LIRIS

UMR 5205 CNRS

Méthodes ensemblistes

Haytham Elghazel

Laboratoire d'InfoRmatique en Image et Systèmes d'information

Pôle Data Science, Equipe DM2L

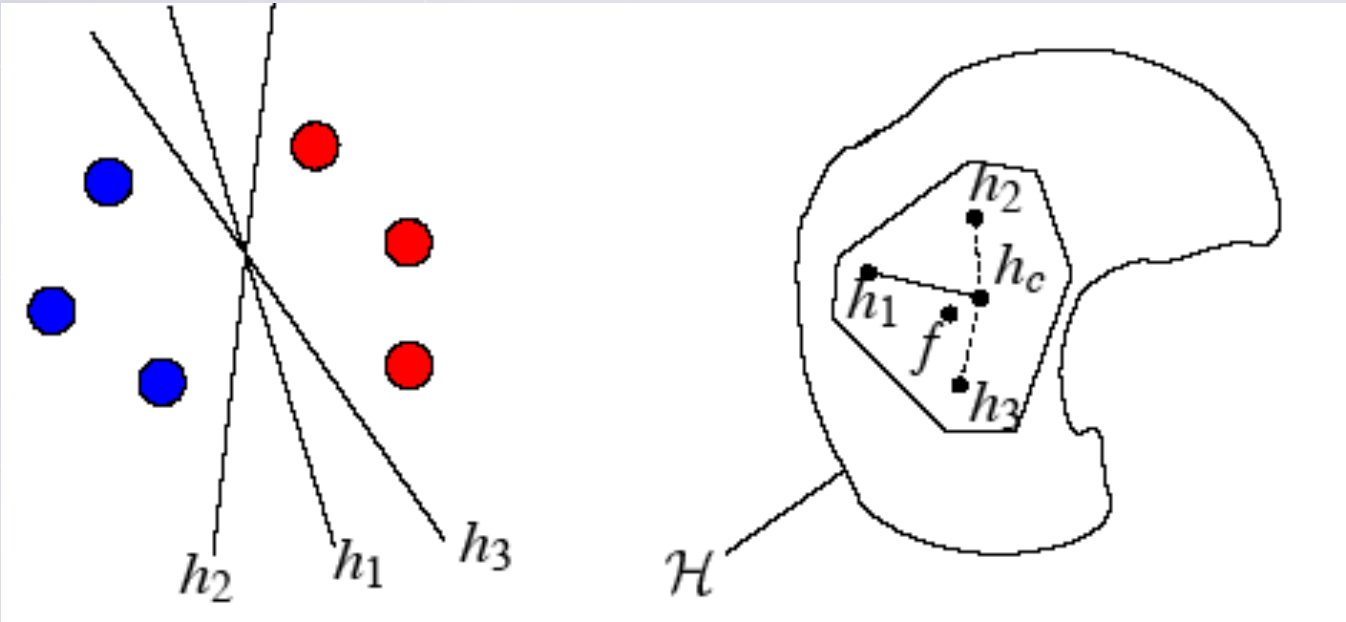
cnrs INSA Lyon 1 UNIVERSITÉ LUMIÈRE LYON 2 ÉCOLE CENTRALE LYON

Apprentissage supervisé

- Produire automatiquement des règles à partir d'une base de données d'apprentissage étiquetées
- But: prédire la classe de nouvelles données observées
- Algorithmes (1ère partie): Arbres de décision, réseaux bayésiens, réseaux de neurones, k-plus proches voisins, etc…

Limites des méthodes induisant une seule hypothèse (1/2)

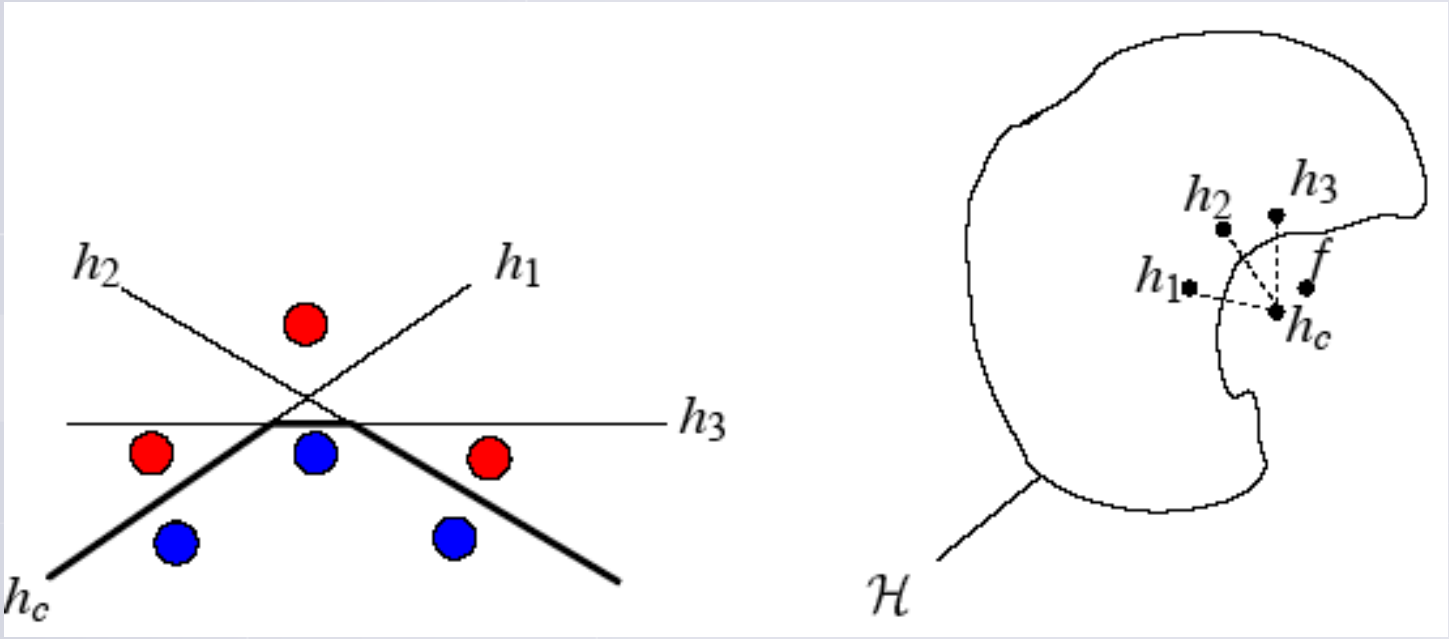
Limite statistique (variance) : si l'espace de recherche est grand proportionnellement au nombre d'exemples, plusieurs hypothèses de même performance peuvent être induites. **L'algorithme est contraint d'en choisir une (surement pas la meilleure.**



Solution : un simple vote peut réduire ce risque

Limites des méthodes induisant une seule hypothèse (2/2)

Limite de représentation (biais) : lorsque la famille d'hypothèses H ne contient pas de bonne approximation de f.



Solution : une combinaison peut permettre d'augmenter l'espace des fonctions possibles

Méthodes ensemblistes

Définition

Une méthode ensembliste combine les décisions individuelles de plusieurs hypothèses h_1 , . . . , h_T pour classer de nouveaux exemples.

Principe essentiel : l'union fait la force

Réunir un «comité d'experts» chacun peut se tromper, mais en combinant les avis, on a plus de chance d'avoir la bonne prédiction !!!

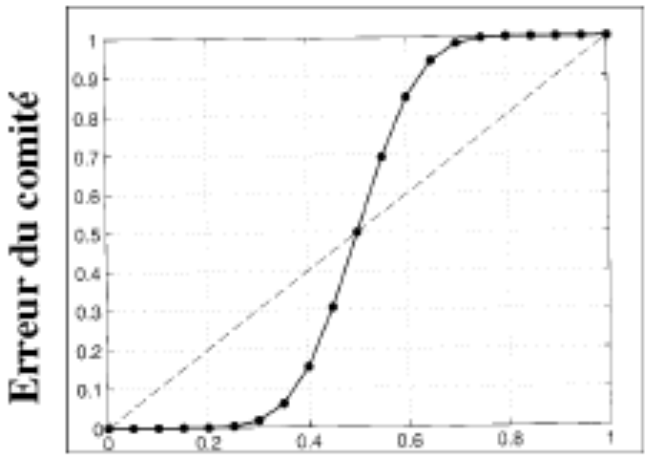
Conditions pour qu'une méthode ensembliste soit efficace :

- **Les hypothèses construites ont un taux de succès meilleur que l'aléatoire (erreur<0.5 dans le cas de deux classes équilibrées).**
- **Les hypothèses présentent une certaine diversité.**

Méthodes ensemblistes

Justification théorique

- Supposons T classifieurs ***indépendants*** faisant chacun une erreur $E_{gen} = \varepsilon$.
- Si on décide «à la majorité», alors on se trompe si et seulement si plus de la moitié du «comité» se trompe



$$Erreur_{ensemble} = \sum_{k=N/2}^{N} C_k^N \varepsilon^k (1-\varepsilon)^{N-k}$$

Décision fortement améliorée,
(sous réserve que ε<0.5 !!)…
…et ce d'autant plus qu'on augmente
le nombre N d'experts

Comment produire les classifieurs à combiner

Méthodes ensemblistes hétérogènes :

- **combinent un ensemble d'hypothèses h_1 , ... , h_T produites par:**
  - Des algorithmes différents L_1 , ... , L_T sur une même distribution **D** des exemples d'apprentissage **A**.
  - Même algorithmes mais avec des paramètres et/ou initialisations différent(e)s.

Comment produire les classifieurs à combiner

Méthodes ensemblistes homogènes :

- combinent un ensemble d'hypothèses h_1 , . . . , h_T produites par un même algorithme d'apprentissage L.

- Idée : faire varier l'ensemble d'apprentissage A. La diversité des hypothèses s'opère en modifiant la distribution de probabilité D_t des exemples utilisés pour construire h_t .

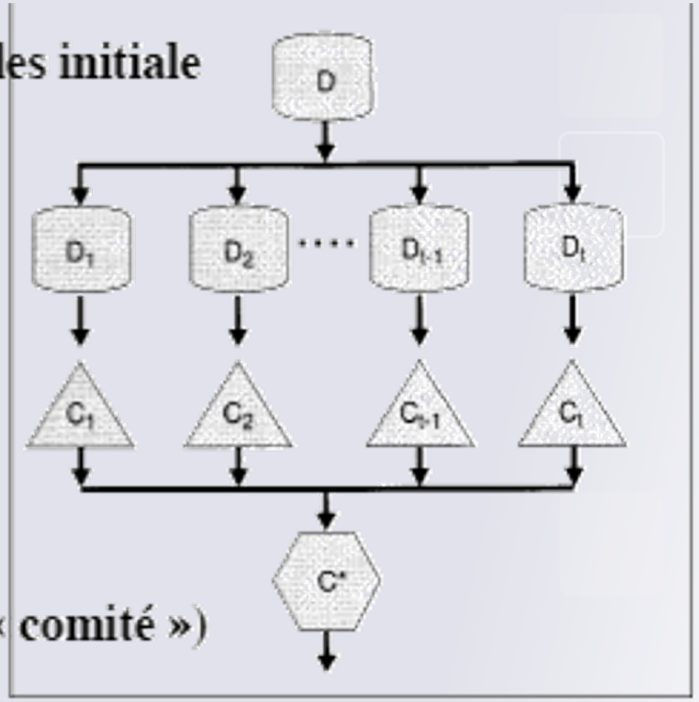
Méthodes ensemblistes homogènes

Base d'exemples initiale

Diverses variantes de la base
(tirages aléatoires, pondérations différentes, …)

Classifieurs élémentaires obtenus par la même procédure,
mais chacun sur une variante différente de la base

Classifieur global (« comité »)



→ Méthodes très générales, applicables
avec n'importe quel algorithme « élémentaire »

Méthodes ensemblistes homogènes

La diversité provient de la distribution des exemples d'apprentissage

- Utilisent des stratégies adaptatives (boosting) ou aléatoires (bagging)
- **Boosting** : utiliser une stratégie adaptative pour booster des performances, applicable à tout type d'algorithme (Réseau de neurones,CART, etc.)
- **Bagging ou Random Forests :** utiliser l'aléatoire pour améliorer les performances d'algorithmes de « faibles » performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.

Boosting : Prédiction de courses hippiques

Comment gagner aux courses ?

- **On interroge des parieurs professionnels**
- **Supposons:**
  - Que les professionnels ne puissent pas fournir **une règle de pari simple et performante**
  - Mais que face à **des cas de courses**, ils puissent toujours produire des règles un peu meilleures que le hasard

LIRIS

Comment gagner aux courses ?

Exemples de règles individuellement peu performantes

- *h_1: "Parier sur le cheval qui a gagné le plus de courses récemment".*
- *h_2: "Parier sur le cheval pour lequel il y a le plus grand nombre de mises".*
- *h_3: "Parier sur le cheval qui préfère les terrains lourds".*
- **Pouvons-nous devenir riche?**

LIRIS

Idée

- Demander à l'expert une heuristique
- Recueillir un ensemble de cas pour lesquels cette heuristique échoue (cas difficiles)
- Ré-interroger l'expert pour qu'il fournisse une heuristique pour les cas difficiles
- Et ainsi de suite…

- Combiner toutes ces heuristiques
- Un expert peut aussi bien être un algorithme d'apprentissage peu performant (*weak learner*)

LIRIS

Questions

- **Comment choisir les courses à chaque étape?**
  - Se concentrer sur les courses les plus "difficiles" (celles sur lesquelles les heuristiques précédentes sont les moins performantes)
- **Comment combiner les heuristiques (règles de prédiction) en une seule règle de prédiction ?**
  - Prendre un vote (pondéré) majoritaire de ces règles

LIRIS

Le boosting (dopage)

- **boosting = méthode générale pour convertir des règles de prédiction peu performantes en une règle de prédiction (très) performante**
- **Plus précisément :**
  - Étant donné un algorithme d'apprentissage **L** "faible" qui peut toujours retourner une hypothèse **h** de **taux d'erreurs ≤ 0.5-γ** (où γ est l'amélioration de **h** par rapport à une décision aléatoire)
  - Un algorithme de boosting peut construire (de manière prouvée) une règle de décision (hypothèse) de taux d'erreur ≤ e

Adaboost [Freund & Schapire 1996]

- Adaboost : Adaptive Boosting
- Algorithme itératif pour l'ajout des classifieurs
- Variantes de la base d'apprentissage obtenues par des pondérations successives des mêmes exemples (calculées pour «se focaliser» sur les exemples «difficiles»)
- On augmente l'importance des exemples mal-classés lors de l'itération précédente

Adaboost : vue formelle

- **Entrée : Base d'exemples : $A=\{(x_1,y_1), \ldots ,(x_m,y_m)\}$, avec $y_i \in \{+1,-1\}$, i=1,…,m, Algorithme de classification "faible » : L, Nombres d'itérations : T**
- **Initialisation : Poids initiaux : $D_1(x_i)= 1/m$ pour tout i=1,…,m**
- **Pour chaque étape t de 1 à T, faire**
  1. Apprendre par l'algorithme **L** une hypothèse de classification h_t sur **A** suivant la distribution D_t.
  2. Calculer l'erreur pondérée ε_t de h_t sur **A** en considérant la distribution D_t. (somme des poids des ex. mal classés)

$$\varepsilon_t = \sum_i D_t(x_i) \times \left| h_t(x_i) - y_i \right|$$

  1. Si $\varepsilon_t \geq 0.5$ alors T= t−1; **exit**
  2. En déduire la «**capacité prédictive**» de h_t : $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) > 0$ $\alpha_t>0$ si $\varepsilon_t<0.5$; $\alpha_t \to +\infty$ si $\varepsilon_t \to 0$
  3. Mettre à jour les **poids**, i.e. pour i de 1 à m

$$D_{t+1}(x_i) = \frac{D_t(x_i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

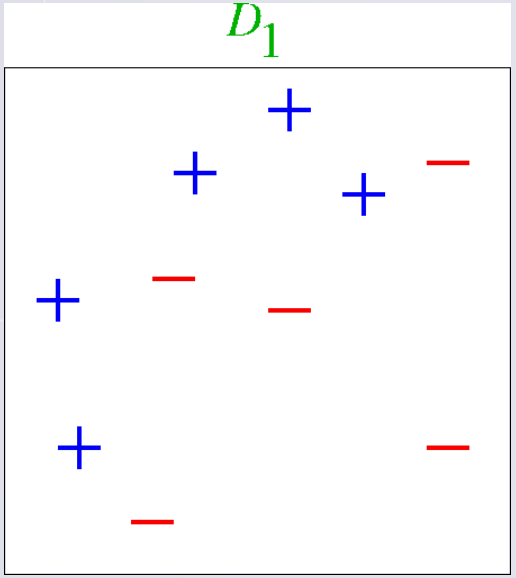
x_i bien classé ; x_i mal classé

Z_t: facteur de normalisation

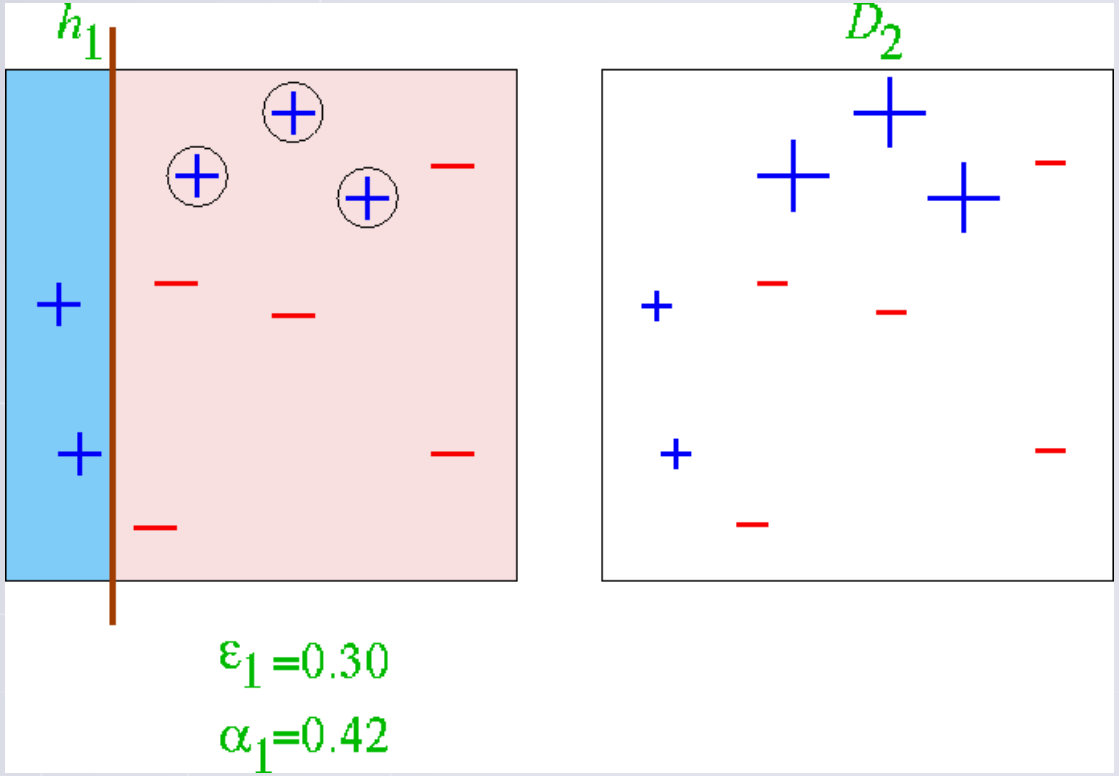
- **Fournir l'hypothèse finale**

$$H_{\text{final}}(x) = \text{sgn}\left(\sum_t \alpha_t h_t(x) \right)$$

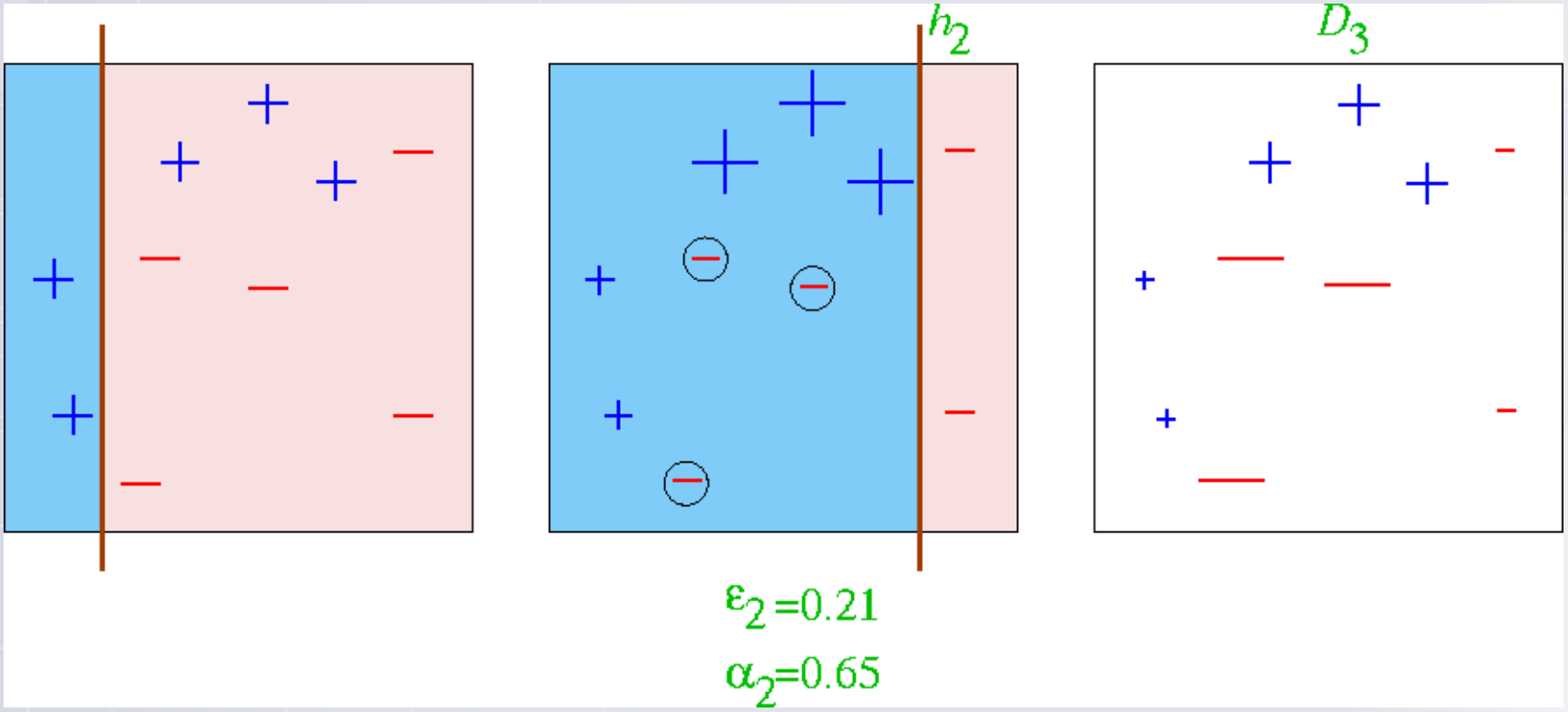
Exemple Illustratif



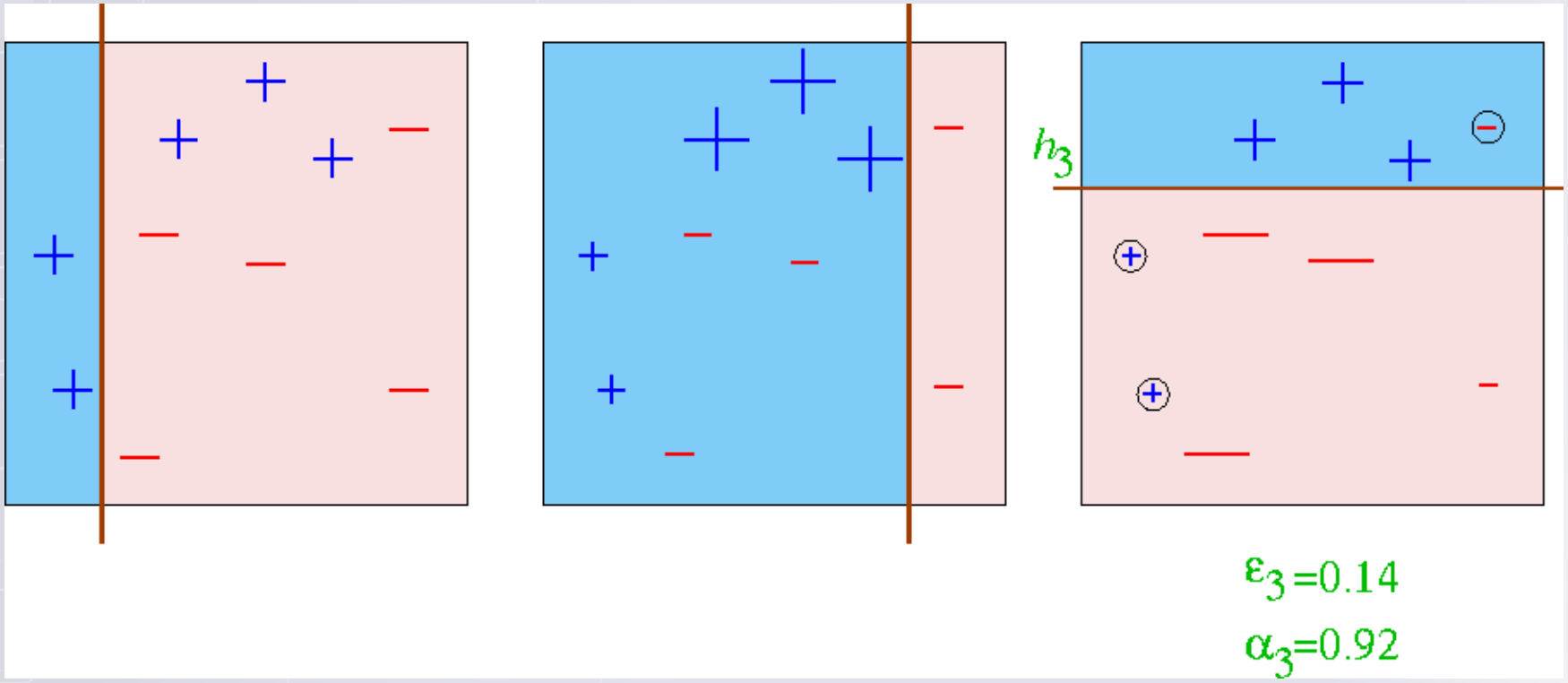
Etape 1



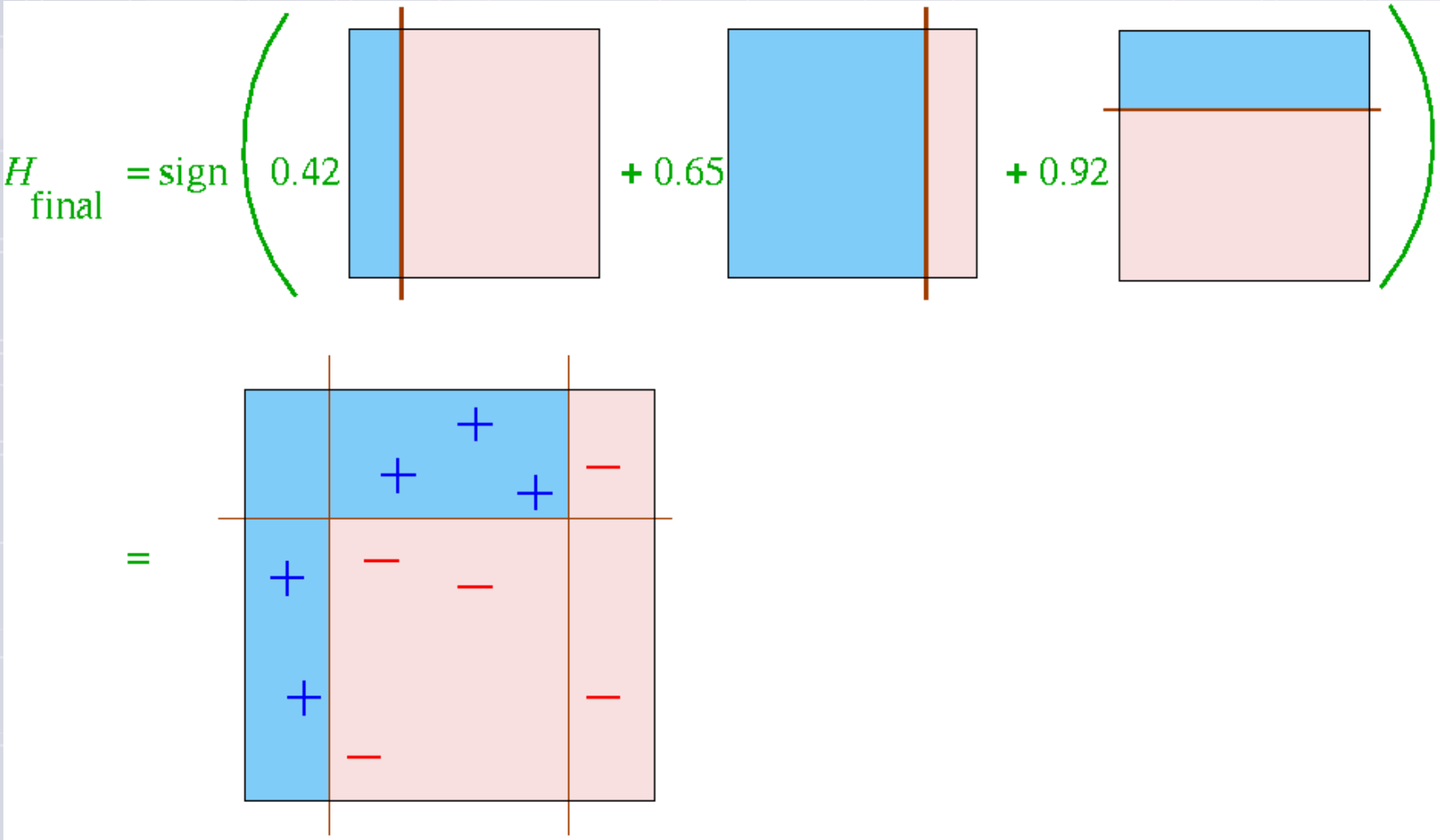
Etape 2



Etape 3



Hypothèse finale



Adaboost

- **Comment apprendre une hypothèse h_t en considérant les pondérations D_t sur les exemples d'apprentissage ?**
  - Utilisé un algorithme de classification L qui intégré les poids des individus dans la phase d'apprentissage (arbres de décision, réseaux de neurones, …)
  - A chaque itération t un nouvel échantillon d'apprentissage est considéré en sélectionnant au hasard avec remise ***m*** exemples suivants les poids.

Boosting: Bilan

- La puissance du boosting vient du rééchantillonnage adaptatif
- Réduit la variance
- Réduit le biais en obligeant l'algorithme à focaliser sur les cas difficiles → hypothèse combinée beaucoup plus flexible
- Convergence rapide
- Sensible au bruit : les apprenants de base classent mal les exemples bruités ⇒ poids augmentent ⇒ surajustement au exemples bruités.

Rappel : Comment produire les classifieurs à combiner

Méthodes ensemblistes hétérogènes :

- **combinent un ensemble d'hypothèses h_1 , . . . , h_T produites par des algorithmes d'apprentissage différents**

Méthodes ensemblistes homogènes :

- **combinent un ensemble d'hypothèses h_1 , . . . , h_T produites par un même algorithme d'apprentissage L.**
  - Utilisent des stratégies adaptatives (**boosting**) ou aléatoires (**bagging**)
    - **Boosting :** utiliser une stratégie adaptative pour booster des performances, applicable à tout type d'algorithme (Réseau de neurones, CART, etc.)
    - **Bagging ou Random Forests :** utiliser l'aléatoire pour améliorer les performances d'algorithmes de « faibles » performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.

Bagging : Bootstrap aggregation [Breiman,96]

- Variantes de la base d'apprentissage obtenues par tirages aléatoires avec remise depuis la base initiale (sorte de «bootstrap»)
- Un ensemble de données « bootstrap » est un ensemble de données obtenu en sélectionnant au hasard avec remise *m* observations parmi les *m* observations de l'ensemble d'entraînement.
- Certaines observations sont dupliquées tandis que d'autres sont absentes; ce qui introduit une part d'aléatoire.
- L'intérêt de telle méthode est de répéter la procédure et d'utiliser chaque ensemble de données pour construire un modèle. Nous disposons alors de différentes réalisations de la statistique estimée (ou du modèle).

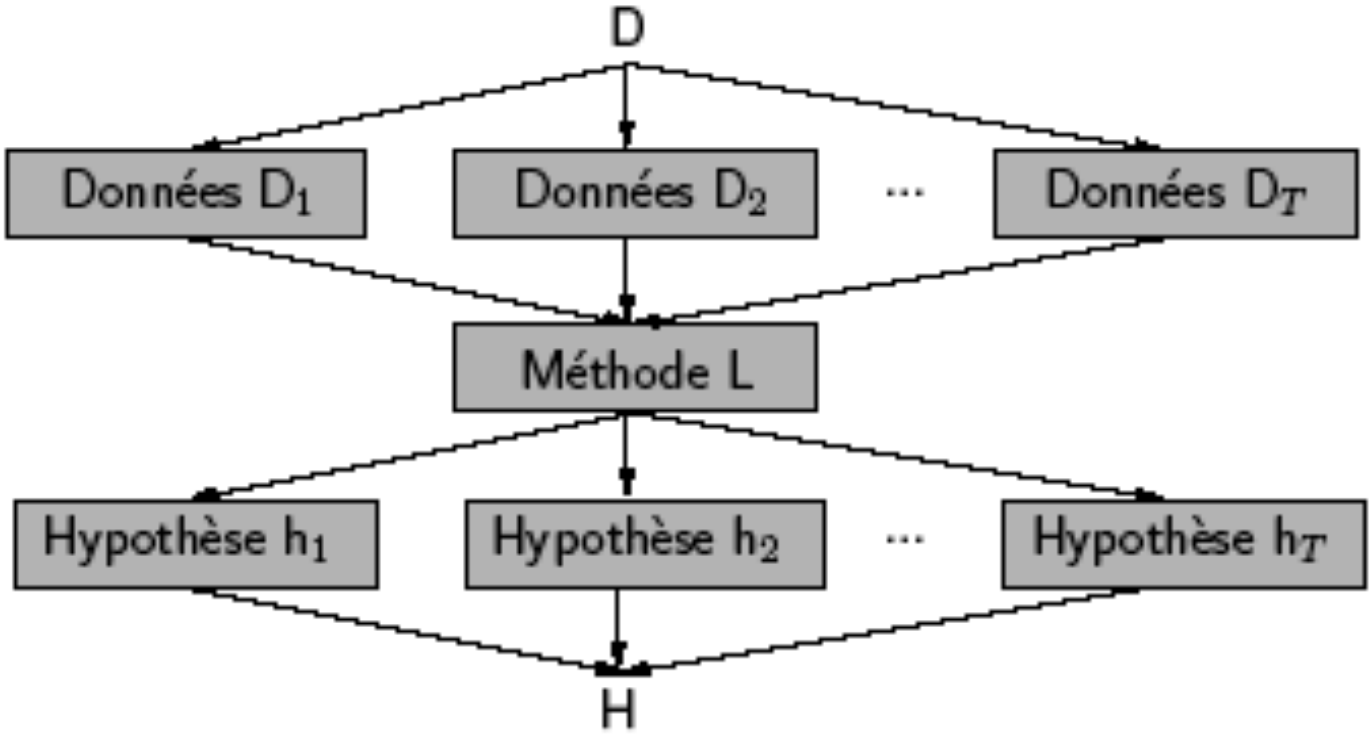
Principe du Bagging

- Génération de *k* échantillons « bootstrap » par tirage avec remise dans l'ensemble d'apprentissage A= $\{(x_1,y_1), \ldots ,(x_m,y_m)\}$.
- Pour chaque échantillon, apprentissage d'un classifieur en utilisant le même algorithme d'apprentissage
- La prédiction finale pour un nouvel exemple est obtenue par vote (simple) des classifieurs

LIRIS

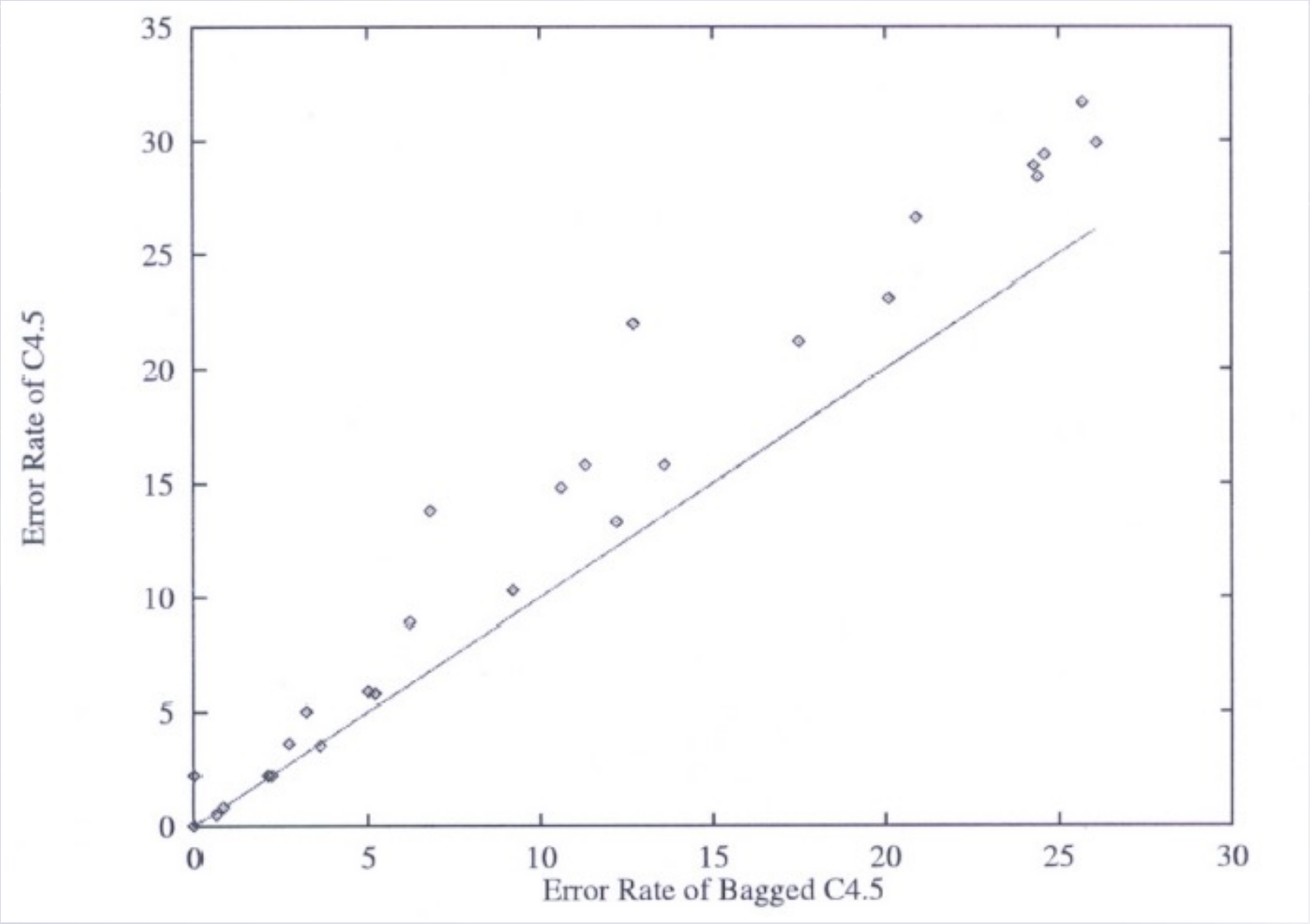
Schéma du Bagging

Diversification : Créer des réplicats bootstrap de D



Intégration : Hypothèse finale $H=f(h_1, h_2, \ldots, h_T)$

C4.5 sans et avec bagging



Bagging : Bilan

- Le but premier du bagging est d'atténuer l'instabilité inhérente à certaines méthodes de discrimination.
- Une méthode de discrimination est dite instable si un changement mineur dans les données provoque un changement assez important de modèle. (Ex: arbres, réseaux de neurones)
- Utile et efficace en particulier si l'algorithme de base utilisé est «instable» car différences entre variantes de base ➔ classifieurs élémentaires très différents
- Evite l'overfitting (sur-apprentissage), car on «moyenne» des classifieurs construits avec différentes réalisations aléatoires des mêmes données.
- Il est souvent dit que le bagging fonctionne en réduisant la variance en laissant le biais inchangé.

Les forêts aléatoires [Breiman,2001]

Principe :

- Est ce qu'à partir d'un ensemble de faible classifieurs (arbres de décision) ont peut créer un classifieur plus performant?
- L'union fait-elle la force?

La réponse est oui à condition que :

- les arbres sont complémentaires.

Le résultat sera donc autant plus intéressant que les arbres sont indépendants et le plus efficaces possible.

Les forêts aléatoires [Breiman,2001]

- On va construire un grand nombre d'arbres de décision différents pour un même problème: une forêt (forest).

Pour construire une forêt on injecte de l'aléatoire:

- on rajoute de l'aléatoire avant ou pendant la construction d'un arbre (Algorithme CART); on construit plusieurs arbres "randomisés"
- on agrège l'ensemble des arbres obtenus: Pour classer un individu on prend la décision finale par un vote majoritaire

Les forêts aléatoires [Breiman,2001]

- **Pour rajouter l'aléa, RF combine la sélection aléatoire d'instances avec la sélection aléatoire de variables.**
  - on lance la construction de l'arbre sur un sous échantillon tiré aléatoirement : Breiman propose d'utiliser le **bagging**,
  - on tire à chaque nœud de l'arbre **mtry** variables uniformément et on cherche la "meilleure" coupure uniquement parmi ces variables-ci.
  - Les arbres de décision sont complets : construits automatiquement sans pre- ou post-élagage.

Les forêts aléatoires : Algorithme

- **Soit A= {(x_1,y_1), … ,(x_m,y_m)} comportant m exemples décrit par p variables. y_i l'étiquette (classe) de x_i**
- **Hyperparamètres : mtry << p = nb de variables à choisir à chaque étape; T = nombre d'arbres à construire**
- **Pour construire chaque arbre**
  1. Créer un réplicat bootstrap **A_i** de **A**
  2. Créer un arbre CART avec les modifications suivantes :
     - A chaque nœud de l'arbre, choisir aléatoirement **mtry** variables à partir desquelles la variable de discrimination sera choisie
     - Ne pas élaguer l'arbre
- **Méthode d'intégration : vote uniforme des T arbres**

Les forêts aléatoires : Points forts

- **À partir d'une base d'apprentissage A de m exemples, on tire des sous-bases bootstrap de m exemples avec remise.**
- **Normalement pour chaque échantillon bootstrap 63,2% des exemples sont uniques de A, le reste étant des doublons.**
- **Donc pour chaque sous base 1/3 des exemples de A ne sont pas sélectionnés et sont considérés comme oob (out of bag). Ils serviront à :**
  - l'évaluation **interne du forêt** (estimation de l'erreur de classification en généralisation du forêt).
  - estimer **l'importance des variables** pour la sélection de variables.

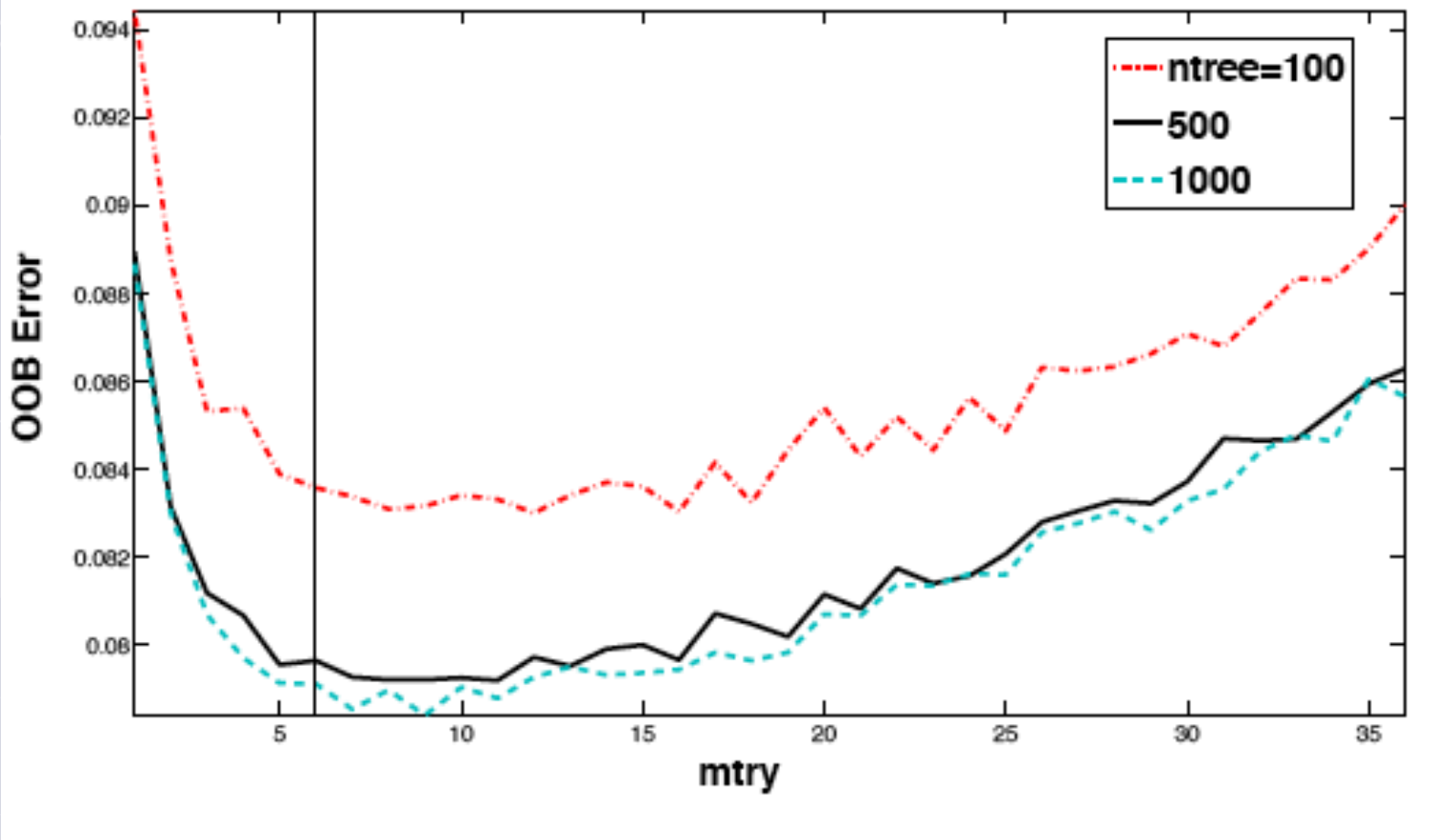
Estimation de l'erreur en généralisation

- Pour chaque élément x_i et pour chaque arbre H_k (parmi les T arbres construits) pour lequel x_i a été sélectionné comme oob, prévoir la classe de x_i selon H_k.
- Déterminer la classe majoritaire y_m de x_i pour les cas trouvés.
- Pour tous les points x_i de $A = \{x_1, \ldots, x_m\}$ $_i$ calculer la moyenne de fois que y_m est différente de sa vrai classe y_i : C'est l'erreur de classification sur l'oob.
- Cette mesure fournit une estimation non biaisée du taux d'erreur en généralisation ε_F de la forêt.

Pour optimiser RF

- **L'erreur ε_F de la forêt est fonction de deux facteurs :**
  - la corrélation entre les arbres de la forêt (**co**) : si **co** ↗ alors ε_F ↗
  - la force f_i des arbres individuels (force = performance = justesse des prédictions : si f_i ↗ alors ε_F ↘
- **Impact du nombre de variables mtry**
  - Si **mtry** ↘ alors **co** ↘ mais f_i ↘ aussi : compromis **co**/f_i à trouver
    - Tester plusieurs valeurs sur une forêt de 20-30 arbres et prendre la valeur de **mtry** donnant la plus petite valeur de ε_f.
    - D'autres travaux proposent d'essayer avec $\sqrt{p}$
- **Impact du nombre d'arbres T**
  - On peut augmenter **T** sans risque de surajustement [Breiman, 2001]. Le taux d'erreur en généralisation ε_F converge vers une valeur limite.

RF: Nombre optimal de variables



m = 6435, p= 36 , nombre de classes = 6

RF pour le clustering

- **Principe:** deux individus proches devraient suivre un cheminement identique dans un arbre de décision.
- La proximité entre deux individus est calculée par la fraction d'arbres générés dans lesquels ces derniers aboutissent dans une même feuille.
- Cette mesure peut être utile lors d'une recherche de structure au sein du jeu de données, et ouvre la voie à l'utilisation des RF en clustering.

RF pour la sélection de variables

- **Principe:** une variable **v** est importante si la modification de sa valeur pour un individu entraine sa mauvaise classification.
- A partir des arbres construits, on peut en déduire une hiérarchisation des variables et d'où une possibilité de sélection de variables
- La méthode consiste à calculer l'augmentation du taux d'erreur oob lorsque les modalités de la variable étudiée sont permutées aléatoirement sur les données OOB, les autres variables restant inchangées.
- Pour chaque arbre et pour tous ses individus oob calculer le nombre de vrai positifs **VP1** (individus bien classés). Permuter ensuite aléatoirement les valeurs de la variable **v** entre tous ces éléments et calculer le nombre de **VP2**.
- La moyenne des écarts **VP1-VP2** sur tous les arbres est l'importance de la variable **v**.

RF pour la gestion des valeurs manquantes

- Faire un premier remplacement de toutes les valeurs manquantes d'une manière inexacte.
- Créer une première forêt aléatoire.
- Calculer les proximités entre les exemples.
- Pour une variable numérique v, l'estimation d'une valeur manquante $x_{i,v}$ pour un individiu x_i est donnée par la moyenne des valeurs de la variable v pour le reste des exemples pour lesquels v est non manquante pondérées par leurs proximités avec x_i.
- Pour une variable qualitative on utilise la fréquence
- Refaire le processus 4 à 6 fois en utilisant les nouvelles valeurs trouvées.

Forêts aléatoires : Bilan

- **Encore une fois très simple à mettre en œuvre**
- **La sélection d'un sous-ensemble de variables explicatives (input) parmi un grand nombre, permet généralement:**
  - réduire de beaucoup les temps de calcul : très fructueuse en grande dimension
  - d'obtenir une plus grande variété de modèles.
- **L'aggrégation des valeurs ou classes prédites (vote majoritaire) par tous les modèles générés devrait alors donner un classifieur plus robuste et plus précis.**