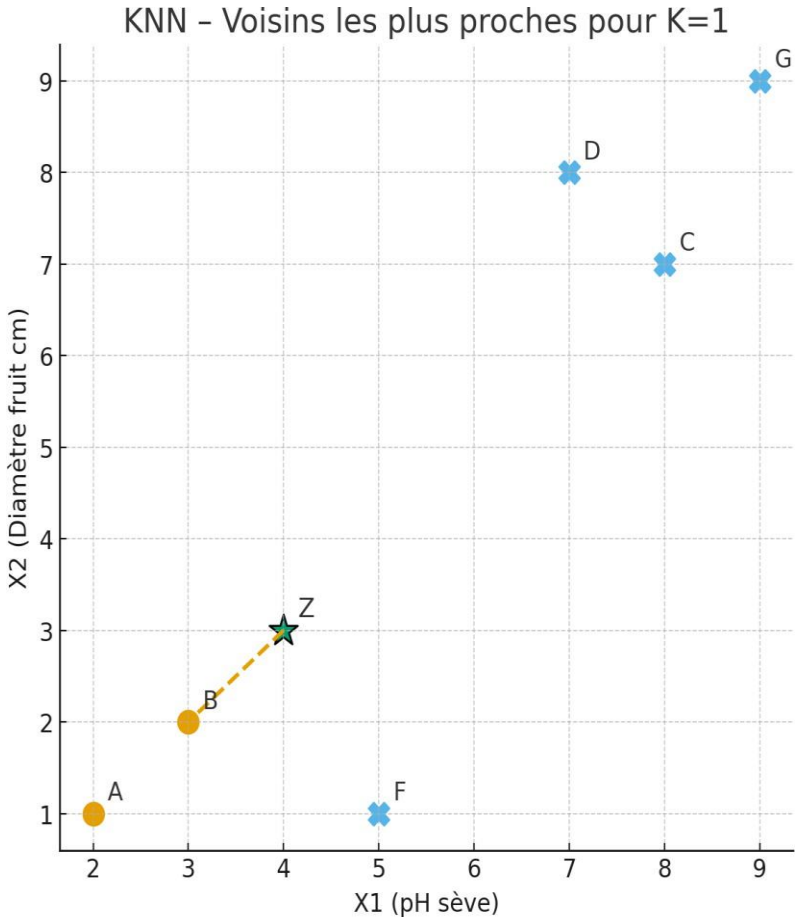


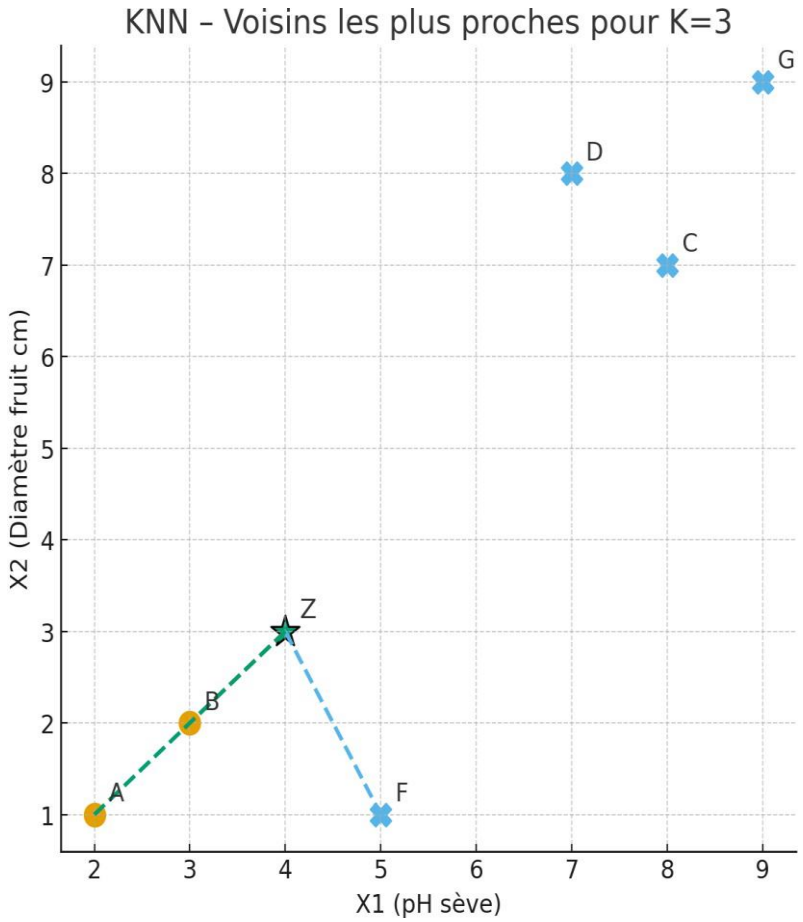
Classement des voisins (du plus proche au plus lointain) :

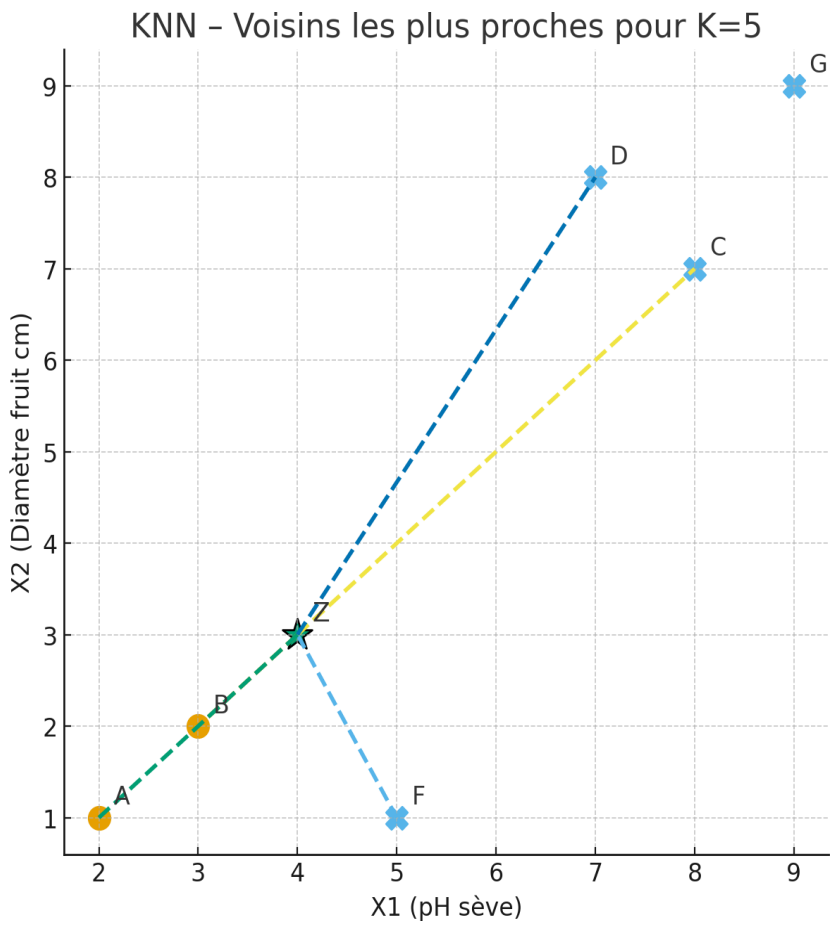
1. Plante B (Dist = 1.41) - Comestible
2. Plante F (Dist = 2.24) - Toxique
3. Plante A (Dist = 2.83) - Comestible
4. Plante C (Dist = 5.66) - Toxique
5. Plante D (Dist = 5.83) - Toxique
6. Plante G (Dist = 7.81) - Toxique

KNN – K Nearest Neighbors

Étape 3 : Le Vote







KNN – K Nearest Neighbors

Étape 3 : Le Vote

Quelle valeur de K choisir?

- **Cas 1 : Pour K=1**
  - On regarde son **1** plus proche voisin.
  - Voisin : Plante B (Comestible).
  - Vote : 1 (Comestible) vs 0 (Toxique).
  - **Décision : COMESTIBLE.**
- **Cas 2 : Pour K=3**
  - On regarde ses **3** plus proches voisins.
  - Voisins : Plante B (Comestible), Plante F (Toxique), Plante A (Comestible).
  - Vote : 2 (Comestible) vs 1 (Toxique).
  - **Décision: COMESTIBLE.**
- **Cas 3 : Pour K=5**
  - On regarde ses **5** plus proches voisins.
  - Voisins : B (Comestible), F (Toxique), A (Comestible), C (Toxique), D (Toxique).
  - Vote : 2 (Comestible) vs 3 (Toxique).
  - **Décision : TOXIQUE.**

Constat : Le choix de K change radicalement la décision.

KNN – K Nearest Neighbors

Étape 4 : L'optimisation du K (Grid Search)

On ne choisit jamais K au hasard. On l'optimise.

Supposant que le data set a 1000 plantes étiquetées:

1. Répartition des données (Train/Test Split):
   - **80% (800 plantes)** ⟶ **Jeu d'Entraînement (Training Set) :** C'est la "mémoire" du K-NN.
   - **20% (200 plantes)** ⟶ **Jeu de Test (Test Set)**

2. Optimisation (Grid Search) : On va tester méthodiquement plusieurs valeurs de K (généralement les impairs : 1, 3, 5, 7, 9...) et voir laquelle donne le meilleur score *sur le Jeu de Test*

KNN – K Nearest Neighbors

Étape 4 : L'optimisation du K (Grid Search)

- Test avec K=1 :

On fait passer les 200 plantes du Jeu de Test. Le modèle K-NN prédit leurs classes. On compare aux vraies étiquettes.
Résultat : 170 prédictions correctes sur 200. **Précision** = 85%. (Il y a du bruit, donc K=1se trompe souvent).

- Test avec K=3:

On refait le test avec K=3.
Résultat : 188 prédictions correctes sur 200. Précision = 94%.

- Test avec K=5 :

On refait le test avec K=5.
Résultat : 192 prédictions correctes sur 200. Précision = 96%.

- Test avec K=7 :

On refait le test avec K=7.
Résultat : 191 prédictions correctes sur 200. Précision = 95.5%.

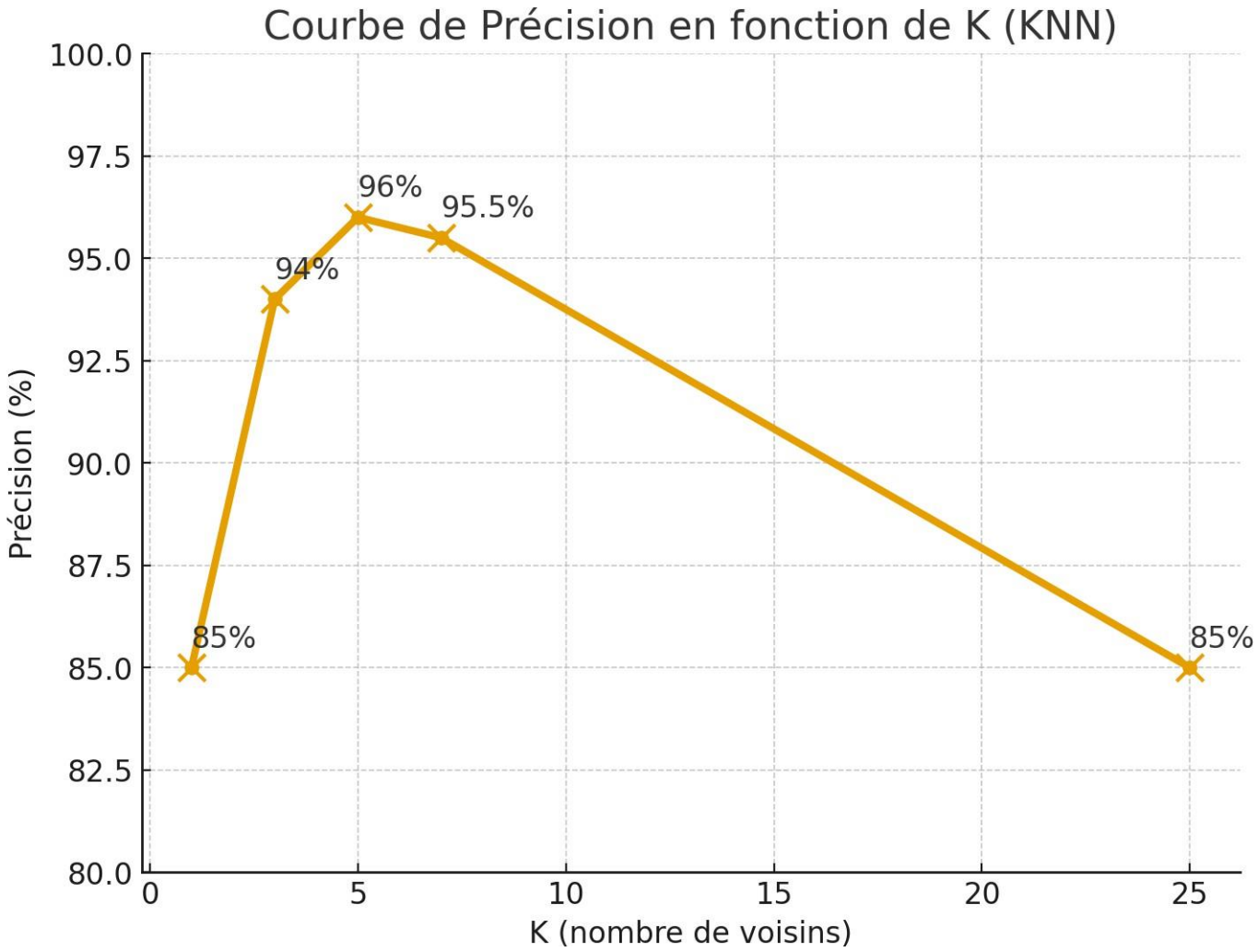
- Test avec K=25 :

On refait le test avec K=25 (un K trop grand).
Résultat : 170 prédictions correctes sur 200. Précision = 85%. (Le modèle devient trop "lisse" et ignore les détails locaux).

KNN – K Nearest Neighbors

Étape 4 : L'optimisation du K (Grid Search)

Le pic de la courbe est à **K=5**. C'est le meilleur compromis Biais/Variance



KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

I. Matrice de confusions:

La Matrice de Confusion est un tableau qui compare les prédictions de votre modèle (K=5) aux *vraies* étiquettes (la réalité).

Pour notre exemple binaire (Toxique vs Comestible), nous avons 4 cas possibles pour chaque plante du jeu de test :

- **Positif** = "Toxique" (Classe 1)
- **Négatif** = "Comestible" (Classe 0)

Prédiction

Réel

	Prédit : Toxique	**Prédit : Comestible**
Réel : Toxique	**TP** (Vrai Positif)	**FN** (Faux Négatif)
Réel : Comestible	**FP** (Faux Positif)	**TN** (Vrai Négatif)

KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

I. Matrice de confusions: Exemple

	Prédit : Toxique	**Prédit : Comestible**
Réel : Toxique	**TP** 94	**FN** 6
Réel : Comestible	**FP** 2	**TN** 98

exemple où K=5 le meilleur score sur 200 plantes de test est 192 prédictions correctes.

Imaginons que les 8 erreurs se répartissent ainsi :

- 6 Faux Négatifs (**FN**) : 6 plantes toxiques on été classifié comme "Comestibles". (Très grave)
- 2 Faux Positifs (**FP**) : 2 plantes comestibles on été classifié comme "Toxiques". (Moins grave)

les 192 succès se répartissent ainsi (par exemple, s'il y avait 100 toxiques et 100 comestibles) :

- **TP = 94** (Il a bien trouvé 94 plantes toxiques)
- **TN = 98** (Il a bien trouvé 98 plantes comestibles)

Total = 94 + 98 + 6 + 2 = 200 plantes.

KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

II. L'Exactitude (Accuracy)

C'est la métrique la plus simple : "Quel est le ratio de prédictions correctes ?"

$$ACCURACY = \frac{Predictions\ correctes}{Total\ des\ Prediction} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Calcul :** (94 + 98) / 200 = 192 / 200
- **Résultat : 0.96 (soit 96%)**

KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

II. La Précision (Precision)

C'est la métrique qui répond à la question : "Parmi les plantes prédites comme 'Toxiques', combien l'étaient vraiment ?"

$$\text{PRECISION} = \frac{TP}{TP+FP}$$

- **Calcul :** 94 / (94 + 2) = 94 / 96
- **Résultat :0.979 (soit 97.9%)**

Le modèle classifie 97.9% des plante toxique comme "Toxique"

La précision indique la fiabilité des prédictions positives.
Plus elle est élevée, moins le modèle fait de fausses alertes

KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

Ⅲ. Le Rappel (Recall)

C'est la métrique qui répond à la question : "Sur toutes les plantes réellement Toxiques qui existent, combien on a trouvé ?"

$$\text{RECALL} = \frac{TP}{TP+FN}$$

- **Calcul :** 94 / (94 + 6) = 94 / 100
- **Résultat : 0.94 (soit 94%)**

Le modele a manqué 6% des plantes toxiques (les FN)

KNN – K Nearest Neighbors

Étape 4 : Métriques d'évaluations

IV. Conclusion

Même si l’accuracy atteint 96%, le rappel de 94% signifie que le modèle laisse passer 6% de plantes toxiques (Faux Négatifs).

Dans ce problème, c’est inacceptable : le rappel est donc la métrique prioritaire.
On préfère avoir plus de Faux Positifs (se tromper sur des plantes comestibles) plutôt que des Faux Négatifs.

Le choix de K doit donc se base sur la métrique qui correspond au risque métier, ici la minimisation des FN, et non seulement sur l’accuracy.

KNN – K Nearest Neighbors

LIMITES:

- Très lent pour grands datasets
- Sensible aux features non normalisées
- Sensible au curse of dimensionality

SVM:
Support Vector Machine

Introduction

Qu'est-ce qu'un SVM ?

Le SVM est un algorithme d'apprentissage supervisé utilisé principalement pour :

- **Classification** : Assigner des données à différentes catégories
- **Régression** : Prédire des valeurs continues (SVR)

Principe fondamental : Trouver le meilleur hyperplan qui sépare les données de différentes classes en maximisant la distance (marge) entre cet hyperplan et les points les plus proches de chaque classe.

Introduction

Applications pratiques

- **Vision par ordinateur :** Reconnaissance de visages, détection d'objets, classification d'images
- **Traitement du langage naturel :** Classification de textes, détection de spam, analyse de sentiments
- **Bioinformatique :** Classification de protéines, diagnostic médical, prédiction de maladies
- **Finance :** Prédiction de défauts de crédit, détection de fraudes

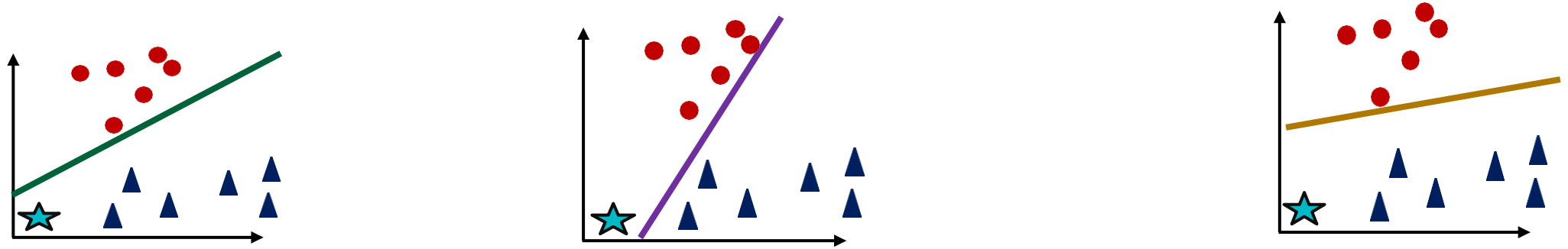
Pourquoi les SVM sont populaires ?

- Performance élevée même avec des données complexes
- Robustesse face au surapprentissage grâce à la maximisation de la marge
- Capacité à gérer des espaces de haute dimension
- Base théorique solide

Données LINÉAIRE

Principe

Classe 1 Classe 2 Nouvelle Donnée



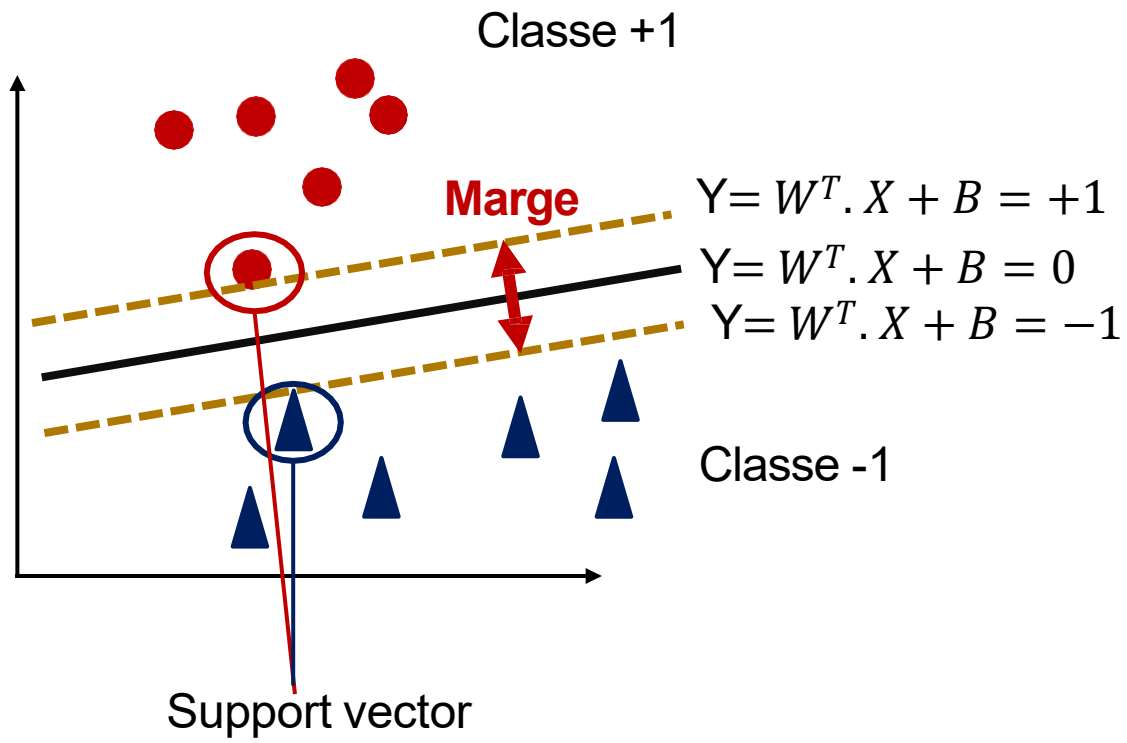
Observons ces trois graphiques représentant le même ensemble de données : des points rouges (Classe 1) et des triangles bleus (Classe 2) clairement séparables. Nous pouvons tracer différentes lignes qui séparent parfaitement ces deux groupes, et toutes atteignent 100% de précision sur les données d'entraînement.

Principe

Face à cette infinité de solutions possibles, le SVM adopte un principe élégant :
choisir l'hyperplan le plus éloigné des points les plus proches de chaque classe. C'est la ligne orange qui illustre cette solution optimale. Cette approche garantit une meilleure robustesse lorsqu'une nouvelle donnée arrive, comme l'étoile cyan représentée ici. Avec les lignes verte ou violette, une légère variation pourrait entraîner une erreur de classification, alors que la ligne orange offre une marge de sécurité maximale.

C'est le principe fondamental du SVM : maximiser la marge pour maximiser la capacité de généralisation du modèle sur des données non vues

Principe



$f(x) = W^T . X + B$

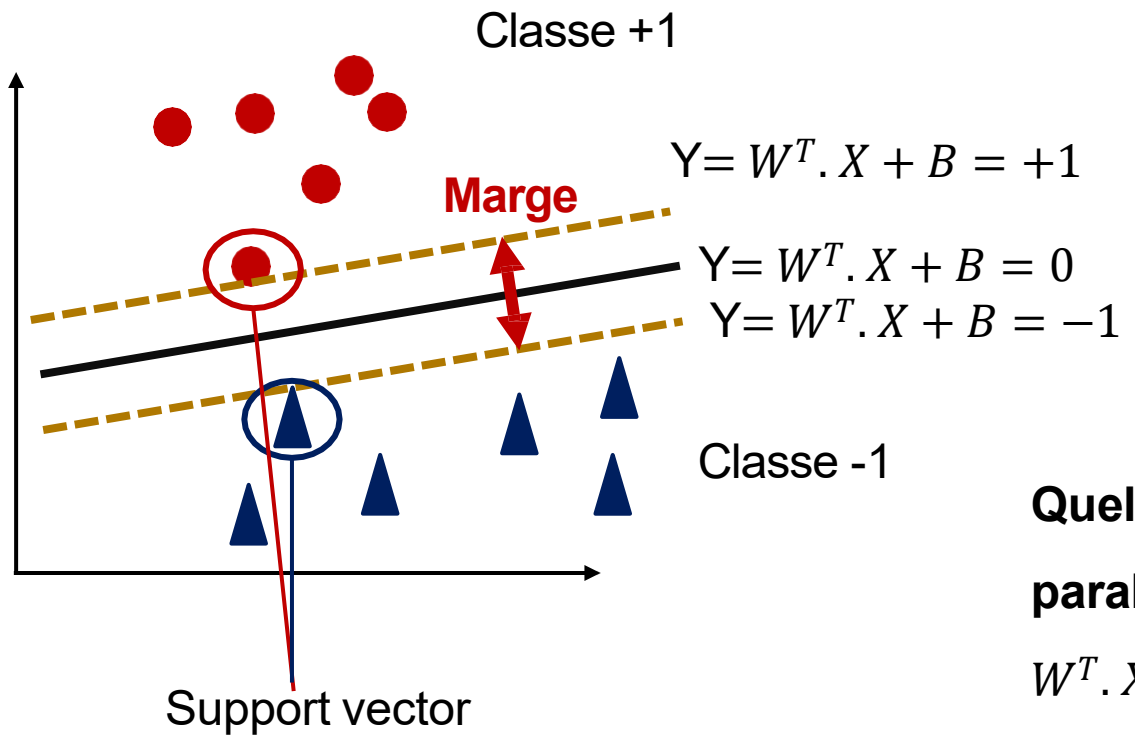
$f(x) > +1\ classe + 1$

$f(x) < -1\ classe\ -1$

$f(x)$=0 La frontière de décision

Principe

Nous avons établi visuellement que le SVM recherche l'hyperplan qui **maximise la marge** - cette zone de sécurité entre les deux classes. Mais pour optimiser mathématiquement, nous devons pouvoir **calculer** cette marge



$Y = W^T . X + B = +1$ Frontière gauche

$Y = W^T . X + B = 0$ Frontière de décision

$Y = W^T . X + B = -1$ Frontière droite

Quelle est la distance géométrique entre ces deux hyperplans parallèles ?

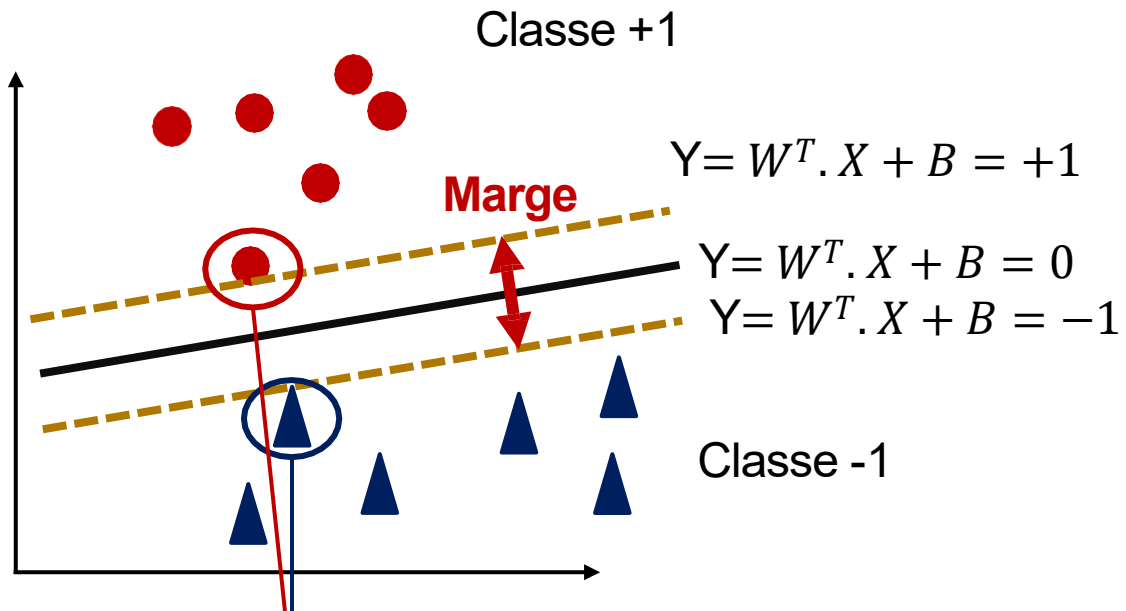
$W^T . X + B = +1$ et $W^T . X + B = -1$

➤ Cette distance = **LA MARGE** que nous voulons maximiser

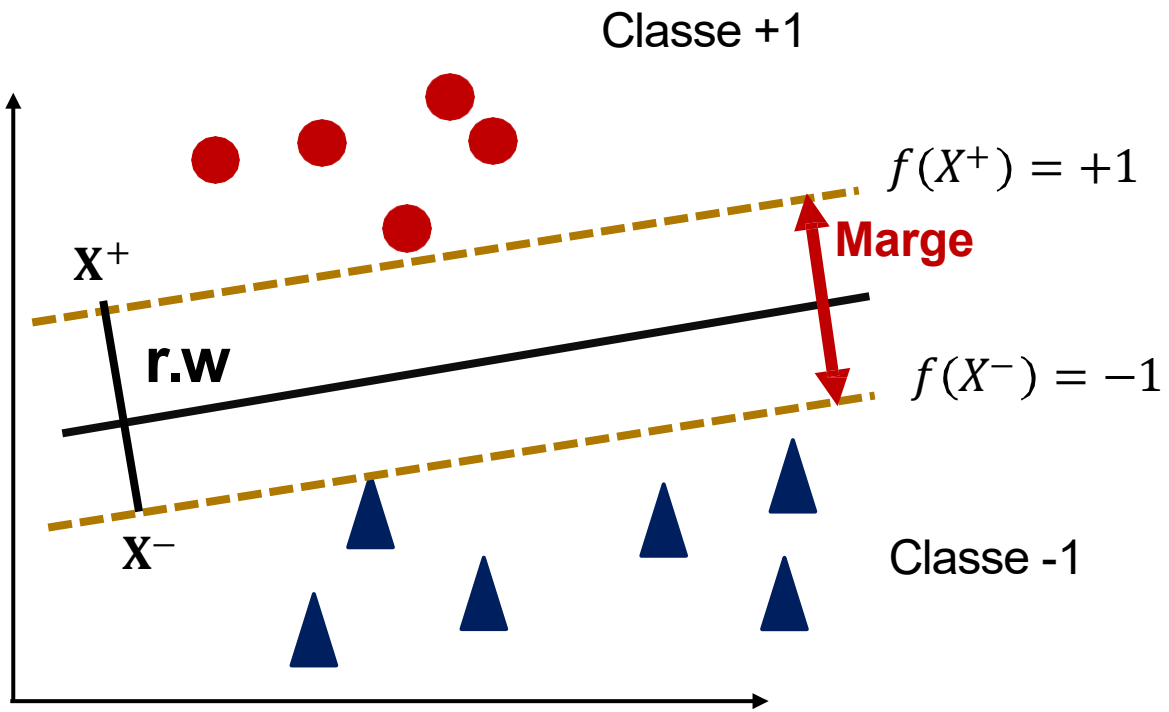
Principe

Pour calculer cette distance, nous allons :

1. Prendre deux points support vectors : x^+ (sur $W^T . X + B = +1$) et x^- (sur $W^T . X + B = -1$)
2. Calculer la distance entre ces deux points **le long de la direction perpendiculaire à l'hyperplan**
3. Cette direction perpendiculaire est donnée par le vecteur **w**
4. Exprimer la marge en fonction de **w** et **b**



Optimisation SVM: La MARGE RIGIDE



$$w = [w_1, w_2, \ldots \ldots . w_n]$$

$$x^+ = x^- + rw$$

Les deux points les plus proches de la marge satisfait:

$$w^T . x^- + b = -1$$

$$w^T . x^+ + b = +1$$

$$w^T . (x^- + rw) + b = +1$$

$$r||w||^2 + w^T . x^- + b = +1$$

$$r||w||^2 - 1 = +1$$

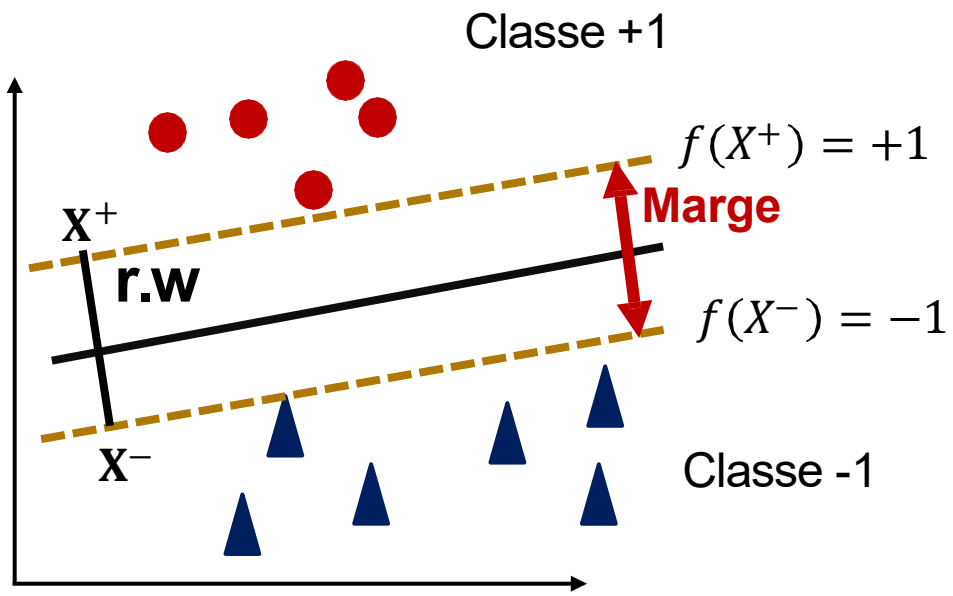
$$r||w||^2 = 2$$

$$r = \frac{2}{||w||^2}$$

Optimisation SVM: La MARGE RIGIDE

Objectif:

1. Une séparation des classes optimale
2. maximisation de la marge



Maximiser la marge:

$$r = \frac{2}{||w||^2}$$

$$\max r \sim \min ||w||^2$$

Avec une bonne classification:

$$w^T . x^i + B \leq -1 \qquad y^i = -1$$

$$w^T . x^i + B > 1 \qquad y^i = +1$$

2 Conditions

$$y^i \left(w^T . x^i + b\right) > +1$$

1 Condition

Optimisation SVM: La MARGE RIGIDE

Objectif

Trouver le w et b qui maximisent la marge tout en séparant correctement les classes.

Cela revient à :

Problème primal

$\min_{w,b} \frac{1}{2}||w||^2$ sous la contrainte $y^i\left(w^T.x^i + b\right) > +1$

Optimisation SVM: La MARGE RIGIDE LAGRANGE

On cherche a optimiser une fonction f(w) mais on n'a pas le droit de faire ce qu'on veut car il y a des contraintes a respecter g(w).

$$f(w) = \frac{1}{2}||w||^2 \qquad g(w) = y^i(w^t x^i + b) > 1$$

Le multiplicateur de Lagrange permet de transformer une optimisation avec contraintes en une optimisation simple.

$$L(w, b, \varphi) = \frac{1}{2}||w||^2 - \sum_i \varphi_i (y^i (w^t x^i + b)) - 1$$

$$\frac{\partial L}{\partial w} = 0 \qquad \frac{\partial L}{\partial b} = 0 \qquad \varphi \geq 0$$

Un multiplicateur de Lagrange sert à :

Intégrer une contrainte dans une fonction à optimiser pour pouvoir optimiser sans se soucier directement de la contrainte en forçant la solution à respecter la contrainte automatiquement

Optimisation SVM: La MARGE RIGIDE LAGRANGE

$$L(w, b, \varphi) = \frac{1}{2}||w||^2 - \sum_i \varphi_i (y_i (w^t x_i + b)) - 1$$

Minimiser L par rapport à w et b:

$$\frac{\partial L}{\partial w} = 0$$

$$\frac{\partial L}{\partial w} = w - \sum_i \varphi_i x_i y_i = 0$$

$$w = \sigma_i \varphi_i x_i y_i$$

$$\frac{\partial L}{\partial b} = 0$$

$$\frac{\partial L}{\partial b} = -\sigma_i \varphi_i y_i = 0$$

$$\sum_i \varphi_i y_i = 0$$

Optimisation SVM: La MARGE RIGIDE LAGRANGE

Contrainte transformée en Pénalité

Fonction objective à minimiser

$$L(w, b, \varphi) = \frac{1}{2}||w||^2 - \sum_i \varphi_i(y^i(w^t x^i + b)) - 1$$

Cas1: Contrainte Respecté

Si $y^i(w^t x^i + b) \geq 1$, alors $[y^i(w^t x^i + b) - 1] \geq 0$

- Le terme $-\varphi_i [y^i(w^t x^i + b) - 1]$ est **négatif ou nul** car $\varphi \geq 0$
  - Cela **diminue** la valeur du Lagrangien
  - Pas de pénalité : la contrainte est satisfaite

Cas1: Contrainte violée

Si $y^i(w^t x^i + b) < 1$, alors $[y^i(w^t x^i + b) - 1] < 0$

- Le terme $-\varphi_i \times$ (valeur négative) = $\varphi_i \times$ (valeur positive)
  - Cela **augmente** la valeur du Lagrangien
  - **C'est une pénalité** qui force l'optimisation à corriger w et b

Optimisation SVM: La MARGE RIGIDE LAGRANGE

$$L_{(w,b,\varphi)} = \frac{1}{2}||w||^2 - \sum_i \varphi_i(y^i(w^t x^i + b)) - 1$$

Le multiplicateur de Lagrange φ_i transforme chaque contrainte en une **pénalité conditionnelle :**

- Pénalité = 0 si la contrainte est respectée
- Pénalité > 0 si la contrainte est violée
- Cette pénalité force le système à corriger w et b pour satisfaire les contraintes

Optimisation SVM: La MARGE RIGIDE LAGRANGE

L'optimisation du Lagrangien impose une condition fondamentale :

Condition KKT :

$\varphi_i(y^i(w^t x^i + b) - 1) = 0$ pour tout point i

➤ Pour que ce produit soit nul, **au moins un des deux termes doit être égal à zéro**

Optimisation SVM: La MARGE RIGIDE LAGRANGE

Cas A - Points loin de la marge

Points bien classifiés :

- ❑ $y^i(w^t x^i + b) \geq 1$
- ❑ Donc $y^i(w^t x^i + b) - 1 \geq 0 \neq 0$

Pour satisfaire la condition KKT :

Il faut que:

- ❑ $\varphi_i = 0$

Conséquence :

- ➢ Ces points **ne contribuent PAS** à la solution finale.
- ➢ Ils peuvent être ignorés dans le calcul de w

Cas B - Points sur la marge (Support Vectors)

Support Vectors :

- ❑ $y^i(w^t x^i + b) = 1$
- ❑ Donc $y^i(w^t x^i + b) - 1 = 0$

La condition KKT est automatiquement satisfaite :

➢ Ces points peuvent avoir:

- ❑ $\varphi_i > 0$

Conséquence :

- ❑ Seuls ces points définissent w.
- ❑ **w = Σ_support vectors** $\varphi_i \, y^i \, x^i$

Optimisation SVM: La MARGE RIGIDE LAGRANGE

Conséquence remarquable: Parcimonie du modèle SVM

Exemple concret :

- Dataset : **10,000 points** d'entraînement
- Support vectors (points sur les frontières) : **50-200 points** seulement (1-2%)
- Points avec $\varphi_i = 0$: **~9,800 points** (98%)
  - **Impact :** Modèle final très compact

    Prédiction rapide et efficace

    Réduction de la complexité

Optimisation SVM: La MARGE RIGIDE LAGRANGE

Problème Dual:

Lagrangien avec multiplicateurs de Lagrange $\alpha_i \geq 0$

$$L(w, b, \alpha) = (1/2)||w||^2 - \sum_i \alpha_i[y_i(w^T x_i + b) - 1]$$

Développement :

$$L(w, b, \alpha) = (1/2)||w||^2 - \sum_i \alpha_i y_i w^T x_i - b\sum_i \alpha_i y_i + \sum_i \alpha_i$$

Optimisation SVM: La MARGE RIGIDE LAGRANGE

Problème Dual:

Conditions KKT

$\partial L/\partial w = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i$

$\partial L/\partial b = 0 \Rightarrow \sum_i \alpha_i y_i = 0$

Injection de w dans L

On remplace w dans le Lagrangien :

$||w||^2 = \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$

$\sum_i \alpha_i y_i w^T x_i = \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$

Donc $L(\alpha) = \sigma_i \alpha_i - 1/2 \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$

Problème dual final:

$\max_\alpha (\sum_i \alpha_i - 1/2 \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle)$

Sous les contraintes : $\alpha_i \geq 0$ et $\sum_i \alpha_i y_i = 0$

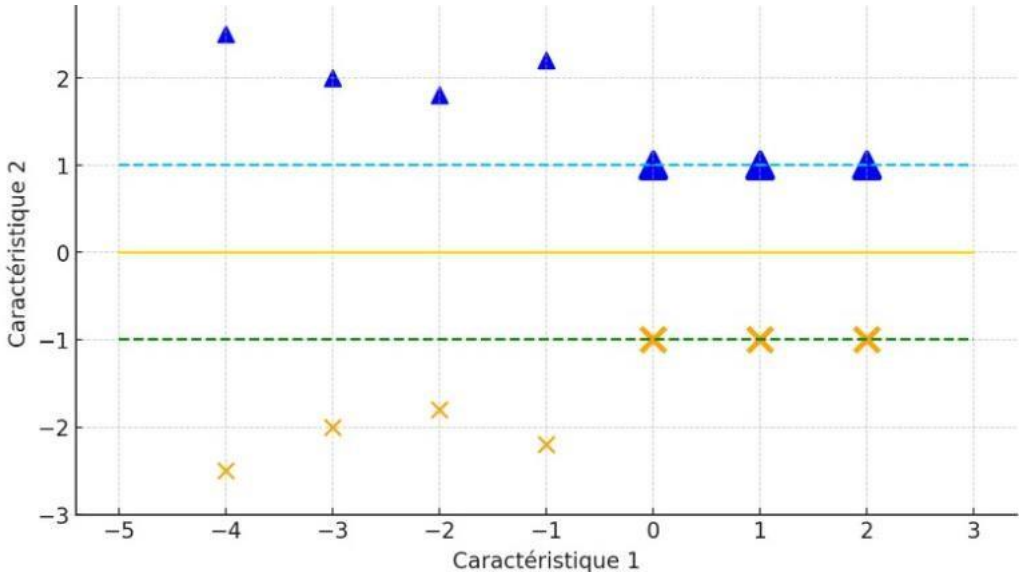
Optimisation SVM: La MARGE SOFT

1. Pourquoi le Hard Margin ne suffit pas ?

Le SVM Hard Margin suppose que les données sont parfaitement séparables :

- Aucun point ne doit être dans la marge,

- Aucun point ne doit être mal classé.

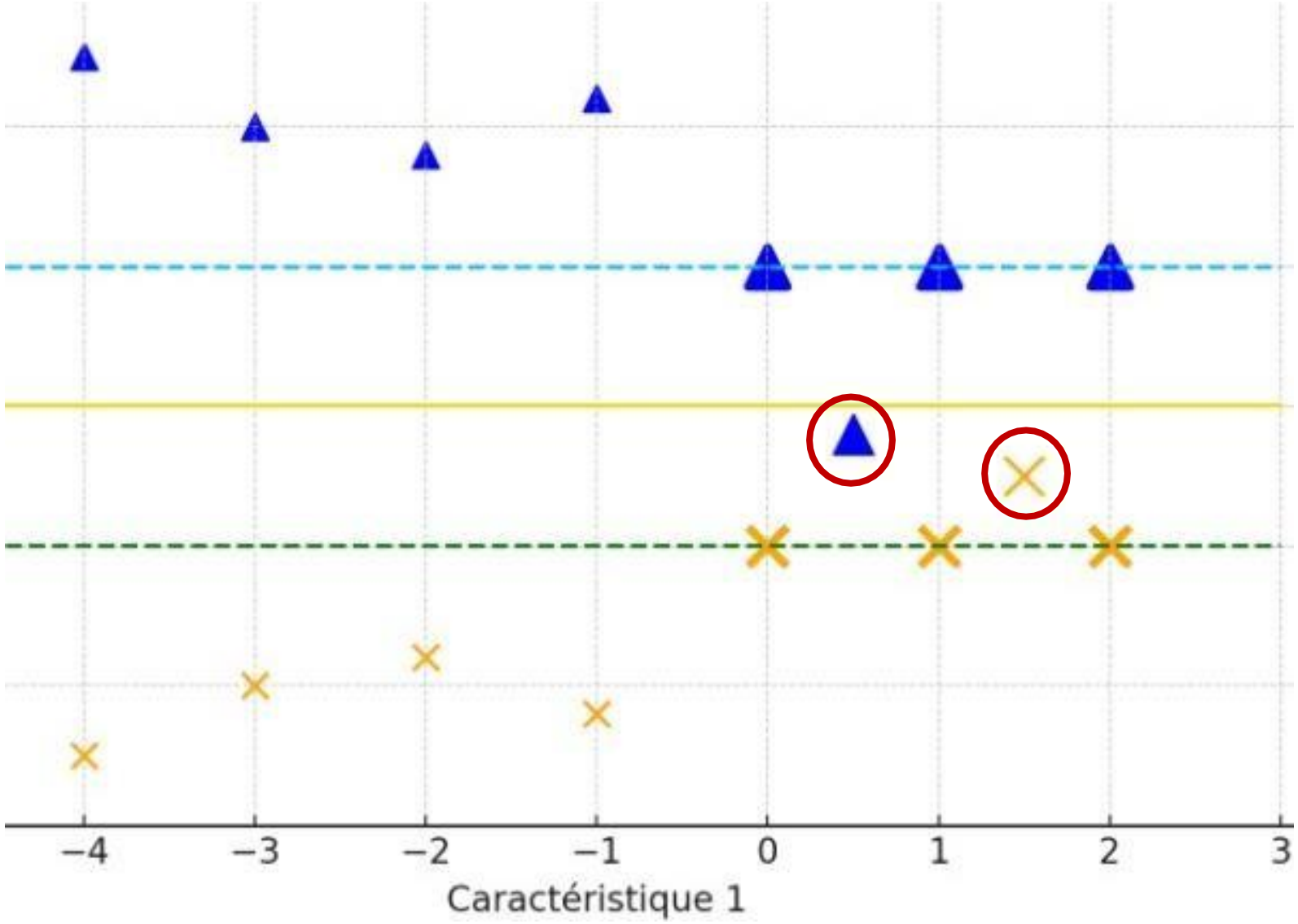


Optimisation SVM: La MARGE SOFT

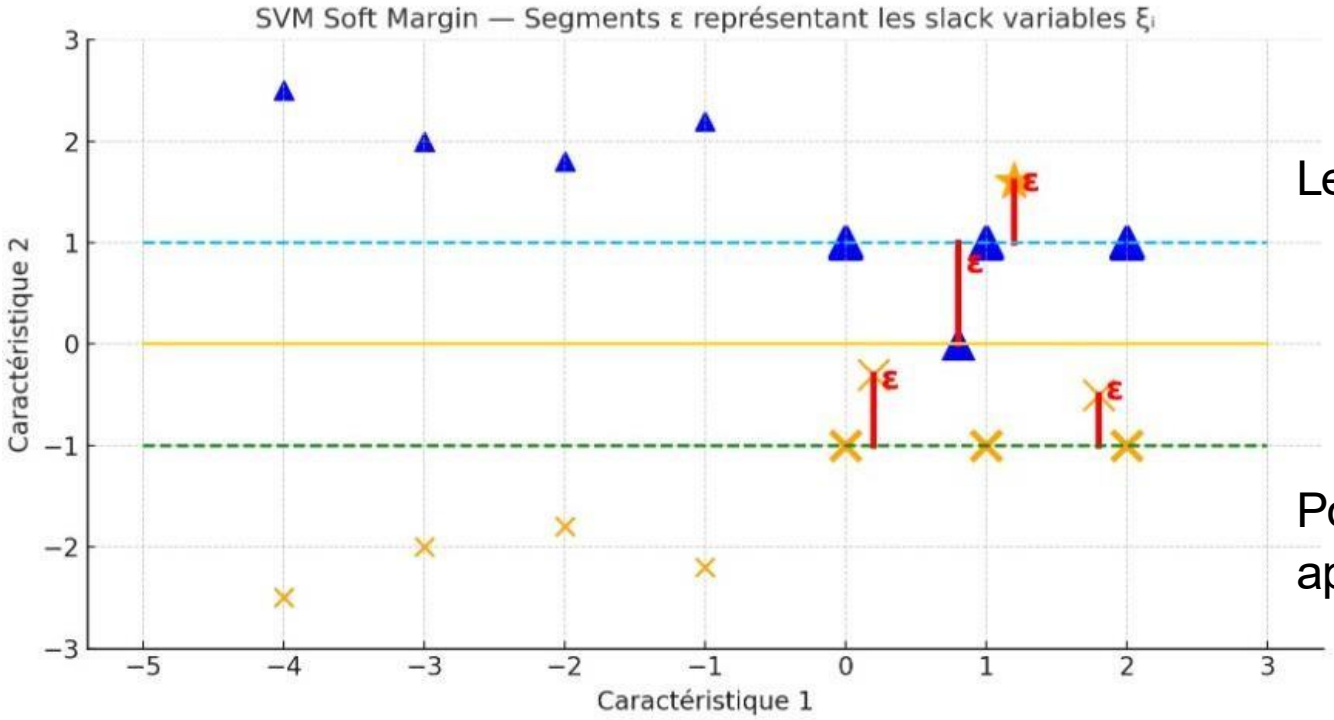
Pourquoi le Hard Margin ne suffit pas ?

- Dans la réalité, les données contiennent du bruit, des valeurs aberrantes et des classes qui se chevauchent.

- Le Hard Margin devient donc impossible à appliquer dans la plupart des cas. D'où la nécessité d'introduire le Soft Margin.



Optimisation SVM: La MARGE SOFT



Le Soft Margin cherche un compromis entre :

1. Maximiser la marge,
2. Autoriser certaines erreurs de classification lorsque cela est nécessaire.

Pour cela, on introduit des variables de relâchement appelées slack variables (ξ_i).

Optimisation SVM: La MARGE SOFT

Les slack variables permettent à un point de violer la contrainte de marge.

La contrainte devient :

$y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \text{avec } \xi_i \geq 0$

Interprétation :

- $\xi_i = 0 \rightarrow$ Point bien classé, hors de la marge.
- $0 < \xi_i < 1 \rightarrow$ Point dans la marge mais bien classé.
- $\xi_i \geq 1 \rightarrow$ Point mal classé.

Optimisation SVM: La MARGE SOFT

Maximiser la marge

$$\text{minimiser} \frac{1}{2} \| w \|^2$$

Minimiser les erreurs

$$\text{minimiser } \sigma_i \xi_i$$

Donc : $$\min(\underbrace{\frac{1}{2} \| w \|^2}_{\textit{maximise la marge}} + \underbrace{C \sigma_i \xi_i}_{\textit{pénalise les erreurs}})$$

Le Soft Margin SVM cherche donc **un équilibre optimal entre marge large et peu d'erreurs.**

Le paramètre C règle l'importance de ces erreurs :

- C grand → les erreurs coûtent cher → modèle très strict
- C petit → plus de flexibilité → modèle plus tolérant

Optimisation SVM: La MARGE SOFT

Contrainte 1 :

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

On réécrit dans la forme standard :

$$y_i(w^T x_i + b) - 1 + \xi_i \geq 0$$

Pour la transformer en équation dans le Lagrangien :

$$\alpha_i \geq 0$$

$$-\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i)$$

Contrainte 2 :

$$\xi_i \geq 0$$

La slack variable mesure :

- ✓ combien un point viole la marge
- ✓ s'il est mal classé,à quel point il rentre dans la marge.

C'est donc une quantité d'erreur, une distance qui doit obligatoirement être positif

Optimisation SVM: La MARGE SOFT

La première partie : ce qu'on veut optimiser

$$\frac{1}{2} \| w \|^2 + C \sum \xi_i$$

La deuxième partie : empêche les violations des contraintes

- **α_i** empêche :

$$y_i(w^T x_i + b) < 1 - \xi_i$$

- **μ_i** empêche :

$$\xi_i < 0$$

Les α_i et μ_i sont comme des **forces** qui tirent la solution dans la zone autorisée.

Optimisation SVM: La MARGE SOFT

$$L = \frac{1}{2}\|w\|^2 + C\sum \xi_i - \sum \alpha_i (y_i(w^T x_i + b) - 1 + \xi_i) - \sum \mu_i \xi_i$$

Fonction objective

Contrainte 1

Contrainte 2

Optimisation SVM: La MARGE SOFT

Conditions KKT pour le Soft Margin

Les dérivations donnent :

$w = \Sigma\ \alpha_i\ y_i\ x_i$

$\Sigma\ \alpha_i\ y_i = 0$

$\alpha_i = C - \mu_i \ \rightarrow\ 0 \leq \alpha_i \leq C$

Cette contrainte $\alpha_i \leq C$ est la seule différence avec le Hard Margin.

Optimisation SVM: La MARGE SOFT

Le dual devient :

$$\max \Sigma \alpha_i - 1/2 \Sigma \Sigma \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

Sous contraintes :

$0 \leq \alpha_i \leq C$

$\Sigma \alpha_i y_i = 0$

C limite la valeur maximale des multiplicateurs de Lagrange.

DONNÉES NON LINÉAIRE

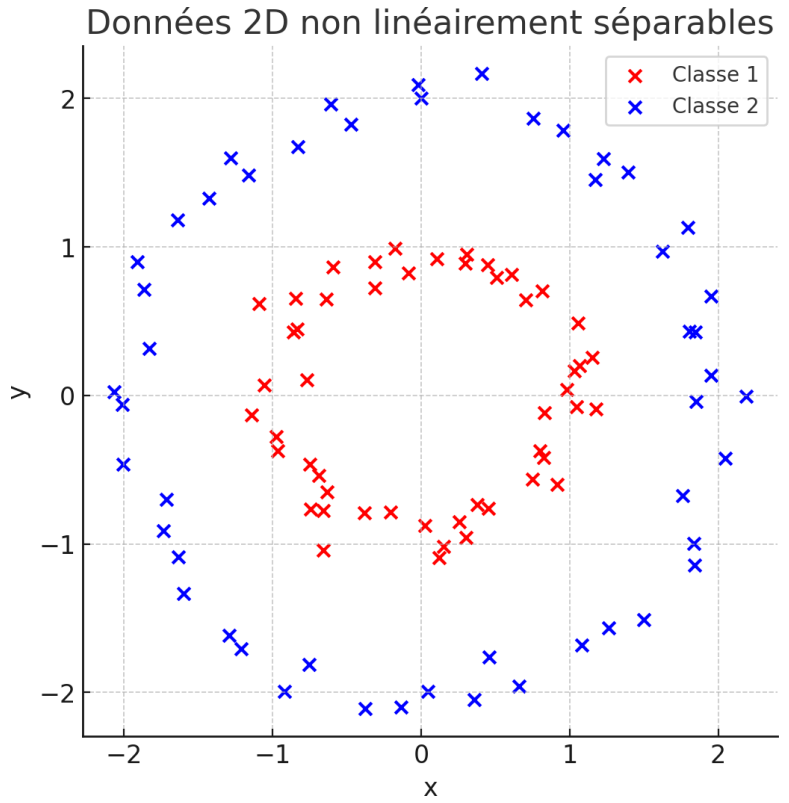
Principe
Introduction

Limitation du SVM linéaire

Le SVM linéaire fonctionne parfaitement lorsque les classes sont séparables par une ligne droite (ou un hyperplan). Mais que faire lorsque les données ont une structure plus complexe, comme ces cercles concentriques ?

Le problème :

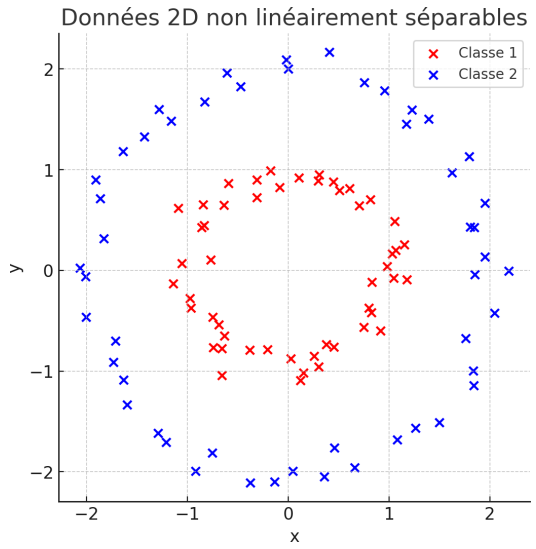
- Points rouges au centre
- Points bleus en périphérie
- **Aucune ligne droite ne peut les séparer**



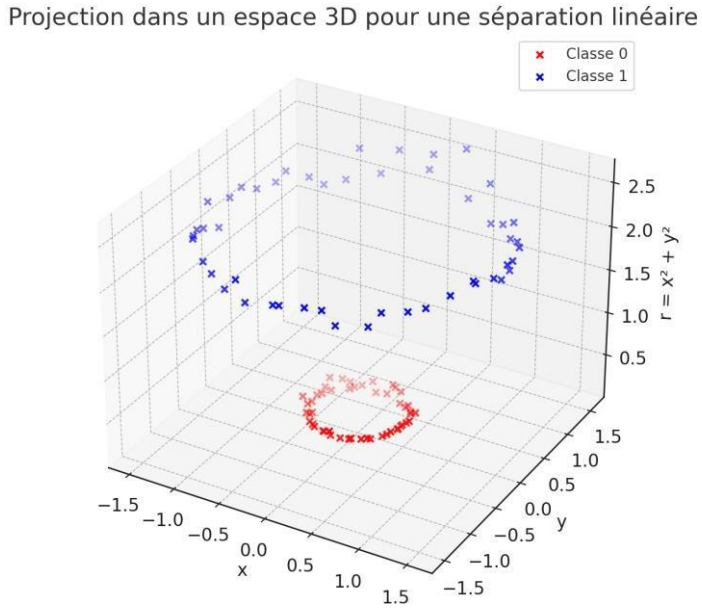
Principe
Introduction

La solution : SVM non-linéaire

Grâce au **kernel trick**, nous allons transformer ces données dans un espace de dimension supérieure où elles deviennent linéairement séparables, sans calculer explicitement cette transformation.



Transformation de données



Principe
Introduction

$$RBF = e^{-||X||^2}$$

X1 X2

$$X = [[F1, F2], [F1, F2], \dots\dots]$$

$X1^2$

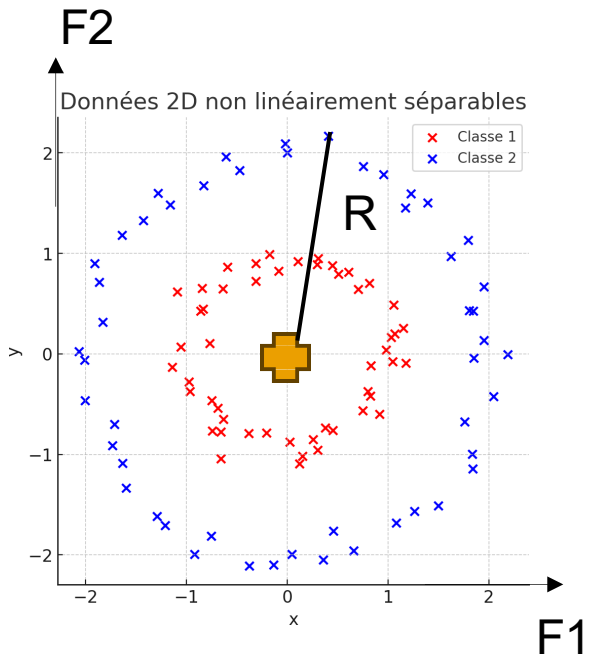
$$X^2 = [[F1^2, F2^2], [F1, F2], \dots\dots]$$

$X2^2$

$$||X|| = F1^2 + F2^2$$

RBF:RADIAL BASIS FUNCTION

$$R1 = e^{-(F1^2+F2^2)}$$



Centre

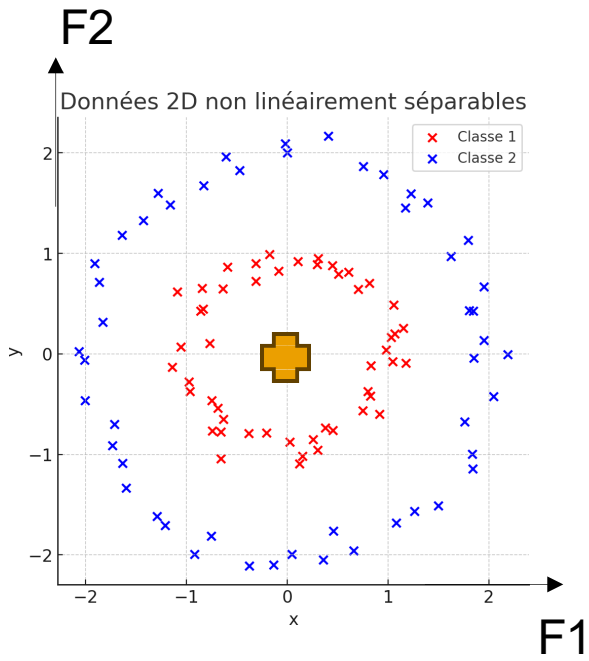
R: distance entre xi et le centre

Principe
Introduction

$$RBF = e^{-||X||^2}$$

$$R1 = e^{-(F1^2+F2^2)}$$

La fonction RBF nous permet de créer une nouvelle donnée R(distance entre le centre et la donnée xi). Cette transformation va nous permettre de passer d'une representation des donnees en 2D (f1,f2) en representation 3D (f1,f2.r)



Centre

R: distance entre xi et le centre

Principe
Introduction

$$RBF = e^{-||X||^2}$$

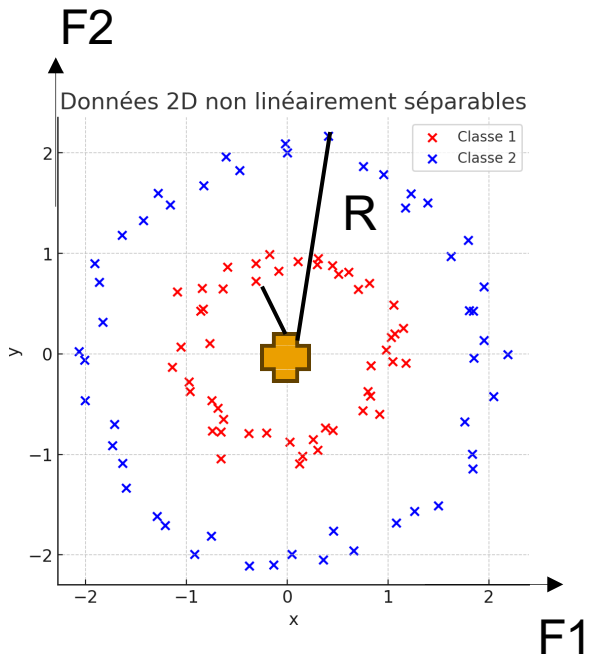
$$R1 = e^{-(F1^2+F2^2)}$$

	F1	F2	R
1	Petite	Petite	1
2	Grande	Grande	0
.			
.			
n			

X proche du centre petite valeur $e^{-(petite\ valeur)}$ = 1

X loin du centre grande valeur $e^{-(grande\ valeur)}$ =0

R AURA DES VELEURS ENTRE 0 ET 1



Centre

R: distance entre xi et le centre

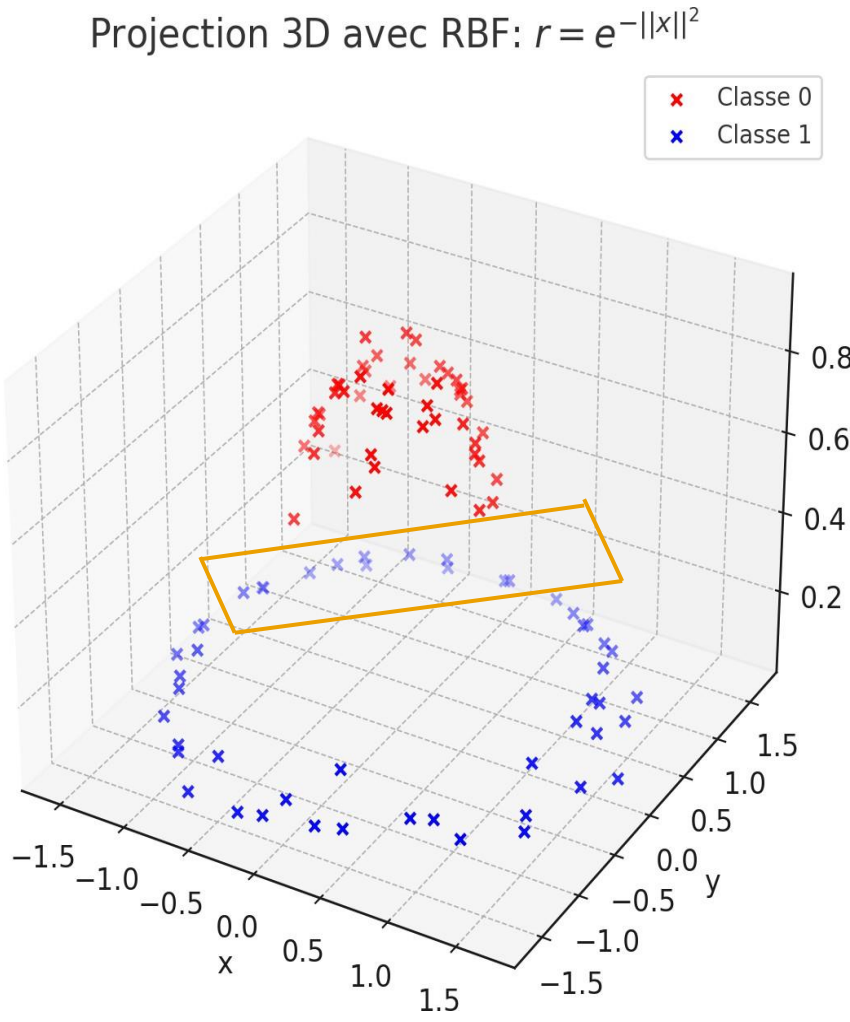
Principe

- Les points rouges (classe 0) sont regroupés autour du centre, avec une valeur r plus élevée.
- Les points bleus (classe 1), plus éloignés du centre, ont une valeur r plus petite.

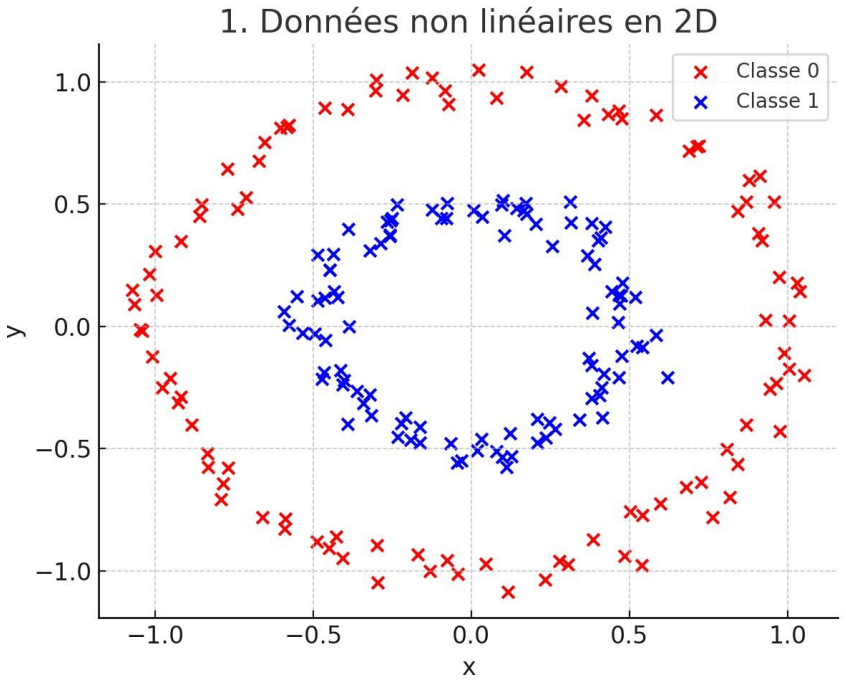
Pourquoi c'est important ?

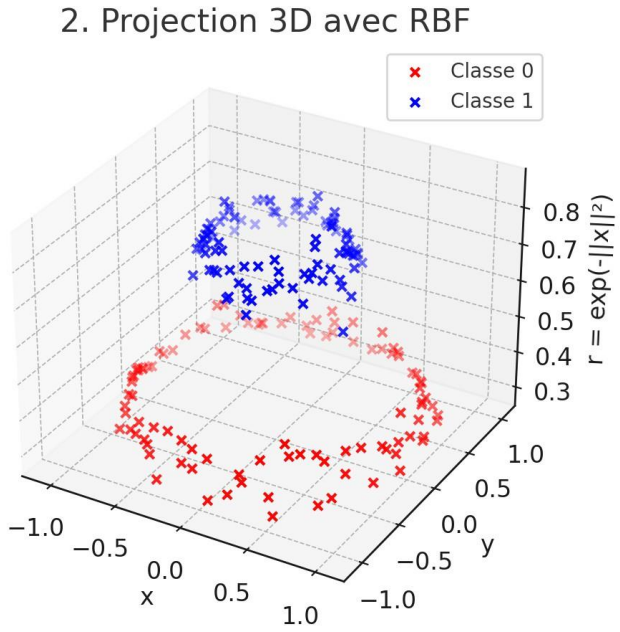
Grâce à cette transformation, les deux classes deviennent séparables par un plan dans l'espace 3D, ce qui n'était pas possible avec une simple droite en 2D.

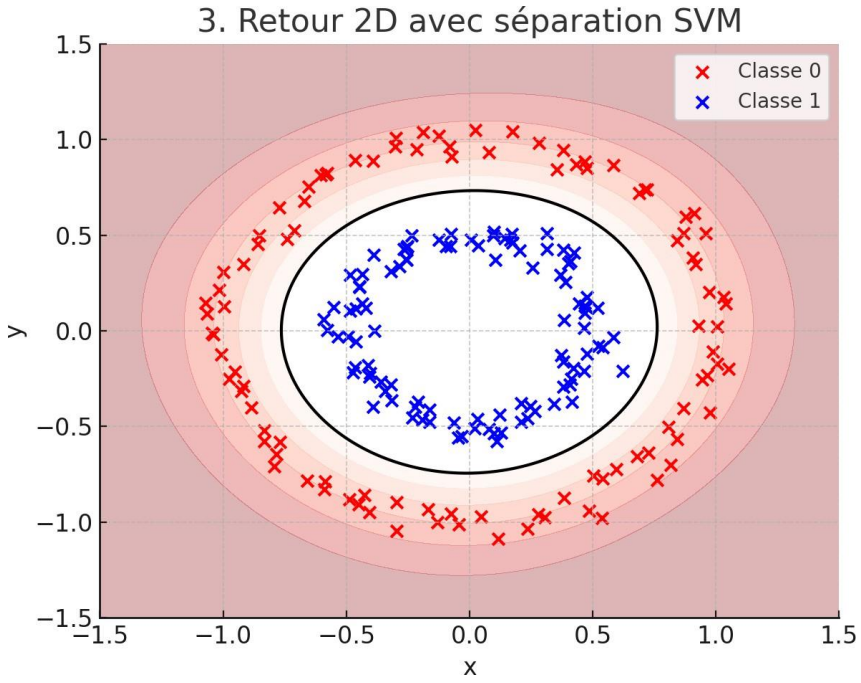
Cela montre le pouvoir du noyau RBF : il rend possible la séparation linéaire de données qui ne l'étaient pas dans leur forme d'origine.



SVM NON LINÉAIRE







Qu’est-ce qu’un noyau (kernel) ?

- **Qu’est-ce qu’un noyau (kernel) ?**

Un noyau est une **fonction de similarité** entre deux points.

Dans un SVM, cette similarité est calculée sous forme de **produit scalaire**.

- Dans le cas linéaire, la similarité entre deux points est :

$$K(xi,xj)=\langle xi,xj\rangle=X_i^T \cdot X_j$$

Pour obtenir des séparations **non linéaires**, on imagine transformer les données :

$$x \rightarrow \phi(x)$$

Dans ce nouvel espace, la similarité devient un **produit scalaire enrichi** :

$$K(x_i, x_j) = \langle \phi, (x_i)\phi(x_j)\rangle$$

Le kernel trick permet de calculer cette similarité comme si l’on travaillait dans l’espace transformé, sans jamais calculer explicitement φ(x).

Qu’est-ce qu’un noyau (kernel) ?

1. **Formulation duale du SVM linéaire (rappel) :**

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i \, x_j \rangle$$

- Les données interviennent uniquement **via des produits scalaires** $\langle x_i, x_j \rangle$

2. **Idée du *kernel trick***

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{K}(\boldsymbol{x_i}, \boldsymbol{x_j})$$

Kernel LINÉAIRE

Expression :

$$K(x_i, x_j) = \langle x_i, x_j \rangle = x_i^T x_j$$

Quand l'utiliser :

- Données **linéairement séparables**
- Données **très haute dimension** (texte, TF-IDF, bag-of-words)
- Modèle très rapide et robuste
- Souvent le meilleur choix quand les features >> nombre d'échantillons

Python:

```
from sklearn.svm import SVC
SVC(kernel='linear')
```

Kernel Polynomiale

Expression :

$$K(x_i, x_j) = (\langle x_i \; x_j \rangle + c)^d$$

où :

d= degré du polynôme c= coefficient (souvent 1)

Quand l'utiliser :

- Relations **non linéaires mais régulières**
- Utile quand on soupçonne une relation **quadratique / cubique** entre les features
- Frontières non linéaires mais "propres"
- Attention : peut devenir coûteux si le degré est élevé

Python:

```
from sklearn.svm import SVC
SVC(kernel='poly', degree=3, coef0=1)
```

Kernel Polynomiale

Kernel polynomial degré 2 :

$$(x^T x_j + c)^2 = (x^T x_j)^2 + 2c(x^T x_j) + c^2$$

- Le terme quadratique crée la non-linéarité
- Les termes linéaires + constant rendent le modèle plus lisse

Quand c devient grand, la frontière est dominée par les termes :

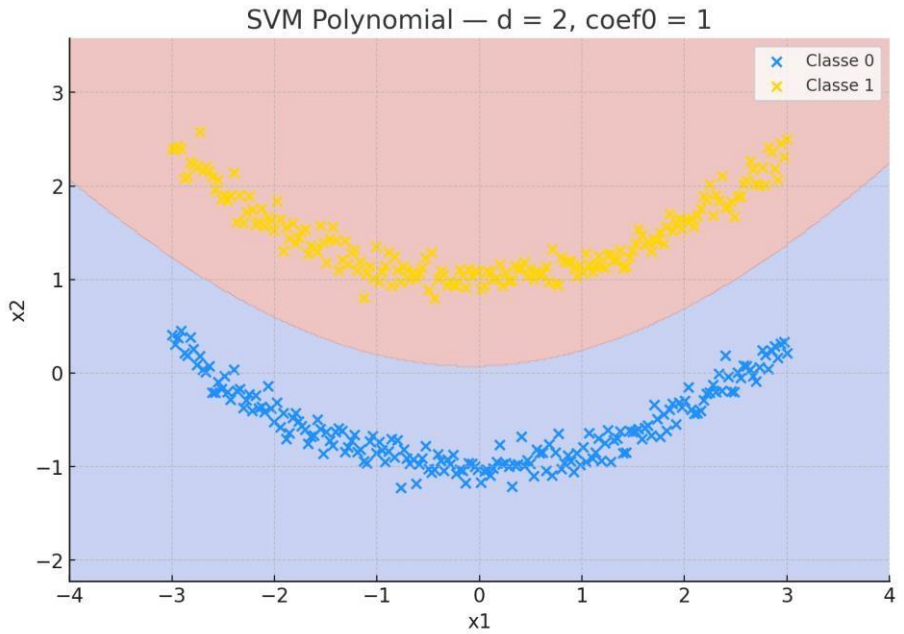
$$2c(x^T x_j) + c^2$$

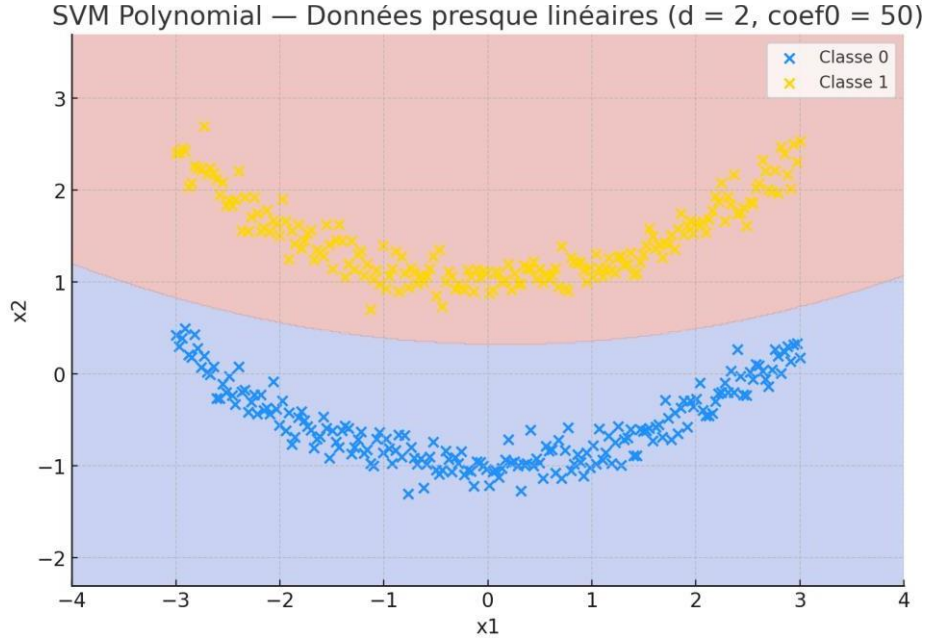
qui se comportent presque comme une frontière linéaire.

- c (coef0) contrôle l'importance des termes de bas degré.

  Petit c = modèle très non linéaire.

  Grand c = modèle plus rigide, presque linéaire





Kernel RBF (Gaussien)

Expression :

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

Quand l'utiliser :

- Données **fortement non linéaires**
- Frontières complexes, tordues ou irrégulières
- Cas général : souvent le kernel le plus performant
- Mapping implicite en **dimension infinie**

Python:

```
from sklearn.svm import SVC
SVC(kernel='rbf', gamma=0.1)
```

Kernel RBF (Gaussien)

Exercice:

On considère trois points :

- $x_i = (0, 0)$
- $A = (1, 0)$
- $B = (3, 0)$

On utilisera le noyau RBF :

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

1. calculer **la similarité entre** x_i et A puis entre x_i et B pour trois valeurs de gamma :

$$\gamma = 0.1,\ 1,\ 5$$

Kernel RBF (Gaussien)

2. Complétez le tableau suivant

Gamma	(K(x_i, A))	(K(x_i, B))	Interprétation
0.1	…	…	…
1	…	…	…
5	…	…	…

3. Pour quel gamma A et B semblent-ils tous les deux encore “proches de” x_i?
4. Pour quel gamma A semble proche mais B semble complètement différent ?
5. À partir de quel gamma la similarité chute presque à 0 ?
6. Expliquez comment gamma change votre définition de “proche” ou “loin”.

Kernel RBF (Gaussien)

D_A=1
D_B=9

Gamma (γ)	Similarité avec A (proche)	Similarité avec B (loin)	Interprétation
0.1	0.90 (Très fort)	0.41 (Moyen)	**Généraliste** (Vue large)
1	0.37 (Moyen)	~0.00 (Nul)	**Équilibré**
5	0.006 (Très faible)	0.00 (Nul)	**Sélectif / Rigide** (Risque d'Overfitting)

Kernel RBF (Gaussien)

Conclusion

- Petit gamma → la similarité reste élevée même si les points sont loin.
- Grand gamma → la similarité chute très vite → seuls les points très proches comptent.

Le SVM utilise ces similarités pour décider la frontière.

ARBRE DE DÉCISION

Introduction

Qu'est-ce qu'un arbre de décision ?

Un **arbre de décision** est un algorithme d'apprentissage supervisé utilisé pour :

- **Classification** (oui/non, classe A/B/C…)
- **Régression** (prédire une valeur numérique)

Il est utilisé pour **représenter visuellement et explicitement les décisions** et la prise de décision pour des problèmes de classification ainsi que pour des problèmes de régression. Il représente aussi **l'élément de base de plusieurs modèles** comme le Random Forest ou XGBoost.

Principe

Le fonctionnement d'un arbre de decision est basé sur une structure descendante composée de **noeuds** : chaque noeud possède une condition qui amène à plusieurs réponses, ce qui dirige à un prochain noeud. Lorsqu'un nœud donne la réponse, on dit que le nœud est **terminal**.

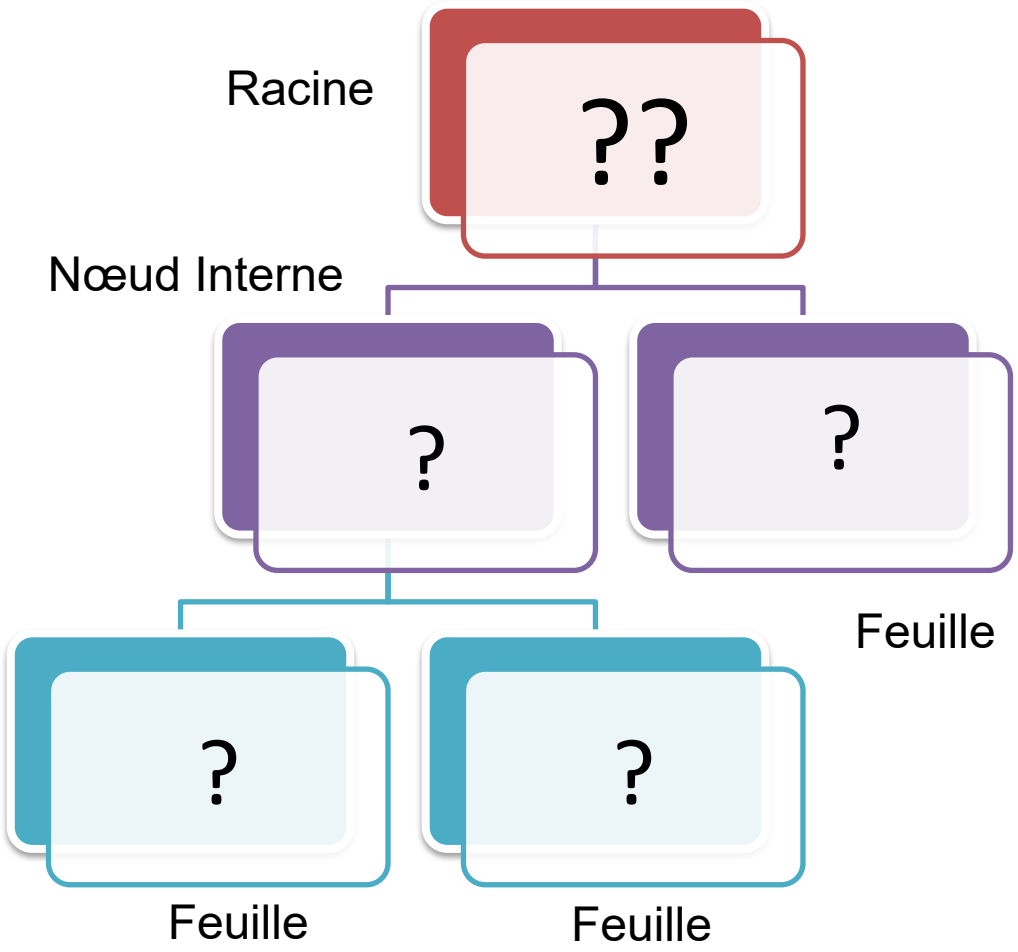
Donc:

- **Racine (root)** : première question
- **Nœuds internes** : décisions intermédiaires
- **Feuilles (leaves)** : décision finale (classe ou valeur)

Construction de l'arbre de decision Exemple1

Problème : prédire si un étudiant **réussit (Oui/Non)**

Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	2	5	Non
C	6	0	Oui
D	1	6	Non
E	7	2	Oui
F	4	3	Non

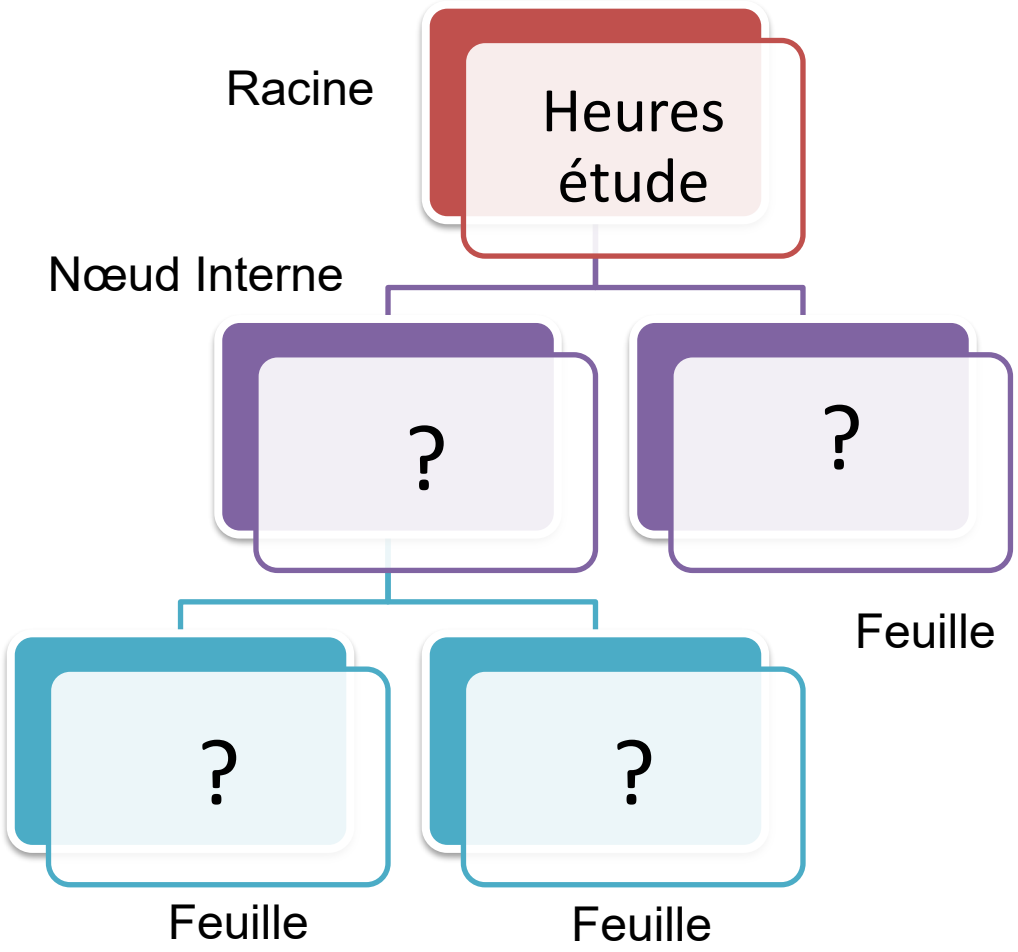


Quelle est la Question possible à la racine??

Construction de l'arbre de decision Exemple1

Problème : prédire si un étudiant **réussit (Oui/Non)**

Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	2	5	Non
C	6	0	Oui
D	1	6	Non
E	7	2	Oui
F	4	3	Non

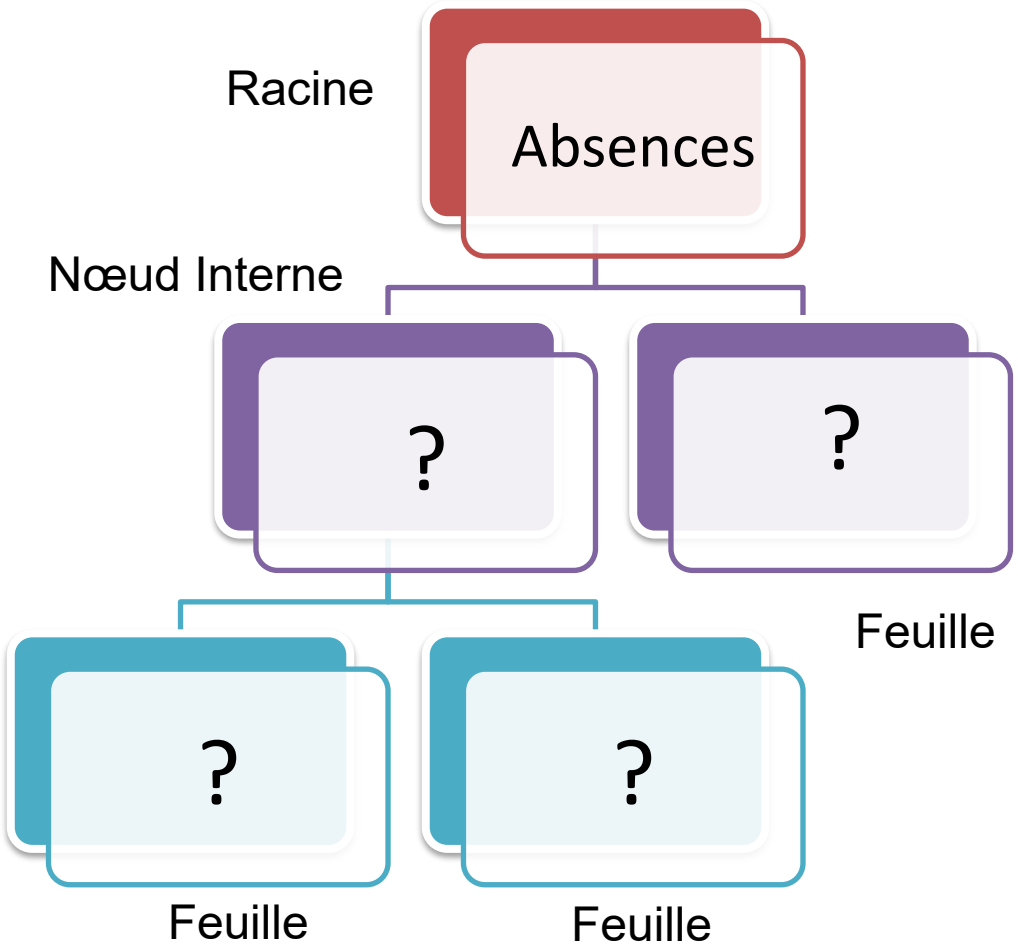


Quelle est la Question possible à la racine??

Construction de l'arbre de decision Exemple1

Problème : prédire si un étudiant **réussit (Oui/Non)**

Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	2	5	Non
C	6	0	Oui
D	1	6	Non
E	7	2	Oui
F	4	3	Non



Quelle est la Question possible à la racine??

Construction de l’arbre de decision Exemple1

Nous avons :

•Variables explicatives :

- **Heures**
- **Absences**

•Variable cible :

- **Classe** ∈ {Oui, Non}

Objectif :

Trouver **la meilleure question à poser à la racine**

Avant de choisir une question, l’arbre mesure :

•à quel point un groupe est mélangé

Deux mesures principales :

1.Entropie

2.Indice de Gini

Entropie

Définition

L'entropie mesure le **désordre** d'un ensemble.

$$Entropy = -\sum p_i \log_2(p_i)$$

Exemple numérique

Dans notre dataset :

Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
D	3	4	Non
E	2	5	Non
F	4	3	Oui

- 4 **Oui**
- 2 **Non**

$$p(Oui) = 4/6, p(Non) = 2/6$$

$$Entropy\ Parent = -(4/6)\log_2(4/6) - (2/6)\log_2(2/6)$$

$$Entropy \approx 0.918$$

Dataset très mélangé

Entropie

Définition

L'entropie mesure le **désordre** d'un ensemble.

$$Entropy = -\sum p_i \log_2 (p_i)$$

Entropie = **0** → groupe parfaitement pur ou homogène
Entropie = **1** → groupe hétérogène

Entropie

Information Gain : choisir la meilleure question

L'arbre teste plusieurs questions et choisit celle qui **réduit le plus l'entropie**.

$$Gain = Entropy(parent) - \sum \frac{N_i}{N} Entropy(child_i)$$

Entropie

Pour une **variable numérique**, un arbre de décision :

1. **teste tous les seuils possibles entre les valeurs observées.**
2. **calcule le gain d'information pour chacun.**
3. **choisit le seuil qui maximise le gain**

Entropie

Pour une **variable numérique**, un arbre de décision :

1. **teste tous les seuils possibles entre les valeurs observées.**

1.1 Calcul du seuil

Étape 1 — Trier les valeurs numériques

Étudiant	Heures	Classe
E	2	Non
D	3	Non
F	4	Oui
C	6	Oui
B	7	Oui
A	8	Oui

Étudiant	Absences	Classe
C	0	Oui
A	1	Oui
B	2	Oui
F	3	Oui
D	4	Non
E	5	Non

Entropie

Pour une **variable numérique**, un arbre de décision :

1. **teste tous les seuils possibles entre les valeurs observées.**

1.1 Calcul du seuil

Étape 2 — Générer les seuils candidats

Un seuil est testé entre deux valeurs consécutives

Formule générale :

$$\theta_k = \frac{x_k + x_{k+1}}{2}$$

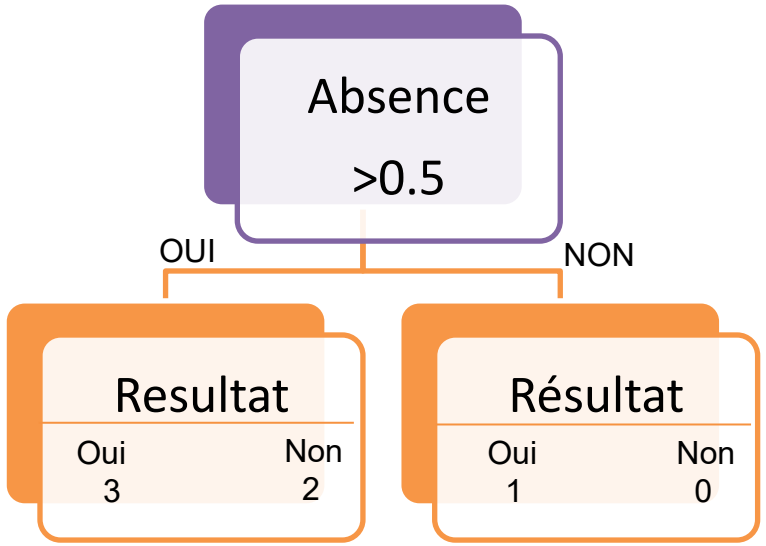
Heure

Entre les valeurs	Seuil
2 et 3	**2.5**
3 et 4	**3.5**
4 et 6	**5.0**
6 et 7	**6.5**
7 et 8	**7.5**

Absence

Entre les valeurs	Seuil
0 et 1	**0.5**
1 et 2	**1.5**
2 et 3	**2.5**
3 et 4	**3.5**
4 et 5	**4.5**

Entropie



Étudiant	Absences	Classe
C	0	Oui
A	1	Oui
B	2	Oui
F	3	Oui
D	4	Non
E	5	Non

P(oui)=3/5 P(Non)=2/5

$H2=-\frac{2}{5}\log_2\frac{2}{5}-\frac{3}{5}\log_2\frac{3}{5}=0.971$

P(oui)=1 P(Non)=0

$H1=-1\log_2 1=0$

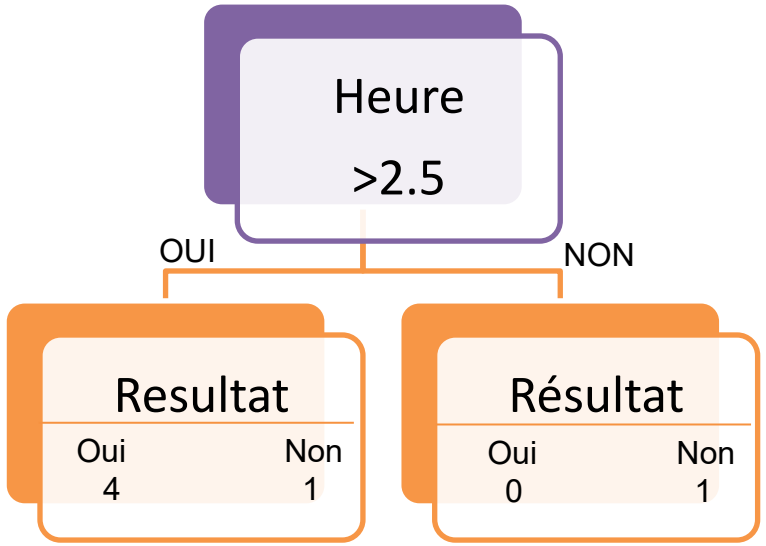
Entropie Pondéré $=\frac{1}{6}\times 0+\frac{5}{6}\times 0.971=0.809$

Gain= $Entropy\ Parent$ -entropie pondéré

=0.918-0.809=0.109

Seuil	Entropie pondérée	Gain d'information
0.5	0.809	0.109
1.5	0.667	0.251
2.5	0.459	0.459
3.5	0.000	0.918
4.5	0.602	0.316

Entropie



Étudiant	Heures	Classe
E	2	Non
D	3	Non
F	4	Oui
C	6	Oui
B	7	Oui
A	8	Oui

P(oui)=3/5 P(Non)=2/5

$H2=-\frac{1}{5}\log_2\frac{1}{5}-\frac{4}{5}\log_2\frac{4}{5}$

$= 0.722$

P(oui)=0 P(Non)=1

$H1=-1\log_2 1=0$

Entropie Pondéré $=\frac{1}{6}\times 0+\frac{5}{6}\times 0.722=0.602$

Gain= $Entropy\ Parent$ -entropie pondéré = 0.918-0.602=0.316

Seuil	Entropie pondérée	Gain d'information
2.5	0.601	0.316
3.5	0	0.918
5.5	0.459	0.459
6.5	0.667	0.251
7.5	0.809	0.109

Entropie

Variable	Gain max
Absences	**0.918**
Heures	0.459

- La variable Absences réduit beaucoup plus l'entropie que Heures
- Elle sépare les données en groupes beaucoup plus homogènes
- Elle permet de prédire la classe avec plus de certitude dès la racine

On choisit à la racine la variable **Absence** qui maximise le gain d'information,

car c'est celle qui apporte le plus d'information sur la classe et qui réduit le plus rapidement l'incertitude.

Indice de Gini

Définition
L'indice de Gini mesure la **probabilité de se tromper** si l'on classe un élément **au hasard** selon la distribution des classes dans un nœud.

$$Gini = 1 - \sum p_i^2$$

- P_i : proportion de la classe i
- **Gini = 0** → nœud parfaitement pur(Homogène)
- **Gini élevé** → classes mélangées

Indice de Gini

Étape 1 — Calcul du Gini du nœud parent
Dans notre dataset :
•Oui = 4
•Non = 2
•Total = 6

$$p(Oui) = \frac{4}{6}, p(Non) = \frac{2}{6}$$

$$Gini_{parent} = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 1 - \frac{16}{36} - \frac{4}{36} = \boxed{0.556}$$

C’est notre point de départ, commun à toutes les variables.

Indice de Gini

Étape 2 — Calcul du Gini après séparation (pour chaque variable)

Comme pour l'entropie, on procède **variable par variable**.

Règle générale

Pour une question donnée :

$$Gini_{apres} = \sum_{i} \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

On calcule un Gini pondéré.

Indice de Gini

Étape 3 — Mesure de la qualité de la séparation (Gain de Gini)

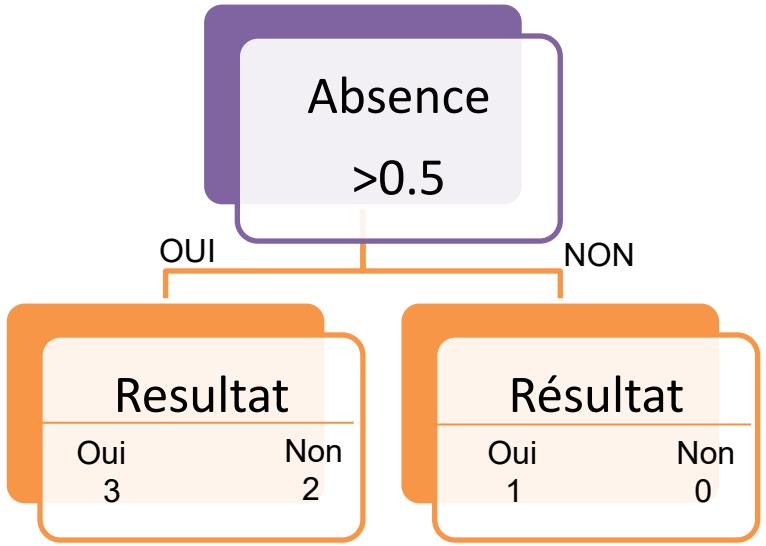
Pour rester parallèle à l'entropie, on définit :

$$\boxed{\Delta Gini = Gini_{parent} - Gini_{apres}}$$

•Plus **ΔGini est grand,** Plus la séparation est **bonne**

Minimiser le Gini après séparation ⇔ Maximiser la réduction de Gini

Gini



Étudiant	Absences	Classe
C	0	Oui
A	1	Oui
B	2	Oui
F	3	Oui
D	4	Non
E	5	Non

P(oui)=3/5 P(Non)=2/5

Gini1=$1-((\frac{3}{5})^2+\left(\frac{2}{5}\right)^2)$
$=0.48$

P(oui)=1 P(Non)=0

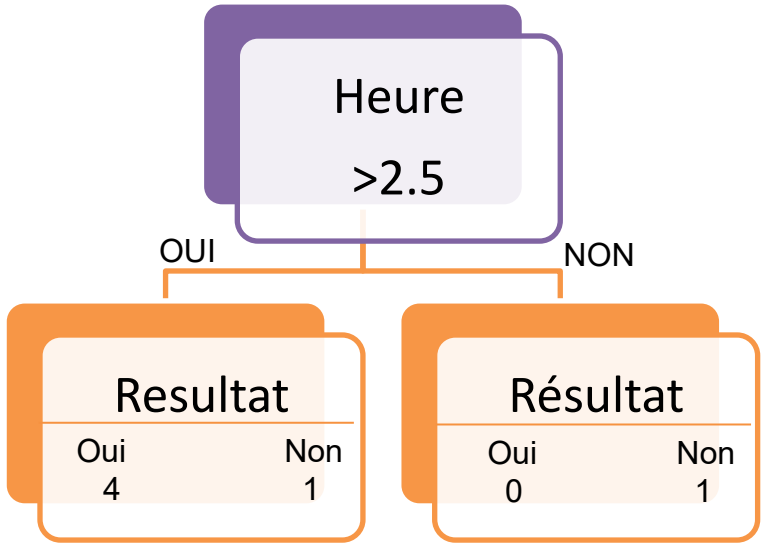
Gini2$=0$

Seuil	Gini pondéré	Réduction de Gini
0.5	0.400	0.156
1.5	0.333	0.223
2.5	0.222	0.316
3.5	**0.000**	**0.918**
4.5	0.267	0.459

Gini pondéré $=\frac{1}{6}\times 0+\frac{5}{6}\times 0.48=0.40$

$\Delta Gini=Gini\_parent-Gini\_pondéré=0.556-0.40=0.156$

Gini



Étudiant	Heures	Classe
E	2	Non
D	3	Non
F	4	Oui
C	6	Oui
B	7	Oui
A	8	Oui

P(oui)=4/5 P(Non)=1/5

Gini1=$1-((\frac{4}{5})^2+\left(\frac{1}{5}\right)^2)$
$=0.32$

P(oui)=0 P(Non)=1

Gini2= 0

Entropie Pondéré =$\frac{1}{6}\times 0+\frac{5}{6}\times 0.32=0.267$

Gain= Gini_parent−Gini_pondéré =
0.556-0.267=0.289

Seuil	Gini pondéré	Réduction de Gini
2.5	0.267	0.289
3.5	0.222	0.334
5.0	**0.222**	**0.334**
6.5	0.333	0.223
7.5	0.400	0.156

Gini

Variable	Réduction max de Gini
Absences	**0.556**
Heures	0.334

Avec l’indice de Gini, on choisit la variable qui minimise la probabilité d’erreur après la séparation.

exactement comme avec l’entropie, **Absences reste la meilleure variable à la racine**,

Quelle métrique utiliser?

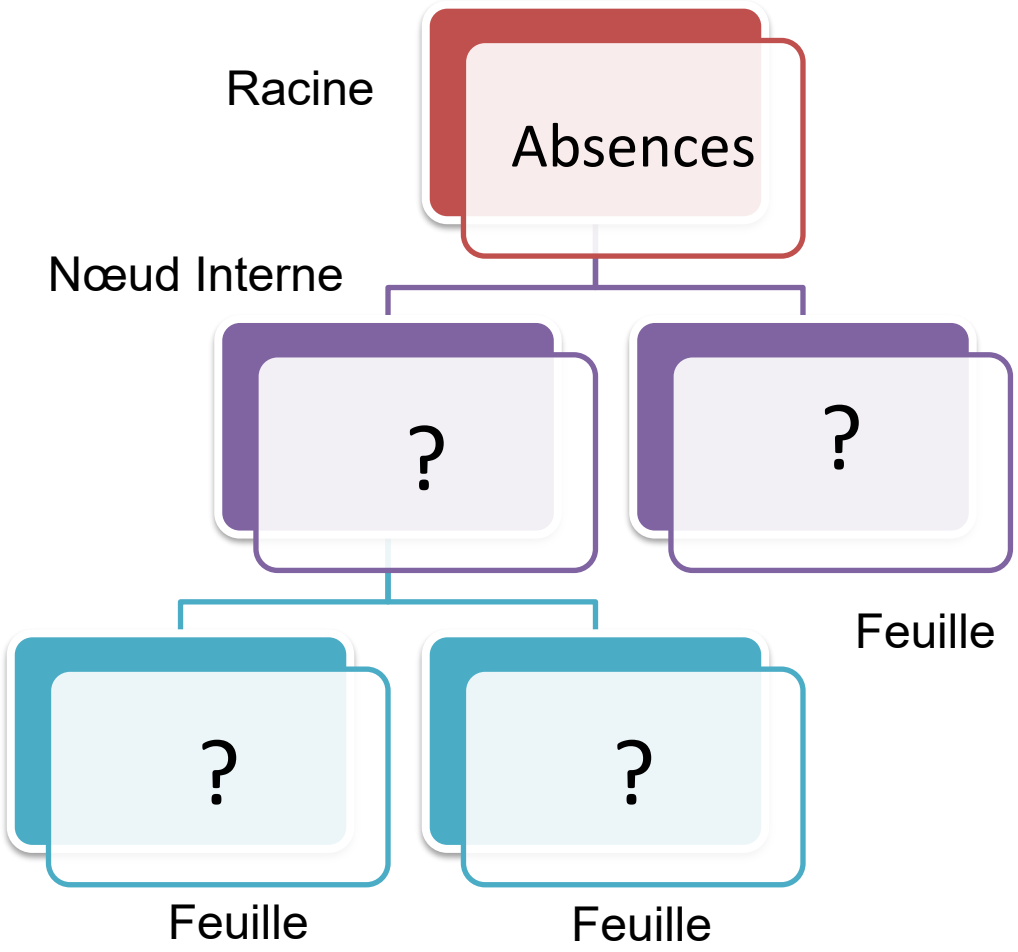
Le choix entre **l'indice de Gini** et **l'entropie** dépend principalement de la façon dont on mesure la pureté des données.

L'entropie évalue le niveau d'incertitude, tandis que le Gini mesure la probabilité d'erreur de classification. En pratique, ces deux critères conduisent généralement à des arbres très similaires. Pour des raisons de simplicité et de rapidité de calcul, **l'indice de Gini est souvent utilisé par défaut**.

Exemple

Problème : prédire si un étudiant **réussit (Oui/Non)**

Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	2	5	Non
C	6	0	Oui
D	1	6	Non
E	7	2	Oui
F	4	3	Non



Quelle est la Question possible à la racine=absence

Exemple

Avec le Gini, on a déjà fait :

1.Calcul du **Gini du nœud parent**

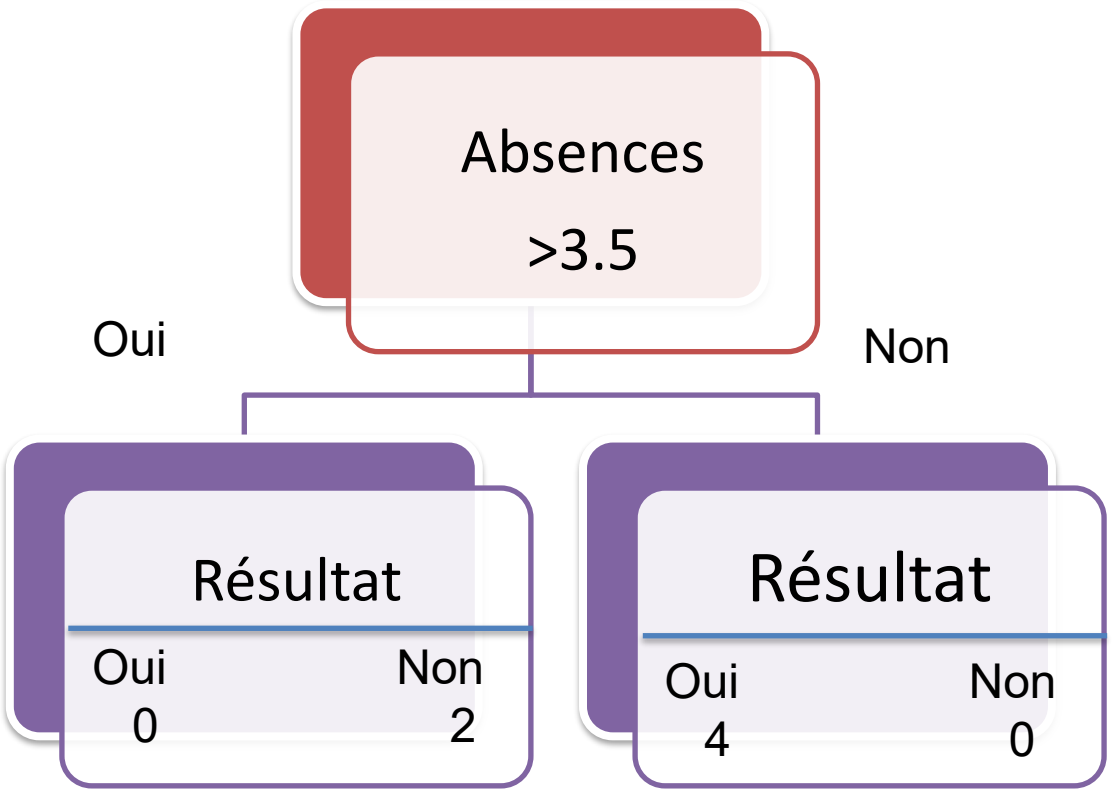
2.Calcul du **Gini pondéré** pour chaque (variable + seuil)

3.Sélection de la **meilleure question**

celle qui **minimise le Gini après séparation**(ou maximise la réduction de Gini)

La question à la racine est maintenant fixée.

Exemple1



Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
D	3	4	Non
E	2	5	Non
F	4	3	Oui

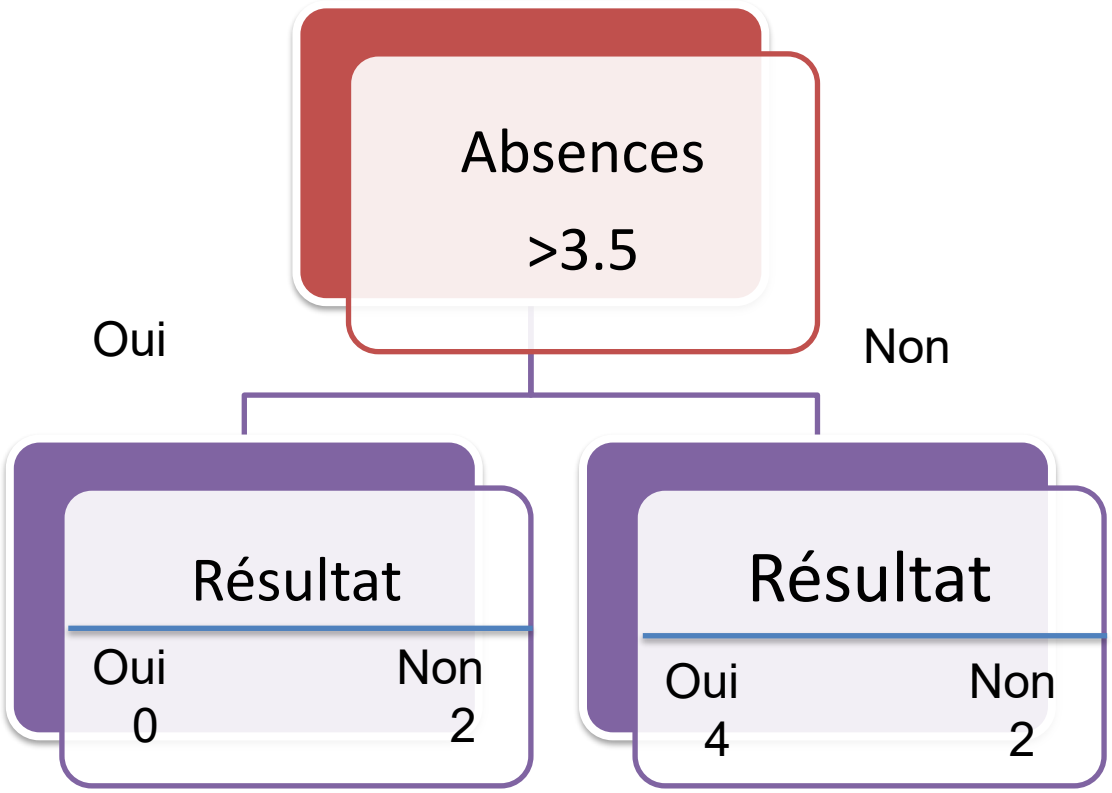
Gini1=0

Gini2=0

$$\text{Total\_Gini}=\frac{2}{6}\times 0+\frac{4}{6}\times 0=0$$

⟹ L'arbre s'arrête

Exemple2



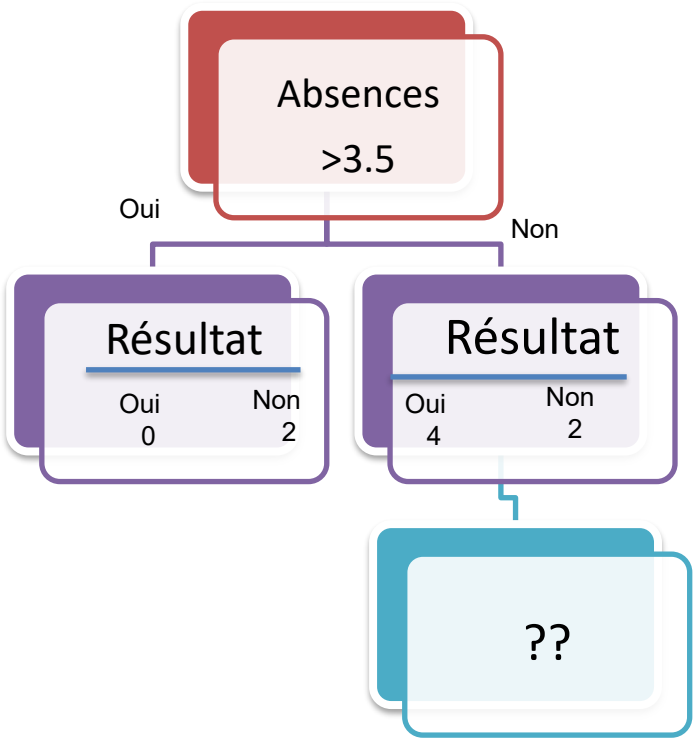
Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
D	3	4	Non
E	2	5	Non
F	4	3	Oui
G	3	2	Non
H	3	3	Non

Gini1=0

Gini2=1-$(\frac{4}{8})^2-\left(\frac{2}{8}\right)^2 = 0.6875$

Total_Gini=$\frac{2}{8}\times 0+\frac{6}{8}\times 0.6875 = 0.52$

Exemple2



Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
F	4	3	Oui
G	3	2	Non
H	5	3	Non

Exemple2

Valeurs observées (absence<=5):

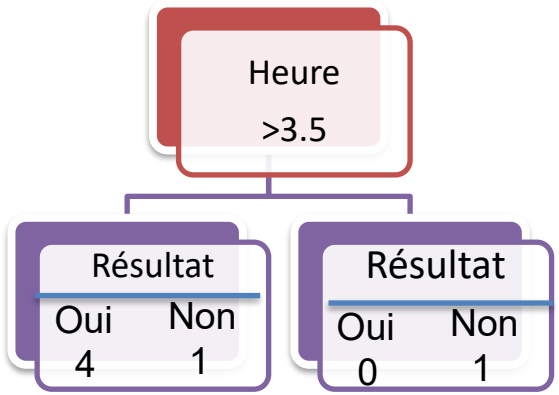
Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
F	4	3	Oui
G	3	2	Non
H	5	3	Non

$$Gini = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 1 - \frac{16}{36} - \frac{4}{36} = \boxed{0.556}$$

Nœud non homogène donc on continue a développer

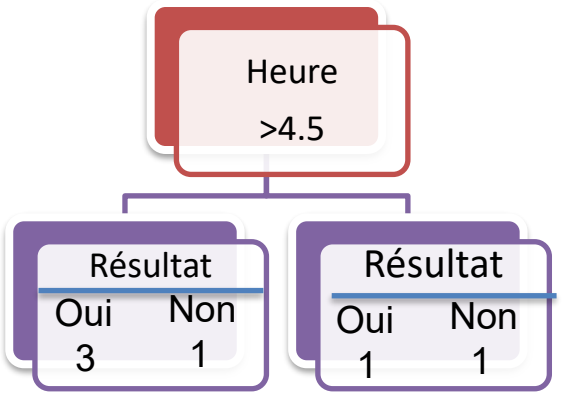
Seuils: 3.5,4.5,5.5,6.5,7.5

Exemple2



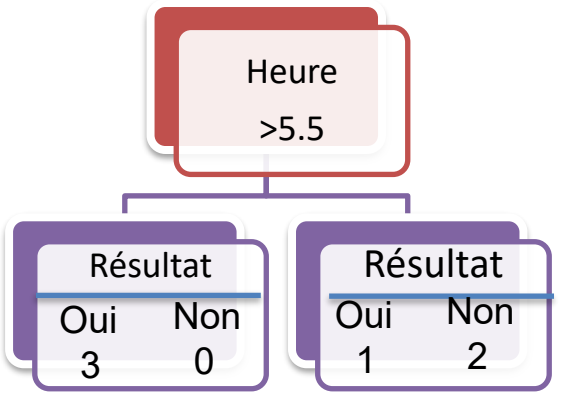
Gini1 =
$1-(\frac{4}{5})^2-(\frac{1}{5})^2 = 0.32$

Gini2=0



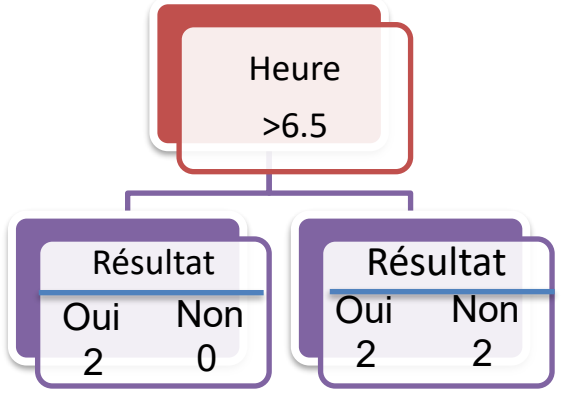
Gini1 =
$1-(\frac{3}{4})^2-(\frac{1}{4})^2$
= 0.375

Gini2 =
$1-(\frac{1}{2})^2-(\frac{1}{2})^2$
= 0.5



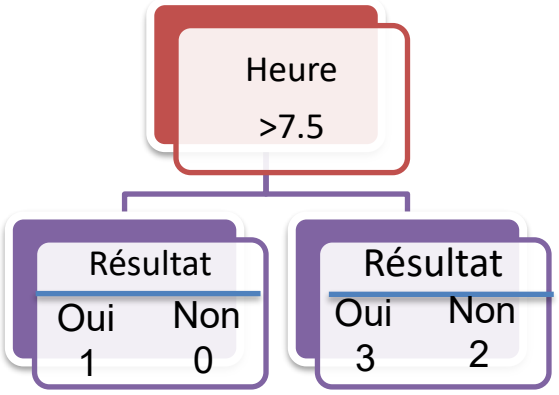
Gini1 = 0

Gini2 =
$1-(\frac{1}{3})^2-(\frac{2}{3})^2$
= 0.444



Gini1 = 0

Gini2 =
$1-(\frac{2}{4})^2-(\frac{2}{4})^2$
= 0.5



Gini1 = 0

Gini2 =
$1-(\frac{3}{5})^2-(\frac{2}{5})^2$
= 0.48

$$G3.5=\frac{1}{6}\times 0+\frac{5}{6}\times 0.32=0.267$$

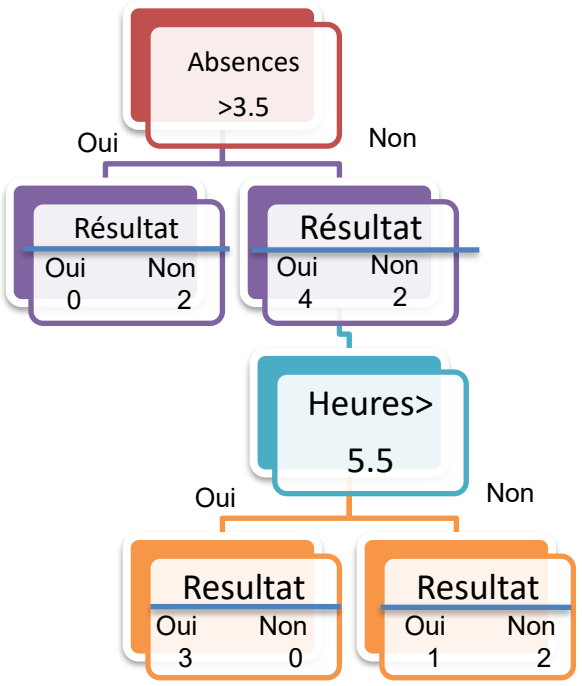
$$G4.5=\frac{4}{6}\times 0.375+\frac{2}{6}\times 0.5=0.417$$

$$G5.5=\frac{3}{6}\times 0+\frac{3}{6}\times 0.444=0.222$$

$$G6.5=\frac{2}{6}\times 0+\frac{4}{6}\times 0.5=0.333$$

$$G7.5=\frac{1}{6}\times 0+\frac{5}{6}\times 0.48=0.4$$

Exemple2



Étudiant	Heures étude	Absences	Résultat
A	8	1	Oui
B	7	2	Oui
C	6	0	Oui
F	4	3	Oui
G	3	2	Non
H	5	3	Non

Gini1 = 0
On arrête le développement de ce côté

Gini1 # 0
Même si un nœud n'est pas homogène, si l'arbre n'a plus de questions à poser, il s'arrête et choisit la classe majoritaire

Construction d’un arbre de décision (Entrainement)

Étape 1 — Nœud initial (racine)
Commencer avec un nœud racine contenant l’ensemble des données d’apprentissage étiquetées.

- Toutes les observations sont au même endroit, la classe est souvent mélangée.

Étape 2 — Choix de la meilleure variable
Trouver la variable (et le seuil) qui sépare le mieux les données.

- On teste plusieurs variables et plusieurs seuils.
- On choisit celui qui maximise le gain d’information ou minimise l’impureté (Gini)

Étape 3 — Partition des données
Chaque séparation divise les données du nœud en sous-groupes.

- Les données sont réparties dans les branches.
- Chaque branche devient un nouveau nœud

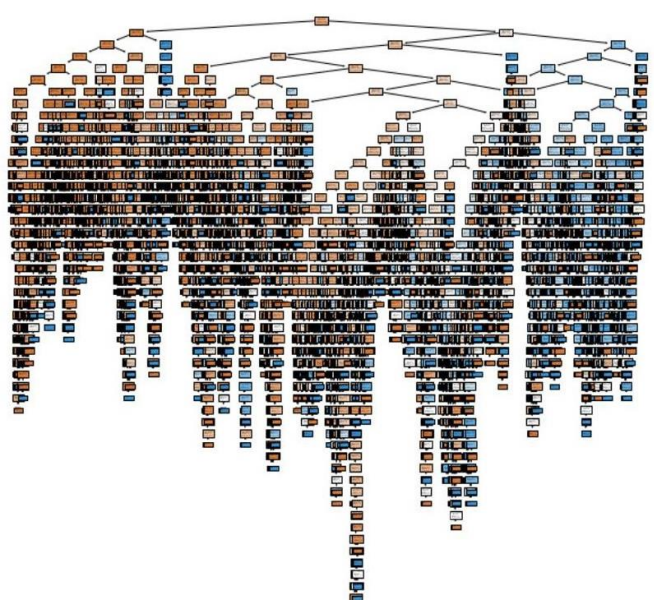
Étape 4 — Répéter le processus
Répéter le même raisonnement pour chaque nouveau nœud.

1. Calculer l’impureté du nœud.
2. Tester les variables restantes.
3. Séparer si possible.
4. S’arrêter si le nœud est pur ou si aucune variable n’est disponible

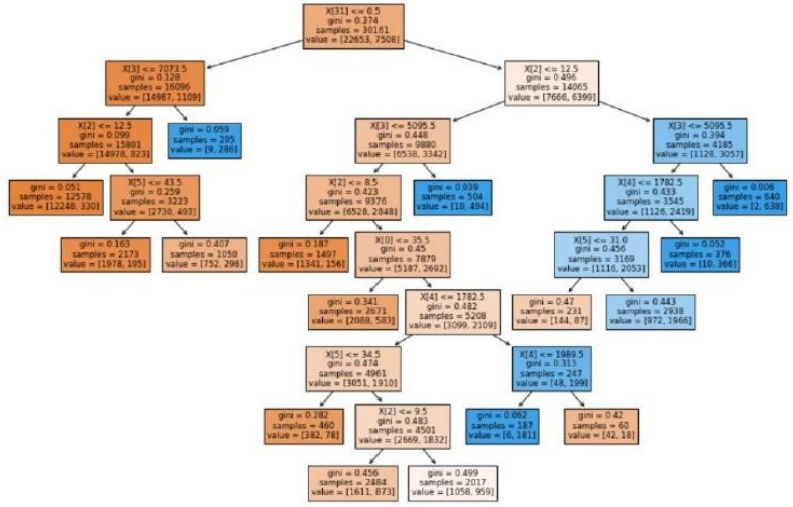
Construction d’un arbre de décision (Problematique d’Entrainement)

Élagage(Pruning)

Même si les arbres de décisions sont relativement simples, ils ont tendance à surajustement, cela se traduit par un arbre comportant beaucoup de nœuds et de division. Après certain niveau l'arbre devient trop compliqué cela réduit la performance du modèle.
L'élagage est une technique qui stoppe le surajustement d'un arbre.



Après élagage



Élagage(Pruning)

L'élagage est une technique qui consiste à supprimer les parties de l’arbre qui ne sont pas nécessaires à la classification Finale, il réduit la complexité du classificateur final et améliore la précision de la prédiction En réduisant le surajustement.

L'élagage est divisé en deux types:

1. Élagage Préalable.

Est effectué pendant L'entraînement du modèle

2. Élagage Postérieur

Est effectué après la génération de l’arbre

Élagage(Pruning)

1. **<u>Élagage Préalable:</u>**
    1. Limitation de la profondeur **(max_depth):**

On fixe une profondeur maximale de l'arbre.

- Empêche des règles trop spécifiques.
- Risque : arbre trop simple → sous-apprentissage

2. Taille Minimale d'un nœud(min_samples_split / min_samples_leaf):

Un nœud n'est divisé que s'il contient suffisamment d'exemples

- Évite des décisions basées sur très peu de données

3. Gain trop faible:

Si la division n'apporte qu'une faible réduction d'impureté, on stoppe.

- Le bruit n'est pas appris comme une règle

Élagage(Pruning)

2. **Élagage Postérieur :**
    1. Réduction d'erreurs.
    2. Coût complexité minimale

Elagage par Reduction d'erreur

Technique d'élagage pour optimiser les arbres de décision

Objectif

Réduire le **surapprentissage** en simplifiant l'arbre après sa construction, tout en maintenant ou améliorant la performance sur de nouvelles données.

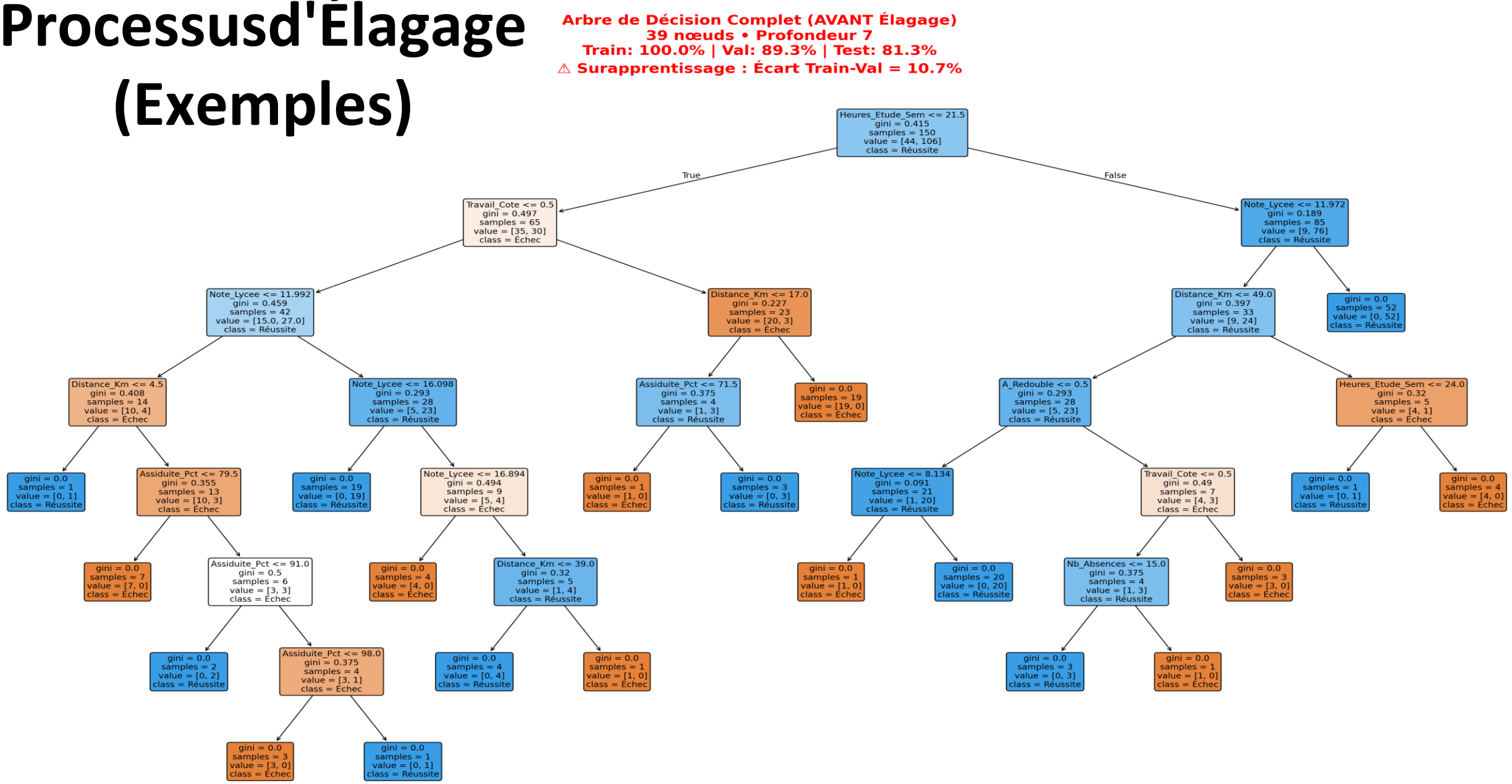
Principe

Utilise un **ensemble de validation** pour tester l'élagage de chaque nœud. Si l'erreur ne s'aggrave pas, le nœud est remplacé par une feuille.

Algorithme

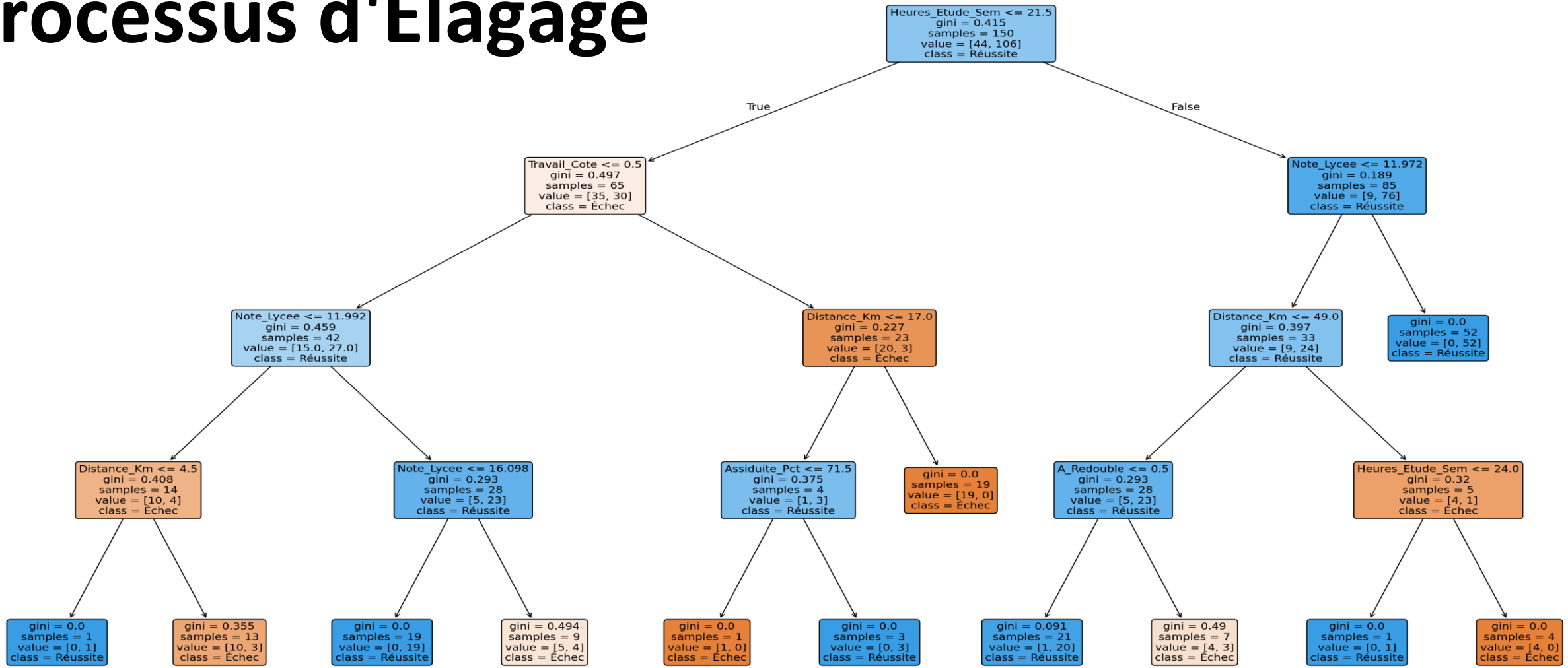
1. Construire un arbre complet sur l'ensemble d'entraînement
2. Pour chaque nœud non-feuille, tester son élagage
3. Évaluer l'erreur sur l'ensemble de validation
4. Si erreur ≤ erreur avant élagage : conserver l'élagage
5. Répéter jusqu'à ce qu'aucun élagage ne soit possible

Processusd'Élagage (Exemples)

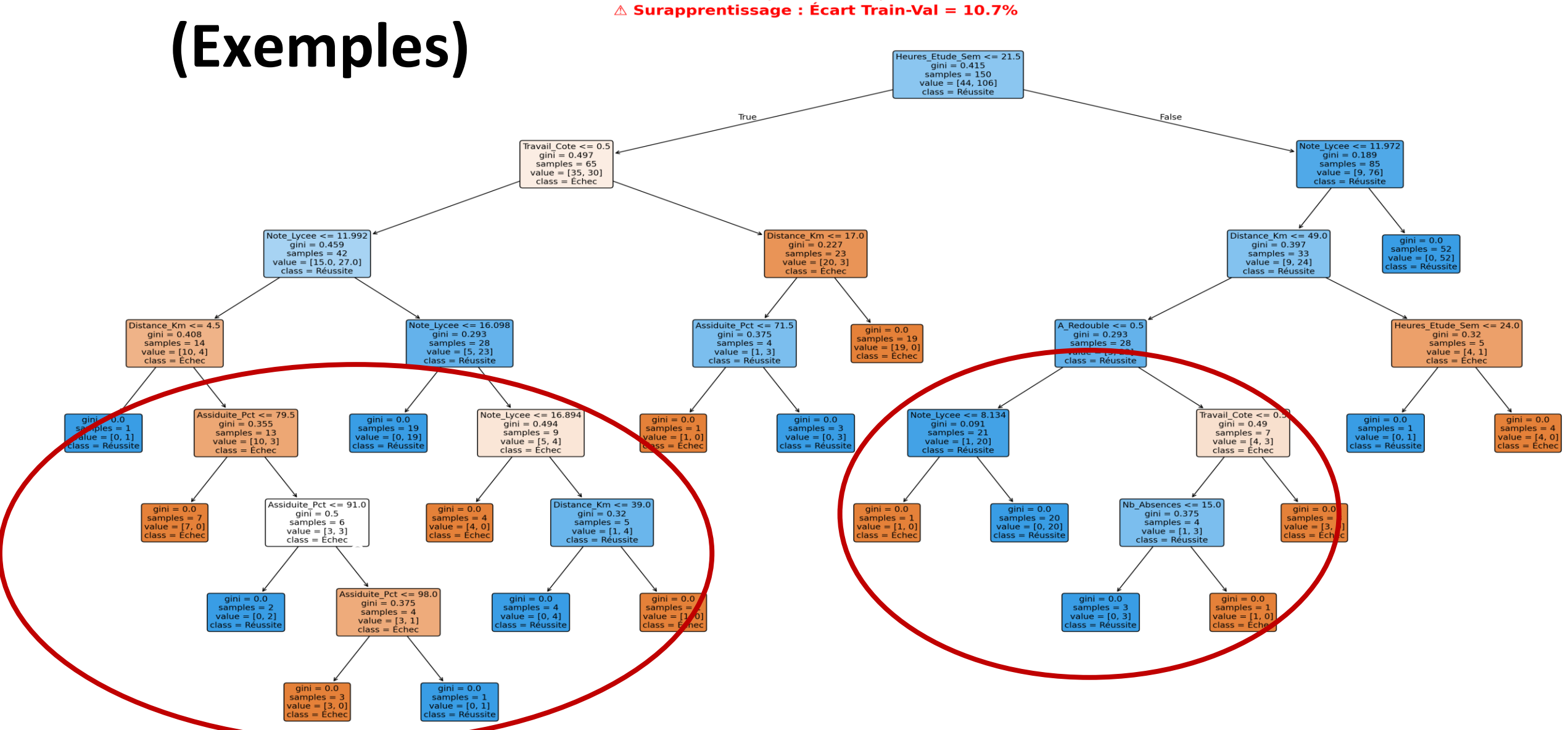


Solution :

Processus d'Élagage



Processusd'Élagage (Exemples)



Solution : Processus d'Élagage (Exemples Réels)

Pour chaque sous-arbre : tester si l'élagage réduit l'erreur de validation

✓

Test : Élaguer "Assiduité?"

Branche: Heures<21.5 → Note<12

Erreur AVANT		Erreur APRÈS
13%	→	11%

✓ ACCEPTER : Amélioration de 2%

L'erreur diminue → On garde l'élagage

✗

Test : Élaguer "Travail_Cote?"

Branche: Heures<21.5

Erreur AVANT		Erreur APRÈS
11%	→	15%

✗ REJETER : Dégradation de 4%

L'erreur augmente → On annule l'élagage

Arbre Initial		Arbre Élagué	
39 nœuds • Erreur 11%	→	23 nœuds • Erreur 11%	-41% nœuds

Solution : Processus d'Élagage (Exemples Réels)

Résultat Final :

Le Reduced Error Pruning a réduit la complexité de 41% tout en maintenant la performance et en réduisant le surapprentissage

Métrique	Arbre Initial	Arbre Élagué	Gain
Nœuds	39	23	**-41%**
Profondeur	7	4	-3 niveaux
Erreur Validation	11%	11%	Maintenue
Surapprentissage	10.7%	3.3%	**-69%**