Apprentissage par combinaison de décisions

Laurent Miclet

IRISA (Lannion)
France

EMINES 2024

1. Le dopage (*boosting*)

2. Le dopage par gradient (*gradient boosting*)

3. Le *bagging*
   - Bootstrap
   - Les forêts aléatoires (*random forests*)
   - Autres méthodes. L'apprentissage en cascade (*cascading*)

Plan

Principe : composer un comité d'experts et poser les bonnes questions

Soit plusieurs " experts " des courses de chevaux. Ils ne donnent pas toute leur expertise, mais seulement des règles *faibles* :

- " Dans les courses de trot, il faut parier sur le cheval ayant gagné le plus grand nombre de courses "
- " L'hiver, il faut parier sur le cheval ayant la plus grande cote "
- etc.

Une règle faible est peu performante, mais meilleure que le hasard.
Si on interroge chaque expert sur des ensembles de courses différents, on obtiendra des règles différentes.

Problèmes

- Quels ensembles de courses présenter à chaque expert en vue d'extraire les règles les plus intéressantes ?
- Comment combiner les avis des experts pour atteindre la meilleure décision ?

Le *boosting* élémentaire

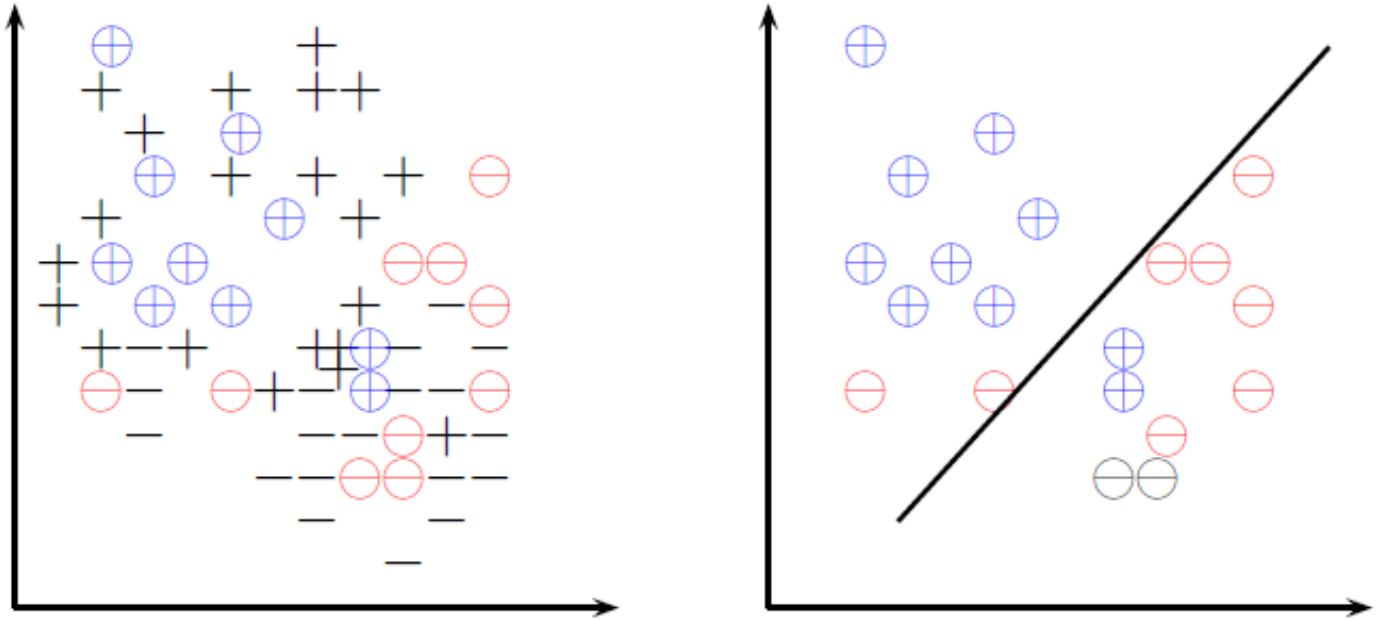
Soit un échantillon d'entraînement $\mathcal{S}$ de taille m.

1. On obtient d'abord une première hypothèse h_1 sur un sous-échantillon $\mathcal{S}_1$ de taille $m_1 < m$.
2. On apprend alors une deuxième hypothèse h_2 sur un échantillon $\mathcal{S}_2$ de taille m_2 choisi dans $\mathcal{S} - \mathcal{S}_1$.
   Choisi comment ? Comme le plus informatif pour enrichir h_1 : La moitié des exemples de $\mathcal{S}_2$ sont mal classés par h_1.
3. On apprend finalement une troisième hypothèse h_3 sur m_3 exemples tirés dans $\mathcal{S} - \mathcal{S}_1 - \mathcal{S}_2$ pour lesquels h_1 et h_2 sont en désaccord.
4. L'hypothèse finale est obtenue par un vote majoritaire des trois hypothèses apprises :

$$H = \text{vote majoritaire}(h_1, h_2, h_3)$$

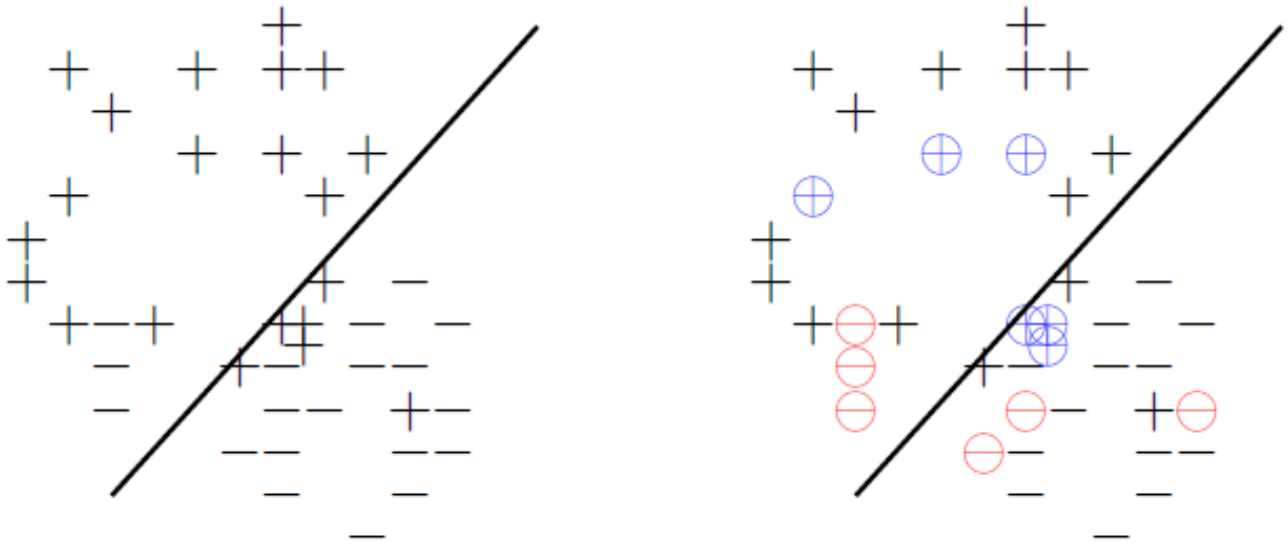
On démontre que H a une performance supérieure à celle de l'hypothèse apprise directement sur $\mathcal{S}$.

Début



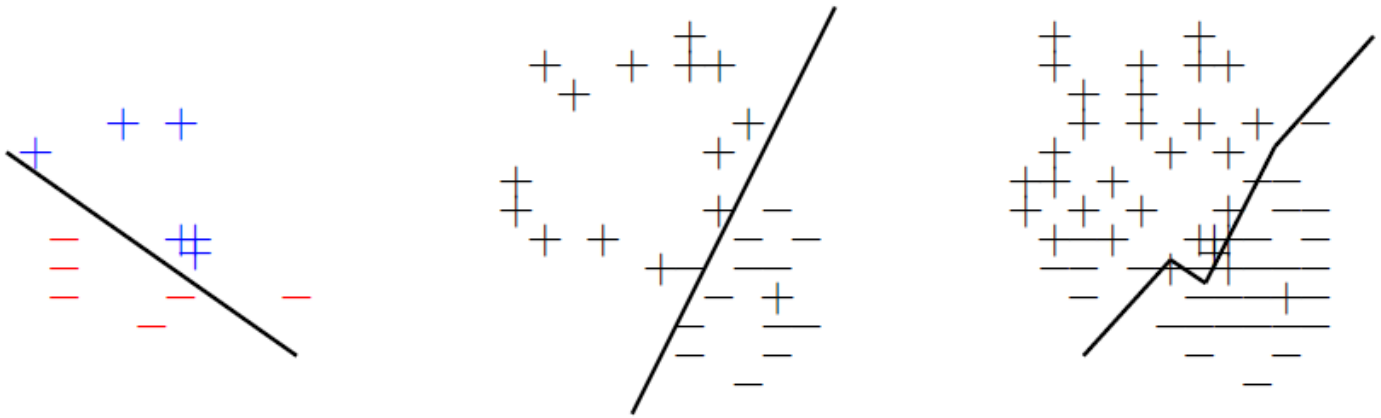
A gauche : *l'ensemble d'apprentissage $\mathcal{S}$ et le sous-ensemble $\mathcal{S}_1$ (points rouges ou bleus entourés).*
A droite : *l'ensemble $\mathcal{S}_1$ et la droite $\mathcal{C}_1$ apprise sur cet ensemble.*

Suite



A gauche : *l'ensemble* $\mathcal{S} - \mathcal{S}_1$ *et la droite* $\mathcal{C}_1$ *apprise sur* $\mathcal{S}_1$.
A droite : *un ensemble* $\mathcal{S}_2$ *inclus dans* $\mathcal{S} - \mathcal{S}_1$ *parmi les plus informatifs pour* $\mathcal{C}_1$ *(points rouges ou bleus et entourés).*

... et fin.



A gauche : $\mathcal{S}_2$ *et la droite* $\mathcal{C}_2$ *apprise sur* $\mathcal{S}_2$.
Au centre : $\mathcal{S}_3 = \mathcal{S} - \mathcal{S}_1 - \mathcal{S}_2$ *et la droite* $\mathcal{C}_3$ *apprise sur* $\mathcal{S}_3$
A droite : $\mathcal{S}$ *et la combinaison des 3 droites.*

Généralisation : le boosting probabiliste

Trois idées pour le *boosting probabiliste* :

1. L'utilisation d'un *comité d'experts* spécialisés que l'on fait voter pour atteindre une décision.
2. La *pondération adaptative* des votes par une technique de mise à jour multiplicative.
3. La *modification de la distribution* des exemples disponibles pour entraîner chaque expert, en surpondérant au fur et à mesure les exemples mal classés aux étapes précédentes.

La technique de base

Paramètre Un apprenant faible h

Entrée Un échantillon $\mathcal{S}$

Initialisation Tous les exemples ont le mê me poids

Pour $t = 1, T$ **faire**

- utiliser h avec $\mathcal{S}$ et la distribution courante
- renforcer le poids des exemples mal classés à l'étape courante

fait

Sortie H : un vote majoritaire pondéré des hypothèses

Le boosting probabiliste : Adaboost

Dopage *adaptatif*

L'algorithme standard s'appelle AdaBoost (*adaptive boosting*).

- L'idée principale est de définir à chacune de ses étapes $1 \leq t \leq T$, une nouvelle distribution de probabilité *a priori* sur les exemples de $\mathcal{S}$ en fonction des résultats de l'algorithme à l'étape précédente : échantillon pondéré.
- Le poids à l'étape t d'un exemple $(\mathbf{x}_i, \mathbf{u}_i)$ d'indice i est noté $D_t(i)$. Initialement, tous les exemples ont un poids identique, puis à chaque étape, les poids des exemples mal classés sont augmentés.
- Ainsi, l'algorithme doit se concentrer sur les exemples difficiles.

Le boosting probabiliste : Adaboost

Boucle de calcul

À chaque étape t, l'apprenant cherche une hypothèse $h_t : \mathcal{X} \to \{-1, +1\}$ bonne pour la distribution D_t sur $\mathcal{X}$. La performance de l'apprenant est mesurée par l'erreur :

$$\varepsilon_t = p_{D_t}[h_t(\mathbf{x}_i) \neq u_i] = \sum_{i : h_t(\mathbf{x}_i) \neq u_i} D_t(i)$$

L'erreur est mesurée en fonction de la distribution D_t sur laquelle l'apprenant est entraîné.

Comment pondérer un échantillon ?

- Soit les poids des exemples sont effectivement modifiés ;
- Soit c'est la probabilité de tirage des exemples qui est modifiée et l'on utilise un tirage avec remise.

Le boosting probabiliste : Adaboost

Comment pondérer une hypothèse ?

- On prend le cas où les classes '+' et '-' sont équiprobables : l'erreur d'une décision aléatoire est 1/2.
- Chaque hypothèse h_t apprise est affectée d'un poids α_t mesurant l'importance qui lui sera donnée dans la combinaison finale.
- Ce poids est positif si $\varepsilon_t \leq 1/2$. Plus l'erreur associée à l'hypothèse h_t est faible, plus celle-ci est dotée d'un coefficient α_t important.

Comment ne pas aller trop loin ?

Le poids des exemples difficiles à apprendre devient dominant. Si on peut trouver une hypothèse performante sur ces exemples (avec $\varepsilon_t \approx 0$), elle sera dotée d'un coefficient α_t considérable.
Les exemples bruités, sur lesquels finit par se concentrer l'algorithme, perturbent le boosting.

Adaboost

Entrée. $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \ldots, (\mathbf{x}_m, \mathbf{u}_m)\}$, avec $\mathbf{u}_i \in \{+1, -1\}, i = 1, m$

Initialisation **Pout tout** $i = 1, m$ **faire** $p_0(\mathbf{x}_i) \leftarrow 1/m$ **fait** ; $t \leftarrow 0$

Boucle

tant que $t \leq T$ **faire**

Tirer $\mathcal{S}_t$ dans $\mathcal{S}$ selon les probabilités p_t

Apprendre une règle de classification h_t sur $\mathcal{S}_t$ par l'algorithme $\mathcal{A}$

Soit ε_t l'erreur apparente de h_t sur $\mathcal{S}$.

Calculer $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

pour $i = 1, m$ **faire**

$p_{t+1}(\mathbf{x}_i) \leftarrow \frac{p_t(\mathbf{x}_i)}{Z_t} e^{-\alpha_t}$ si $h_t(\mathbf{x}_i) = u_i$ (bien classé par h_t)

$p_{t+1}(\mathbf{x}_i) \leftarrow \frac{p_t(\mathbf{x}_i)}{Z_t} e^{+\alpha_t}$ si $h_t(\mathbf{x}_i) \neq u_i$ (mal classé par h_t).

Z_t est telle que $\sum_{i=1}^{m} p_t(\mathbf{x}_i) = 1$

fin pour

$t \leftarrow t + 1$

fin tant que

Sortie. T hypothèses h_t de poids α_t. $H(\mathbf{x}) = \text{signe}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

Le fonctionnement de Adaboost

Hypothèse finale et calcul des α_t

À la fin de cet algorithme, chaque règle de classification h_t est pondérée par une valeur α_t calculée en cours de route. La règle finale est :

$$H(\mathbf{x}) \;=\; \text{signe}\left(\sum_{t=1}^{t=T} \alpha_t\, h_t(\mathbf{x})\right)$$

Le poids α_t est calculé pour minimiser la fonction de perte :

$$G(\alpha) = \sum_{i=1}^{m} \exp\left[- u_i \left(\alpha h_t(\mathbf{x}_i) + f_{t-1}(\mathbf{x}_i) \right) \right]$$

$f_{t-1}(\mathbf{x}_i)$ est la combinaison des hypothèses aux itérations précédentes :

$$f_{t-1}(\mathbf{x}_i) = \sum_{r=1}^{t-1} \alpha_r h_r(\mathbf{x}_i)$$

L'erreur empirique de AdaBoost

Écrivons l'erreur ε_t de h_t comme : $\frac{1}{2} - \gamma_t$, où γ_t mesure l'amélioration apportée par l'hypothèse h_t par rapport à l'erreur de base 1/2.
On montre que l'erreur empirique (sur l'échantillon d'entraînement $\mathcal{S}$) de l'hypothèse finale
H est bornée par :

$$\prod_{t=1}^{T} \left[2\sqrt{\varepsilon_t(1-\varepsilon_t)} \right] = \prod_t \sqrt{1-4{\gamma_t}^2} \leq exp\left(-2\sum_t {\gamma_t}^2 \right)$$

Ainsi, si chaque hypothèse faible est légèrement meilleure que le hasard, ($\gamma_t \geq \gamma > 0$), alors l'erreur empirique diminue exponentiellement avec t.

L'erreur en généralisation de AdaBoost

L'erreur en généralisation de H peut être bornée par une formule avec l'erreur empirique $R_{Emp}(H)$, le nombre m d'exemples d'apprentissage, la dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$ de l'espace des hypothèses et le nombre d'étapes T de boosting.

$$R_{Réel}(H) = R_{Emp}(H) + \mathcal{O}\left(\sqrt{\frac{T \cdot d_{\mathcal{H}})}{m}}\right)$$

Le boosting devrait tendre à surapprendre lorsque T devient grand, puisque le deuxième terme augmente.

En fait, cela ne se produit pas : le risque réel tend à diminuer même longtemps après que le risque empirique soit devenu nul.
L'explication à ce paradoxe vient du lien entre le boosting et les méthodes *séparatrices à vaste marge* (SVM).

AdaBoost et les marges

Soit un exemple $(\mathbf{x}, y)$, avec $y = +1$ ou -1. Sa *marge* se définit par :

$$\text{marge}(\mathbf{x}, y) \;=\; \frac{y \sum_{t=1}^{T} \alpha_t \, h_t(\mathbf{x})}{\sum_{t=1}^{T} \alpha_t}$$

Ce nombre est dans $[-1, +1]$; il est positif si H classifie bien l'exemple.
La marge est une mesure de fiabilité de la prédiction.
Il a été prouvé que l'erreur en généralisation peut être bornée, avec une forte probabilité, pour m assez grand et pour tout $\theta > 0$ par :

$$R_{Réel}(H) \;\leq\; \hat{Pr}[\text{marge}(\mathbf{x}, y) \leq \theta] + \mathcal{O}\left(\sqrt{\frac{d_{\mathcal{H}}}{m\theta^2}}\right)$$

Cette borne est indépendante de T, le nombre d'étapes de boosting.
Le boosting cherche effectivement à augmenter la marge : il se concentre sur les exemples difficiles à classer, ceux dont la marge est la plus faible.
AdaBoost et les méthodes SVM ont en commun d'effectuer une recherche de classificateurs *à large marge* dans des espaces de grandes dimensions.
Ces classificateurs sont dans les deux cas des combinaisons linéaires.

Plan

On apprend sur $\mathcal{S}$ une fonction (de régression, de classification) h_1, en utilisant un algorithme approprié. L'erreur commise par h_1, mesurée par la fonction de perte ℓ, est :

$$E_{h_1} = \sum_{i=1}^{m} \ell\left(y_i, h_1(\mathrm{x}_i)\right)$$

La quantité $e_i = y_i - h_1(\mathrm{x}_i)$ est appelée le *résidu* (*residual*) de h_1 en x_i.
On recherche d'une fonction h_2 telle que, pour tout $i \in [\![1 \cdots m]\!]$ $|h_2(\mathrm{x}_i) - e_i| < \epsilon$, avec ϵ petit. Dans ce cas, $F = h_1 + h_2$ aura une erreur E_F plus petite que E_{h_1}.
Si par exemple la fonction de perte est définie au point x par l'erreur quadratique :

$$\ell\left(y, h_1(\mathrm{x})\right) = \frac{1}{2}\left(y - h_1(\mathrm{x})\right)^2$$

alors le résidu s'écrit :

$$e = y - h_1(\mathrm{x}) = -\frac{\partial}{\partial h_1(\mathrm{x})}\ell\left(y, h_1(\mathrm{x})\right)$$

et le résidu est alors l'opposé du gradient. Appliqués en $\mathcal{S}$, ces résidus

définissent un nouvel ensemble $\tilde{\mathcal{S}} = \{x_i, e_i\}_{1 \leq i \leq m}$ sur lequel donc h_2 est

appris.

Plan

1. Le dopage (*boosting*)
2. Le dopage par gradient (*gradient boosting*)
3. Le *bagging*
   - Bootstrap
   - Les forêts aléatoires (*random forests*)
   - Autres méthodes. L'apprentissage en cascade (*cascading*)

Plan

Bootstrap

- Pour estimer une statistique θ (moyenne, médiane, variance, etc.) d'une distribution inconnue à partir d'un échantillon S de taille N. Aucune hypothèse sur cette distribution.
- Une étape : on tire N fois dans S avec remplacement. On obtient un *échantillon bootstrap* de taille N.
- On répète B fois cette étape : on possède B échantillons bootstrap.
- On calcule la statistique θ_i sur chaque échantillon bootstrap et on fait la moyenne $\hat{\theta} = \frac{1}{B}\sum_{i=1}^{B}\theta_i$.

Échantillon $S = \{1, 2, 3, 1, 2, 1, 1, 1\}$ de taille 8 d'une distribution dont on veut estimer la moyenne θ. On tire 3 échantillons bootstrap de taille 8.
$\{1, 2, 1, 2, 1, 1, 3\}$; $\theta_1 = 1.375$ $\{1, 1, 2, 2, 3, 1, 1\}$; $\theta_2 = 1.375$
$\{1, 2, 1, 2, 1, 1, 2\}$; $\theta_3 = 1.25$
Moyenne estimée de la distribution : $\hat{\theta} = (1.375 + 1.375 + 1.25)/3 = 1.3333$

Bootstrap Aggregation : Bagging

- On entraîne un algorithme d'apprentissage sur B ensembles $(S_1, \cdots, S_B)$ obtenus par tirage avec remise de m' $(m' < m)$ exemples dans l'ensemble d'entraînement $\mathcal{S}$.
- Pour chaque ensemble S_i de taille m', on obtient une hypothèse h_i. L'hypothèse finale est la moyenne des h_i sur les B tirages :

$$H(\mathrm{x}) = \frac{1}{B} \sum_{i=1}^{B} h_i(\mathrm{x})$$

- Les hypothèses h_i calculées pour chaque tirage S_i ont une variance importante, mais leur moyenne H aura une variance réduite.
- D'autres stratégies de *bagging* n'utilisent pas le *bootstrap* :

Random subspaces Les S_i sont construits à partir d'un sous-ensemble des composantes des vecteurs ;

Pasting Les S_i sont construits sans remise ;

Wagging Chaque classifieur est entraîné sur $\mathcal{S}$ et chaque instance se voit affecter un poids aléatoire.

Plan

Les forêts aléatoires

- Le *bagging* est performant quand il est appliqué à des hypothèses de faible biais mais de forte variance, comme les arbres de décision.
- La *méthode des forêts aléatoires (random forests)* modifie l'algorithme du *bagging* appliqué en ajoutant une *dé-corrélation* entre les arbres.
- La technique consiste à sélectionner aléatoirement un sous-ensemble de m variables à à chaque étape de choix du meilleur nœud de l'arbre.
- Typiquement, n a comme valeur $\sqrt{n}$ quand n est la taille de la dimension de l'espace d'entrée (le nombre d'attributs).
- Lorsque l'on calcule la moyenne de B variables aléatoires, chacune de variance σ^2, la variance globale est de $\frac{1}{B}\sigma^2$. Si les variables sont identiquement distribuées, mais non nécessairement indépendantes, et de même corrélation par paires ρ, alors la variance globale est $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$, dans laquelle le second terme tend vers 0 quand B augmente.
- Les performances de cette méthode sont souvent très bonnes et l'algorithme soit facile à mettre en œuvre.

Plan

La cascade

- Dans l'apprentissage en cascade, les classifieurs agissent en série.
- Le premier classifieur est lancé sur la donnée. Si le résultat est considéré comme fiable, il est conservé.
- Sinon, c'est le second classifieur élémentaire qui traite la donnée. Si son résultat est fiable, on le conserve, sinon on passe la main au classifieur élémentaire suivant, et ainsi de suite.

- Par exemple, pour $C = 2$, le premier classifieur est un hyperplan. Si la marge est trop faible, on passe la main à un classifieur bayésien paramétrique. Si la différence des probabilités n'est pas suffisante, on lance les k plus proches voisins.
- On optimise le temps et la qualité de la décision mais il faut savoir évaluer la fiabilité des décisions.
- La marge ou la probabilité de classement sont des bons indicateurs.

Validations en cascade

On peut aussi évaluer la fiabilité d'un classifieur de la cascade par un ensemble de validation.

- On divise l'ensemble d'apprentissage $\mathcal{S}$ en deux parties $\mathcal{A}_1$ et $\mathcal{V}_1$.
- On apprend le premier classifieur sur $\mathcal{A}_1$. On classe les éléments de $\mathcal{V}_1$.
- Cette classification indique les éléments non fiables de $\mathcal{V}_1$: ils forment un nouvel ensemble $\mathcal{S}_2$, que l'on divise en deux parties $\mathcal{A}_2$ et $\mathcal{V}_2$.
- On apprend le second classifieur sur $\mathcal{A}_2$. On classe les éléments de $\mathcal{V}_2$. Cette classification permet de trouver les éléments non fiables de $\mathcal{V}_2$
- Et ainsi de suite.