Agrégation de modèles - Partie 3

Mathieu Pigeon

UQAM

Retour sur la problématique

- On cherche à « expliquer » une variable $Y \in \mathcal{Y}$ par p variables explicatives $\mathbf{X} \in \mathbb{R}^p$.
- $\mathcal{Y} = \mathbb{R}$ pour de la régression, $\mathcal{Y} = \{0, 1\}$ ou $\{-1, 1\}$ (ou autre si plus de deux catégories) pour de la classification.
- Règle de prédiction : une fonction mesurable $r : \mathbb{R}^p \to \mathcal{Y}$ obtenue en optimisant une mesure de la performance (fonction objectif).

Retour sur la problématique

Pour une problématique de régression, on peut mesurer la performance d'une règle de prédiction par son erreur quadratique moyenne (MSE)

$$\begin{aligned}\widehat{r}(\mathbf{X}) &= \arg\min_{r:\mathbb{R}^p\to\mathbb{R}} \mathbb{E}\left[(Y - r(\mathbf{X})^2\right] \\ &= \mathbb{E}\left[Y|\mathbf{X}=\mathbf{x}\right].\end{aligned}$$

Pour une problématique de classification, on peut mesurer la performance d'une règle de prédiction par sa probabilité d'erreur

$$\begin{aligned}\widehat{r}(\mathbf{X}) &= \arg\min_{r:\mathbb{R}^p\to\{-1,1\}} \Pr\left(r(\mathbf{X}) \neq Y\right) \\ &= \begin{cases} 1, & \Pr\left(Y=1|\mathbf{X}=\mathbf{x}\right) \geq 0.5 \\ -1, & \text{sinon.}\end{cases}\end{aligned}$$

Fonction objectif

1. De façon générale, on cherche une règle $\widehat{r}$ en résolvant un problème d'optimisation empirique

$$\widehat{r} = \arg\min_{r \in \mathcal{R}} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, r(\boldsymbol{X}_i)),$$

   où $L()$ est une fonction objectif (ou fonction de perte) quelconque.
2. Si l'optimisation est faite sur l'ensemble des fonctions r possibles et imaginables, le problème est trop complexe pour être résolu (même numériquement !).

Fonction objectif

1. Par contre, si on limite la classe $\mathcal{R}$ aux fonctions linéaires, par exemple

$$\widehat{r}(\mathbf{X}_i) = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_p X_{ip},$$

et si on choisit une fonction objectif quadratique, alors on obtient un modèle de régression linéaire classique.

Fonction objectif

- Une solution possible est de « convexifier » la fonction objectif afin de rendre le problème plus simple d'un point de vue numérique.
- L'algorithme *AdaBoost* présenté au cours précédent répond à ce principe de minimisation pour un risque « convexifié » donné par

$$L(Y_i, r(\boldsymbol{X}_i)) = \exp\left(-Y_i r(\boldsymbol{X}_i)\right).$$

- On va également réaliser l'optimisation de façon séquentielle plutôt que globale (*greedy*).

Fonction objectif

On peut démontrer que la règle de classification à l'étape t de l'agorithme *AdaBoost* peut s'écrire comme étant [1]

$$r_t(\mathbf{X_i}) = r_{t-1}(\mathbf{X}_i) + \alpha_t R_t(\mathbf{X}_i),$$

où

$$(\alpha_t, R_t) = \arg\min_{(\alpha,r)\in\mathbb{R}\times\mathcal{R}} \sum_{i=1}^{n} e^{-Y_i(r_{t-1}(\mathbf{X}_i)+\alpha r(\mathbf{X}_i))}$$

et où $\mathcal{R}$ est l'espace des choix possibles pour la règle r.

1. On suppose que la règle de classification r_t s'obtient en minimisant le taux de mauvaise classification sur la base de données complète.

Généralisation

- On a vu que l'algorithme *AdaBoost* peut être vu comme une méthode récursive optimisant, à chaque étape, une fonction objectif empirique sur une classe de fonctions donnée.
- On peut généraliser en cherchant la règle r qui minimise $\mathbb{E}\left[L(Y, r(\mathbf{X})\right]$ pour une fonction objectif convexe $L : \mathbb{R}^2 \to \mathbb{R}$.
- Pour y parvenir, on va utiliser une approche de descente de gradient fonctionnelle : le $\ll$ paramètre $\gg$ que l'on cherche à minimiser est une fonction (r).

Méthode de Newton-Raphson

- On considère une fonction strictement convexe $J : \mathbb{R} \to \mathbb{R}$ et un problème de minimisation dont la solution est $\tilde{x}$.
- On cherche une suite $x_0, x_1, x_2, \ldots$ qui converge vers la solution $\tilde{x}$.
- À partir d'un point de départ arbitraire x_0, on cherche $x_1 = x_0 + h$ tel que $J'(x_1) \approx 0$.

Méthode de Newton-Raphson

- Par un développement limité, on a

$$J'(x_0 + h) \approx J'(x_0) + hJ''(x_0).$$

- Puisque $J'(x_1) = J'(x_0 + h) \approx 0$, on a

$$h \approx \frac{-J'(x_0)}{J''(x_0)}.$$

- Avec $\alpha = (J''(x_0))^{-1} > 0$, on obtient la formule récursive

$$x_k = x_{k-1} - \alpha J'(x_{k-1}).$$

Version vectorielle

En utilisant la même idée mais dans un cadre vectoriel, on obtient la formule récursive

$$r_t(\mathbf{X}_i) = r_{t-1}(\mathbf{X}_i) - \alpha \left[\frac{\partial L(Y_i, r(\mathbf{X}_i)}{\partial r(\mathbf{X}_i)} \right] |_{\{r(\mathbf{X}_i)=r_{t-1}(\mathbf{X}_i)\}}.$$

On obtient ainsi une suite de règles $r_0(\mathbf{X}_i), r_1(\mathbf{X}_i), r_2(\mathbf{X}_i), \ldots$ que l'on peut évaluer pour les valeurs présentes dans la base de données $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$.

Version vectorielle

Pour pouvoir évaluer cette suite de règles en tout points $\mathbf{X} \in \mathbb{R}^p$, on effectue, à chaque étape, une régression avec

$$U_i = -\left[\frac{\partial L(Y_i, r(\mathbf{X}_i)}{\partial r(\mathbf{X}_i)}\right] |_{\{r(\mathbf{X}_i)=r_{t-1}(\mathbf{X}_i)\}}, \qquad i = 1, \ldots, n$$

comme variable réponse (minimisation de la fonction de perte quadratique).

Boosting par descente de gradient fonctionnelle

1. Initialiser

$$r_0(\mathbf{X}_i) = \arg\min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, c).$$

2. Pour les étapes $t = 1, \ldots, T$,
   2a. Calculer

$$U_i = - \left[\frac{\partial L(Y_i, r(\mathbf{X}_i))}{\partial r(\mathbf{X}_i)} \right] |_{\{r(\mathbf{X}_i) = r_{t-1}(\mathbf{X}_i)\}}, \qquad i = 1, \ldots, n.$$

   2b. Ajuster le *weak learner* sélectionné sur l'échantillon composé des éléments $(\mathbf{X}_1, U_1), (\mathbf{X}_2, U_2), \ldots, (\mathbf{X}_n, U_n)$ afin d'obtenir la pseudo-règle $h_t(\mathbf{X}_i)$.
   2c. Effectuer la mise à jour $r_t(\mathbf{X}_i) = r_{t-1}(\mathbf{X}_i) + \alpha h_t(\mathbf{X}_i)$.
3. La prédiction finale est donnée par $r_T(\mathbf{X}_i)$.

Remarques

- Le choix du paramètre α (taux d'apprentissage) est peu important. On recommande généralement de prendre une petite valeur (0.01 ou 0.001).
- Si on pose $\alpha = 1$ et $L(y, r) = \exp(-yr)$, on retrouve presque l'algorithme *AdaBoost*.

Exemple 1 : Boston

```
round(head(Boston), 2)

crim zn indus chas  nox   rm  age  dis rad tax
0.01 18  2.31    0 0.54 6.58 65.2 4.09   1 296
0.03  0  7.07    0 0.47 6.42 78.9 4.97   2 242
0.03  0  7.07    0 0.47 7.18 61.1 4.97   2 242

ptratio   black lstat  medv
   15.3  396.90  4.98  24.0
   17.8  396.90  9.14  21.6
   17.8  392.83  4.03  34.7
set.seed(1001)
train <- sample(1:506, size = 420, replace = FALSE)
```

Exemple 1a : Forêts aléatoires

```
modele1 <- randomForest(Boston[train,-14],
           y = Boston[train,]$medv , ntree = 500,
           importance = TRUE)

### Importance des variables (IncMSE)
round(sort(rf_model$importance[,1], decreasing=TRUE), 2)

lstat       rm      nox    indus     crim  ptratio ...
63.32    35.69    10.61     8.05     7.34     6.86 ...
```

Exemple 1a : Forêts aléatoires

```
### MSE Out-of-bag
oob_pred <- predict(modele1)

oob <- mean(as.numeric((oob_pred - Boston[train,]$medv)^2))
oob
9.490969

### vMSE
y_pred_rf <- predict(modele1 , Boston[-train,])
test_mse <- mean(((y_pred_rf - Boston[-train,]$medv)^2))
test_mse
13.73492
```

Exemple 1b : Boosting

```
library(xgboost)

dtrain <- xgb.DMatrix(data =  as.matrix(Boston[train, -14]),
                      label = Boston[train,]$medv)
dtest <- xgb.DMatrix(data =  as.matrix(Boston[-train, -14]),
                     label = Boston[-train,]$medv)
```

Exemple 1b : Boosting

```
watchlist <- list(train = dtrain, test = dtest)

bst <- xgb.train(data = dtrain,
                 max.depth = 8,
                 eta = 0.3,
                 nthread = 2,
                 nround = 1000,
                 watchlist = watchlist,
                 objective = "reg:linear",
                 early_stopping_rounds = 50,
                 print_every_n = 500)
```

Exemple 1b : Boosting

```
[1] train-rmse:17.178850 test-rmse:16.753817
Multiple eval metrics are present.
Will use test_rmse for early stopping.
Will train until test_rmse hasn't improved in 50 rounds.

Stopping. Best iteration:
[49] train-rmse:0.042135 test-rmse:3.520537
3.520537^2
12.39418
```

Exemple 1b : Boosting

```
bst_slow <- xgb.train(data = dtrain,
                      max.depth=5,
                      eta = 0.01,
                      nthread = 2,
                      nround = 10000,
                      watchlist = watchlist,
                      objective = "reg:linear",
                      early_stopping_rounds = 50,
                      print_every_n = 500)
rf_benchmark <- 13.73492
bst_slow$best_score^2 / rf_benchmark
0.717572
```

Exemple 1b : Boosting

```
sample <- sample.int(n = nrow(Boston[train,]),
           size = floor(.8*nrow(Boston[train,])),
           replace = FALSE)

train_t <- (Boston[train,])[sample, ]
valid  <- (Boston[train,])[-sample, ]

train_y <- train_t$medv

train_x <- train_t[, -14]

valid_y = valid$medv
valid_x = valid[, -14]
```

Exemple 1b : Boosting

```
gb_train <- xgb.DMatrix(data = as.matrix(train_x),
                        label = train_y )
gb_valid <- xgb.DMatrix(data = as.matrix(valid_x),
                        label = valid_y )
watchlist <- list(train = gb_train, valid = gb_valid)

bst_slow <- xgb.train(data= gb_train,
                      max.depth = 5,
                      eta = 0.01,
                      nthread = 2,
                      nround = 10000,
                      watchlist = watchlist,
                      objective = "reg:linear",
                      early_stopping_rounds = 50,
                      print_every_n = 500)
```

Exemple 1b : Boosting

```
[1] train-rmse:23.909586 valid-rmse:23.097574

[500] train-rmse:1.173579 valid-rmse:2.276383
```

Exemple 1b : Boosting

```
y_hat_valid <- predict(bst_slow, dtest)

test_mse <- mean(((y_hat_valid - Boston[-train,]$medv)^2))
test_rmse = test_mse
test_rmse
10.59443

test_rmse/rf_benchmark
0.77135
```

Exemple 1b : Boosting

```
max.depths <- c(3, 5, 7, 9, 11)
etas <- c(0.1, 0.01, 0.001)

best_params = 0
best_score = 0
```

Exemple 1b : Boosting

```
count = 1
for( depth in max.depths ){
  for( num in etas){

    bst_grid = xgb.train(data = gb_train,
                         max.depth = depth,
                         eta=num,
                         nthread = 2,
                         nround = 10000,
                         watchlist = watchlist,
                         objective = "reg:linear",
                         early_stopping_rounds = 50,
                         verbose=0)
...
```

Exemple 1b : Boosting

```
...
    if(count == 1){
      best_params = bst_grid$params
      best_score = bst_grid$best_score
      count = count + 1
    }
    else if( bst_grid$best_score < best_score){
      best_params = bst_grid$params
      best_score = bst_grid$best_score
    }
  }
}
```

Exemple 1b : Boosting

```
best_params
$max_depth
[1] 3

$eta
[1] 0.1

$nthread
[1] 2

best_score
valid-rmse
  2.132227
```

Exemple 1b : Boosting

```
bst_tuned <- xgb.train( data = gb_train,
                        max.depth = 3,
                        eta = 0.1,
                        nthread = 2,
                        nround = 10000,
                        watchlist = watchlist,
                        objective = "reg:linear",
                        early_stopping_rounds = 50,
                        print_every_n = 500)
Stopping. Best iteration:
[126] train-rmse:1.210756 valid-rmse:2.132227
```

Exemple 1b : Boosting

```
y_hat_xgb_grid <- predict(bst_tuned, dtest)
test_mse <- mean(((y_hat_xgb_grid - Boston[-train,]$medv)^2))
test_rmse <- test_mse
test_rmse
[1] 10.22936

test_rmse/rf_benchmark
[1] 0.7447702
```

Exemple 1 : Résultats

Modèle		vMSE
Forêts aléatoires	-	13.7349
Boosting	générique (sans cv)	12.3942
	slow (sans cv)	9.8558
	slow (cv)	10.5944
	tuned (cv)	10.2294