

Apprentissage Automatique

k-NN, Arbres de décision & méthodes ensemblistes

S. Herbin

stephane.herbin@onera.fr

Rappel du dernier cours

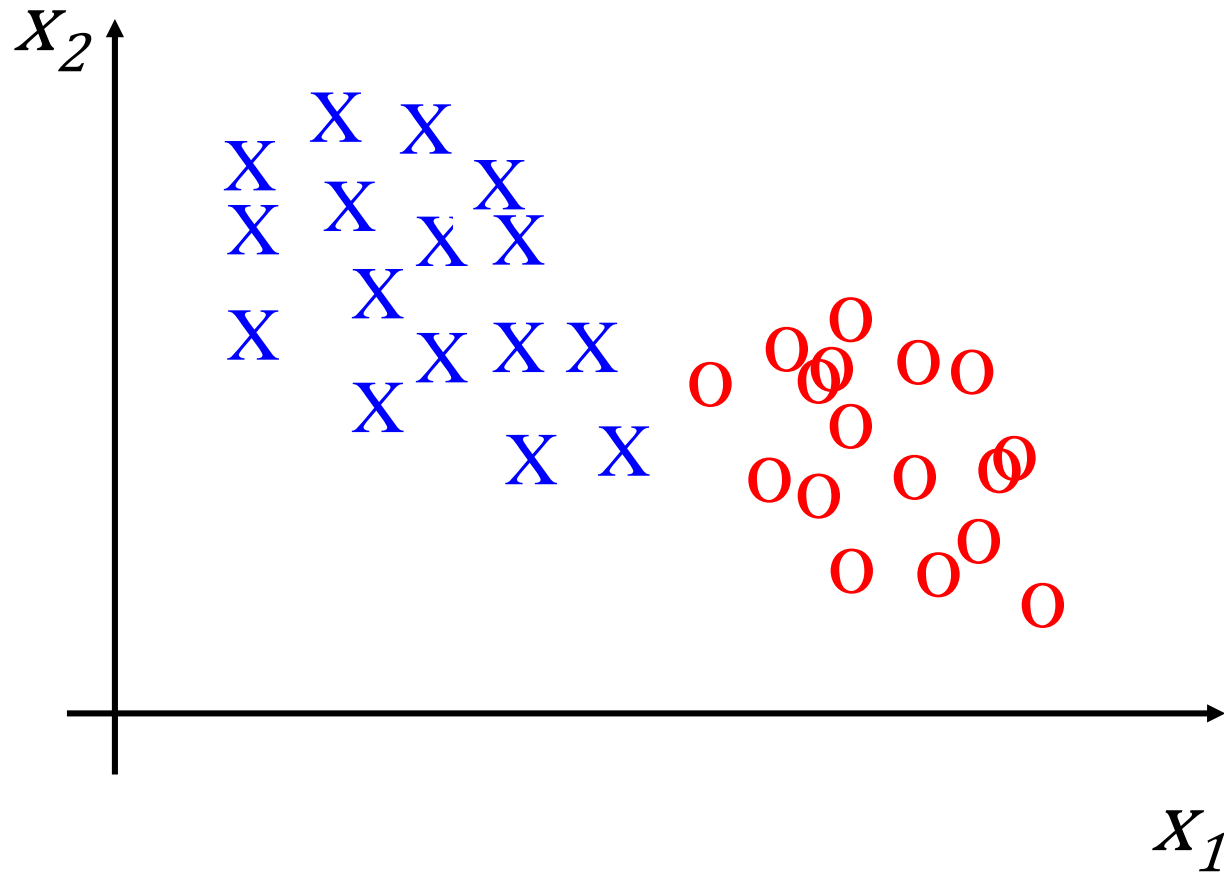
- ▶ Principes généraux d'apprentissage : données apprentissage/validation/test, optimisation, évaluation
- ▶ Deux familles d'algorithmes élémentaires de *classification supervisée* : classifieur Bayésien et discrimination linéaire.

Objectifs de ce cours

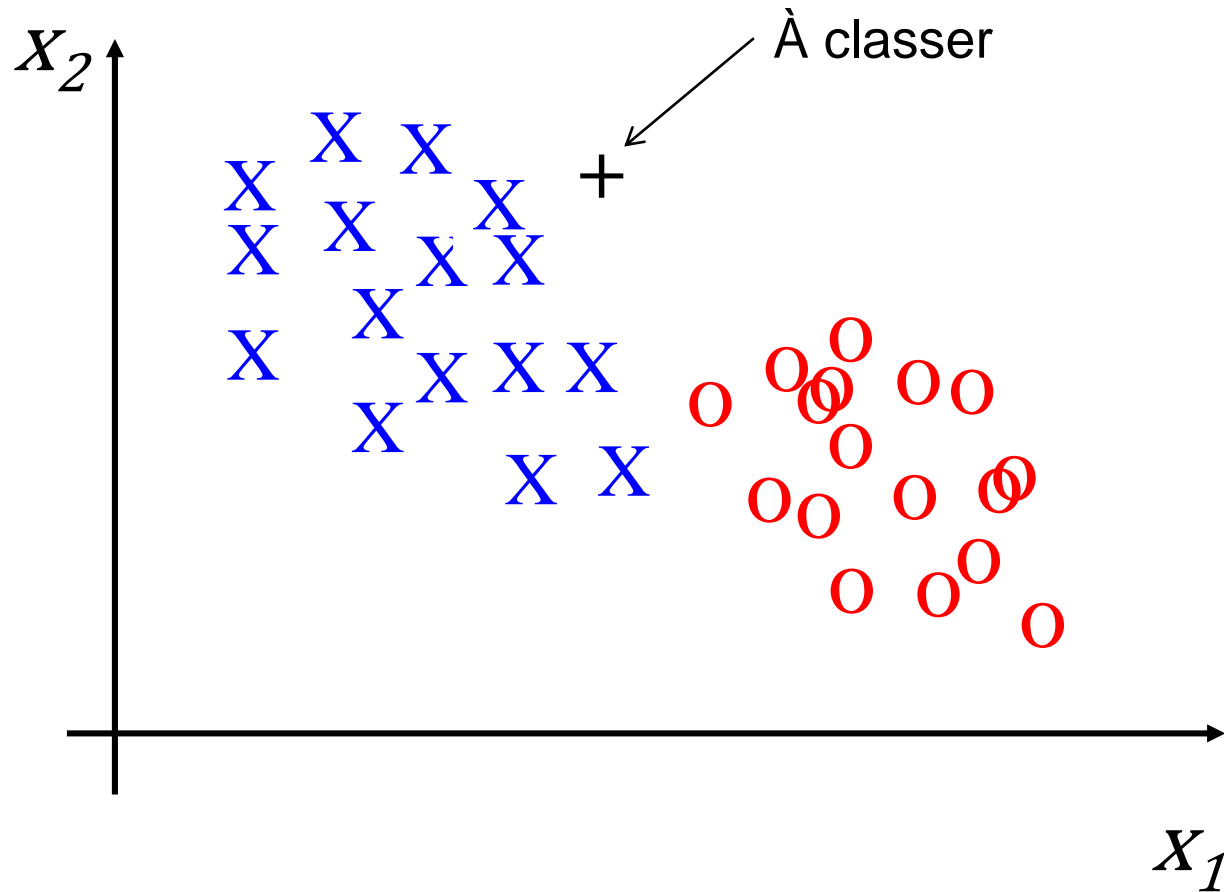
- ▶ Deux nouveaux algorithmes : plus proches voisins et arbre de décision
- ▶ Un principe de conception : les approches ensemblistes
Intuition : « un groupe prend plus souvent de meilleures décisions qu'un individu »

Plus proches voisins

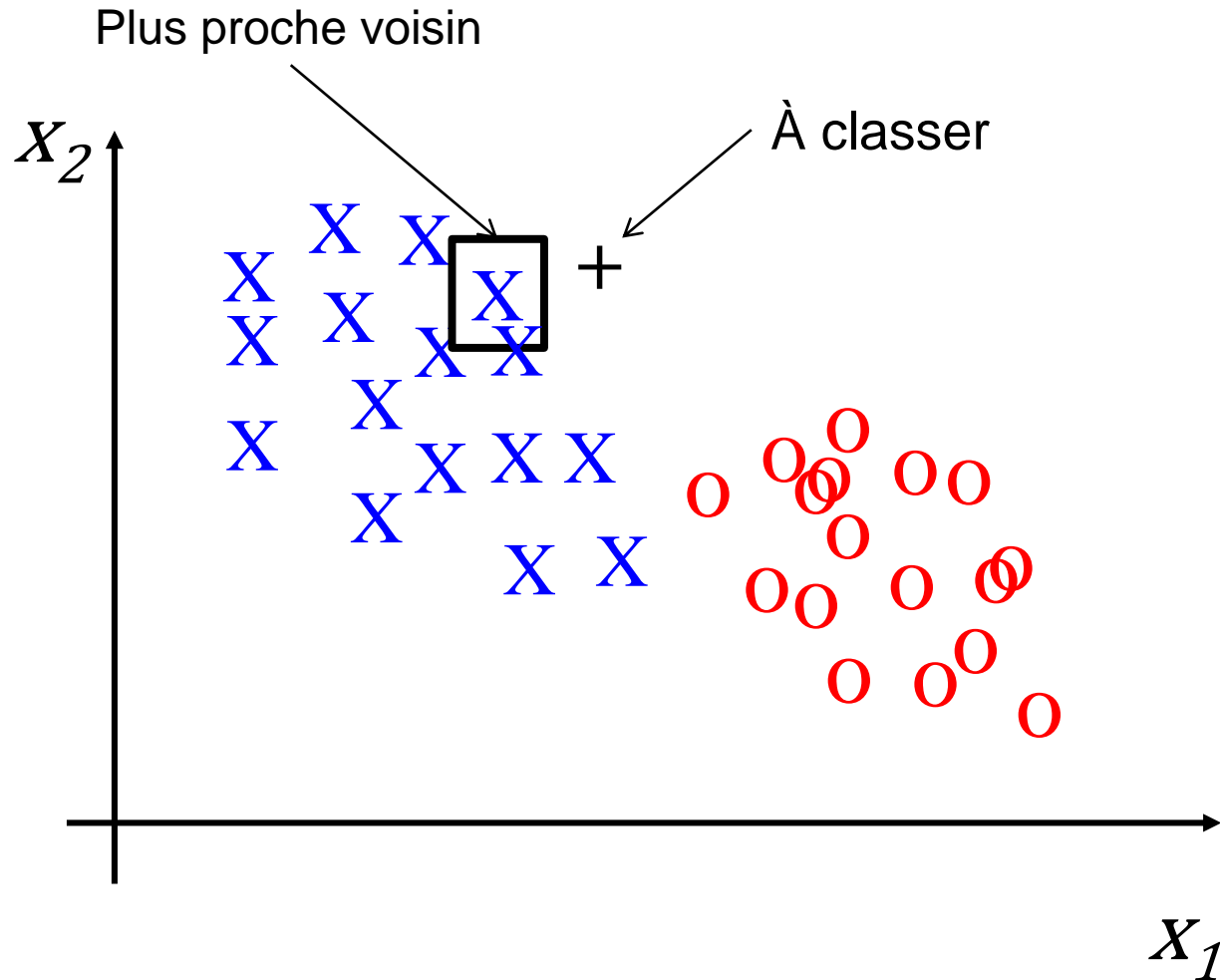
Classification ppv



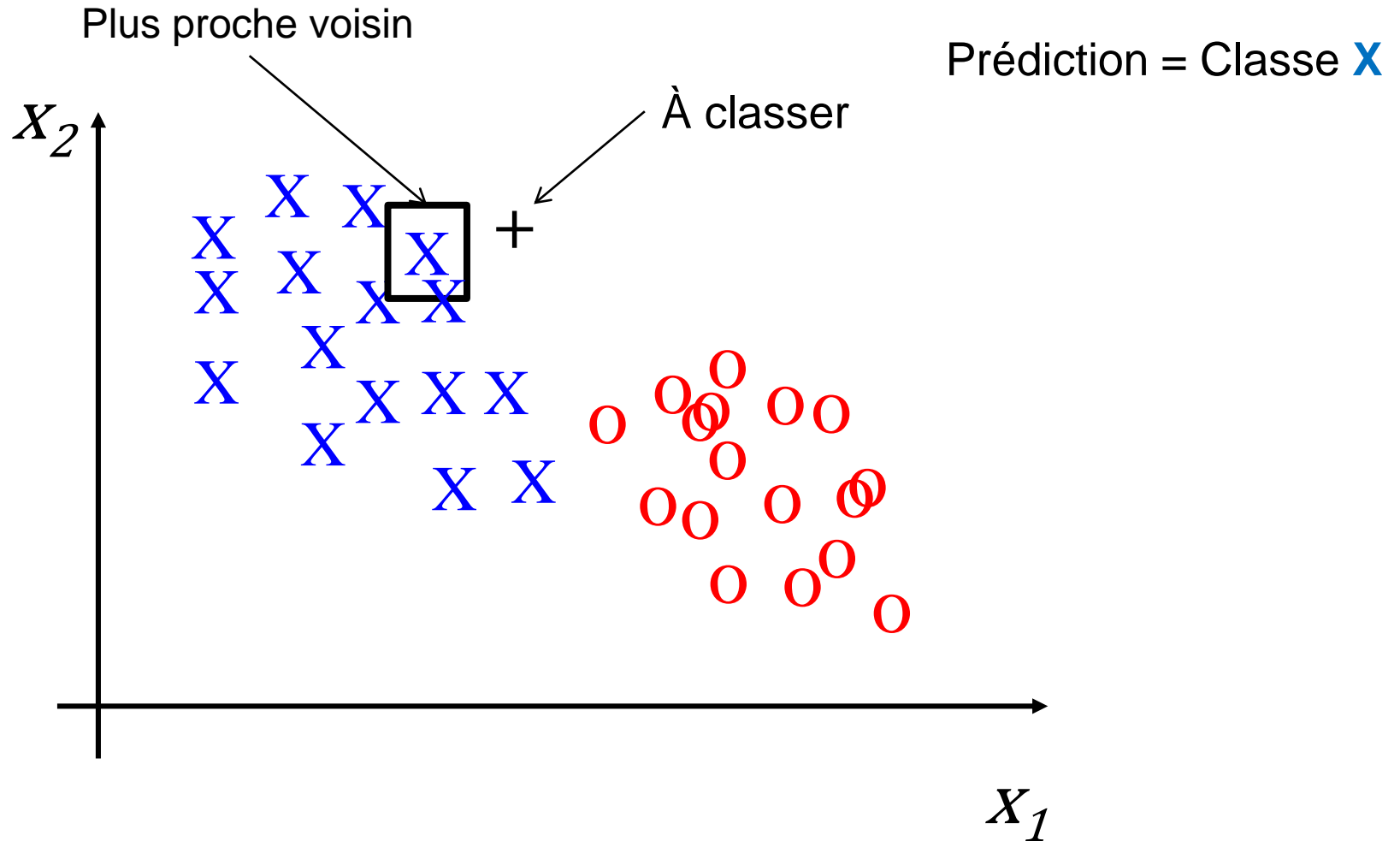
Classification ppv



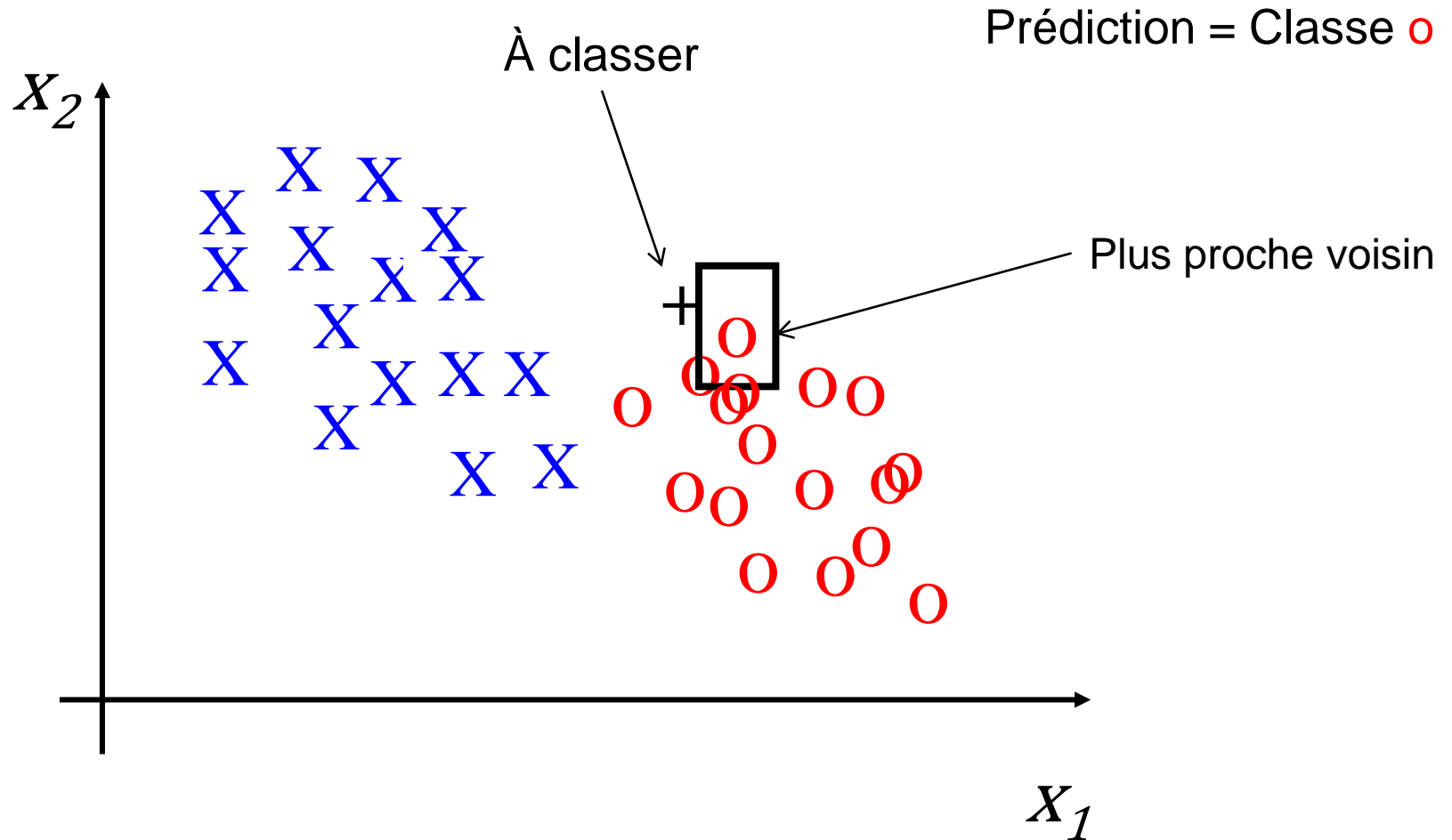
Classification ppv



Classification ppv



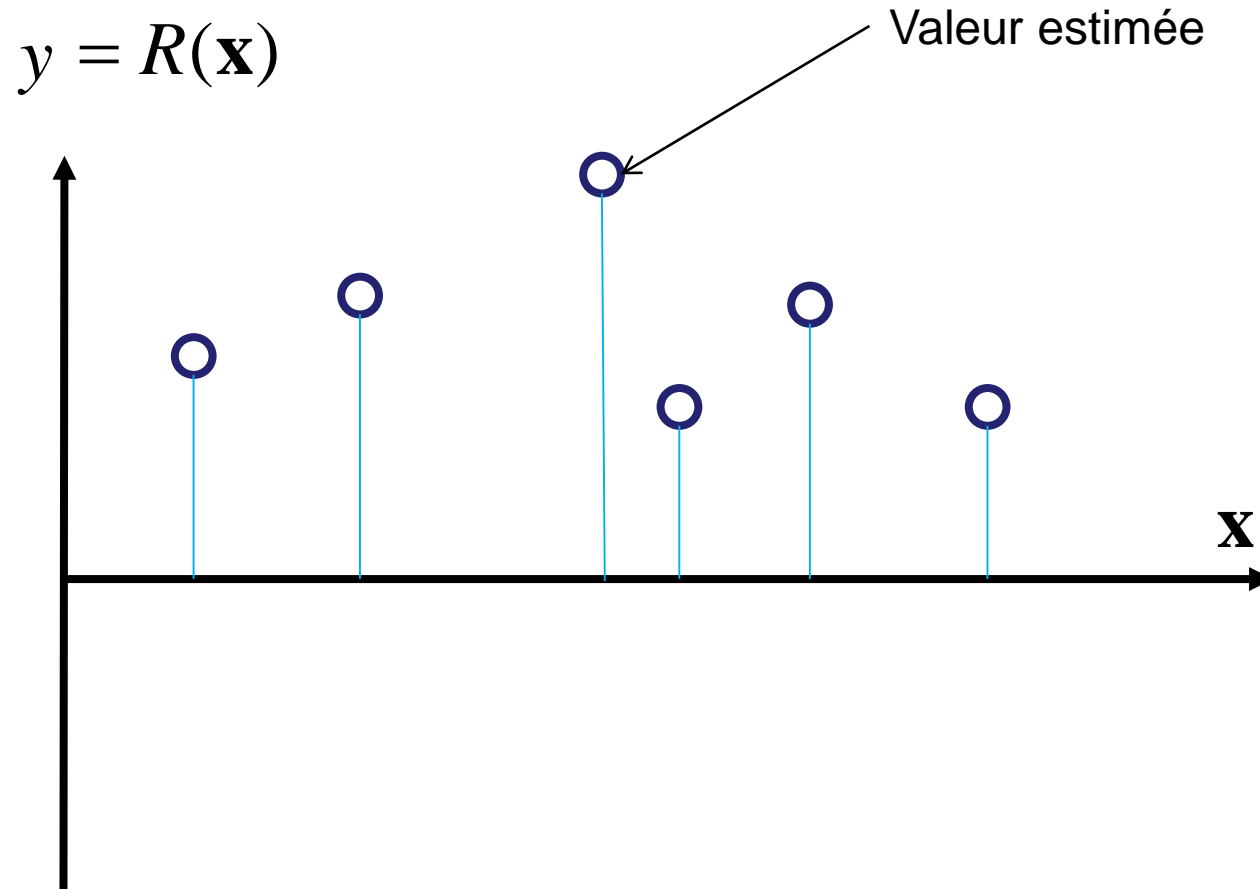
Classification ppv



Plus proche(s) voisin(s)

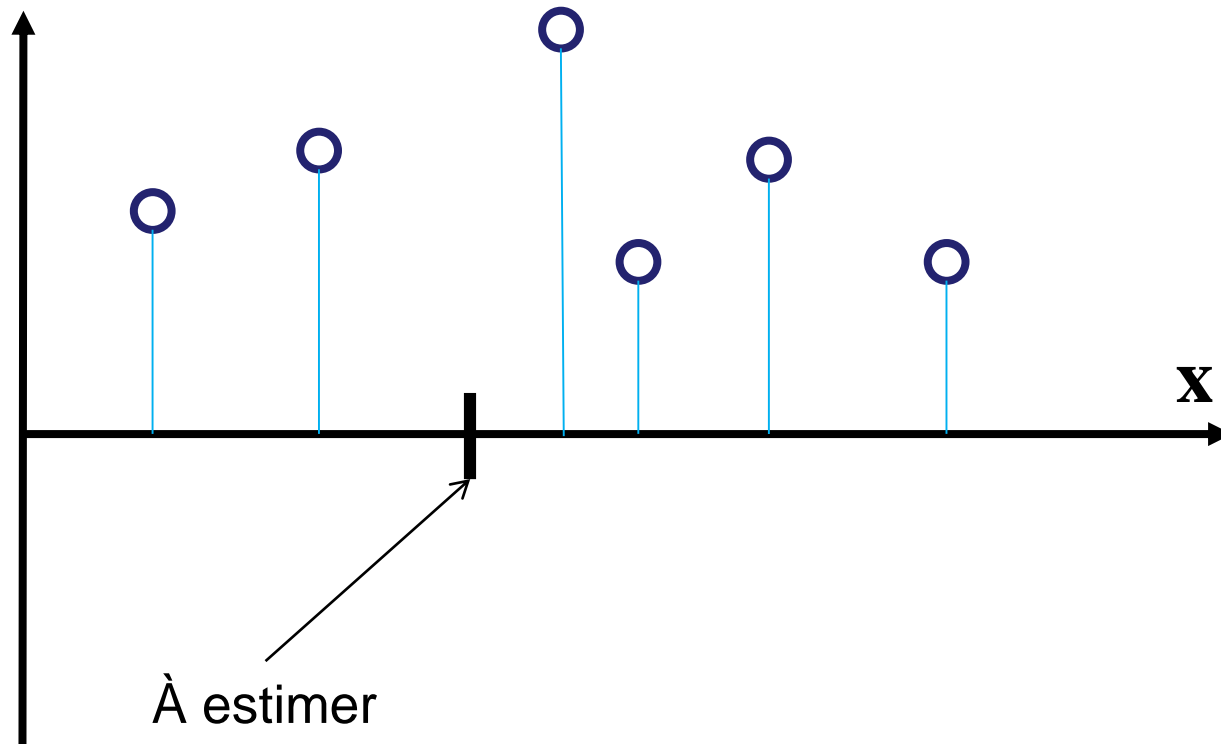
- Principe:
 - Deux échantillons **proches** dans l'espace de représentation ont les mêmes prédictions
 - Pour prédire, il suffit de trouver l'exemple annoté **le plus** proche, et d'associer son annotation (étiquette, valeur...)
- Que veut dire « proche » ?
 - Nécessite la définition d'une métrique ou mesure de similarité $d(\mathbf{x}, \mathbf{x}')$
 - Plusieurs métriques possibles: distance euclidienne (L2), city-block (L1), Minkowski, Mahalanobis...
 - On peut aussi « apprendre » la métrique ou mesure de similarité
- Que veut dire « le plus proche » ?
 - Base d'échantillons annotés $\mathcal{L} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots (\mathbf{x}_N, y_N)\}$
 - Recherche de l'échantillon le plus proche: $i^* = \arg \min_i d(\mathbf{x}, \mathbf{x}_i)$
 - Attribue comme prédiction l'annotation du plus proche: $y^* = y_{i^*}$

Régression PPV

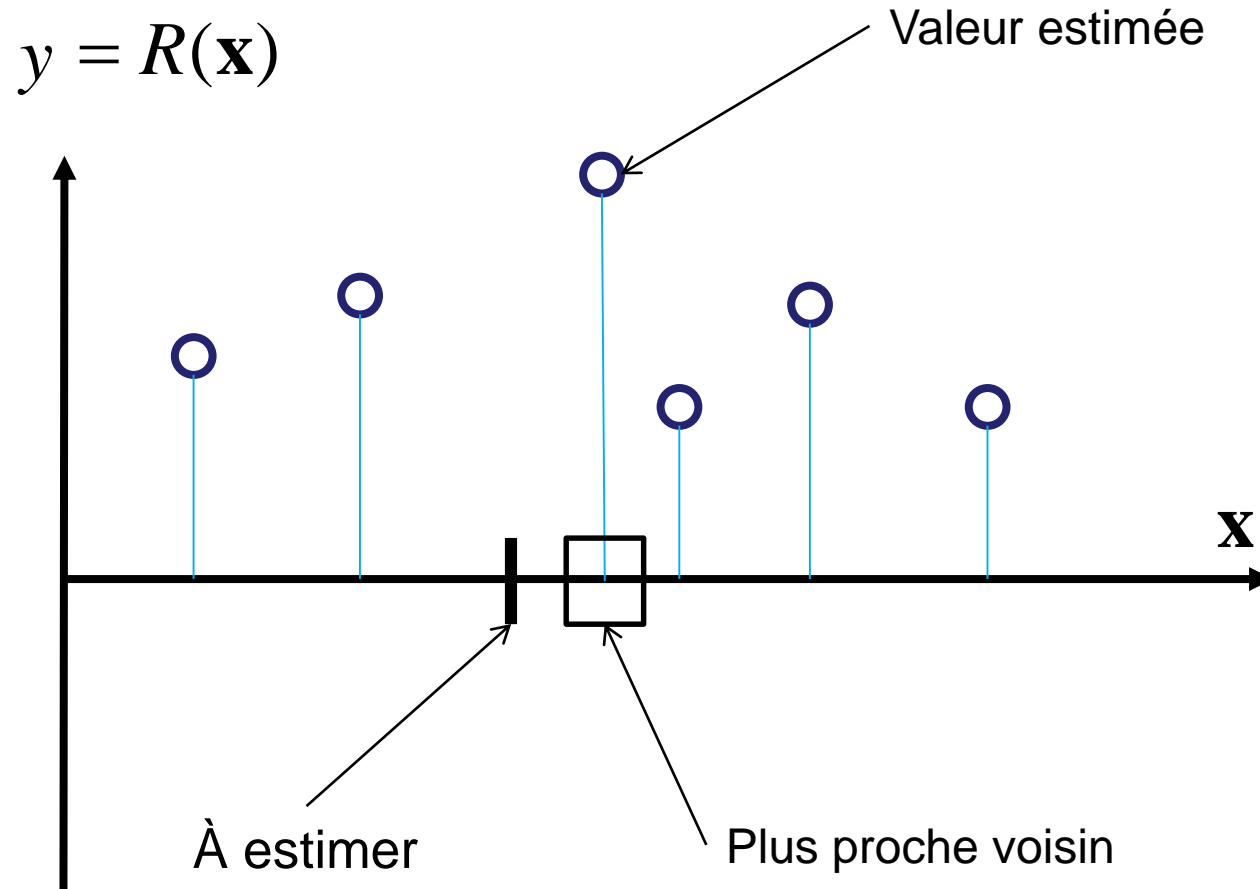


Régression PPV

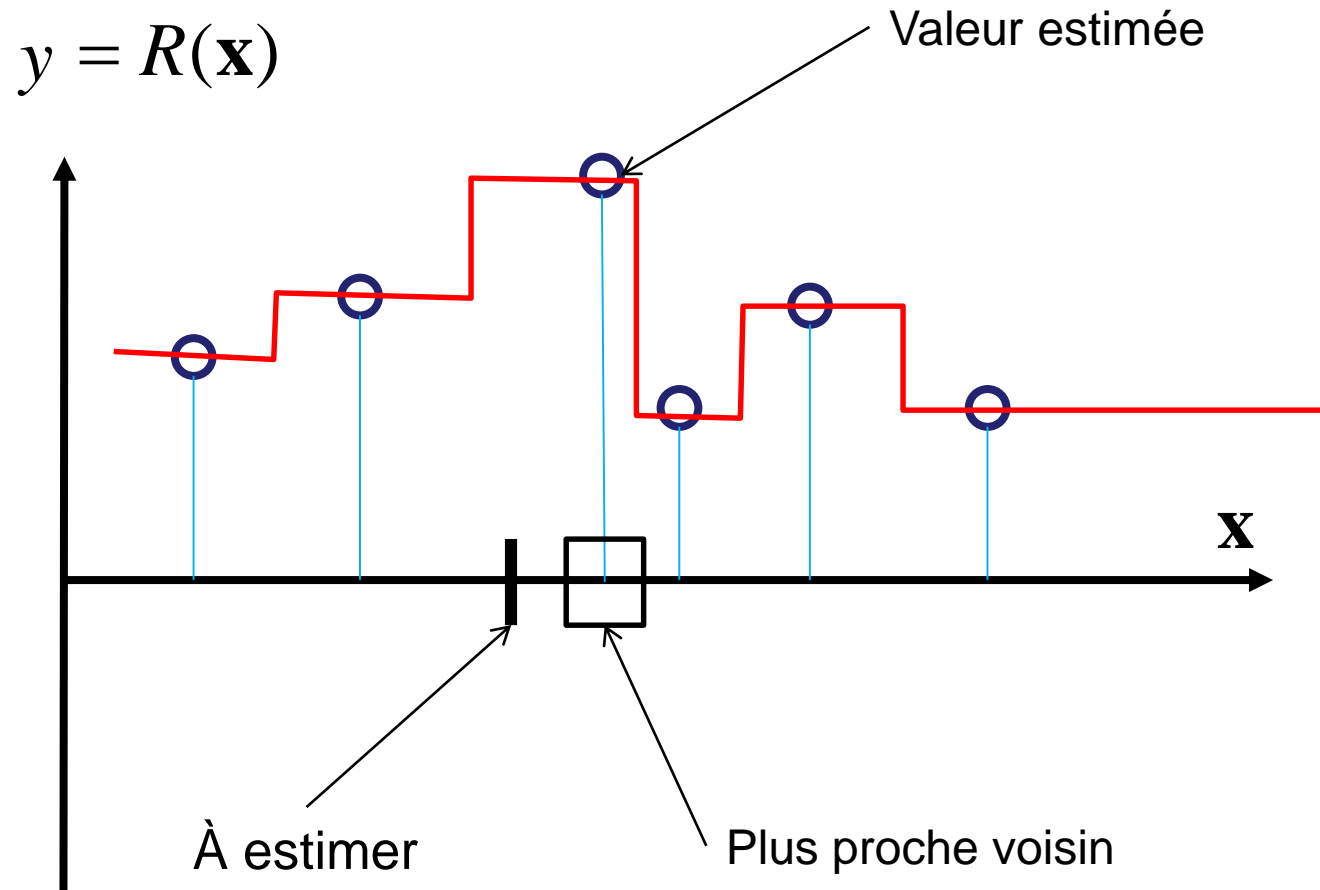
$$y = R(\mathbf{x})$$



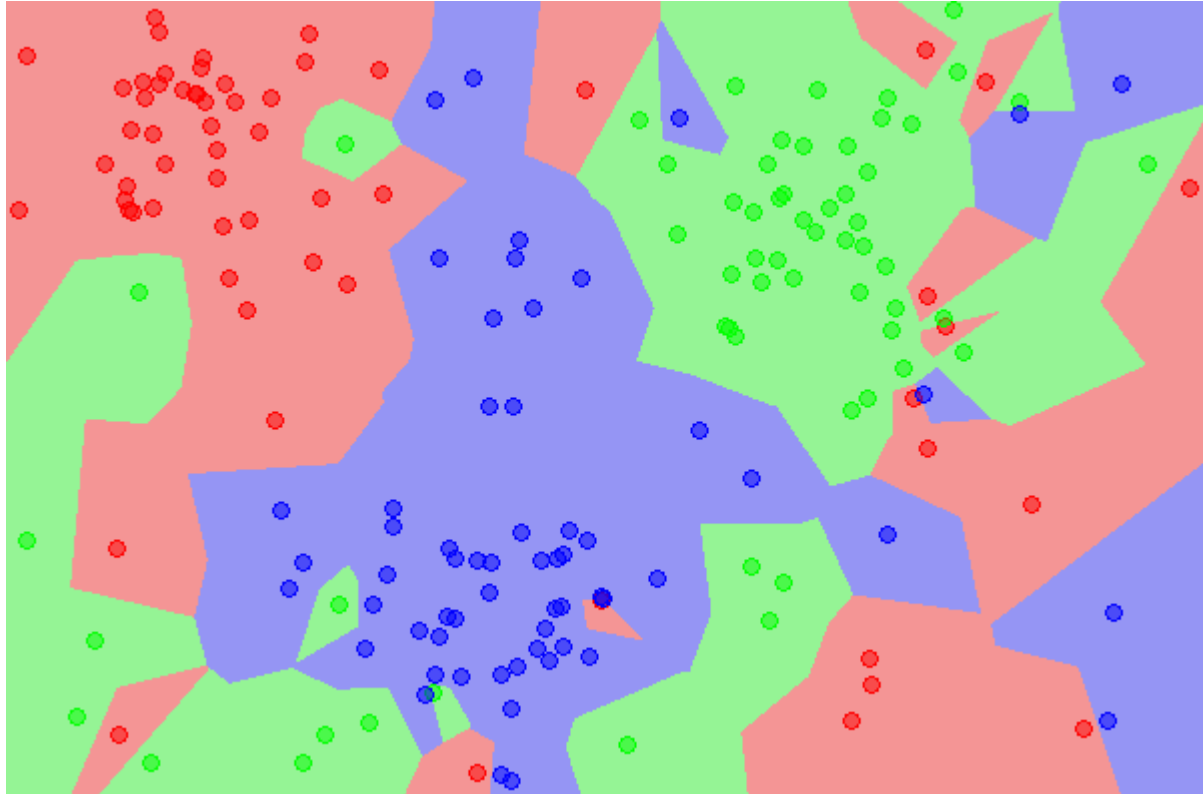
Régression PPV



Régression PPV



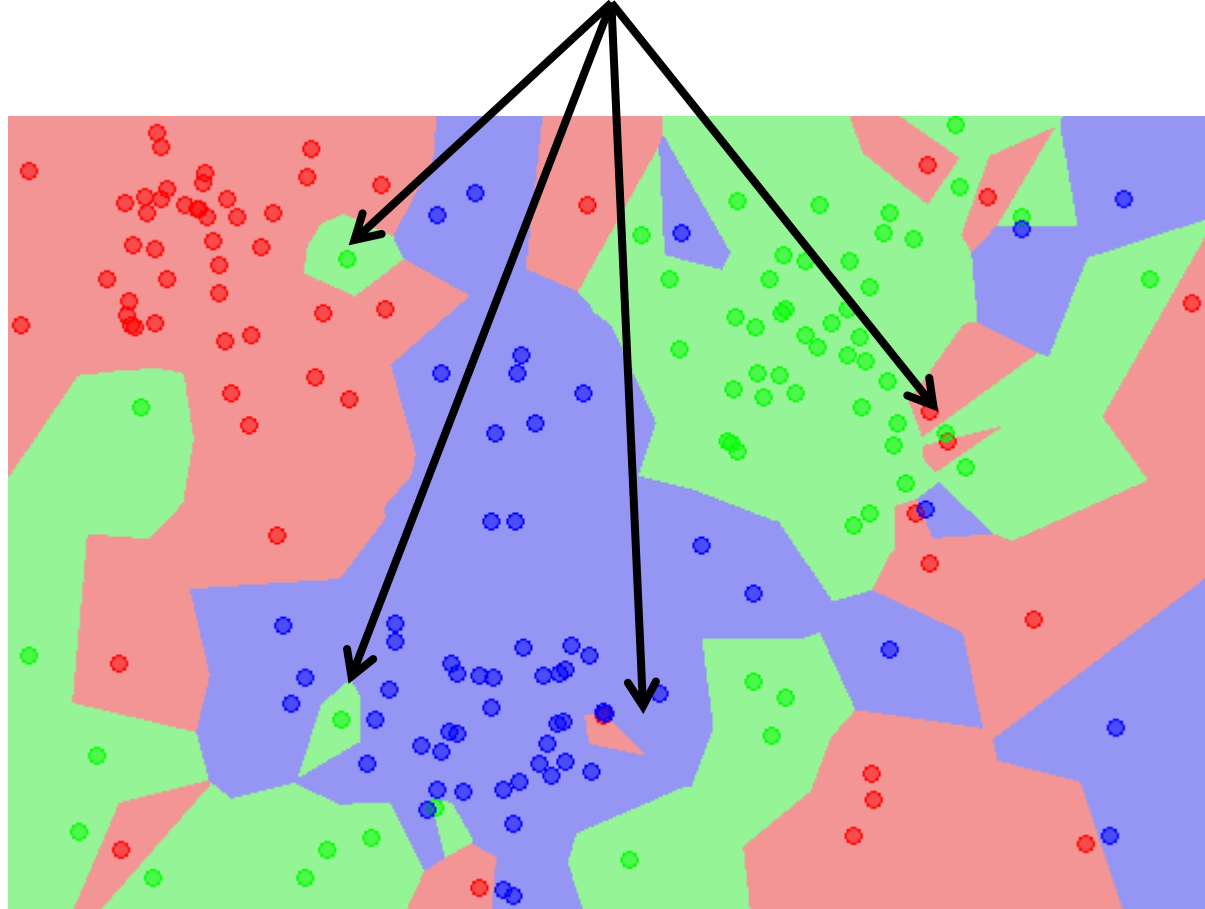
Fonction de classification



Chaque échantillon définit une région homogène de l'espace de représentation

Fonction de classification

Données bruitées → Régions isolées → mauvaise régularité des prédictions



Chaque échantillon définit une région homogène de l'espace de représentation

k-plus proches voisins (« k-NN »)

- Principe: décision à partir de plusieurs exemples de la base de données d'apprentissage
- On ordonne les échantillons d'apprentissage en fonction de leur distance à la donnée à classer:

$$d(\mathbf{x}, \mathbf{x}_{(1)}) \leq d(\mathbf{x}, \mathbf{x}_{(2)}) \leq \dots \leq d(\mathbf{x}, \mathbf{x}_{(N)})$$

- On choisit les k plus proches
- On prédit en choisissant la classe recueillant le plus de votes

$$y^* = \arg \max_y \sum_{i=1}^k \delta(y, y_{(i)})$$

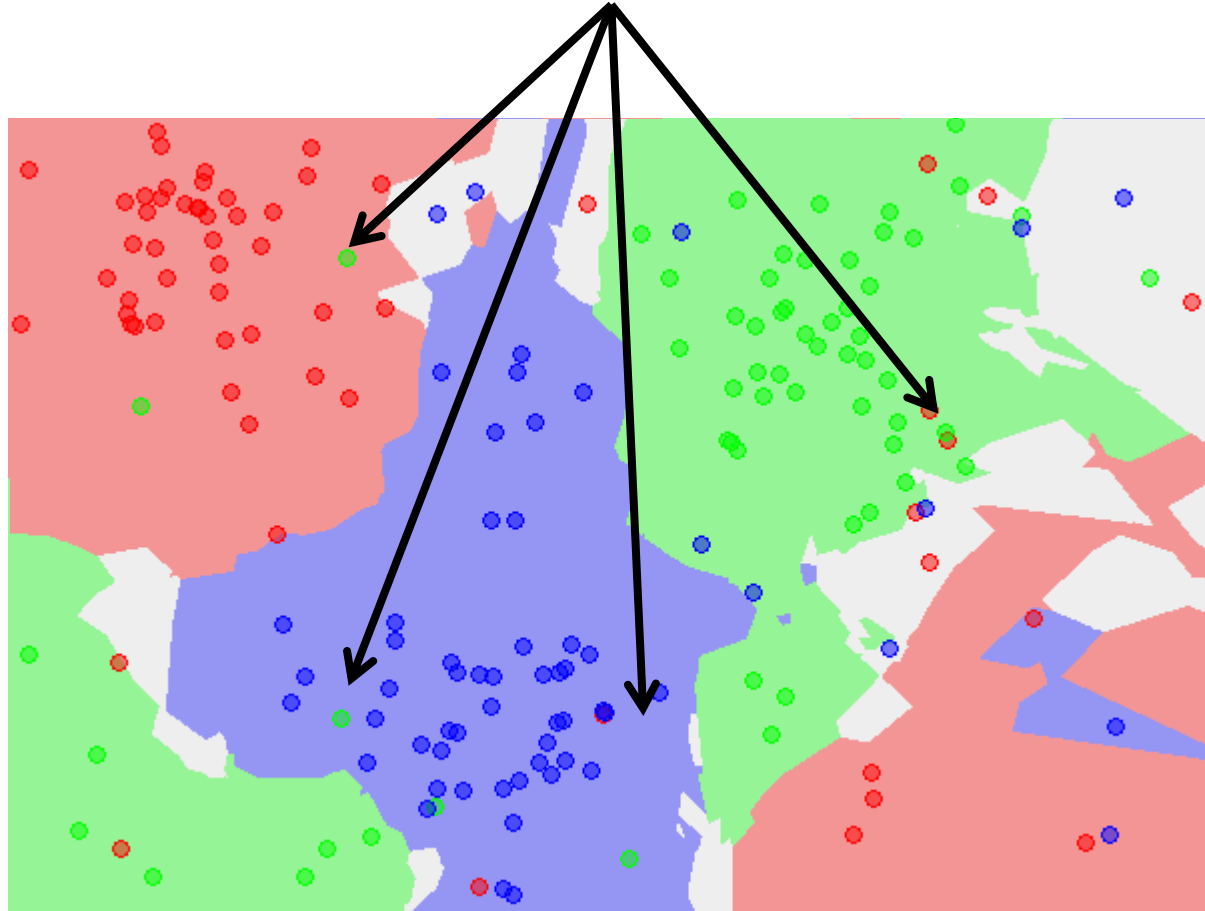
Où δ est la fonction de Kronecker (elle vaut 1 si égal, 0 sinon)

- Si pas de max (ambiguïté sur la prédiction) on ne décide pas!
- On peut aussi pondérer les votes:

$$y^* = \arg \max_y \sum_{i=1}^k K(\mathbf{x}, \mathbf{x}_{(i)}) \delta(y, y_{(i)})$$

Fonction de classification 5 ppv

Données bruitées → Régions isolées → mauvaise régularité des prédictions



Chaque échantillon définit une région homogène de l'espace de représentation

Propriétés statistiques

Bornes statistiques asymptotiques ($N \rightarrow \infty$)

$$E \leq E_{kNN} \leq E \left(2 - \frac{LE}{L-1} \right)$$

Où E est l'erreur théorique optimale (Bayes), L est le nombre de classes et E_{kNN} est l'erreur des k-ppv.

« L'erreur du k-NN est au plus deux fois moins bonne que l'erreur minimale théorique. »

Coût de la prédiction du k-ppv

- Calcul de la prédiction dépend pour chaque exemple x d'un calcul + tri par rapport aux N exemples de la base:

$$d(x, x_{(1)}) \leq d(x, x_{(2)}) \leq \dots \leq d(x, x_{(N)})$$

- Pour N et d grands, coût important de la recherche exhaustive $O(Nd)$. Il existe:
 - Des algorithmes efficaces de recherche pour problèmes de tailles moyennes (KDtree)
J. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transaction on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.
 - Des algorithmes d'approximation pour les grandes bases ($>10^6$).
Jegou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1), 117-128.
- Autre manière: pré-calculer les surfaces de séparation entre classes. La complexité de prédiction est alors liée à la complexité de la surface et/ou de son approximation. On verra comment d'autres approches permettent de l'estimer directement.

La malédiction des grandes dimensions

- Lorsque la dimension d de l'espace de représentation augmente, les points sont tous aussi proches ou aussi loin.
- On peut montrer, pour une distribution quelconque de N points tirés de manière indépendante dans $[0,1]^d$, que:

$$\lim_{d \rightarrow \infty} E \left[\frac{d_{max} - d_{min}}{d_{min}} \right] = 0$$

- Ce n'est plus vrai si les distributions ont une structure...heureusement!
- On peut interpréter les techniques de Machine Learning comme des moyens de repérer les bonnes corrélations entre données.
- Conséquence pour les approches « plus proches voisins »:
 - Ca ne marche que pour les faibles dimensions
 - Ou il faut **réduire** les dimensions de représentation avant de calculer les distances → apprentissage non supervisé

Comportement des PPV

- Avantages
 - Schéma flexible, facile à mettre en œuvre, dépendant de la définition d'une similarité entre données.
 - Bonnes propriétés statistiques ($N \rightarrow \infty$)
- Mais...
 - Temps de calcul prohibitif pour grandes bases
 - Algorithmes efficaces de recherche optimaux ou sous-optimaux
 - Régularité dépend des données, pas de l'apprentissage
 - Le k-PPV (« kNN ») pour lisser et réduire le bruit
 - Malédiction des grandes dimensions (« Curse of dimensionality »)
 - Réduire la dimension de représentation

« Plus proches voisins »: résumé

- Hypothèse de régularité = Si observations proches, même comportement
- Deux questions:
 - Que veut dire « proche »?
 - Comment trouver les plus proches?
- Apprentissage
 - Aucun
- Prédiction
 - Tri des distances aux échantillons + vote
- Quand l'utiliser? (limitations)
 - Efficace sur petits problèmes (dimensions & nombre d'exemples)
 - Pb du « curse of dimensionality » + temps de calcul
 - Disposer d'une mesure de similarité adaptée aux données

Arbres de décision

Explicabilité I

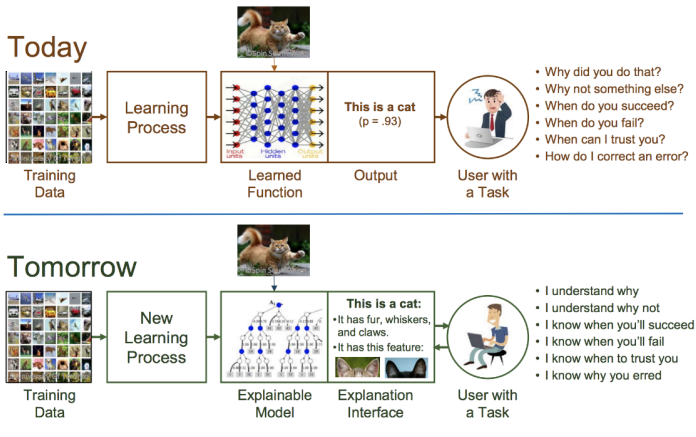


Figure 1 – Programm XAI de la Darpa <https://www.darpa.mil/program/explainable-artificial-intelligence>

Explicabilité II

- ▶ Beaucoup de fonctions de prédiction sont opaques (réseaux de neurones) : il est souvent difficile de comprendre la logique de leur calcul.
- ▶ **Explicabilité** : Fournir des éléments de compréhension du fonctionnement des prédicteurs est un des éléments pour construire une Intelligence Artificielle de confiance.

On va décrire un prédicteur plus facilement interprétable : l'arbre de décision.

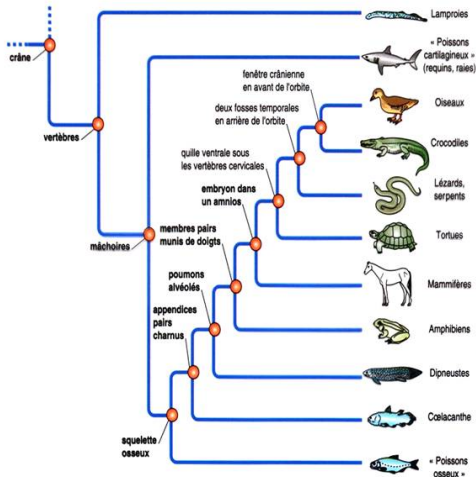
Jeux de déduction



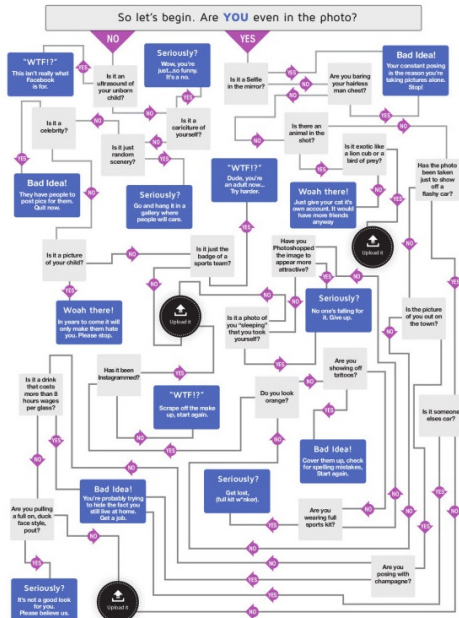
⇒ Quelle meilleure question poser ? (Comment fait <https://fr.akinator.com/> ?)



Classification hiérarchique



Choisir ma photo de profil



Qu'y a-t-il de commun à ces exemples ?

- ▶ Décompose une prédiction globale en une **séquence** de **décisions** (questions+réponses) locales pour
- ▶ **sélectionner** une prédiction pré-estimée.

Les séquences de décisions peuvent être représentées globalement par un **arbre de décision**.

⇒ La question du jour : comment construire les séquences de décision (l'arbre) pour une bonne prédiction ?

Arbres de décision [3, 9]

Principe

- ▶ Prédiction en posant une séquence de questions fermées (= nombre fini de réponses possibles)
- ▶ Questions organisées sous forme d'arbre : la question suivante dépend de la réponse à la question précédente

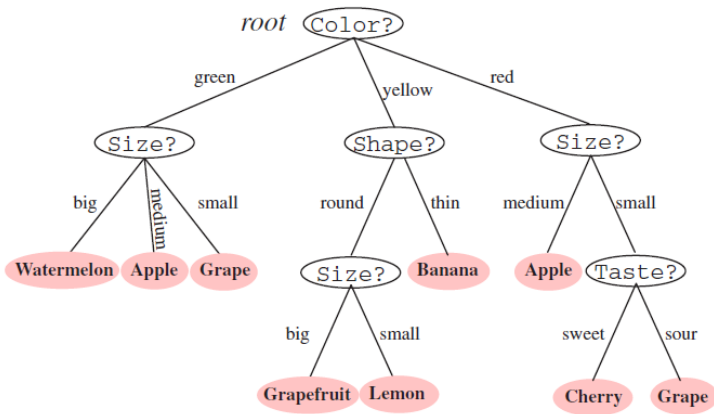
Types de questions

- ▶ Sur la valeur d'un attribut caractéristique : « x est rouge ? »
- ▶ Sur la véracité d'une clause logique :
 $\text{rouge}(x) \wedge \text{rond}(x) = \text{True}?$
- ▶ Sur l'appartenance à un intervalle ou un sous-ensemble : $1_{x > 0.5}$

Prédiction

- ▶ Estimation de la valeur prédite à partir des données pour lesquelles la séquence de questions est vraie

Exemple d'arbre



Arbres de décision : structure

- ▶ **Données** codées comme ensemble d'attributs (ex : attributs d'un fruit = couleur, taille forme, goût...)
- ▶ **Noeud de décision** associé à un **test** ou une **question** sur un des attributs
- ▶ **Branches** qui représentent les valeurs possibles de l'attribut testé ou des réponses aux questions
- ▶ Noeud terminal ou **feuille** définissant la prédiction

Arbres de décision et partition

- ▶ Les questions découpent (partitionnent) l'espace des données à chaque étape
- ▶ Le noeud terminal code un élément de la partition
- ▶ Toutes les données codées par le noeud terminal ont la même prédiction

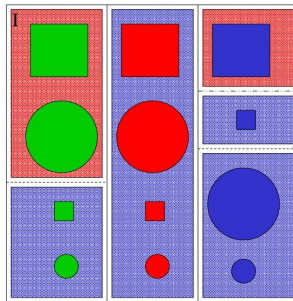
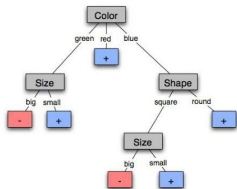


Figure 2 – Partition sur des données symboliques. Les questions portent sur la valeur d'un attribut discret.

Arbres de décision et partition

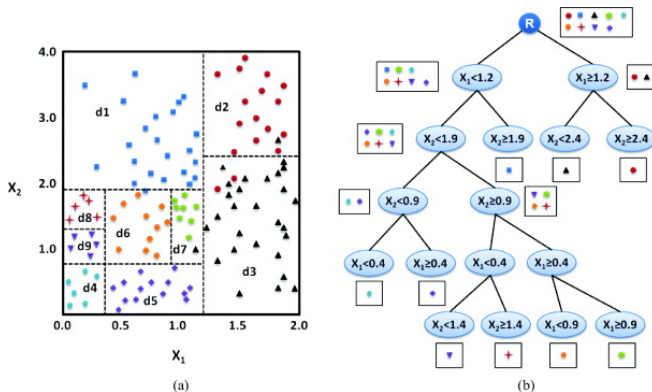
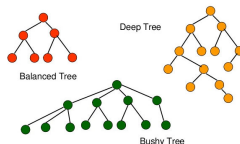


Figure 3 – Prédiction multi-classe. Les questions sont des tests comparant la valeur d'une dimension à un seuil.

Arbres de décision

Quelles questions se poser pour construire un arbre ?



Questions globales :

- ▶ Quelle **structure** choisir ? (profond, équilibré,...) ?
- ▶ Combien de **découpages** par noeud ? (binaire, plus)
- ▶ Quand s'**arrêter** de découper ?

Questions locales :

- ▶ Quel **attribut** choisir, et quel **test** lui appliquer ?
- ▶ Si l'arbre est trop grand, **comment l'élaguer** ?
- ▶ Si une feuille n'est pas pure, **quelle classe attribuer** ?

Arbres de décision : apprentissage

Soit $D = \{(x_j, y_j)\}_{j \leq N}$ un ensemble d'apprentissage où chaque donnée est caractérisée par ensemble d'attributs $x_j = \{a_i^j\}_{1 \leq i \leq M}$, avec a_i^j à valeurs numériques ou symboliques.

Principes pour construire l'arbre de décision compatible avec D :

- ▶ « Rasoir d'Occam » : trouver l'hypothèse explicative la plus simple possible (principe local)
- ▶ « Minimum Description Length » : trouver l'ensemble des hypothèses qui produit le plus petit nombre d'opérations (principe global)

Recherche optimale impossible (problème NP-complet) [10]

⇒ Heuristique assurant un arbre cohérent avec les données d'apprentissage.

Arbres de décision : algorithme élémentaire

Principe général

Construction incrémentale d'un arbre.

Trois étapes

1. Décider si un noeud est **terminal**
2. Si un noeud n'est pas terminal, choisir un attribut, un test et des **branches** possibles
3. Si un noeud est terminal, lui associer une **prédiction** (une classe, une valeur, etc.)

Remarques

- ▶ il est courant de n'utiliser que des tests *binaires* (vrai/faux).
- ▶ il existe des formulations non récursives plus globales [8].

Arbres de décision : Formulation récursive

Fonction Construire_arbre(D)

Si les données de D ont des valeurs *homogènes* (critère d'arrêt)

- ▶ créer une feuille et une prédiction estimée avec la valeur des données

Sinon

- ▶ choisir un attribut a_i et un test T ayant J réponses possibles pour créer un nouveau noeud et une question associée
- ▶ la question partitionne D en J sous-ensembles $\{D_j\}_{j=1\dots J}$ associés à chaque branche j
- ▶ répéter Construire_arbre(D_j) pour chaque branche j

Arbres de décision : comment choisir la bonne question ?

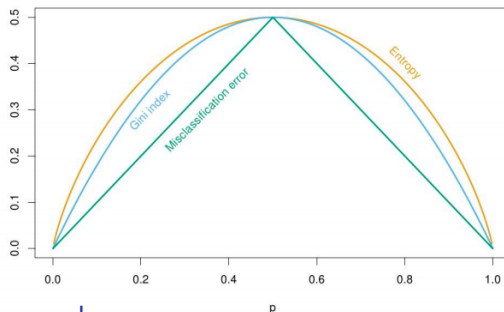
- ▶ A chaque noeud non homogène, on associe une question = attribut + test sur J valeurs.
- ▶ On dispose d'un critère d'hétérogénéité $I(D)$ caractérisant une population de données D .
- ▶ À chaque noeud, on choisit de T^* maximisant le gain en homogénéité :

$$T^* = \arg \max_T \text{Gain}(D, T)$$

où $\text{Gain}(D, T) = I(D) - \sum_j p(D_j|D, T)I(D_j)$
et $p(D_j|D, T) = |D_j|/|D|$ est la proportion de données dans D sélectionnées par la branche j

Remarque : En pratique, le nombre de tests à évaluer peut être très grand. La recherche du $\arg \max$ peut faire intervenir des heuristiques sous-optimales.

Arbres de décision : critères d'homogénéité



Trois critères usuels

- ▶ Entropie : $I(D) = -\sum_k p_k(D) \log_2(p_k(D))$
- ▶ Indice de Gini : $I(D) = \sum_k p_k(D)(1 - p_k(D))$
- ▶ Indice d'erreur : $I(D) = 1 - \max_k(p_k(D))$

où $p_k(D) = N_k(D)/|D|$ est la probabilité d'avoir une donnée de classe k dans l'ensemble D .

Quand s'arrêter ?

Critères structuraux

- ▶ Profondeur maximale
- ▶ Nombre de feuilles minimal

Critères statistiques

- ▶ Indice d'homogénéité minimal
- ▶ Nombre minimal de données en chaque noeud (avant ou après répartition)

Que prédire ?

On exploite la population de données associée à chaque feuille de l'arbre.

Classification

- ▶ Classe la plus probable
- ▶ Distribution de classes

Régression

- ▶ Moyenne, médiane de la population

Exemple simulé

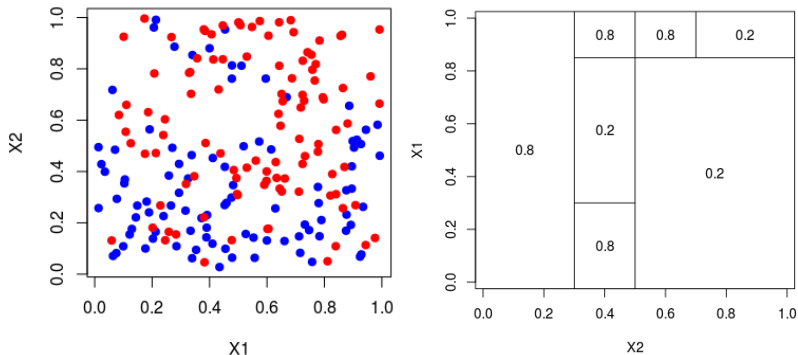


Figure 4 – Distribution simulée et arbre théorique optimal.

Exemple simulé : Recherche du meilleur test

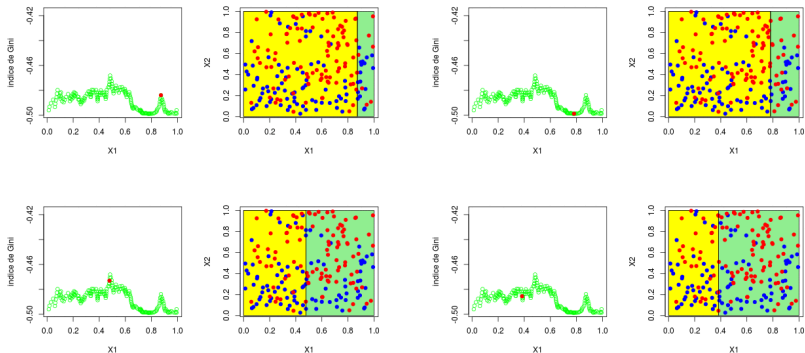


Figure 5 – Recherche sur premier axe avec indice de Gini.

Exemple simulé : Recherche du meilleur test

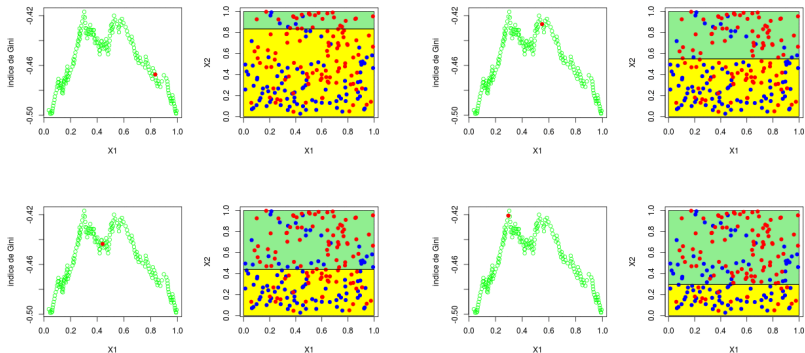


Figure 6 – Recherche sur deuxième axe avec indice de Gini.

Exemple simulé : résultat (indice de Gini)

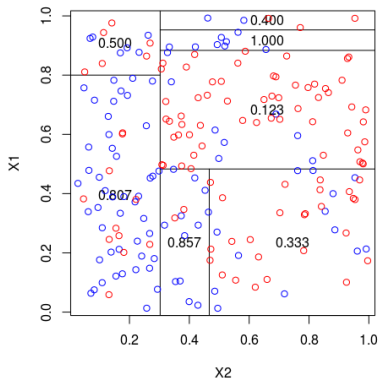
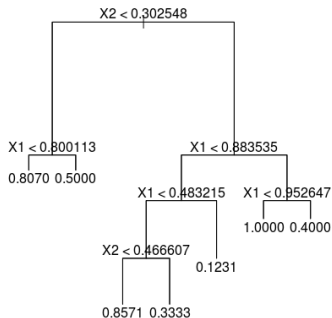
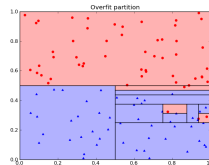
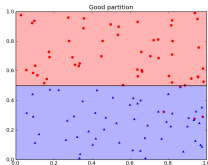
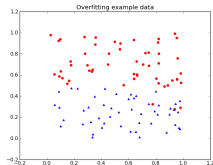


Figure 7 – Arbre et partition finale avec probabilité de classe bleue pour chaque région.

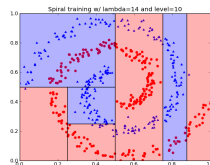
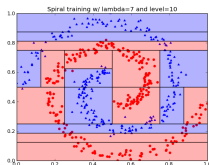
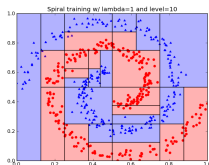
Arbres de décision : comportement statistique



Sur-apprentissage

- ▶ Un arbre trop précis risque de mal généraliser (cf. k-NN)
- ▶ Les arbres peuvent être mal équilibrés
- ▶ On peut utiliser des techniques d'élagage (« pruning ») pour améliorer a posteriori la qualité des arbres

Arbres de décision : comportement statistique



La complexité peut être contrôlée

- ▶ en limitant la profondeur
- ▶ en minorant le gain en homogénéité
- ▶ en ajoutant une pénalisation de complexité dans le coût
- ▶ en garantissant une bonne estimation des coûts (par ex. un nombre minimal d'échantillons par noeud)

Arbres de décision : Résumé

Points clés des arbres de décision

- + Interprétabilité
- + Apprentissage et classification rapides et efficaces, y compris en grande dimension.
- Tendance au surapprentissage (mais moyen de contrôle de la complexité)
- Sensibilité au bruit et aux points aberrants, instabilité

Utilisations

- + Classification ou régression...
- + Capable de traiter des données numériques, mais aussi symboliques

Méthodes ensemblistes

Méthodes ensemblistes

Définition

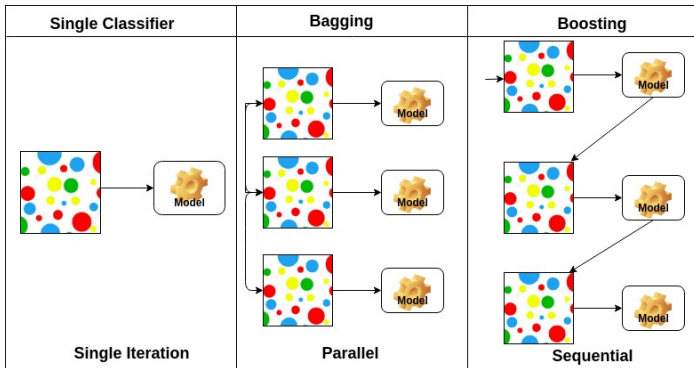
- ▶ Méthodes agrégeant des *ensembles* de classifieurs ;
- ▶ Produire une variété de classifieurs : en échantillonnant différemment les données, en modifiant les structures de classifieurs ;
- ▶ Classe finale = fusion des prédictions.

Principe

- ▶ *L'union fait la force* : tirer parti de plusieurs classifieurs peu performants (« faibles ») pour construire un classifieur performant (« fort »)
 - ⇒ Réduit la variance d'apprentissage et moyenne les erreurs

Méthodes ensemblistes

Deux grandes approches : bagging et boosting



Bagging [1]

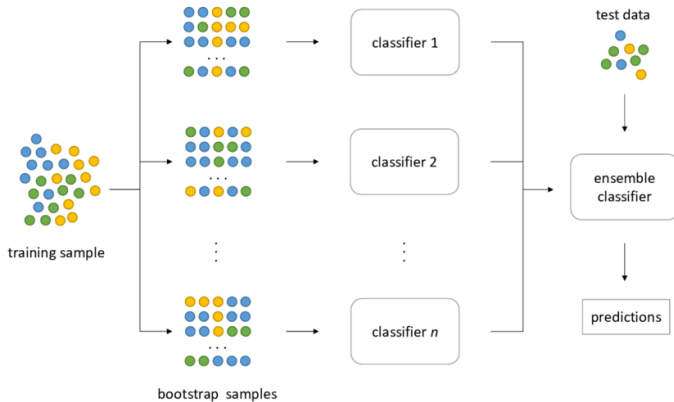
Génération de jeux de données multiples

- ▶ Construction de $\tilde{X}_1, \dots, \tilde{X}_K$ par tirage avec remise sur X .
- ▶ \tilde{X}_k similaires, mais pas trop (proba d'un exemple de ne pas être sélectionné $p = (1 - 1/N)^N$. Quand $N \rightarrow \infty$, $p \rightarrow 0.3679$.)
- ▶ Entraîner K fois le même algorithme f_k (arbre, réseau de neurones, SVM..) sur chaque \tilde{X}_k et agréger par vote majoritaire ou moyenne $f(x) = \frac{1}{K} \sum f_k(x)$

Conséquence

- ▶ Chaque classifieur commet des erreurs différentes, liées à \tilde{X}_k
→ l'agrégat a une plus faible variance d'apprentissage
- ▶ Méthode pour *régulariser* le processus de prédiction.

Bagging



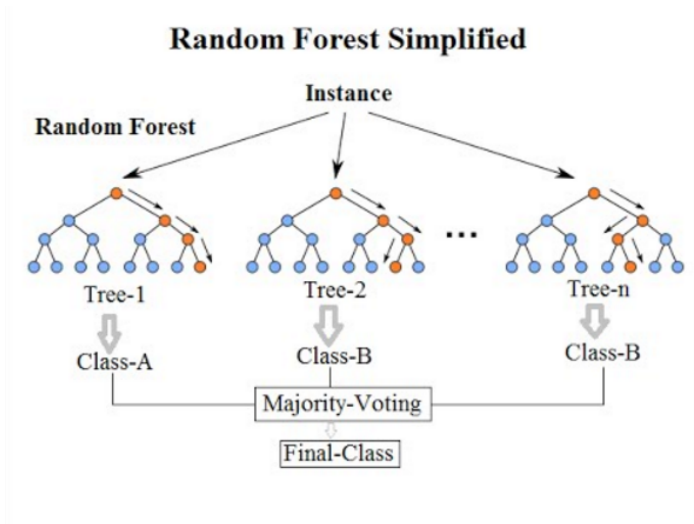
Random Forests [2]

Forêts aléatoires ou *Random forests*

⇒ Combiner hasard et bagging pour construire un ensemble d'arbres de décision encore plus varié (=forêt)

- ▶ La partie calculatoire des arbres de décision est la construction incrémentale de leur structure (meilleure paire attribut & test)
- ▶ Structure = paramètre de contrôle des arbres (profondeur max, critère de pureté des noeuds, nombre d'échantillons par noeud...) + aléatoire sur attributs/données/tests

Random Forests



Random Forests

Forêts aléatoires ou *Random forests*

Algorithme :

POUR $k = 1 \dots K$:

- ▶ Bagging : tirage de \tilde{X}_k de même taille que X
- ▶ Tirage (avec remise) de q attributs A_i parmi les M possibles
- ▶ Construction de l'arbre G_k avec des seuils aléatoires
- ▶ Construction de f_k la fonction de décision de G_k dont les feuilles sont remplies avec \tilde{X}_k

Agrégation :

- ▶ $f(x) = \frac{1}{K} \sum f_k(x)$ (régression)
- ▶ $f(x) = \text{Vote majoritaire}(f_1(x), \dots, f_K(x))$

Intérêt des approches ensemblistes

On introduit une source d'aléatoire supplémentaire : choix des splits, du sous ensemble de variables, etc.

Lorsque les prédicteurs individuels sont sans biais (c'est le cas avec les arbres), la variance du prédicteur ensembliste est :

$$\text{var} \left(\hat{f}_D(x) \right) = \rho \sigma^2 + \frac{1 - \rho}{K} \sigma^2$$

σ variance d'un prédicteur individuel et ρ corrélation entre deux prédicteurs.

On voit que l'on a intérêt à construire des prédicteurs individuels indépendants ($\rho \approx 0$), et en grand nombre (K grand).

Points clés des forêts aléatoires

- + Bonnes performances
- + Arbres plus décorrélés que par simple bagging
- + Grandes dimensions
- + Robustesse
 - Temps d'entraînement (mais aisément parallélisable).

Utilisation

- Choix d'une faible profondeur (2 à 5), autres hyper-paramètres à estimer par validation croisée
- Classification et régression
- Données numériques et symboliques

Boosting [5]

Principe

- ▶ $X = \{(x_i, y_i)\}_{i=1}^N$ un ensemble de données où $y_i \in \{-1, 1\}$
- ▶ H un ensemble ou une famille de classifieurs $f \mapsto -1, 1$, pas forcément performants → appelés *weak learners*

Objectif du boosting :

- ▶ Construire un classifieur performant $F(x) = \sum_{k=1}^K \alpha_k f_k(x)$
→ appelé *strong learner*
 - ▶ Moyenne pondérée des *weak learners*
- ▣ Comment trouver les poids ?

Boosting

AdaBoost

- ▶ Adaboost = « Adaptive boosting algorithm », algorithme minimisant l'erreur globale de F de manière itérative
- ▶ Principe : à chaque itération k , modifier F^k de manière à *donner plus de poids aux données difficiles* (mal-classées) qui permettent de corriger les erreurs commises par F^{k-1}

Boosting

AdaBoost : algorithme

Initialiser les poids liés aux données :

$$d^0 \leftarrow \left(\frac{1}{K}, \frac{1}{K}, \dots, \frac{1}{K} \right)$$

POUR $t = 1 \dots K$:

- ▶ Entraîner f_k sur les données X pondérées par d^{k-1}
($f_k = \arg \min_f \sum_i d_i^{k-1} [y_i \neq f(x_i)]$)
- ▶ Prédire $\hat{y} = y^i \leftarrow f_k(x_i), \forall i$
- ▶ Calculer l'erreur pondérée $\epsilon^k \leftarrow \sum_i d_i^{k-1} [y_i \neq \hat{y}_i]$
- ▶ Calculer les paramètres adaptatifs $\alpha^k \leftarrow \frac{1}{2} \log \left(\frac{1-\epsilon^k}{\epsilon^k} \right)$
- ▶ Re-pondérer les données $d^k = d_i^k \leftarrow d_i^{k-1} \exp(-\alpha^k y_i \hat{y}_i)$

Classifieur (pondéré) final : $F(x) = \text{sgn} \left(\sum_{k=1}^K \alpha_k f_k(x) \right)$

Boosting

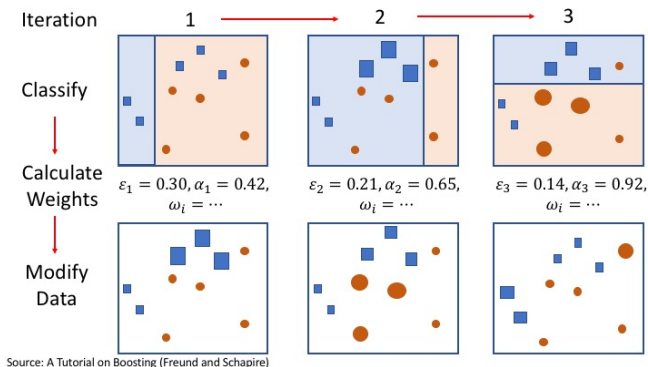
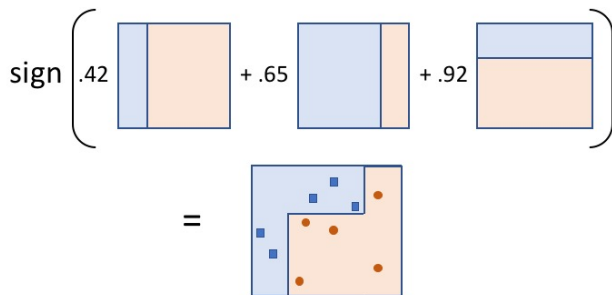


Figure 8 – Apprentissage séquentiel des classifieurs et des pondérations.

Boosting



Source: A Tutorial on Boosting (Freund and Schapire)

Figure 9 – Classifieur final.

Gradient Boosting [6, 7]

Gradient Boosting

Variante : version additive pas-à-pas

- ▶ $X = \{(x_i, y_i)\}_{i=1}^N$ un ensemble de données où $y_i \in \{-1, 1\}$
- ▶ H un ensemble de classifieurs $f \mapsto -1, 1$, pas forcément performants \rightarrow appelés *weak learners*

Objectif du gradient boosting :

- ▶ Construire itérativement un classifieur performant
$$F_T(x) = \sum_{t=1}^T \alpha_t f_t(x) = F_{T-1}(x) + \alpha_T f_T(x)$$
 où f_t est l'un des weak learners h .
- ▶ Il s'agit à chaque étape de minimiser le risque empirique :
$$\mathcal{L}(F_T) = \sum_{n=1}^N l(y_n, F_T(x_n))$$
 où l est un coût (*loss*)

Gradient Boosting

Coûts

- ▶ Adaboost \rightarrow gradient boost avec fonction de coût
 $l(y, f(x)) = \exp(-y.f(x))$
- ▶ Adaboost peut être vu comme la construction itérative d'un classifieur optimal par minimisation du risque empirique à chaque pas.
- ▶ Cadre plus général : d'autres pénalités sont possibles :
 - ▶ LogitBoost : $l(y, f(x)) = \log_2(1 + \exp[-2y.f(x)])$
 - ▶ L_2 Boost : $l(y, f(x)) = (y - f(x))^2/2$
 - ▶ DoomII : $l(y, f(x)) = 1 - \tanh(y.f(x))$
 - ▶ Savage : $l(y, f(x)) = \frac{1}{(1 + \exp(2y.f(x)))^2}$
- ▶ DoomII et Savage sont non-convexes \rightarrow plus robustes aux données bruitées

Gradient Boosting

Pourquoi *Gradient Boosting* ?

- ▶ Chaque étape minimise le risque empirique :
 $\mathcal{L}(F_T) = \sum_{n=1}^N l(y_n, F_T(x_n))$ où l est un coût (*loss*)
- ▶ Lors de la variante additive d'adaboost, $\alpha_T f_T(x)$ peut donc être vu comme le *weak learner* qui approxime le mieux le pas d'une descente de gradient dans l'espaces des fonctions de classification
- ▶ Une version exacte de la descente de gradient donne les Gradient Boosting Models :
$$F_T(x) = F_{T-1}(x) + \alpha_T \sum_{i=1}^N \nabla_{F_{T-1}} l(y_i, f_{T-1}(x_i))$$

Boosting : Résumé

Points clés du boosting

- ▶ Agrégation adaptative de classifieurs moyens
- + Résultats théoriques sur la convergence et l'optimalité du classifieur final
- + Très efficace (améliore n'importe quel ensemble de classifieurs)
- + Assez facile à mettre en oeuvre (moins vrai pour Gradient Boosting)
- Sensibilité aux données aberrantes, surapprentissage

Utilisations

- ▶ Choix du *weak learner* : ne doit pas être trop bon, sinon surapprentissage
- ▶ Choix de la pénalité en fonction du bruit des données
- ▶ Variantes pour la classification et la régression

Cours n°2 : Arbres de décision et méthodes ensemblistes

Notions phares du jour

- ▶ Arbres de décision (vote, homogénéité)
- ▶ Aggrégation de classifieurs
- ▶ Bagging, Random Forests
- ▶ Boosting, GradientBoost

Concepts généraux

- ▶ Classification / régression
- ▶ Bagging et randomisation (Forêts aléatoires)
- ▶ Construction adaptative à partir de *weak learners* et optimisation dans l'espace des classifieurs (Boosting)

Références I

- [1] Leo Breiman.
Bagging predictors.
Machine learning, 24(2) :123–140, 1996.
- [2] Leo Breiman.
Random forests.
Machine learning, 45(1) :5–32, 2001.
- [3] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone.
Classification and regression trees.
Routledge, 2017.
- [4] Tianqi Chen and Carlos Guestrin.
Xgboost : A scalable tree boosting system.
In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [5] Yoav Freund and Robert E Schapire.
A decision-theoretic generalization of on-line learning and an application to boosting.
Journal of computer and system sciences, 55(1) :119–139, 1997.
- [6] Jerome H Friedman.
Greedy function approximation : a gradient boosting machine.
Annals of statistics, pages 1189–1232, 2001.
- [7] Jerome H Friedman.
Stochastic gradient boosting.
Computational statistics & data analysis, 38(4) :367–378, 2002.
- [8] Donald Geman and Bruno Jedynak.
Model-based classification trees.
IEEE Transactions on Information Theory, 47(3) :1075–1082, 2001.

Références II

- [9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman.
The elements of statistical learning.
Springer, 2009.
- [10] Hyafil Laurent and Ronald L Rivest.
Constructing optimal binary decision trees is np-complete.
Information processing letters, 5(1) :15–17, 1976.