COURS DE DEUXIÈME ANNÉE
INGÉNIEUR ISIMA

Machine Learning
Méthodes d'ensembles

Vincent Barra

Table des matières

1 Introduction

Les méthodes d'ensemble sont des algorithmes d'apprentissage fondés sur l'idée qu'une combinaison de classifieurs simples (dits faibles), si elle est bien faite, doit donner de meilleurs résultats que chacun des classifieurs pris séparément. Le principe général suivant est adopté : il s'agit de construire une famille de modèles qui sont ensuite agrégés (moyenne pondérée des estimations,vote,...). Suivant la famille de modèles considérés (modèles dépendant les uns des autres, modèles indépendants), on aboutit à des stratégies différentes (boosting dans le premier cas, bagging, forêts aléatoires dans le second cas)
Dans ces méthodes, les notions de classifieurs faible est fort est fondamentale. Considérons un problème de classification binaire, à valeurs dans $\{-1,1\}$. Soit un ensemble d'exemples $Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1,1\}\}$, les x_i étant des échantillons d'une certaine distribution P, et $y_i = f(x_i)$, f règle de classification. Un classifieur h est dit fort si, pour n suffisamment grand, il produit pour tout P, f, $\varepsilon \geq 0$ et $\delta \leq 1/2$ une sortie avec

probabilité plus petite que $1-\delta$ telle que $\mathbb{P}_p(h(x) \neq f(x)) \leq \varepsilon$. De plus, la complexité temporelle de h doit au plus être polynomiale en $1/\varepsilon$, $1/\delta$ et n.
A l'inverse, un classifieur faible produit, pour un certain $\varepsilon_0 \geq 0$, un certain $\delta_0 \leq 1/2$, une sortie avec probabilité plus petite que $1-\delta_0$ telle que $\mathbb{P}_p(h(x) \neq f(x)) \leq \varepsilon_0$. En pratique, souvent, les classifieurs faibles produisent des résultats à peine meilleurs que l'aléatoire (dans le cas de la classification binaire, un tirage uniforme sur $\{-1,1\}$).

2 Algorithmes de boosting

Le boosting considère la construction d'une famille de modèles dépendant les uns des autres. Chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées ou mal prédites. Intuitivement, ces algorithmes concentrent donc leurs efforts sur les observations les plus difficiles à ajuster tandis que l'agrégation de l'ensemble des modèles permet d'échapper au surajustement.

Les algorithmes de boosting diffèrent par plusieurs caractéristiques :
- la façon de pondérer c'est-à-dire de renforcer l'importance des observations mal estimées lors de l'itération précédente ;
- leur objectif selon le type de la variable a prédire : binaire, qualitative à C classes, réelles ;
- la fonction perte, qui peut être choisie plus ou moins robuste aux valeurs atypiques, pour mesurer l'erreur d'ajustement ;
- la façon de pondérer les classifieurs successifs.

Le premier algorithme de boosting a été proposé dans [6], dans lequel un classifieur fort est construit par combinaison de classifieurs faibles (algorithme 1)

2.1 AdaBoost

AdaBoost (Adaptive Boosting, [3]) propose d'utiliser des versions pondérées du même ensemble d'apprentissage, plutôt que des sous-ensembles produits aléatoirement. Cela induit que n ne doit nécessairement pas être grand, contrairement à l'algorithme 1.

2.1.1 Algorithme

Soit un problème de classification à deux classes. On dispose d'un ensemble d'apprentissage $Z = \{(x_i, y_i)_{1 \leq i \leq n}, x_i \in X, y_i \in \{-1,1\}\}$ et on cherche à évaluer la classe d'un point $x \in X$. On dispose de M classifieurs faibles h_i donnant une classification faible de x. On souhaite construire un classifieur h, à valeurs dans $\{-1,1\}$, combinaison linéaire des h_i.
Dans cette algorithme, le modèle de base retourne l'identité d'une classe, il est encore nommé AdaBoost discret (algorithme 2). Il est facile de l'adapter à des modèles retournant une valeur réelle comme une probabilité d'appartenance à une classe.
Les poids de chaque observation sont initialisés à $\frac{1}{n}$ pour l'estimation du premier modèle, puis évoluent à chaque itération donc pour chaque nouvelle estimation. Le poids d'un

Algorithme 1 : Algorithme de boosting pour la classification

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
Résultat : un classifieur h

1. Z_1 : sous-ensemble de $n_1 < n$ exemples de Z, tirés aléatoirement sans remise
2. apprentissage d'un classifieur faible h_1 sur Z_1
3. Z_2 : sous-ensemble de $n_2 < n$ exemples de Z, dont la moitié sont des exemples mal classés par h_1
4. apprentissage d'un classifieur faible h_2 sur Z_2
5. Z_3 : ensemble des exemples sur lesquels h_1 et h_2 sont en désaccord
6. apprentissage d'un classifieur faible h_3 sur Z_3

$$h(x) = sign\left(\sum_{i=1}^{3} h_i(x)\right)$$

exemple est inchangé si ce dernier est bien classé, il croît sinon proportionnellement au défaut d'ajustement du modèle. Estimer h_i sur Z pondéré par w signifie trouver le classifieur faible parmi une famille de classifieurs satisfaisant

$$\sum_{j=1}^{n} w_j \mathbb{I}_{h_i(x_j) \neq y_j} \leq \frac{1}{2} - \varepsilon$$

pour un petit ε.
L'agrégation finale des prévisions :

$$\sum_{i=1}^{M} \alpha_i h_i(x)$$

est une combinaison pondérée par les qualités d'ajustement de chaque modèle. Sa valeur absolue appelée marge est proportionnelle à la confiance que l'on peut attribuer à son signe qui fournit le résultat de la prévision.
Après M itérations, les exemples avec un très fort poids sont des exemples durs à apprendre, et possiblement des points aberrants. AdaBoost peut donc servir à détecter des outliers sur un ensemble d'apprentissage Z donné.
Notons qu'il est possible [1] d'étendre AdaBoost à des problèmes de régression.

2.1.2 Interprétation statistique

Il est possible d'interpréter l'algorithme 2 en termes statistiques, pour justifier en partie le bon comportement du boosting en classification. Pour ce faire, on définit les classifieurs faibles comme des fonctions paramétriques $h_{\theta_i}(x) = h_i(x, \theta)$, $\theta \in \Theta$, et le résultat de la classification comme une combinaison linéaire de ces classifieurs faibles

Algorithme 2 : Algorithme AdaBoost en classification binaire

Entrées :
$Z = \left\{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\right\}$
$x \in X$
M le nombre de classifieurs faibles
Résultat : un classifieur h

Initialisation des poids w : $\forall i \in \{1 \cdots n\}, (w_i = \frac{1}{n})$
pour $i \leftarrow 1$ **à** M **faire**
  Estimer h_i sur Z pondéré par w
  Calculer le taux d'erreur $\varepsilon_i = \sum_{j=1}^{n} \mathbb{I}_{h_i(x_j) \neq y_j}$
  Calculer le poids du classifieur faible $\alpha_i \leftarrow \frac{1}{2} log\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j \leftarrow w_j exp\left[-\alpha_i y_j h_i(x_j)\right]$
  fin
  Renormaliser les poids : $W = \sum_{j=1}^{n} w_j$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j \leftarrow w_j / W$
  fin
fin

$$h(x) = sign\left[\sum_{j=1}^{M} \alpha_j h_j(x)\right]$$

$h(x) = \sum_{i=1}^{n} \alpha_i h_{\theta_i}(x)$.

Une approche pour déterminer les θ_i et les α_i est d'ajouter séquentiellement au problème d'apprentissage les classifieurs faibles, sans ajuster les paramètres et les coefficients de la solution courante (algorithme 3).

Algorithme 3 : Modèle additif pas à pas.

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M le nombre de classifieurs faibles
L une fonction de perte
Résultat : un classifieur h

$h_0(x) = 0$
pour $i \leftarrow 1$ **à** M **faire**
  $(\alpha_i, \theta_i) = arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{j=1}^{n} L(y_j h_{i-1}(x_j) + \alpha h_{\theta_i}(x_j))$
  $h_i(x) = h_{i-1}(x) + \alpha_i h_{\theta_i}(x)$
fin
$h(x) = \sum_{i=1}^{n} \alpha_i h_{\theta_i}(x)$

Cet algorithme, lorsque l'on utilise la fonction de perte exponentielle $L(y, f(x)) = e^{-yf(x)}$, est équivalent à l'algorithme ADABOOST. En effet, à chaque itération, la minimisation suivante est effectuée :

$$(\alpha_i, \theta_i) = arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{j=1}^{n} exp\left[-y_j(h_{i-1}(x_j) + \alpha h_{\theta_i}(x_j))\right] \quad (1)$$

$$= arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{j=1}^{n} exp\left[-y_j h_{i-1}(x_j)\right] exp\left[-y_j \alpha h_{\theta_i}(x_j)\right] \quad (2)$$

$$= arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{j=1}^{n} w_j^i exp\left[-y_j \alpha h_{\theta_i}(x_j)\right] \quad (3)$$

où $w_j^i = exp\left[-y_j h_{i-1}(x_j)\right]$ n'affecte pas le problème d'optimisation. Pour tout $\alpha > 0$, la

fonction objectif peut être réécrite

$$\begin{aligned}
\theta_i &= arg\min_{\theta\in\Theta}\left[e^{-\alpha}\sum_{y_j=h_{\theta_i}(x_j)} w_j^t + e^{\alpha}\sum_{y_j\neq h_{\theta_i}(x_j)} w_j^i\right] \\
&= arg\min_{\theta\in\Theta}\left[(e^{-\alpha}+e^{\alpha})\sum_{j=1}^{n} w_j^i \mathbb{I}_{\{y_j\neq h_{\theta_i}(x_j)\}} + e^{\alpha}\sum_{j=1}^{n} w_j^i\right] \\
&= arg\min_{\theta\in\Theta}\sum_{j=1}^{n} w_j^i \mathbb{I}_{\{y_j\neq h_{\theta_i}(x_j)\}}
\end{aligned}$$

Le classifieur faible minimisant (3) minimisera donc également le taux d'erreur pondéré, que l'on réinjecte dans (3) pour trouver

$$\alpha_i = \frac{1}{2} log \frac{1-\varepsilon_i}{\varepsilon_i}$$

avec

$$\varepsilon_i = \sum_{j=1}^{n} w_j^i \mathbb{I}_{\{y_j\neq h_{\theta_i}(x_j)\}}$$

Enfin, la mise à jour du modèle $h_i(x) = h_{i-1}(x) + \alpha_i h_{\theta_i}(x)$ est équivalente à la mise à jour des poids dans AdaBoost, puisque

$$w_j^{i+1} = w_j^i e^{-y_j h_i(x_j)}$$

En résumé, AdaBoost peut être interprété comme un algorithme minimisant la fonction de perte exponentielle par ajout itératif de classifieurs faibles.

2.1.3 Interprétation comme problème de maximisation d'une marge

Il est également possible de relier AdaBoost à un problème de séparation à vaste marge. Le principe est de construire un espace de coordonnées de dimension égale au nombre de classifieurs faibles, M, et pour $u \in \mathbb{R}^M$ de considérer les coordonnées u_i comme les sorties des classifieurs faibles h_i. On montre alors qu'AdaBoost est un algorithme itératif qui résout le problème d'optimisation suivant

$$\hat{w} = arg\max_{w\in\mathbb{R}^M}\min_{u_j\in\mathbb{R}^M}\frac{y_j < w, u_j >}{\|w\|_{L_1}}$$

où $u_j = (h_1(x_j)\cdots h_M(x_j))$ et où le classifieur final est

$$h(x) = \sum_{i=1}^{M} \hat{w}_i^M h_i(x)$$

ce qui correspond à un problème de maximisation de la plus petite des distance entre des points de classes différentes, soit un problème de maximisation de marge.

2.2 Variations autour d'AdaBoost

De nombreuses variantes de l'algorithme 2 ont été proposées. Nous en décrivons quelques unes, dans le cas de la classification, et dans le cas de la régression.

2.2.1 Real AdaBoost

Real Adaboost ([4]) remplace la classification par h_i des points de Z et l'estimation du taux d'erreur ε_i par le calcul d'une valeur réelle p_i, qui détermine avec quelle probabilité un point x appartient à une classe, étant donnée la distribution des poids sur l'ensemble Z. La contribution de h_i au classifieur fort est la moitié de la transformation logit de p_i (algorithme 4).

Algorithme 4 : Algorithme Real AdaBoost en classification binaire

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M le nombre de classifieurs faibles
Résultat : un classifieur h

Initialisation des poids w : $\forall i \in \{1 \cdots n\}, (w_i = \frac{1}{n})$
pour $i \leftarrow 1$ **à** M **faire**
  Calculer $p_i(x) = P_w(y = 1|x)$ en utilisant w_i
  $h_i = \frac{1}{2} log\left(\frac{p_i(x)}{1-p_i(x)}\right)$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j \leftarrow w_j exp\left[-y_j h_i(x_j)\right]$
  fin
  $W = \sum_{j=1}^{n} w_j$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j \leftarrow w_j / W$
  fin
fin
$h(x) = sign\left[\sum_{i=1}^{M} h_i(x)\right]$

2.2.2 LogitBoost

LogitBoost [4] considère AdaBoost comme un modèle additif généralisé et applique la fonction de perte L de la régression logistique, plutôt que la fonction de perte exponentielle comme AdaBoost. Vu comme un problème d'optimisation, LogitBoost s'écrit donc

$$h = Arg \min_f \sum_{i=1}^{n} log\left(1 + e^{-y_i f(x_i)}\right)$$

L'algorithme 5 détaille l'implémentation de cette méthode.

Algorithme 5 : Algorithme LogitBoost en classification binaire

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M
Résultat : un classifieur h

Initialisation des poids w : $\forall i \in \{1 \cdots n\}, (w_i = \frac{1}{n})$
Initialisation des p : $\forall i \in \{1 \cdots n\}, p(x_i) = \frac{1}{2}$
$h = 0$
pour $i \leftarrow 1$ **à** M **faire**
  pour $j \leftarrow 1$ **à** n **faire**
    $u_j = \frac{y_j - p(x_j)}{p(x_j)(1-p(x_j))}$
    $w_j = p(x_j)(1 - p(x_j))$
  fin
  Calculer h_i par moindres carrés pondérés des u_i sur les x_i, avec les poids w_i
  $h(x) = h(x) + \frac{1}{2} h_i(x)$
  $p(x) = \frac{e^{h(x)}}{e^{h(x)} + e^{-h(x)}}$
fin

2.2.3 Emphasis Boost

Dans cette version de boosting [5], chaque x_j est pondéré par un critère paramétré par $\lambda \in [0, 1]$, de sorte que l'entraînement se concentre sur les points critiques (près de la frontière de décision), sur l'erreur quadratique de chaque point, ou sur toute autre situation intermédiaire. La fonction de pondération est définie par :

$$w_j = exp\left[\lambda \left(\sum_{i=1}^{M} (\alpha_i h_i(x_j) - y_j)^2\right) - (1-\lambda)\left(\sum_{i=1}^{M} h_i(x_j)\right)^2\right]$$

Suivant la valeur de l'hyperparamètre λ, l'algorithme accorde plus ou moins d'importance aux points critiques :

— si $\lambda = 0$, seule la proximité à la frontière de décision est prise en compte
— si $\lambda = 1/2$, on retrouve une fonction de pondération de type Real AdaBoost
— si $\lambda = 1$, le poids est mis sur l'erreur quadratique de chaque point.

2.2.4 Régression par boosting

Différentes adaptations du boosting ont été proposées pour le cas de la régression, c'est-à-dire lorsque la variable à prédire est quantitative. En voici un exemple, où la fonction de perte L peut être exponentielle, quadratique ou, plus robuste, la valeur absolue. On note

Algorithme 6 : Algorithme Emphasis AdaBoost en classification binaire

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M le nombre de classifieurs faibles
λ
Résultat : un classifieur h

Initialisation des poids w : $\forall i \in \{1 \cdots n\}, (w_i = \frac{1}{n})$
pour $i \leftarrow 1$ **à** M **faire**
  Estimer h_i sur Z pondéré par w
  Calculer le taux d'erreur $\varepsilon_i = \dfrac{\sum_{j=1}^{n} w_j y_j h_i(x_j)}{\sum_{j=1}^{n} w_j}$
  Calculer le poids du classifieur faible $\alpha_i \leftarrow \frac{1}{2} log\left(\frac{1+\varepsilon_i}{1-\varepsilon_i}\right)$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j = exp\left[\lambda\left(\sum_{k=1}^{i}(\alpha_k h_k(x_j) - y_j)^2\right) - (1-\lambda)\left(\sum_{k=1}^{m} h_k(x_j)\right)^2\right]$
  fin
  Renormaliser les poids : $W = \sum_{j=1}^{n} w_j$
  pour $j \leftarrow 1$ **à** n **faire**
    $w_j \leftarrow w_j / W$
  fin
fin
$h(x) = sign\left[\sum_{i=1}^{M} \alpha_i h_i(x)\right]$

$L_i = \sup_{1 \leq j \leq n} L(h_i(x_j), y_j) = \sup_{1 \leq j \leq n} l_i(j)$ le maximum de l'erreur observée par le classifieur h_i sur l'échantillon initial. On introduit enfin la fonction g telle que

$$g(l_i(j)) = \beta_i^{1-l_i(j)/L_i}$$

avec

$$\beta_i = \frac{\varepsilon_i}{L_i - \varepsilon_i}$$

Algorithme 7 : Boosting pour la régression

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M le nombre de classifieurs faibles
Résultat : un classifieur h
Initialiser p à une distribution uniforme : $\forall i \in \{1 \cdots n\}, (p_i = \frac{1}{n})$
pour $i \leftarrow 1$ **à** M **faire**
  Tirer avec remise dans Z un sous-échantillon Z_i suivant p
  Estimer h_i sur Z_i
  pour $j \leftarrow 1$ **à** n **faire**
    $l_i(j) = L(h_i(x_j), y_j), \forall j \in \{1 \cdots n\}$
    $w_j = g(l_i(j)) p_j$
  fin
  $\varepsilon_i = \sum_{j=1}^{n} p_j l_i(j)$
  $L_i = \sup_{1 \leq j \leq n} l_i(j)$
  $\beta_i = \frac{\varepsilon_i}{L_i - \varepsilon_i}$
  $W = \sum_{j=1}^{n} w_j$
  pour $j \leftarrow 1$ **à** n **faire**
    $p_j \leftarrow \frac{w_j}{W}$
  fin
fin
$h(x) = \sum_{i=1}^{M} log\left(\frac{1}{\beta_i}\right) h_i(x)$

L'algorithme produit M prédicteurs construits sur des échantillons Z_i dont le tirage dépend de probabilités p mises à jour à chaque itération. Cette mise à jour est fonction d'un paramètre β_j qui est un indicateur de la performance, sur l'échantillon Z_i, du i-ième prédicteur estimé sur l'échantillon Z_i. La mise à jour des probabilités dépend donc à la fois de cet indicateur global et de la qualité relative $l_i(j)/L_i$ de l'estimation du j-ème individu. L'estimation finale est enfin obtenue à la suite d'une moyenne (ou médiane en fait, pour s'affranchir des prévisions atypiques) des prévisions pondérées par la qualité respective de chacune de ces prévisions.

3 COMBINAISON DE CLASSIFIEURS INDÉPENDANTS

La combinaison de classifieurs indépendant transforme l'ensemble d'apprentissage initial $Z = \{(x_i, y_i)_{1 \leq i \leq n}\, y_i \in \{-1, 1\}\}$ en plusieurs ensembles, disjoints ou non, sur lesquels plusieurs classifieurs indépendants sont entraînés. Une méthode de combinaison est ensuite appliquée pour produire la classification finale.

3.1 Bootstrapping

Le bootstrapping est une méthode d'échantillonnage statistique dans laquelle un certain nombre d'ensembles d'entraînement non disjoints sont définis en tirant aléatoirement et avec remise dans un ensemble de n données $Z = (z_1 \cdots z_n)$.. Après n tirages (pour n grand), la probabilité qu'un exemple n'ai pas encore été sélectionnée est

$$\left(1 - \frac{1}{n}\right) \approx e^{-1}$$

et chaque ensemble d'apprentissage contient donc $(1 - e^{-1}) \approx 63.2\%$ des exemples.
Le bootstrapping est utilisé pour inférer une statistique S sur une population $\mathcal{P}$, à partir d'un échantillon de n individus de $\mathcal{P}$. M tirages avec remise de n éléments sont effectués sur Z, permettant d'estimer M valeurs de S. Une moyenne permet ensuite d'obtenir l'estimation finale de la statistique S, et d'obtenir une estimation de la variance et des intervalles de confiance.

3.2 Bagging

Le bagging (bootstrap aggregation) utilise le bootstrapping pour réduire la variance et/ou pour améliorer la précision d'un prédicteur donné. Si $z_i = (x_i, y_i), 1 \leq i \leq n$, avec y_i la classe de x_i (classification) ou une valeur de $\mathbb{R}^d$ (régression), le bagging apprend un ensemble de M prédicteurs (obtenus par M bootstrapping sur Z), et calcule un prédicteur final en combinant les prédicteurs appris. Cette combinaison réduit la variance de l'estimation de la sortie, d'autant plus que M est grand.
A noter que d'autres stratégies de bagging existent, qui n'utilisent pas le bootstrapping :

- lorsque les ensembles d'entraînement sont construits sans remise, on parle de Pasting
- lorsque les ensemble d'entraînement sont construits à partir d'un sous-ensemble des composantes des vecteurs $x \in \mathbb{R}^d$, on parle de random subspaces

3.3 Wagging

Le wagging est une variante du bagging, dans laquelle chaque classifieur est entraîné sur Z, mais où chaque instance se voir affecter un poids aléatoire.

3.4 Forêts aléatoires

Les forêts aléatoires (algorithme 8) utilisent des arbres de décision comme classifieurs partiellement indépendants. L'algorithme entraîne chaque arbre de décision sur un échantillonnage de Z obtenu par bootstrapping selon un algorithme d'apprentissage sur les arbres (CART sans élagage par exemple), en limitant sa croissance par validation croisée. Chaque noeud de chaque arbre est choisi comme split optimal parmi k variables tirées aléatoirement dans les entrées, $k << |X|$.
On retrouve dans le choix de k un compromis biais-variance :

Algorithme 8 : Forêt aléatoire

Entrées :
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}$
$x \in X$
M le nombre d'arbres
k
Résultat : un classifieur h

pour $i \leftarrow 1$ **à** M **faire**
    Tirer avec remise dans Z un sous-échantillon Z_i (bootstrap)
    Construire un arbre CART $h_i(.)$ sur Z_i, chaque coupure est sélectionnée en minimisant la fonction de coût de CART sur un ensemble de k variables choisies au hasard
fin
$h(x) = \frac{1}{M} \sum_{i=1}^{M} h_i(x)$

— lorsque k diminue, la tendance est à se rapprocher d'un choix aléatoire des variables de découpe des arbres. Dans le cas où $k = 1$, les axes de la partition des arbres sont choisis au hasard, et seuls les points de coupure utiliseront l'échantillon. Ainsi, la corrélation entre les arbres a tendance à diminuer également, ce qui entraîne une baisse de la variance de l'estimateur agrégé. En revanche, choisir les axes de découpe des arbres de manière (presque) aléatoire se traduit par une moins bonne qualité d'ajustement des arbres sur l'échantillon d'apprentissage, d'où une augmentation du biais pour chaque arbre h_i ainsi que pour h.
— lorsque k augmente, les phénomènes inverses se produisent.

Le choix de k est donc lié aux choix des paramètres de l'arbre, notamment au choix du nombre d'observations dans ses noeuds terminaux. En pratique, le nombre maximum d'observations dans les noeuds est par défaut pris relativement petit (de l'ordre de 5 en régression, 1 en classification), ou alors optimisé par validation croisée.
A noter que cette procédure, lorsqu'elle est appliquée avec $k = n$, s'apparente au bagging.

3.5 Méthodes de combinaison

La combinaison de classifieurs indépendants peut se réaliser suivant deux principes : soit par pondération, soit par méta-apprentissage. La première classe de méthodes trouve son

intérêt lorsque les classifieurs réalisent la même tâche avec approximativement le même succès. La seconde est plus particulièrement adaptée lorsque certains classifieurs classifient systématiquement bien (ou mal) certains exemples.

3.5.1 Méthodes de pondération

Il existe de très nombreuses méthodes de pondération. Nous présentons dans la suite certaines d'entre elles, à partir desquelles d'autres peuvent facilement être dérivées.

Vote par majorité Un point x est affecté à la classe qui a reçu le plus grand nombre de votes de la part des M classifieurs.

Pondération par performance Le poids de chaque classifieur est proportionnl à sa précision sur un ensemble de validation :

$$\alpha_i = \frac{1 - A_i}{\sum_{j=1}^{M} (1 - A_j)}$$

où A_i est un indice de performance de h_i calculé sur l'ensemble de validation.

Sommation de distributions L'idée de cette méthode de pondération est de sommer les vecteurs de probabilités conditionnelles obtenues par chaque classifieur. Un point x est affecté à la classe de score le plus élevé dans le vecteur somme :

$$classe(x) = arg \max_{v \in domaine(y)} \sum_{i=1}^{M} \hat{P}_{h_i}(y = v|X)$$

Combinaison bayésienne Le poids de chaque classifieur est la probabilité a posteriori du classifieur, étant donné Z :

$$classe(x) = arg \max_{v \in domaine(y)} \sum_{i=1}^{M} P(h_i|Z).\hat{P}_{h_i}(y = v|X)$$

où $P(h_i|Z)$ est la probabilité que h_i soit correct, étant donné Z, et dont l'estimation dépend de la représentation de h_i.

3.5.2 Méthodes par méta apprentissage

Empilement L'objectif de cette méthode est d'augmenter la capacité de généralisation du classifieur fort h. Des méta données sont crées à partir des prédictions des classifieurs sur les données de Z . Un nouvel algorithme d'apprentissage est alors utilisé sur ces méta données, pour prédire quelles combinaisons des classifieurs faibles donne de bons résultats.

Apprentissage par grading Cette méthode transforme les classifications produites par les M classifieurs en M ensembles d'entrainement , en utilisant les exemples x_i M fois, avec un nouveau label binaire pour chaque occurrence, indiquant si le k-ième classifieur a correctement ou non classé x_i. Pour chaque classifieur, un méta-classifieur est entrainé, qui prédit quand le classifieur tend à mal classer un exemple donné. Le classifieur fort h est construit à partir des classifieurs faibles reconnus comme fiables par les méta classifieurs. Cette méthode s'apparente à une généralisation de la sélection par validation croisée.

4 Partie pratique

Le module sklearn.ensemble contient une série de méthodes d'ensemble pour la classification et la régression.
On considère un ensemble $Z = \left\{(x_i, y_i), 1 \leq i \leq n, x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\right\}$ de n individus, mesurés par un très grand nombre (d grand) de variables (ou caractéristiques). L'objectif de ce challenge est d'utiliser l'algorithme AdaBoost pour sélectionner les variables pertinentes, et ainsi diminuer la taille de l'ensemble d'apprentissage avant d'entraîner un classifieur sur ces données.

Chaque caractéristique est d'abord vue comme un classifieur faible. AdaBoost sélectionne alors les plus pertinentes d'entre elles, pour créer un nouvel ensemble d'apprentissage $Z' = \left\{(x_i, y_i), 1 \leq i \leq n, x_i \in \mathbb{R}^{d'}, y_i \in \{-1, 1\}\right\}$, avec $d' << d$. Ce nouvel ensemble d'apprentissage est alors fourni en entrée d'un classifieur qui réalise l'apprentissage et répond au problème posé à partir de Z.

On donne dans le fichier `CNAE.txt` un jeu de données de $n = 1080$ descriptions de compagnies brésiliennes [2]. Originellement composée de texte, chaque description a été prétraitée pour obtenir la description numérique fournie dans le jeu de données (une catérogie décrivant l'entreprise, codée entre 1 et 9, suivie de d=856 variables, représentant les fréquences dans le texte de chaque mot d'un dictionnaire). La fréquence étant codée en entier, la matrice1080×856 ainsi construite est très creuse (plus de 99% des coefficients sont à 0).
Appliquer la sélection de variable par AdaBoost pour construire un ensemble d'apprentissage de $n = 1080$ individus, représentés par $d' << d$ variables. Brancher en sortie un classifieur de votre choix (dans scikit-learn) pour réaliser la classification de ces textes et mesurer la performance de votre classification. Comparer à une classification réalisée sur l'ensemble d'apprentissage de départ, avec le même classifieur.

Références

[1] Ran Avnimelech and Nathan Intrator. Boosting regression estimators. *Neural Computation*, 11(2) :499–520, 1999.

[2] Patrick Marques Ciarelli and Elias Oliveira. Agglomeration and elimination of terms

for dimensionality reduction. In *ISDA*, pages 547–552. IEEE Computer Society, 2009.

[3] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression : a statistical view of boosting. *Annals of Statistics*, 28 :2000, 1998.

[5] Vanessa Gómez-Verdejo, Manuel Ortega-Moral, Jerónimo Arenas-García, and Aníbal R. Figueiras-Vidal. Boosting by weighting critical and erroneous samples. *Neurocomput.*, 69(7-9) :679–685, March 2006.

[6] Robert E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5 :197–227, 1990.