Gradient Boosting

Fabrice Rossi

L'objectif de ce devoir est de programmer en `R` l'algorithme de *gradient boosting* dans sa version la plus complète (avec *stochastic gradient boosting* avec *shrinkage* sur des fonctions de perte variées). Le sujet demande explicitement de faire des tests lors des premières questions. Il faut bien entendu tester le reste des développements demandés, même quand cela n'est pas indiqué directement.

1 Rappels sur le *gradient boosting*

On suppose donné un ensemble d'apprentissage $\mathcal{D} = ((\mathbf{X}_i,Y_i))_{1\leq i\leq N}$ et une fonction de perte l. La fonction de perte est supposée dérivable par rapport à la prédiction et on note l' la dérivée correspondante. Par exemple si

$$l(p,v) = l_2(p,v) = (p-v)^2,$$

on a

$$l'(p,v) = l_2'(p,v) = 2(p-v).$$

L'algorithme 1.1 est la version de base *Gradient boosting*.

Algorithm 1.1 Gradient boosting

fixer g_0 au meilleur modèle constant $g_0(\mathbf{x}) = \arg\min_{\mathbf{y}} \sum_i l(\mathbf{y},Y_i)$
for $k = 1$ to K **do**
  calculer $r_{k,i} = -l'(g_{k-1}(\mathbf{X}_i),Y_i)$
  apprendre le modèle f_k sur l'ensemble $(\mathbf{X}_i,r_{k,i})_{1\leq i\leq N}$
  calculer

$$\rho_k = \arg\min_{\rho} \sum_i l(g_{k-1}(\mathbf{X}_i) + \rho f_k(\mathbf{X}_i),Y_i)$$

  calculer le nouveau modèle $g_k = g_{k-1} + \rho_k f_k$
end for
modèle final g_K

L'objectif du devoir est de programmer en `R` une version du *gradient boosting* s'appuyant sur des arbres de décision simples pour apprendre les f_k.

2 Version simple

On commence par programmer une version de l'algorithme limitée au cas de la perte quadratique l_2.

2.1 Fonctions liées à la fonction de perte

Pour faciliter l'extension du programme à d'autres fonctions de perte, on n'intègre pas directement la perte dans le code du *gradient boosting*.

Question 1 Écrire une fonction `qloss` qui calcule $l_2(p,v)$ pour p et v deux vecteurs.

Question 2 Écrire une fonction `qloss_d` qui calcule $l_2'(p,v)$ pour p et v deux vecteurs.

Question 3 Écrire une fonction `qloss_best_cst` qui calcule $\arg\min_{\mathbf{y}} \sum_i l_2(\mathbf{y},Y_i)$ pour un vecteur $(Y_i)_{1\leq i\leq N}$.

2.2 Représentation du modèle

Pour représenter le modèle construit par l'algorithme, on utilisera une liste contenant les éléments suivants :

- `cst` : la constante g_0 ;
- `rho` : le vecteur des coefficients ρ_k calculés lors de l'apprentissage ;
- `f` : la liste des modèles f_k. Chaque modèle est supposé utilisable avec la fonction `predict` de `R`.

La liste représente ainsi le modèle donné par

$$g(\mathbf{X}) = g_0 + \sum_{k=1}^{K} \rho_k f_k(\mathbf{X}).$$

2.2.1 *Bagging*

Pour tester cette structure, on l'utilise pour stocker un modèle de *bagging*. On rappelle que le *bagging* consiste simplement à entraîner K modèles sur des échantillons échantillons *bootstrap* des données. Dans le modèle ci-dessus, on aura donc $g_0 = 0$ et $\rho_k = \frac{1}{K}$ pour tout k.

Question 4 Écrire une fonction `bagging` qui prend comme paramètre une formule h, une data frame `df` et un entier K renvoie une liste au format ci-dessus obtenu par *bagging* de K arbres entraînés sur des échantillons bootstrap de `df` pour la tâche décrite par la formule h.

Question 5 Tester brièvement la fonction sur le jeu de données `BostonHousing` de la bibliothèque `mlbench` : l'objectif est de prédire la variable `medv`.

2.2.2 Prédiction

Question 6 Écrire une fonction `predict_ens` qui prend comme paramètre une liste au format décrit ci-dessus et une data frame, et qui renvoie les prédictions du modèle sur la data frame.

Question 7 Poursuivre les tests sur le jeu de données `BostonHousing`. On pourra par exemple étudier l'évolution des performances en fonction du nombre d'arbres.

2.3 Minimisation unidimensionnelle

Pour déterminer ρ_k, on doit minimiser la fonction m_k définie par

$$m_k(\rho) = \sum_i l(g_{k-1}(\mathbf{X}_i) + \rho f_k(\mathbf{X}_i),Y_i).$$

La fonction `optimise` permet de faire ce type d'optimisation.

Question 8 Écrire une fonction `mrisk` qui prend en paramètre un réel ρ et trois vecteurs, g, f et y, et renvoie en résultat la valeur de

$$\sum_i l_2(g_i + \rho f_i, y_i).$$

Question 9 Tester l'utilisation de `optimise` pour trouver le minimum de `mrisk` par rapport à ρ pour différentes valeurs des vecteurs g, f et y.

2.4 *Gradient boosting*

En combinant les différentes briques construites ci-dessus, on peut construite l'algorithme du *gradient boosting*, en se restreignant au cas de la fonction de perte l_2.

Question 10 Écrire une fonction `gradient_boosting` prenant en paramètre une formule h, une data frame `df` et un entier K, et renvoyant en résultat un modèle sous la forme décrite plus haut.

Pour accéder à la variable cible de la data frame, on pourra utiliser la formule

```
eval(formula(h)[[2]], df)
```

Pour obtenir le nom de la variable cible, on pourra utiliser la formule

```
as.character(formula(h)[[2]], df)
```

Question 11 Tester la fonction sur les données `BostonHousing`, en comparant notamment au bagging.

Question 12 Ajouter à la fonction `predict_ens` un paramètre optionnel permettant d'arrêter le calcul de l'ensemble de modèles avant sa taille complète K, c'est-à-dire de calculer $g_m(\mathbf{X}) = g_0 + \sum_{k=1}^{m} \rho_k f_k(\mathbf{X})$ pour un m quelconque inférieur à K.

3 Améliorations simples

3.1 Bagging

Question 13 Modifier la fonction `bagging` pour qu'elle puisse prendre en paramètre de façon optionnelle une liste crée par `rpart.control` afin de contrôler la construction des arbres utilisés dans l'algorithme.

Question 14 Mettre en place un paramétrage par défaut adapté quand le nouveau paramètre n'est pas fourni.

Question 15 Inclure dans la fonction `bagging` le calcul optionnel (selon un paramètre) des prévisions *out-of-bag* pour les données d'apprentissage.

3.2 *Gradient boosting*

Deux stratégies ont été proposées par Friedman pour améliorer le *gradient boosting* :

1. le *shrinkage* consiste à remplacer le ρ_k optimal par $\nu\rho_k$ où $\nu \in]0;1]$ est un méta-paramètre de l'algorithme (Friedman recommande de prendre $\nu = 0.1$) ;

2. le *stochastic gradient boosting* consiste à entraîner f_k sur un sous-ensemble aléatoire des données (tirage sans remise). Le taux de sélection (le ratio entre la taille du sous-ensemble de N) est un autre méta-paramètre de l'algorithme. Friedman recommande des taux compris entre 0.5 et 0.8.

Question 16 Intégrer les deux mécanismes dans la fonction `gradient_boosting`.

Question 17 Inclure dans la fonction le calcul optionnel des prévisions *out-of-bag* pour les données d'apprentissage quand le *stochastic gradient boosting* est utilisé.

On remarque d'autre part que le problème d'optimisation de $m_k(\rho)$ peut se résoudre de façon analytique dans le cas spécifique de la fonction de perte l_2.

Question 18 Déterminer la solution de $\arg\min_\rho \sum_i l_2(g_i+\rho f_i,y_i)$ en utilisant l'équation d'Euler.

Question 19 Écrire une fonction `qloss_solve` qui prend comme paramètres les trois vecteurs g, f et y, et renvoie la solution de $\arg\min_\rho \sum_i l_2(g_i+\rho f_i,y_i)$.

Question 20 Modifier `gradient_boosting` pour qu'elle utilise `qloss_solve`.

4 Généralisation à d'autres fonctions de perte

4.1 Représentation des fonctions de perte

Un des intérêts principaux du *gradient boosting* est la possibilité d'utiliser des fonctions de perte arbitraires. Informatiquement, il faut pouvoir décrire la fonction de perte, sa dérivée, le calcul de la meilleure prévision par une constante et la résolution de $\arg\min_\rho \sum_i l(g_i+\rho f_i,y_i)$. En `R`, on peut regrouper ces informations dans une liste contenant les éléments suivants :

— `loss` : fonction de calcul de la perte, sur le modèle de `qloss` ;
— `deriv` : fonction de calcul de la dérivée de la perte, sur le modèle de `qloss_d` ;
— `best_cst` : fonction de calcul de $\arg\min_{\mathbf{y}} \sum_i l(\mathbf{y},Y_i)$, sur le modèle de `qloss_best_cst` ;
— `solve` : fonction de calcul de $\arg\min_\rho \sum_i l_2(g_i+\rho f_i,y_i)$, sur le modèle de `qloss_solve`.

Question 21 Écrire une fonction `quadratic_loss` qui renvoie une liste de la forme ci-dessus pour le cas de la perte quadratique l_2.

Question 22 Modifier la fonction `gradient_boosting` de manière à ce qu'elle prenne un nouveau paramètre décrivant la fonction de perte qui sera utilisé à la place des fonctions `qloss`, `qloss_d`, `qloss_best_cst` et `qloss_solve`.

4.2 Autres fonctions de perte

Dans les questions suivantes, on tentera dans la mesure du possible de proposer un calcul efficace de $\arg\min_\rho \sum_i l_2(g_i+\rho f_i,y_i)$. Quand cela n'est pas possible, on utilisera `optimise`.

Question 23 Écrire une fonction `absolute_loss` qui renvoie une liste de la forme ci-dessus pour le cas de la perte quadratique $l_1(p,v)=|p-v|$.

Question 24 Écrire une fonction `huber_loss` qui renvoie une liste de la forme ci-dessus pour le cas de la perte de Huber. La perte de Huber est définie par

$$l_{h,\delta}(p,v)=\begin{cases}\frac{1}{2}(p-v)^2 & \text{if } |p-v|\geq\delta,\\ \delta\left(|p-v|-\frac{\delta}{2}\right) & \text{if } |p-v|>\delta.\end{cases}$$

La perte dépendant d'un paramètre δ, celui-ci devra être fourni à la fonction `huber_loss`.

Question 25 Écrire des fonctions adaptées pour les pertes de substitution utilisées en classification supervisée comme la perte logistique, la perte exponentielle, etc.