IFT-7002 Fondements de l'apprentissage machine

Longueur de description minimale et l'apprentissage non uniforme

Shai Shalev-Shwartz
The Hebrew University of Jerusalem

Traduit et adapté par Mario Marchand
Université Laval

Hiver 2024

Sommaire

Comment exprimer notre connaissance a priori ?

- Jusqu'à maintenant, la connaissance a priori de l'apprenant était exprimée par une classe $\mathcal{H}$ d'hypothèses.

Autre moyens d'exprimer notre connaissance a priori

Occam's Razor: "A short explanation is preferred over a longer one"

Sommaire

Une fonction w pour exprimer nos préférences

- Soit une classe $\mathcal{H}$ dénombrable (possiblement infini).
  - Il est donc possible d'énumérer les éléments de $\mathcal{H}$ avec $\mathbb{N}$.
  - On a donc $\mathcal{H} = \{h_1, h_2, \ldots\}$.
- Soit $w : \mathcal{H} \to \mathbb{R}_+$ tel que $\sum_{h \in \mathcal{H}} w(h) \leq 1$.
  - $w(h)$ reflète notre connaissance a priori sur l'importance de h.

Un langage de description pour $\mathcal{H}$

- Un language de description pour $\mathcal{H}$ est obtenu en décrivant chaque $h \in \mathcal{H}$ par un mot fini $\mathbf{d}(h) \in \{0,1\}^*$.
  - Ex : la classe (dénombrable) de tous les programmes Python.
- Soit $d(\mathcal{H}) \stackrel{\text{def}}{=} \{\mathbf{d}(h) : h \in \mathcal{H}\}$, un language de description pour $\mathcal{H}$.
- Supposons que $d(\mathcal{H})$ est sans préfixe : pour toute paire (h, h') telle $h \neq h'$, $\mathbf{d}(h)$ n'est pas un préfixe de $\mathbf{d}(h')$.
  - Toujours réalisable en réservant une séquence fini de bits comme un symbole de fin-de-mot.
- Soit $|h| \stackrel{\text{def}}{=} |\mathbf{d}(h)| \stackrel{\text{def}}{=}$ la longueur de $\mathbf{d}(h)$.
  - $|h|$ est appelée la longueur de description de h.
- Alors utilisons une fonction de $|h|$ pour notre $w(h)$.

Un $w(h)$ qui dépends de $|h|$

- Inégalité de Kraft : $\sum_{h \in \mathcal{H}} 2^{-|\mathbf{d}(h)|} \leq 1$ lorsque $d(\mathcal{H})$ est sans préfixe.
- On peut donc utiliser $w(h) = 2^{-|h|}$ lorsque $d(\mathcal{H})$ est sans préfixe.
- **Preuve de l'inégalité de Kraft :**
  - Définissons un processus de génération de $d(\mathcal{H})$ comme suit :
    - on tire un bit d_1 (au hasard uniforme dans $\{0, 1\}$).
    - Si le code (d_1) n'est pas dans $d(\mathcal{H})$, on tire un autre bit d_2 (au hasard uniforme).
    - Si le code (d_1, d_2) n'est pas dans $d(\mathcal{H})$, on recommence, et ce, jusqu'à ce que $(d_1, d_2, \ldots, d_k) = \mathbf{d}$ soit dans $d(\mathcal{H})$.
  - Puisque $d(\mathcal{H})$ est sans préfixe, on a que pour tout $\mathbf{d} \in d(\mathcal{H})$: la probabilité que ce processus génère $\mathbf{d}$ est exactement de $2^{-|\mathbf{d}|}$.
    - S'il existait $\mathbf{d}' \in d(\mathcal{H})$ préfixe de $\mathbf{d}$, ce processus ne pourrait pas générer $\mathbf{d}$.
  - La somme de ces probabilités est égale à la probabilité que ce processus s'arrête sur un mot de $d(\mathcal{H})$, ce qui doit donc être ≤ 1. ■
    - Note : cette somme est $\leq 1/2$ lorsqu'aucun $\mathbf{d} \in d(\mathcal{H})$ débute par 0.

Biaisé envers les descriptions courtes

Théorème (La borne de la longueur de description (MDL))

Soit $\mathcal{H}$ une classe dénombrable de classificateurs binaires et $w : \mathcal{H} \to \mathbb{R}_+$ tel que $\sum_{h \in \mathcal{H}} w(h) \leq 1$. Alors, pour tout $\mathcal{D}$ et pour tout $\delta \in (0, 1)$ on a

$$\mathop{\mathbb{P}}_{S \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : \ |L_{\mathcal{D}}(h) - L_S(h)| \leq \sqrt{\frac{-\log(w(h)) + \log(2/\delta)}{2m}} \right) \geq 1 - \delta$$

- Notez que $-\log(w(h)) = \log(1/w(h)) = |h| \log 2 < |h|$ avec le choix précédent pour w. Pour cette raison, $-\log(w(h))$ est souvent appelé la longueur de description de h.
- Comparaison avec la borne VC (qui est une borne uniforme) :

$$\mathop{\mathbb{P}}_{S \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : \ |L_{\mathcal{D}}(h) - L_S(h)| \leq C \sqrt{\frac{\mathrm{VCdim}(\mathcal{H}) + \log(2/\delta)}{2m}} \right) \geq 1 - \delta$$

Preuve

- Pour démontrer ce théorème, nous allons démontrer que

$$\mathcal{D}^m \left(\left\{ S : \exists h \in \mathcal{H} : |L_S(h) - L_{\mathcal{D}}(h)| > \sqrt{\frac{\log(2/\delta_h)}{2m}} \right\} \right) \leq \delta \,,$$

  avec $\delta_h \stackrel{\text{def}}{=} w(h) \cdot \delta$.

- Notez que nous avons :

$$\sum_{h \in \mathcal{H}} \delta_h \leq \delta \,.$$

- De plus, selon l'inégalité de Hoeffding, pour tout h on a

$$\mathcal{D}^m \left(\left\{ S : |L_S(h) - L_{\mathcal{D}}(h)| > \sqrt{\frac{\log(2/\delta_h)}{2m}} \right\} \right) \leq \delta_h \,.$$

Preuve (suite)

- En utilisant la borne de l'union et l'inégalité de Hoeffding, on trouve

$$
\begin{aligned}
&\mathcal{D}^m\left(\left\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \sqrt{\frac{\log(2/\delta_h)}{2m}}\right\}\right) \\
&= \mathcal{D}^m\left(\cup_{h\in\mathcal{H}}\left\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \sqrt{\frac{\log(2/\delta_h)}{2m}}\right\}\right) \\
&\leq \sum_{h\in\mathcal{H}} \mathcal{D}^m\left(\left\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \sqrt{\frac{\log(2/\delta_h)}{2m}}\right\}\right) \\
&\leq \sum_{h\in\mathcal{H}} \delta_h \leq \delta \ .
\end{aligned}
$$

■

Minimization de la borne MDL

- borne MDL : $\forall h \in \mathcal{H},\ L_D(h) \leq L_S(h) + \sqrt{\frac{-\log(w(h))+\log(2/\delta)}{2m}}$
- borne VC : $\forall h \in \mathcal{H},\ L_D(h) \leq L_S(h) + C\sqrt{\frac{\mathrm{VCdim}(\mathcal{H})+\log(2/\delta)}{2m}}$
- Le but de l'apprenant est de minimiser $L_{\mathcal{D}}(h)$ sur $h \in \mathcal{H}$
- Minimiser la borne VC nous donne l'algorithme $\mathrm{ERM}_{\mathcal{H}}(S)$
- Minimiser la borne MDL nous donne l'algorithme $\mathrm{MDL}_{\mathcal{H}}(S)$:

$$\mathrm{MDL}_{\mathcal{H}}(S) \in \operatorname*{argmin}_{h\in\mathcal{H}} \left[L_S(h) + \sqrt{\frac{-\log(w(h)) + \log(2/\delta)}{2m}} \right]$$

- Lorsque $w(h) = 2^{-|h|}$, nous avons $-\log(w(h)) = |h|\log(2)$
- Nous donne un compromis entre le risque empirique $L_S(h)$ et la longueur de description $|h|$: à risque empirique égal, on préfère le prédicteur ayant la longueur de description la plus courte (principe du razoir de Occam).

La garantie pour l'algorithme $\mathrm{MDL}_{\mathcal{H}}(S)$

La borne MDL, nous donne la garantie suivante sur $\mathrm{MDL}_{\mathcal{H}}(S)$.

Corollaire (Garantie pour $\mathrm{MDL}_{\mathcal{H}}(S)$)

Soit $\mathcal{H}$ une classe dénombrable de classificateurs et $w : \mathcal{H} \to \mathbb{R}_+$ telle que $\sum_{h \in \mathcal{H}} w(h) \leq 1$. Alors, avec probabilité $\geq 1 - \delta$ sur $S \sim \mathcal{D}^m$ nous avons

$$\forall h \in \mathcal{H} : L_{\mathcal{D}}(\mathrm{MDL}_{\mathcal{H}}(S)) \leq L_{\mathcal{D}}(h) + 2\sqrt{\frac{-\log(w(h)) + \log(2/\delta)}{2m}}$$

Preuve:

- Utilisons le théorème précédent et considérons

$$\epsilon(m, \delta, h) \stackrel{\text{def}}{=} \sqrt{\frac{-\log(w(h)) + \log(2/\delta)}{2m}} \, .$$

- Alors, avec prob. $\geq 1 - \delta$ sur $S \sim \mathcal{D}^m$, on a simultanément $\forall h \in \mathcal{H}$:

$$[L_{\mathcal{D}}(h) \leq L_S(h) + \epsilon(m, \delta, h)] \text{ et } [L_S(h) \leq L_{\mathcal{D}}(h) + \epsilon(m, \delta, h)] \, .$$

La garantie pour l'algorithme $\mathrm{MDL}_{\mathcal{H}}(S)$

- Puisque $\mathrm{MDL}_{\mathcal{H}}(S)$ est un h qui minimise $L_S(h) + \epsilon(m, \delta, h)$, avec prob. $\geq 1 - \delta$ sur $S \sim \mathcal{D}^m$, on a simultanément $\forall h \in \mathcal{H}$:

$$[L_{\mathcal{D}}(\mathrm{MDL}_{\mathcal{H}}(S)) \leq L_S(h) + \epsilon(m, \delta, h)] \text{ et } [L_S(h) \leq L_{\mathcal{D}}(h) + \epsilon(m, \delta, h)] .$$

- Donc, plus simplement, avec prob. $\geq 1 - \delta$ sur $S \sim \mathcal{D}^m$, on a simultanément $\forall h \in \mathcal{H}$:

$$L_{\mathcal{D}}(\mathrm{MDL}_{\mathcal{H}}(S)) \leq L_{\mathcal{D}}(h) + 2\epsilon(m, \delta, h) .$$

■

Contradiction avec la théorème fondamental de l'apprentissage ?

- Donc si $\mathcal{H}$ est la classe (dénombrable) de tous les classificateurs représentables en langage Python et qu'il existe $h^* \in \mathcal{H}$ tel que $L_{\mathcal{D}}(h^*) = 0$, alors pour tout ϵ, δ, si

$$m \geq \frac{2}{\epsilon^2} \left(-\log(w(h^*)) + \log(2/\delta) \right) ,$$

  on a que $\mathcal{D}^m(\{S : L_{\mathcal{D}}(\mathrm{MDL}_{\mathcal{H}}(S)) \leq \epsilon\}) \geq 1 - \delta$.
- Donc $\mathrm{MDL}_{\mathcal{H}}(S)$ apprends cette classe $\mathcal{H}$
- Or ici, $\mathrm{VCdim}(\mathcal{H}) = \infty$. Donc, selon le théorème fondamental de l'apprentissage, il est impossible d'apprendre $\mathcal{H}$ au sens PAC !
- Comment résoudre cette contradiction apparente ?
- Réponse : Le nombre d'exemples pour apprendre $\mathcal{H}$ dépends de h^*, ce qui est interdit pour les critères PAC et PAC agnostique.

Sommaire

Apprendre non uniformément vs. apprentissage PAC

Définition (Apprenable non uniformément)

$\mathcal{H}$ est *apprenable non uniformément* si $\exists A$ et $m_{\mathcal{H}}^{\text{NUL}} : (0,1)^2 \times \mathcal{H} \to \mathbb{N}$ tel que $\forall \epsilon, \delta \in (0,1), \forall h \in \mathcal{H}$, si $m \geq m_{\mathcal{H}}^{\text{NUL}}(\epsilon, \delta, h)$ alors $\forall \mathcal{D}$,

$$\mathcal{D}^m \left(\{ S : L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon \} \right) \geq 1 - \delta \ .$$

- Le nombre d'exemples requis dépends de ϵ, δ, ET h.

Définition (Agnostiquement PAC apprenable)

$\mathcal{H}$ est agnostiquement PAC apprenable si $\exists A$ et $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ tel que $\forall \epsilon, \delta \in (0,1)$, si $m \geq m_{\mathcal{H}}(\epsilon, \delta)$, alors $\forall \mathcal{D}$,

$$\mathcal{D}^m \left(\{ S : \forall h \in \mathcal{H}, L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon \} \right) \geq 1 - \delta \ .$$

- Le nombre d'exemples requis dépends de ϵ, δ.

Apprentissage non uniforme vs. apprentissage PAC

Corollaire (au théorème de la borne MDL)

Soit $\mathcal{H} \subset \{0,1\}^{\mathcal{X}}$ la classe de toutes les fonctions Booléennes calculables (implémentables dans un langage de programmation)

- $\mathrm{MDL}_{\mathcal{H}}$ *apprends* $\mathcal{H}$ *non uniformément avec la complexité d'échantillon*

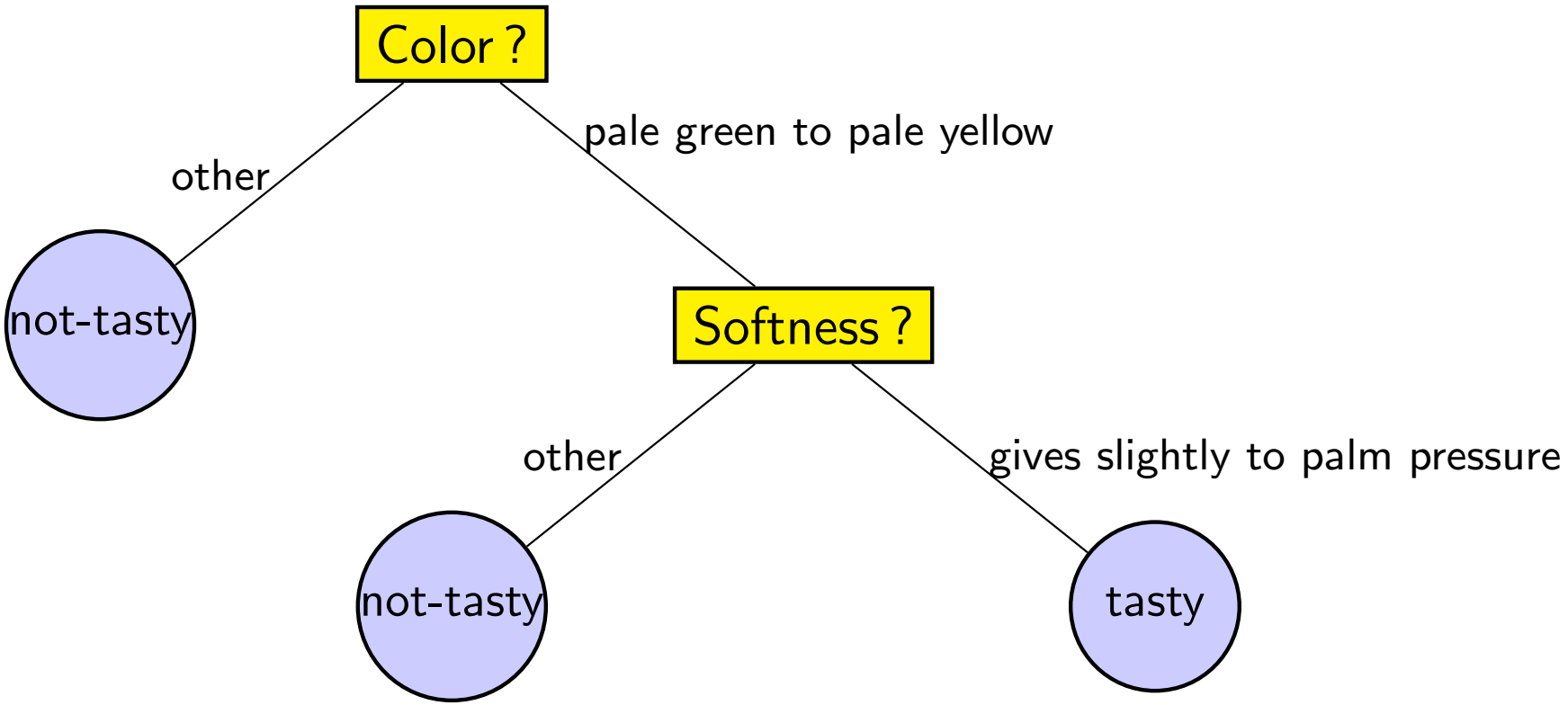
$$m_{\mathcal{H}}^{NUL}(\epsilon, \delta, h) \leq 2\frac{\log(1/w(h)) + \log(2/\delta)}{\epsilon^2}$$

- $\mathcal{H}$ *n'est pas agnostiquement PAC apprenable*

- La dimension VC caractérise l'apprentissage au sens PAC agnostique pour les classes de classificateurs binaires.
  - Car $\mathcal{H}$ est agnostiquement PAC apprenable ssi $\mathrm{VCdim}(\mathcal{H}) < \infty$.
- Dans le manuel il est démontré qu'une classe de classificateurs binaires est apprenable non uniformément ssi elle est l'union dénombrable de classes agnostiquement PAC apprenables.

Sommaire

Arbre de décision pour papayes



- Avantage pour les arbres de décisions : ils sont faciles à interpréter.
- Un arbre de décision effectue une partition disjointe de $\mathcal{X}$.
  - k feuilles implique $k-1$ noeuds internes.

La dimension VC des arbres de décision

- Soit la classe $\mathcal{H}_{T_k}$ des fonctions booléennes qu'il est possible de construire avec un arbre T_k de structure fixe ayant k feuilles (et $k-1$ noeuds internes).
  - Seule l'étiquette $\in \{0, 1\}$ associée à chaque feuille peut varier.
  - On peut alors générer 2^k fonctions booléennes avec ces arbres.
- Cette classe de fonctions peut donc pulvériser k instances.
- Aucun ensemble de $k+1$ instances est pulvérisé.
  - car l'ensemble des feuilles couvre $\mathcal{X}$ et on aura donc au moins une feuille couvrant 2 instances.
- Alors $\mathrm{VCdim}(\mathcal{H}_{T_k}) = k$.
- La VCdim est $+$ élevée lorsque la structure de T_k peut varier.
- Examinons un cas où la structure de T_k peut varier.

Un langage de description pour les arbres de décision

- Considérons $\mathcal{X} = \{0,1\}^d$ et l'ensemble des d règles de décision ("splitting rules") de la forme $\mathbb{1}_{[x_i=1]}$ pour tous les attributs $i \in [d]$.
- Considérons la classe $\mathcal{H}_d$ de tous les arbres de décision sur $\mathcal{X}$ pouvant être construits avec cet ensemble de règles de décision.
- Affirmation : Cette classe contient $\{0,1\}^{\mathcal{X}}$ (exercice 18.1).
- Alors $\mathrm{VCdim}(\mathcal{H}_d) = |\mathcal{X}| = 2^d$. (fini mais très élevé lorsque $d \gg 1$)
- Approche MDL : soyons biaisés envers les petits arbres de décision.
- Chaque arbre $h \in \mathcal{H}_d$ de n_h noeuds est décrit par une séquence de n_h blocs de $\lceil \log_2(d+2) \rceil$ bits.
- Chaque bloc représente un noeud de l'arbre qui est soit :
  - Un noeud interne de la forme $\mathbb{1}_{[x_i=1]}$ pour un $i \in [d]$.
  - Une feuille de valeur 1.
  - Une feuille de valeur 0.
- La séquence est le résultat du parcours pré-ordre de l'arbre (qui permet de reconstruire l'arbre lorsque les feuilles sont identifiées).

Algorithmes pour arbres de décision

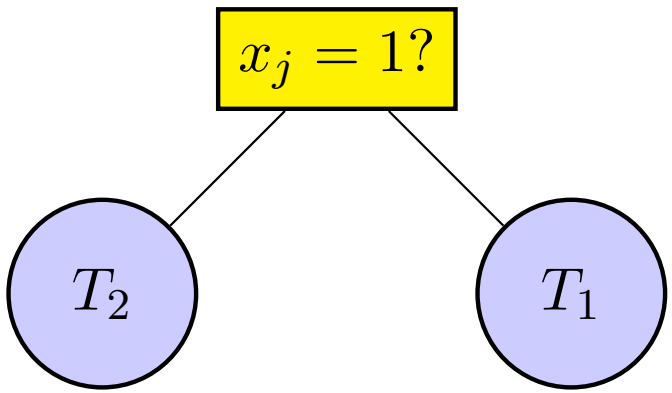
- Ce langage de description est sans préfixe car un arbre de décision (avec feuilles et noeuds internes identifiés) ne peut pas être un sous arbre d'un autre arbre.
- Utilisons alors $w(h) = 2^{-|h|}$ avec $|h| = n_h \lceil \log_2(d+2) \rceil$.
- L'algorithme $\mathrm{MDL}_{\mathcal{H}}(S)$ retourne un arbre h de n_h noeuds minimisant

$$L_S(h) + \sqrt{\frac{n_h \lceil \log_2(d+2) \rceil \log(2) + \log(2/\delta)}{2m}}\,.$$

- Or, il est $\mathcal{NP}$-difficile de trouver l'arbre de $\mathcal{H}_d$ minimisant cette borne.
- Alternative : utiliser un algorithme glouton (comme à la page suivante) pour construire l'arbre.
  - C'est ce que font CART (Breiman et al., 1984) et ID3 (Quinlan, 1986).
- Ces algorithmes sont conformes à l'approche MDL car ils tentent de trouver un arbre avec peu de noeuds ayant un faible risque empirique.

Algorithme glouton $\texttt{TE}(S)$ pour construire un arbre

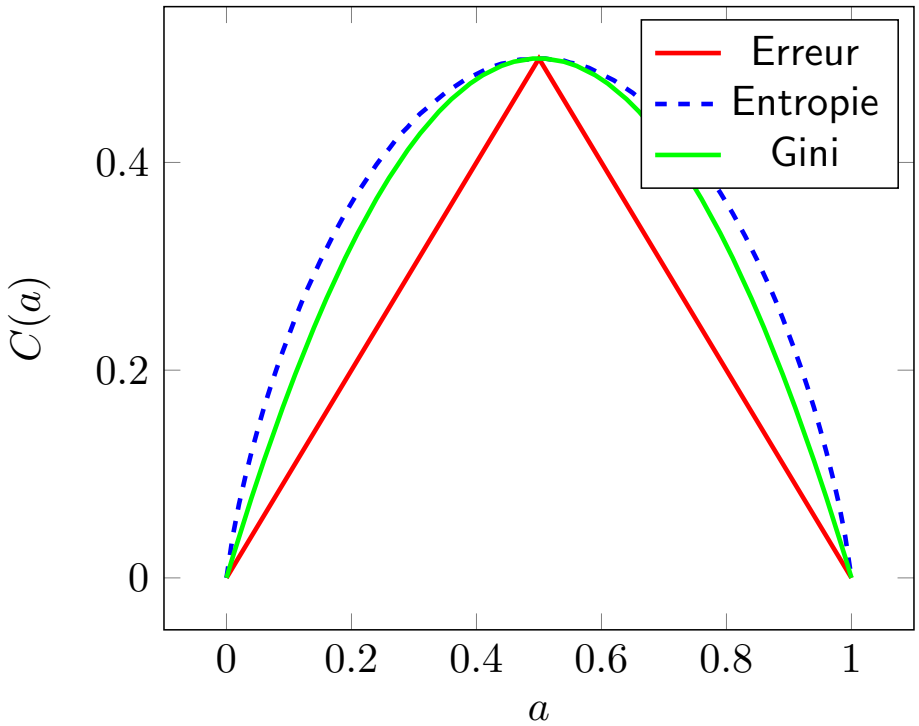
- Entrée : un ensemble S d'exemples
- A est l'ensemble des attributs
- **si** $y = 1$ pour tout $(\mathbf{x}, y) \in S$, retourner une feuille 1
- **si** $y = 0$ pour tout $(\mathbf{x}, y) \in S$, retourner une feuille 0
- soit $j = \text{argmax}_{i \in A} \texttt{Gain}(S, i)$
- **si** $\texttt{Gain}(S, j) \leq 0$ retourner une feuille de la classe majoritaire de S
- soit T_1 l'arbre retourné par $\texttt{TE}(\{(\mathbf{x}, y) \in S : x_j = 1\})$
- soit T_2 l'arbre retourné par $\texttt{TE}(\{(\mathbf{x}, y) \in S : x_j = 0\})$
- retourner l'arbre :



Critères utilisés pour le calcul du Gain

$$\texttt{Gain}(S, i) \overset{\text{def}}{=} C\big(\mathop{\mathbb{P}}_{S}[y = 1]\big) - \big(\mathop{\mathbb{P}}_{S}[x_i = 1]\, C(\mathop{\mathbb{P}}_{S}[y = 1 | x_i = 1]) + \mathop{\mathbb{P}}_{S}[x_i = 0] C(\mathop{\mathbb{P}}_{S}[y = 1 | x_i = 0])\big).$$

- Erreur empirique : $C(a) = \min\{a, 1 - a\}$
- Entropie : $C(a) = -a \log_4(a) - (1 - a) \log_4(1 - a)$
- Index Gini : $C(a) = 2a(1 - a)$



Quel critère choisir ?

- $C(\mathbb{P}_S[y = 1])$ est une mesure du degré d'impureté d'un noeud.
- Le Gain d'une partition du noeud est la différence entre l'impureté du noeud et une combinaison convexe de l'impureté des enfants.
- Si l'impureté du noeud est > 0 et que la pureté d'un de ses enfants augmente, cette différence n'est jamais nulle pour le critère de l'entropie et de l'index Gini car ces fonctions sont localement strictement concaves partout.
- Par contre, il arrive souvent que cette différence soit nulle pour le critère d'erreur car cette fonction est linéaire par morceaux.
- Donc le Gain est souvent nul pour le critère de l'erreur alors qu'il est toujours positif pour les critères Gini et Entropie lorsque le noeud n'est pas pur et que la pureté d'un des enfants est supérieure.

Quel critère choisir ?

- En effet, considérons un noeud que l'on désire partitionner avec l'attribut x. Définissons

$$a \overset{\text{def}}{=} \mathop{\mathbb{P}}_{S}[y=1] \quad ; \quad \alpha \overset{\text{def}}{=} \mathop{\mathbb{P}}_{S}[x=1]$$
$$a_1 \overset{\text{def}}{=} \mathop{\mathbb{P}}_{S}[y=1|x=1] \quad ; \quad a_0 \overset{\text{def}}{=} \mathop{\mathbb{P}}_{S}[y=1|x=0]\,.$$

- Puisque $a = \alpha a_1 + (1-\alpha)a_0$, le Gain de partitionner est donné par

$$C(\alpha a_1 + (1-\alpha)a_0) - \left[\alpha C(a_1) + (1-\alpha)C(a_0)\right].$$

- Ce gain est donc positif ssi C est une fonction strictement concave dans l'intervalle entre a_0 et a_1.
  - ce qui est toujours le cas pour l'entropie et l'index Gini.
  - mais ce n'est pas le cas pour l'erreur lorsque lorsque a_0 et a_1 sont tous les deux $\leq 1/2$ ou tous les deux $\geq 1/2$ (voir graphiques).

Quel critère choisir ?

- L'approche MDL nous dit que le partitionnement devrait être refusé si l'on ajoute des noeuds sans décroitre l'erreur empirique.
- Par contre, partitionner serait globalement une excellente décision si on parvient, par la suite, à partitionner un noeud enfant ayant une grande impureté en deux noeuds purs ; ce qui aurait pour effet de diminuer grandement l'erreur empirique.
- Pour cette raison, l'index Gini et l'entropie sont beaucoup plus utilisés que le critère d'erreur.
- Mais cela illustre bien les limitations de l'approche vorace pour notre problème d'optimisation $\mathcal{NP}$-difficile.
  - Hyafil et Rivest (1976) ont démontré qu'il est $\mathcal{NP}$-difficile de trouver l'arbre de décision de taille minimale minimisant les erreurs de classification.

Utilisation d'attributs à valeurs réelles

- Pour chaque attribut x à valeur réelle, nous pouvons considérer des souches de décisions avec différents seuils θ.
- On trie d'abord les m exemples en ordre croissant selon x pour obtenir $x_1 \leq x_2 \leq \ldots \leq x_m$.
- Ensuite on considère $m+1$ souches de décisions (à valeur $\{0, 1\}$), chacune ayant un seuil $\theta_i = (x_i + x_{i+1})/2$ lorsque $x_i < x_{i+1}$.
  - avec $\theta_0 = x_1 - 1$ et $\theta_m = x_m + 1$.
- En effectuant ce tri, le calcul du Gain pour toutes ces $m+1$ souches est en $O(m \log m)$.
  - Le tri s'effectue en $O(m \log m)$ et, ensuite, le calcul du Gain pour les $m+1$ seuils possibles s'effectue en $O(m)$.
- Donc si on a d attributs réels, on peut trouver la souche de décision maximisant le Gain en $O(dm \log m)$.
  - L'idée est la même que celle présentée à la section 10.1.1 du manuel pour le Boosting des souches de décision.

Élagage de l'arbre

- Généralement, l'algorithme glouton TE retourne un arbre effectuant très peu d'erreurs sur S, mais contenant beaucoup de noeuds.
  - TE possède donc une faible erreur d'approximation et une grande erreur d'estimation ; le compromis n'est généralement pas bon.
- Pour trouver un arbre plus petit que celui retourné par TE (et effectuant un peu plus d'erreurs) on a les solutions suivantes :
  - Arrêter TE lorsqu'un nombre maximum de noeuds est atteint.
  - Contraindre la hauteur de l'arbre.
  - Élaguer l'arbre retourné par TE.
- L'élagage procède des feuilles jusqu'à la racine, typiquement en tentant de remplacer un noeud interne par un de ses sous arbres ou une feuille.
  - La décision de remplacer ou non se base sur le calcul d'une fonction comme celle de la borne MDL ou une estimation du risque calculé sur un ensemble de validation.
    - Une procédure générique se trouve à la section 18.2.2

Logiciels pour arbres de décisions

- ID3 utilise TE, mais avec la contrainte que les règles de décisions sont toutes distinctes le long de chaque chemin menant à une feuille.
- C4.5 est une version améliorée de ID3 qui incorpore l'élagage et qui peut utiliser les données avec valeurs manquantes.
- C5.0 est une version améliorée, accélérée, et commerciale de C4.5.
  - Elle incorpore le Boosting d'arbres de décision.
- CART construit des arbres pour la classification ET la régression.
  - Pour la classification, CART utilise TE pour construire un arbre jusqu'à ce que toutes les feuilles soient pures (autant que possible).
  - Il utilise ensuite une procédure d'élagage sophistiquée pour le réduire.
- scikit-learn decision tree utilise l'algorithme TE, mais en permettant de contraindre la hauteur de l'arbre, le nombre minimum d'exemples dans une feuille, le nombre maximum de feuilles, etc... Et permet aussi de l'élaguer.

Bagging et forêts aléatoires

- Les algorithms de construction d'arbres sont instables : une petite modification dans les données peut provoquer un grand changement dans l'arbre produit. Ceci peut conduire au "overfitting".
- Un façon d'éliminer cette source d'overfitting est de faire un moyennage d'arbres à l'aide du "Bagging" ("Boostrapt aggregating").
  - Cela consiste à générer différents échantillons bootstrap à partir de S. Chaque échantillon bootstrap S' est obtenu en tirant au hasard uniforme (et avec remplacement) m exemples de S.
  - Un arbre est construit sur chaque échantillon bootsrap.
  - Le classificateur final est un vote de majorité non pondéré de chacun des arbres.
- Dans l'approche des forêts aléatoires ("random forest") : pour chaque échantillon bootstrap S' on tire au hasard k attributs parmi les d attributs disponibles pour construire l'arbre sur S'.
  - chaque arbre ainsi produit a tendance à être très différents des autres.
  - Le classificateur final est un vote de majorité non pondéré des arbres.

Résumé

- Nous possédons maintenant deux moyens d'exprimer notre connaissance a priori :
  - À l'aide d'une classe d'hypothèses (critère PAC, algorithme $\mathrm{ERM}_{\mathcal{H}}$) satisfaisant la propriété de convergence uniforme.
  - À l'aide d'une classe d'hypothèses satisfaisant une propriété de convergence non uniforme. (critère non uniforme, algorithmes $\mathrm{MDL}_{\mathcal{H}}$).
- Ordonnancement de ces deux critères d'apprentissage
  - PAC-apprenable $\Rightarrow$ apprenable non uniformément.
- Algorithmes pour arbres de décisions : conforme avec l'approche MDL (apprentissage non uniforme).