Gradient Boosting

Technique ensembliste pour l'analyse prédictive

Introduction explicite d'une fonction de coût

Ricco RAKOTOMALALA

Université Lumière Lyon 2

PLAN

1. Préambule
2. Gradient boosting en régression
3. Gradient boosting en classement
4. Régularisation (shrinkage, stochastic gradient boosting)
5. Pratique du gradient boosting – Logiciels et outils
6. Bilan – Avantages et inconvénients
7. Références bibliographiques

Préambule

Boosting et Descente du gradient

Boosting

Le BOOSTING est une technique ensembliste qui consiste à agréger des classifieurs (modèles) élaborés séquentiellement sur un échantillon d'apprentissage dont les poids des individus sont corrigés au fur et à mesure. Les classifieurs sont pondérés selon leurs performances [RAK, page 28].

Entrée : B nombre de modèles, ALGO algorithme d'apprentissage, Ω un ensemble de données de taille n avec y cible binaire {-1, +1}, X avec p prédicteurs.

MODELES = {}
Les individus sont uniformément pondérés $\omega^1_i = 1/n$
Pour b = 1 à B Faire
  Construire un modèle M_b sur $\Omega(\omega^b)$ avec ALGO (ω^b pondération à l'étape b)
  Ajouter M_b dans MODELES
  Calculer le taux d'erreur pondéré pour M_b : $\varepsilon_b = \sum_{i=1}^{n} \omega_i^b \times I(y_i \neq \hat{y}_i)$
  Si $\varepsilon_b > 0.5$ ou $\varepsilon_b = 0$, arrêt de l'algorithme
  Sinon
    Calculer $\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b}$
    Les poids sont remis à jour $\omega_i^{b+1} = \omega_i^b \times \exp[\alpha_b . I(y_i \neq \hat{y}_i)]$
    Et normalisés pour que la somme fasse 1
Fin Pour

Un vote pondéré (α_b) sur les décisions des classifieurs est utilisé lors de la prédiction (on a un modèle additif)

$$f(x) = sign \sum_{b=1}^{B} \alpha_b \times M_b(x)$$

Descente du gradient

La descente du gradient est une technique itérative qui permet d'approcher la solution d'un problème d'optimisation. En apprentissage supervisé, la construction du modèle revient souvent à déterminer les paramètres (du modèle) qui permettent d'optimiser (max ou min) une fonction objectif (ex. Réseaux de neurones – Perceptron – Critère des moindres carrés, pages 11 et 12).

$$J(y, f) = \sum_{i=1}^{n} j(y_i, f(x_i))$$

f() est le classifieur, paramétré

j() est une fonction de coût confrontant la valeur observée de la cible et la prédiction du modèle pour une observation

J() est une fonction de perte globale, calculée additivement sur l'ensemble des observations

➔ L'objectif est de minimiser J() au regard de f() c.-à-d. des paramètres de f()

$$f_b(x_i) = f_{b-1}(x_i) - \eta \times \nabla j(y_i, f(x_i))$$

$f_b()$ est le classifieur à l'étape ''b''

η est la constante d'apprentissage qui permet de piloter le processus jusqu'à la convergence

∇ est le gradient c.-à-d. la dérivée partielle première de la fonction de coût par rapport au modèle

$$\nabla j(y_i, f(x_i)) = \frac{\partial j(y_i, f(x_i))}{\partial f(x_i)}$$

Boosting = Descente du gradient

On montre que le processus ADABOOST consiste à optimiser une fonction de coût exponentielle c.-à-d. chaque modèle M_t sur la base des individus pondérés à l'issue de M_{t-1} permet de minimiser une fonction globale particulière [BIS, page 659 ; HAS, page 343]

$$J(f) = \sum_{i=1}^{n} \exp\left(-y_i \times f(x_i)\right)$$

$y \in \{-1, +1\}$

J() est la fonction de coût à minimiser

f() est le classifieur agrégé composé à partir d'une combinaison linéaire de classifieurs individuels M_b

$$f_b = f_{b-1} + \frac{\alpha_b}{2} \times M_b$$

Le modèle agrégé à l'étape ''b'' est corrigé avec le classifieur individuel M_b élaboré sur l'échantillon repondéré (ω). M_b joue le rôle de gradient ici c.-à.d. chaque modèle intermédiaire construit permet de réduire le coût du modèle agrégé global.

$$\omega_i^b = \omega_i^{b-1} \times \exp\left[\alpha_{b-1}.I\left(y_i \neq M_{b-1}(i)\right)\right]$$

Le modèle ''gradient'' est issu d'un échantillon où les poids des individus dépendent des performances du modèle précédent (l'idée de corrections itératives est bien présente)

GRADIENT BOOSTING : généraliser l'approche avec d'autres fonctions de coûts

Gradient Boosting en Régression

Gradient Boosting = Descente du gradient + Boosting

Problème de régression

La régression s'inscrit dans l'analyse prédictive : y est la variable cible, quantitative ; X est un ensemble de variables explicatives quelconques

$$y_i = M_1(x_i) + \varepsilon_{1i}$$

ε résume les insuffisances du modèle (x non pertinents, forme de la fonction M non adaptée)

M peut être tout type de méthode, nous privilégierons les arbres de régression

$$e_{i1} = y_i - M_1(x_i)$$

e correspondent aux résidus, valeurs observées des insuffisances du modèle. Plus la valeur est élevée (en valeur absolue), plus le point pose problème, il faut le traiter

L'idée est de modéliser ce résidu avec un second modèle M2 et de l'associer au précédent pour une meilleure prédiction.

$$e_{i1} = M_2(x_i) + \varepsilon_{2i}$$

On peut continuer ainsi sur le résidu e_2, etc.

$$\hat{y}_i = M_1(x_i) + M_2(x_i)$$

Le rôle de M_2 est de compenser (additivement) les insuffisances de M_1, puis on pourra avoir M_3, etc.

Fonction de coût global

Relation avec la descente du gradient

La somme des carrés des erreurs est un indicateur global de qualité des modèles privilégié en régression

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

$$J(y, f) = \sum_{i=1}^{n} j(y_i, f(x_i))$$

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial\left[\frac{1}{2}(y_i - f(x_i))^2\right]}{\partial f(x_i)} = f(x_i) - y_i$$

Calcul du gradient. Il correspond effectivement au résidu mais avec un signe opposé c.-à-d. résidu = gradient négatif

$$\begin{aligned} f_b(x_i) &= f_{b-1}(x_i) + M_b(x_i) \\ &= f_{b-1}(x_i) + (y_i - f_{b-1}(x_i)) \\ &= f_{b-1}(x_i) - 1 \times \frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} \\ &= f_{b-1}(x_i) - \eta \times \nabla j(y_i, f(x_i)) \end{aligned}$$

De fait, si l'on revient sur le modèle additif, on a bien un processus itératif. La modélisation des résidus à l'étape ''b'' (arbre de régression M_b) correspond à un gradient. Au final, nous minimisons la fonction de coût global J()

La constante d'apprentissage η est égale à 1 ici.

Algorithme Gradient Boosting

Pour la régression

Nous avons un processus itératif où, à chaque étape, nous utilisons la valeur négative du gradient : $-\nabla j(y,f)$ [WIK]

Ou, plus simplement, POUR m = 1, …, B (B : paramètre de l'algorithme)

Un arbre trivial réduit à la racine c.-à-d. prédiction = moyenne de y

Doit être calculé pour l'ensemble des individus de la base (i = 1, …, n)
Pour j() = carré de l'erreur ➔ gradient négatif = résidu

```
Créer le modèle initial f0()
REPETER JUSQU'À CONVERGENCE
  Calculer le gradient négatif -∇j(y,f)
  Construire un arbre de régression Mb sur -∇j(y,f)
  fb = fb-1 + γb.Mb
```

La profondeur de l'arbre est un paramètre possible de l'algorithme

Les modèles vont s'empiler additivement

γ_b est choisi à chaque étape de manière à minimiser la quantité
(via une optimisation numérique simple)

$$\gamma_b = \arg\min_{\gamma} \sum_{i=1}^{n} j(y_i, f_{b-1}(x_i) + \gamma . M_b(x_i))$$

L'énorme avantage de cette formulation générique est que l'on peut utiliser d'autres fonctions de coûts et les gradients associés.

Gradient Boosting

Autres fonctions de coûts

Autres fonctions de coût

➔ Autres formulations du gradient

➔ Autres comportements de l'algorithme d'apprentissage

Fonction de coût	$-\nabla j(y_i, f(x_i))$	Intérêt / inconvénient
$\frac{1}{2}(y_i - f(x_i))^2$	$y_i - f(x_i)$	Sensibilité aux écarts, mais peu robuste par rapport aux points aberrants
$\lvert y_i - f(x_i) \rvert$	$signe[y_i - f(x_i)]$	Moins sensible aux écarts mais meilleure robustesse par rapport aux points aberrants
Huber	$y_i - f(x_i)$ si $\lvert y_i - f(x_i) \rvert \leq \delta_b$ $\delta_b . signe[yi - f(xi)]$ si $\lvert y_i - f(x_i) \rvert > \delta_b$ Où δ_b est un quantile de $\{\lvert y_i - f(x_i) \rvert\}$	Mixer les avantages de l'erreur carrée (plus sensible pour valeurs faibles du gradient) et de l'erreur absolue (plus robuste pour les valeurs élevées du gradient)

Gradient Boosting en Classement

Travailler sur les indicatrices des classes

Fonction de coût et gradient pour le classement

La variable cible catégorielle Y est à K modalités {1,…,K}
L'algorithme reste globalement identique mais : il faut définir une fonction de coût adaptée au classement, et en dériver le gradient.

y^k est une variable indicatrice (K variables indicatrices en tout)

$$y_i^k = \begin{cases} 1 \text{ si } Y_i = k \\ 0 \text{ sinon} \end{cases}$$

$$\pi^k(x_i) = \frac{e^{f^k(x_i)}}{\sum_{k=1}^{K} e^{f^k(x_i)}}$$

π_k correspond à la probabilité conditionnelle d'appartenance à la classe ''k'' (modalité k de Y)

$$j(y_i, f(x_i)) = -\sum_{k=1}^{K} y_i^k \times \log \pi^k(x_i)$$

Fonction de coût : DEVIANCE MULTINOMIALE (cf. Cours de Régression Logistique Polytomique, pages 4 et 6).

$$\nabla j(y_i, f(x_i)) = y_i^k - \pi^k(x_i)$$

Gradient

Pour la classe ''k'', le gradient correspond à l'écart entre l'indicatrice correspondante et la probabilité d'appartenance à cette classe

On doit travailler sur des indicatrices (y^k), et construire des arbres de régression sur les gradients négatifs (1 arbre par classe). f^k est le modèle additif issu de ces arbres successifs, nécessaire pour calculer les π^k

Algorithme pour le classement

Le processus n'est pas modifié. Le schéma global reste le même, sauf qu'il faudra passer par des indicatrices.

A partir de Y est généré K variables indicatrices y^k

Créer les K modèles initiaux $f^k_0()$ pour chaque indicatrice

REPETER JUSQU'À CONVERGENCE

  Calculer les K gradients négatifs $-\nabla j(y^k, f^k)$

  Construire un arbre de régression M^k_b sur chaque $-\nabla j(y^k, f^k)$

  $f^k_b = f^k_{b-1} + \gamma_b . M^k_b$

On obtient à la sortie K modèles additifs f^k. La probabilité d'appartenance est obtenue avec

$$\pi^k(x_i) = \frac{e^{f^k(x_i)}}{\sum_{k=1}^{K} e^{f^k(x_i)}}$$

Même si on est dans le cadre du classement, le mécanisme interne repose toujours sur un arbre de régression. On parle de GRADIENT TREE BOOSTING.

La règle d'affectation est immuable : $\hat{Y}_i = \arg\max_k \pi^k(x_i)$

Régularisation

Pistes pour contrer la sur-dépendance à l'échantillon d'apprentissage

(sur-apprentissage)

Autres que la limitation de la taille de l'arbre

Shrinkage

Introduire un terme de régularisation dans la mise à jour des modèles

$$f_b = f_{b-1} + \nu.\gamma_b.M_b$$

On réintroduit un paramètre ($0 < \nu \leq 1$) pour « lisser » les mises à jour. Empiriquement, on constate que ν faible ($\nu < 0.1$) améliore les performances prédictives, mais au prix d'une convergence plus lente (nombre d'itérations **B** plus élevées).

Stochastic Gradient Boosting

Introduire l'échantillonnage. A chaque étape, seule une fraction β ($0 < \beta \leq 1$) de l'échantillon (sans remise) est utilisé pour la construction des arbres M_b [HAS, page 365]

$\beta = 1$, on a l'algorithme usuel. Typiquement, $0.5 \leq \beta \leq 0.8$ convient sur des tailles modérées de dataset [WIK]. Avantages :

1. Réduction du temps de calcul.
2. Introduction de l'aléatoire dans l'algorithme, cette forme de régularisation permet de se prémunir contre le sur-apprentissage (cf. Random Forest et Bagging)
3. Estimation OOB de l'erreur (cf. Bagging)

Pratique du gradient boosting

Logiciels et outils

Python (scikit-learn)

class `sklearn.ensemble.` **`GradientBoostingClassifier`** (*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) [source]

Gradient Boosting for classification.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

```
#importation des données
import pandas
dtrain = pandas.read_table("ionosphere-train.txt",sep="\t",header=0,decimal=".")
print(dtrain.shape)
y_app = dtrain.as_matrix()[:,32]
X_app = dtrain.as_matrix()[:,0:32]
#importation de la classe GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
#affichage des paramètres
print(gb)
#apprentissage
gb.fit(X_app,y_app)
#importation des données test
dtest = pandas.read_table("ionosphere-test.txt",sep="\t",header=0,decimal=".")
print(dtest.shape)
y_test = dtest.as_matrix()[:,32]
X_test = dtest.as_matrix()[:,0:32]
#prédiction
y_pred = gb.predict(X_test)
#evaluation : taux d'erreur = 0.085
from sklearn import metrics
err = 1.0 - metrics.accuracy_score(y_test,y_pred)
print(err)
```

Il y a aussi un mécanisme d'échantillonnage des variables pour la segmentation, comme pour Random Forest. cf. Site web SKLEARN

Python (scikit-learn)

Paramétrage

Scikit-learn propose un dispositif pour identifier par validation croisée les paramètres ''optimaux'' de la méthode.

```
#outil grille de recherche
from sklearn.grid_search import GridSearchCV
#combinaisons de paramètres à tester : Scikit-learn va évaluer toutes les combinaisons
#Les calculs vont être conséquents en validation croisée !!!
parametres = {"learning_rate":[0.3,0.2,0.1,0.05,0.01],"max_depth":[2,3,4,5,6],"subsample":[1.0,0.8,0.5]}

#classifieur à utiliser – gradient boosting
gbc = GradientBoostingClassifier()

#instanciation de la recherche
grille = GridSearchCV(estimator=gbc,param_grid=parametres,scoring="accuracy")
#lancer l'exploration
resultats = grille.fit(X_app,y_app)
#meilleur paramétrage : {'subsample': 0.5, 'learning_rate': 0.2, 'max_depth': 4}
print(resultats.best_params_)
#prédiction avec le ''meilleur'' modèle identifié
ypredc = resultats.predict(X_test)

#performances du ''meilleur'' modèle : taux d'erreur = 0.065
err_best = 1.0 - metrics.accuracy_score(y_test,ypredc)
print(err_best)
```

R

(package "gbm")

gbm: Generalized Boosted Regression Models

An implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, t-distribution loss, quantile regression, logistic, multinomial logistic, Poisson, Cox proportional hazards partial likelihood, AdaBoost exponential loss, Huberized hinge loss, and Learning to Rank measures (LambdaMart).

```
#importer les données
dtrain <- read.table("ionosphere-train.txt",header=T,sep="\t")
dtest <- read.table("ionosphere-test.txt",header=T,sep="\t")

#package "gbm"
library(gbm)

#apprentissage
gb1 <- gbm(class ~ ., data = dtrain, distribution="multinomial")

#prediction : predict renvoie un score (la valeur seuil est 0)
p1 <- predict(gb1,newdata=dtest,n.trees=gb1$n.trees)
y1 <- factor(ifelse(p1[,1,1] > 0, "b", "g"))

#matrice de confusion et taux d'erreur
m1 <- table(dtest$class,y1)
err1 <- 1 - sum(diag(m1))/sum(m1)
print(err1)
```

Distribution=« bernoulli » était possible aussi, mais le package s'attend à ce que l'on recode la variable cible en {0,1} dans ce cas.

R

(package "mboost")

mboost: Model-Based Boosting

Functional gradient descent algorithm (boosting) for optimizing general risk functions utilizing component-wise (penalised) least squares estimates or regression trees as base-learners for fitting generalized linear, additive and interaction models to potentially high-dimensional data.

De très nombreuses possibilités proposées.

```
#package "mboost"
library(mboost)
#apprentissage avec les paramètres par défaut (cf. doc en ligne)
gb2 <- blackboost(class ~ ., data = dtrain, family=Multinomial())
#prediction
y2 <- predict(gb2,newdata=dtest,type="class")
#matrice de confusion et taux d'erreur = 11.5%
m2 <- table(dtest$class,y2)
err2 <- 1 - sum(diag(m2))/sum(m2)
print(err2)

#modifier les paramètres de l'arbre sous jacent (pour un arbre plus grand)
library(party)
parametres <- ctree_control(stump=FALSE,maxdepth=10,minsplit=2,minbucket=1)
#apprentissage avec les nouveaux paramètres
gb3 <- blackboost(class ~ ., data = dtrain, family=Multinomial(),tree_controls=parametres)
#prediction
y3 <- predict(gb3,newdata=dtest,type="class")
#matrice de confusion et taux d'erreur = 12.5% (pas bon les arbres plus grands ici)
m3 <- table(dtest$class,y3)
err3 <- 1 - sum(diag(m3))/sum(m3)
print(err3)
```

Details

This function implements the 'classical' gradient boosting utilizing regression trees as base-learners. Essentially, the same algorithm is implemented in package gbm. The main difference is that arbitrary loss functions to be optimized can be specified via the `family` argument to `blackboost` whereas gbm uses hard-coded loss functions. Moreover, the base-learners (conditional inference trees, see `ctree`) are a little bit more flexible.

R

(package "xgboost")

Des variantes « performantes » comme xgboost introduisent une implémentation parallèle, rendant le calcul possible sur de très grandes bases (ainsi que d'autres modèles sous-jacents que les arbres, l'échantillonnage des variables dans la construction des arbres)

```
#package "xgboost"
library(xgboost)
#transformation des données d'apprentissage en un format géré par xgboost
XTrain <- as.matrix(dtrain[,1:32])
yTrain <- ifelse(dtrain$class=="b",1,0) #codage 1/0 de la cible
#construction du modèle avec les param. par défaut (eta=0.3, max.depth=6)
gb4 <- xgboost(data=XTrain,label=yTrain,objective="binary:logistic",nrounds=100)
#prédiction
XTest <- as.matrix(dtest[,1:32])
p4 <- predict(gb4,newdata=XTest)
#on dispose de PI("b") - on transforme en affectation
y4 <- factor(ifelse(p4 > 0.5,"b","g"))
#matrice de confusion et taux d'erreur = 9.5%
m4 <- table(dtest$class,y4)
err4 <- 1 - sum(diag(m4))/sum(m4)
print(err4)
#construction du modèle avec d'autres paramètres
gb5 <- xgboost(data=XTrain,label=yTrain,objective="binary:logistic",eta=0.5,max.depth=10,nrounds=100)
#prédiction
p5 <- predict(gb5,newdata=XTest)
y5 <- factor(ifelse(p5 > 0.5, "b","g"))
#matrice de confusion et taux d'erreur = 9%
m5 <- table(dtest$class,y5)
err5 <- 1 - sum(diag(m5))/sum(m5))
print(err5)
```

Attention au paramétrage des méthodes

Gradient boosting repose sur de nombreux paramètres qui pèsent fortement sur les performances. Ils peuvent interagir entre eux, rendant leur manipulation difficile. L'enjeu est d'arbitrer entre exploiter pleinement les données disponibles et se prémunir contre le sur-apprentissage.

Caractéristiques des arbres individuels	Profondeur des arbres, effectifs pour segmenter, effectifs d'admissibilité des feuilles. Arbre petit : résistance au sur-apprentissage, mais danger de sous-apprentissage. Inversement si arbre grand.
Constante d'apprentissage ν	Trop faible, lenteur de convergence car corrections timides ; trop élevé, oscillations, sur-apprentissage. « Bonne valeur » autour de 0.1. Si on le réduit, on doit augmenter en parallèle le nombre d'arbres.
Nombre d'arbres	Plus il y en a, mieux c'est. Le risque de sur-apprentissage est faible. Mais le temps de calcul augmente naturellement.
Taux d'échantillonnage des individus β	Stochastic gradient boosting. $\beta = 1$, algorithme utilisant toutes les observations. $\beta < 1$ réduit la dépendance à l'échantillon, résistance au sur-apprentissage par réduction de la variance. Valeur possible autour de 0.5
Echantillonnage des variables	Mécanisme analogue à celui des Random Forest. Permet de « diversifier » les arbres et donc réduire la variance. Peut-être manipulé conjointement avec les caractéristiques de taille des arbres (arbres grands ➔ moins de biais). Mécanisme présent dans certains packages seulement (xgboost, scikit-learn)

Bilan

Gradient Boosting

Le « gradient boosting » est une technique ensembliste qui généralise le boosting en offrant la possibilité d'introduire d'autres fonctions de coût.

Les schémas globaux sont identiques : algorithme sous-jacent = arbre, construction pas à pas des modèles, indicateur « variable importance » pour évaluer la pertinence des prédictives, problématiques similaires pour le paramétrage.

Mais, à la différence du boosting, même en classement, l'algorithme sous-jacent reste un arbre de régression.

Des outils/logiciels existent, mais il faut vraiment aller dans le détail de la documentation pour comprendre ce qu'il y a derrière les implémentations et la manipulation des très nombreux paramètres.

Gradient boosting (GBM : Gradient Boosting Machine)

Avantages et inconvénients

Surtout en classement qui nous intéresse au premier chef dans ce cours.

Avantages

- Par rapport au boosting ''usuel'', GBM s'intéresse à l'amplitude de l'erreur ($y-\pi$) dans la construction des arbres de régression intermédiaires avec la fonction de coût DEVIANCE
- Beaucoup de souplesses avec le choix des fonctions de coûts, adaptables aux spécificités des problèmes étudiés
- GBM a montré son efficacité dans les challenges !!!

Inconvénients

- Modèle non explicite (problème de toutes les méthodes ensemblistes)
- Paramètres nombreux, on s'y perd (taille de l'arbre, nombre d'itérations, paramètre de régularisation, etc.)
- Danger du sur-apprentissage (stratégies de régularisation interagissent entre elles)

- Lourdeur et intensité des calculs (#arbres peut être très élevé)
- Occupation mémoire de tous les arbres en déploiement

Références

Articles de référence

[BIS] Bishop C.M., « Pattern Recognition and Machine Learning », Springer, 2006.

[HAS] Hastie T., Tibshirani R., Friedman J., « The elements of Statistical Learning - Data Mining, Inference and Prediction », Springer, 2009.

[LI] LI C., « A Gentle Introduction to Gradient Boosting », 2014.

[NAT] Natekin A., Knoll A., « Gradient boosting machines, a tutorial », in *Frontiers in NeuroRobotics*, December 2013.

[RAK] Rakotomalala R., « Bagging – Random Forest – Boosting », 2015.

[WIK] Wikipédia, « Gradient boosting », consulté en Avril 2016.