Machine learning with Python: boosting

October 14, 2017

Outline

Introduction

Le boosting : les données

Données d'apprentissage supervisé

- Features $X_i \in \mathbb{R}^d$
- Labels $Y_i \in \mathbb{R}$ (régression) $Y_i \in \{-1, 1\}$ (classification)

Weak learners

- Considérons un ensemble de "weak learners" $\mathcal{H}$
- Chaque learner $h : \mathbb{R}^d \to \mathbb{R}$ ou $\mathbb{R}^d \to \{-1, 1\}$ est un learner très simple
- Chaque learner simple est à peine meilleur que celui appris avec les Y_i (moyenne)

Exemples de weak learners

- Pour la régression : arbres de régression simple de faible profondeur, modèle linéaire à quelques variables
- Pour la classification : arbre de décision simple de faible profondeur, modèle logistique à quelques variables.

Principe du boosting

Un "strong learner"

On combine additivement des weak learners

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

avec $\eta^{(b)} \geq 0$ pour espérer en obtenir un meilleur. $g^{(b)}$ est un learner dans le sens où on $\hat{Y}_i = g^{(b)}(X_i)$.

- Chaque $b = 1, \ldots, B$ est un pas/itération de boosting
- Le boosting fait partie des **ensemble methods** puisqu'il combine des learners faibles pour en fabriquer un meilleur.

Pour aller plus loin : voir Schapire 1999 et Friedman 2001

Principe du "gradient boosting"

On cherche donc des fonctions $h^{(b)}$ du dictionnaire $\mathcal{H}$ et des réels $\eta^{(b)}$ tels que

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

minimise le risque empirique

$$\sum_{i=1}^{n} L(Y_i, g^{(b)}(X_i)),$$

où L est la fonction de coût (de perte) que l'on se fixe suivant le problème

- en régression linéaire $L(y, u) = (1/2)(y - u)^2$
- plus généralement, on peut prendre L comme la log-densité dans le modèle considéré
- on verra qu'il y a d'autres choix possible en classification.

L'algorithme "greedy"

Si c'était possible, on pourrait définir

$$\hat{g}^{(B)} = \sum_{b=1}^{B} \hat{\eta}^{(b)} \hat{h}^{(b)}(x)$$

avec

$$(\hat{\eta}^{(1)}, \ldots, \hat{\eta}^{(b)}, \hat{h}^{(1)}, \ldots, \hat{h}^{(b)}) = \underset{\eta^{(1)}, \ldots, \eta^{(b)} \in \mathbb{R}, h^{(1)}, \ldots, h^{(b)} \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} L(Y_i, \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x))$$

mais même à $\eta^{(1)}, \ldots, \eta^{(b)}$ fixés il faut chercher parmi $\#(\mathcal{H})^B$ solutions possibles.

Gradient-boosting

Algorithme "greedy-stagewise"

On procède en fait pas-à-pas en définissant une suite $\hat{g}^{(b)}$ avec pour tout $b = 1, \ldots, B$ avec

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)} \hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta \in \mathbb{R}, h \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h)$$

Dans nos problèmes, cette minimisation est également un problème difficile, on va alors procéder

- par descente de gradient
- avec la contrainte que la direction du pas de descente soit une fonction de $\mathcal{H}$.

Rappel

Descente de gradient

Considérons le problème de la minimisation de la fonction convexe et différentiable $J : \mathbb{R}^p \to \mathbb{R}$, la suite d'itérés $\theta^{(b)}$ définis par

$$\theta^{(b+1)} = \theta^{(b)} - \gamma \nabla J(\theta^{(b)})$$

alors

$$J(\theta^{(b+1)}) \leq J(\theta^{(b)})$$

(sous des conditions sur J et γ - cf cours d'optimisation du second semestre).

Descente non-contrainte I

A l'itération b,

- nous avons le learner $\hat{g}^{(b)} = \hat{g}^{(b)} + \eta\, \vec{0}$,
- on cherche à faire un pas de descente de gradient (pour l'instant quelconque) pour la fonction à optimiser $u \mapsto \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u(X_i))$.

Attention : u est une fonction de $\mathbb{R}^p \to \mathbb{R}$.

Mais on ne s'intéresse qu'à ses valeurs aux points $X_1, \ldots, X_n$, on l'identifie donc à vecteur de $\mathbb{R}^n$

$$\left(\nabla_{u_i} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i)\right)_{u(X_i)} = \eta \left(\nabla_{u_i} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i)\right)_{u(X_i)}$$

Descente non-contrainte à rester dans $\mathcal{H}$

Le pas de gradient à considérer est donc dans la direction

$$\delta^{(b+1)} = \begin{pmatrix} \left(\nabla_{u_1} L(Y_1, \hat{g}^{(b)}(X_1) + \eta u_1) \right)_0 \\ \dots \\ \left(\nabla_{u_n} L(Y_i, \hat{g}^{(b)}(X_n) + \eta u_n) \right)_0 \end{pmatrix}$$

Exemple dans le modèle linéaire

Dans le modèle linéaire, on prend

- $L(y, v+u) = (1/2)(y-v-u)^2$ avec
- $\nabla_u L(y, v+u) = -(y-v-u)$.

Le pas de gradient est donc dans la direction

$$\delta_{lm}^{(b+1)} = \begin{pmatrix} \left(\nabla_{u_1} L(Y_1, \hat{g}^{(b)}(X_1) + \eta u_1)\right)_0 \\ \dots \\ \left(\nabla_{u_n} L(Y_n, \hat{g}^{(b)}(X_n) + \eta u_n)\right)_0 \end{pmatrix} = \begin{pmatrix} -(Y_1 - \hat{g}^{(b)}(X_1)) \\ \dots \\ -(Y_n - \hat{g}^{(b)}(X_n)). \end{pmatrix}$$

Retour au problème contraint

Le problème d'optimisation au départ est

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)}\hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta\in\mathbb{R}, h\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h)$$

il faudrait donc que $\delta^{(b+1)}$ soit dans $\mathcal{H}$, ce qui ne peut pas être assuré à cette étape.

Pour s'assurer de rester dans le dictionnaire $\mathcal{H}$, on va prendre la fonction de $\mathcal{H}$ la plus proche de $\delta^{(b+1)}$ au sens des moindres carrés (de la norme ℓ_2 aux points d'observation):

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h\in\mathcal{H}, \nu\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\nu h(X_i) - \delta^{(b+1)}(i)\right)^2.$$

La contrainte dans le modèle linéaire

Dans le modèle linéaire, cela s'écrit

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h \in \mathcal{H}, \nu \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \left(\nu\{-(Y_i - \hat{g}^{(b)}(X_i))\} - \delta^{(b+1)}(i) \right)^2.$$

On essaie donc d'apprendre un modèle (faible) sur les résidus $(-(Y_i - \hat{g}^{(b)}(X_i)))$ du modèle précédent : aux points où $g^{(b)}$ n'est pas très performant (grand résidus).

Algorithme du gradient boosting

On optimise enfin en η pour obtenir l'algorithme suivant.

Algorithm 1 Gradient boosting

1: Posons $\hat{g}^{(0)} = \hat{a}$ avec $\hat{a} = \text{argmin}_{a \in \mathbb{R}} \sum_{i=1}^{n} L(Y_i, a)$
2: **for** $b = 1, \ldots, B$ **do**
3: $\quad \delta^{(b+1)} \leftarrow -\Big(\nabla_u L(Y_i, \hat{g}^{(b)}(X_i) + \eta u)\Big)_0 \; i = 1, \ldots, n$
4: $\quad \hat{h}^{(b+1)} \leftarrow \hat{h}$ avec $(\hat{h}, \hat{\nu}) = \text{argmin}_{h \in \mathcal{H}, \nu \in \mathbb{R}} \sum_{i=1}^{n} \left(\nu h(X_i) - \delta^{(b+1)}(i)\right)^2$.
5: $\quad \hat{\eta}^{(b+1)} \leftarrow \text{argmin}_{\eta \in \mathbb{R}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h^{(b+1)})$
6: **end for**
7: **return** Un boosting learner $g^{(B)}(x) = \left(\sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)\right)$

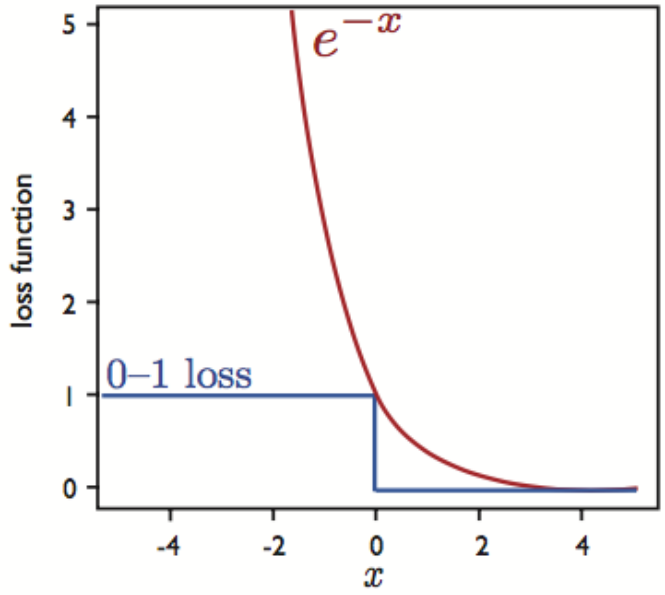
AdaBoost

Pour la classification

L'algorithme de boosting le plus connu en classification est AdaBoost (ADAptive BOOSTing). Il optimise la perte exponentielle

$$L(y, u) = \exp(-yu)$$

qui est une approximation de la perte 0/1 $\mathbf{1}_{y \neq u}$.



On va montrer qu'on peut aussi le voir comme un algorithme qui pondère séquentiellement les observations mal-classées

Rappels et risque pondéré

La perte 0/1 moyenne (ou l'"accuracy") d'un classifieur h est donnée par

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{Y_i \neq h(X_i)} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{Y_i h(X_i)<0}$$

Pour pouvoir donner plus de poids à certaines observations, définissons un risque pondéré

$$\sum_{i=1}^{n} p^{(b)}(i)\mathbf{1}_{Y_i h(X_i)<0},$$

où $\sum_{i=1}^{n} p^{(b)}(i) = 1$ et b est le numéro de l'itération boosting courante.

Initialisation

Au départ, on ne sait pas quelles observations sont difficiles à classer, on commence donc avec $p^{(0)}(i) = \frac{1}{n}$ pour $i = 1, \ldots, n$, et on choisit

$$g^{(0)} = h^{(0)} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} p^{(0)}(i) \mathbf{1}_{Y_i h(X_i) < 0}$$

on calcule son erreur moyenne

$$\varepsilon(0) = \sum_{i=1}^{n} p^{(0)}(i) \mathbf{1}_{Y_i h^{(0)}(X_i) < 0}$$

Comment changer les poids ?

On voudrait

- à l'itération suivante donner plus de poids aux mal-classés : $p^{(1)}(i)$ doit être grand si $Y_i h^{(0)}(X_i) < 0$
- pouvoir calculer simplement $p^{(b+1)}$ à partir des calculs de l'itération b, et on a
$$g^{(b+1)}(x) = g(b)(x) + \eta^{(b+1)} h^{(b+1)}(x).$$
- donc une bonne idée est de prendre
$$p^{(1)}(i) = \frac{e^{-\eta^{(0)} Y_i h^{(0)}(X_i)}}{\text{cst}},$$
car $e^{x+y} = e^x e^y$.

A l'itération b

On pose alors

$$p^{(b+1)}(i) = p^{(b)}(i) \frac{e^{-\eta^{(b)} Y_i h^{(b)}(X_i)}}{Z^{(b)}},$$

de telle sorte que

$$p^{(b+1)}(i) = \frac{e^{-Y_i \sum_{a=1}^{b} \eta^{(a)} h^{(a)}(X_i)}}{n \prod_{a=1}^{b} Z^{(a)}} = \frac{e^{-Y_i g^{(b)}(X_i)}}{n \prod_{a=1}^{b} Z^{(a)}}$$

Les poids (1)

Comme on veut $\sum_{i=1}^{n} p^{(b)}(i) = 1$, il faut poser

$$\frac{1}{n}\sum_{i=1}^{n} e^{-Y_i g^{(b)}(X_i)} = \sum_{i=1}^{n} p^{(b)}(i) \prod_{a=1}^{b} Z^{(a)} = \prod_{a=1}^{b} Z^{(a)}$$

et

$$\begin{aligned} Z^{(b)} &= \sum_{i=1}^{n} p^{(b)}(i) e^{-\eta^{(b)} Y_i h^{(b)}(X_i)} \\ &= \sum_{i:\, Y_i h^{(b)}(X_i)=1} p^{(b)}(i) e^{-\eta^{(b)}} + \sum_{i:\, Y_i h^{(b)}(X_i)=-1} p^{(b)}(i) e^{\eta^{(b)}} \end{aligned}$$

Les poids (1)

Comme

$$\varepsilon(b) = \sum_{i=1}^{n} p^{(b)}(i) \mathbf{1}_{Y_i h^{(B)}(X_i) < 0}$$

on obtient

$$\begin{aligned} Z^{(b)} &= \sum_{i: Y_i h^{(b)}(X_i) = 1} p^{(b)}(i) e^{-\eta^{(b)}} + \sum_{i: Y_i h^{(b)}(X_i) = -1} p^{(b)}(i) e^{\eta^{(b)}} \\ &= (1 - \varepsilon^{(b)}) e^{-\eta^{(b)}} + \varepsilon^{(b)} e^{\eta^{(b)}} \end{aligned}$$

$$Z^{(b)} = (1 - \varepsilon^{(b)}) \sqrt{\frac{\varepsilon^{(b)}}{1 - \varepsilon^{(b)}}} + \varepsilon^{(b)} \sqrt{\frac{1 - \varepsilon^{(b)}}{\varepsilon^{(b)}}} = 2\sqrt{\varepsilon^{(b)}(1 - \varepsilon^{(b)})}.$$

Les poids (2)

On met le poids aux obsservations mal-classées d'une part et bien classées d'autre, on prend donc $\eta^{(b)}$ tel que

$$(1 - \varepsilon^{(b)})e^{-\eta^{(b)}} = \varepsilon^{(b)} e^{\eta^{(b)}},$$

ou bien

$$\eta^{(b)} = \frac{1}{n} \log\left(\frac{1 - \varepsilon^{(b)}}{\varepsilon^{(b)}}\right)$$

Algorithm 2 Adaboost

Posons $p_1(i) = 1/n$ pour $i = 1, \ldots, n$
for $b = 1, \ldots B$ **do**
    $h^{(b)} \in \text{argmin}_{h \in H} \sum_{i=1}^{n} p^{(b)}(i) \mathbf{1}_{Y_i h(X_i) < 0}$
    $\varepsilon^{(b)} \leftarrow \sum_{i=1}^{n} p^{(b)}(i) \mathbf{1}_{Y_i h^{(b)}(X_i) < 0}$
    $\eta^{(b)} \leftarrow \frac{1}{2} \log\left(\frac{1-\varepsilon^{(b)}}{\varepsilon^{(b)}}\right)$
    $Z^{(b)} \leftarrow 2\sqrt{\varepsilon^{(b)}(1-\varepsilon^{(b)})}$
    $p^{(b+1)}(i) \leftarrow p^{(b)}(i) \dfrac{e^{-\eta^{(b)} Y_i h^{(b)}(X_i)}}{Z^{(b)}}$
end for
return Un boosting classifieur $g^{(B)}(x) = \text{signe}\Big(\sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)\Big)$

En pratique

Il reste à fixer les hyper-paramètres

- pour le dictionnaire $\mathcal{H}$:
  - si on choisit des arbres, il reste à fixer leurs profondeurs
  - si on choisir des glm, il reste à fixer le nombre de variables qu'ils contiennent
- et B le nombre d'itérations

Régularisation

Pour rendre la performance de l'algorithme moins dépendant du choix de B, on régularise en ajoutant le paramètre

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \lambda \hat{\eta}^{(b+1)} \hat{h}^{(b+1)}$$

que l'on choisit à la fin par cross-validation.

Voir le GitHub d'XGBoost pour plus de détails sur les paramètres
https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

References I

Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

Robert E Schapire. "A brief introduction to boosting". In: *Ijcai*. Vol. 99. 1999, pp. 1401–1406.