Agrégation de modèles - Partie 2

Mathieu Pigeon

UQAM

1. Boosting
2. Approche par sous-ensembles (algorithme historique)
3. Approche probabiliste (algorithme AdaBoost)

Boosting

- Comme le *Bagging* et les forêts aléatoires, le *Boosting* est un méta-algorithme qui permet d'améliorer le pouvoir prédictif des arbres de décision.
- L'idée peut également s'appliquer à d'autres méthodes.
- En français, on pourrait parler (mais en pratique, on ne le fait jamais) d'amplification ou de stimulation.

Comparaison : Bagging et Boosting

- Le *Bagging* vise à construire B modèles à partir de B bases de données obtenues par *bootstrap*. La **construction** de ces B modèles se fait de façon **parallèle**, c'est-à-dire de façon totalement indépendante.
- Le *Boosting* vise à construire B modèles de façon **séquentielle** : le b^{e} modèle dépend du $(b-1)^{\text{e}}$ modèle qui dépend lui-même du $(b-2)^{\text{e}}$ modèle, etc. Chacun nouveau modèle est construit spécifiquement afin d'améliorer les prédictions faites par le modèle précédent.

Comparaison : Bagging et Boosting

- Avec le *Bagging*, une prédiction est obtenue en calculant la moyenne empirique des prédictions faites par chacun des B modèles. Chacun des modèles $b = 1, \ldots, B$ est généralement un modèle $\ll$ complexe $\gg$: grand arbre non élagué.
- Avec le *Boosting*, une prédiction est obtenue en calculant la moyenne empirique des prédictions faites par chacun des B modèles. Chacun des modèles $b = 1, \ldots, B$ est généralement très $\ll$ simple $\gg$: petit arbre avec très peu de noeud(s) $\rightarrow$ apprenant faible (*weak learner*).

Algorithme

On s'intéresse à un problème de classification binaire (0 ou 1) à partir d'une base de données $\mathcal{D} = \{Y_i; \mathbf{X}_i\}_{i=1,\ldots,n}$.

1. On apprend une première règle de classification r_1 à partir d'un sous-ensemble de taille $n_1 < n$, $\mathcal{D}_1 \subset \mathcal{D}$ (par exemple, un petit arbre avec une racine et deux feuilles).
2. On applique cette règle r_1 aux observations de la base de données initiales.
3. On apprend une deuxième règle de classification r_2 à partir d'un sous-ensemble de taille $n_2 < n$, $\mathcal{D}_2 \subset \mathcal{D} - \mathcal{D}_1$, en s'assurant qu'il contient 50% d'observations mal classées par la règle r_1.
4. On applique cette deuxième règle aux observations de la base de données initiales.

Algorithme (suite)

5. On apprend une troisième règle de classification r_3 à partir d'un sous-ensemble de taille $n_3 < n$, $\mathcal{D}_3 \subset \mathcal{D} - \mathcal{D}_1 - \mathcal{D}_2$, composé d'observations pour lesquelles r_1 et r_2 sont en désaccord.
6. On applique cette troisième règle aux observations de la base de données initiales.
7. Pour une observation, la prédiction finale est le mode des prédictions faites par r_1, r_2 et r_3 :

$$R = \text{mode}(r_1, r_2, r_3).$$

Remarques

- De façon informelle, un apprenant faible est un modèle, ou un ensemble de règles, qui est performe légèrement mieux que le hasard seul.
- On peut démontrer que la performance de R est supérieure à la performance d'une règle qui aurait été apprise à partir de $\mathcal{D}$.
- Cette méthode peut être utilisée récursivement avec $9, 27, \ldots$ sous-ensembles.
- Cette approche a été proposée en 1990 par Robert E. Schapire, dans *The strength of weak learnability*, publié dans *Machine Learning, 5(2) :197– 227*.

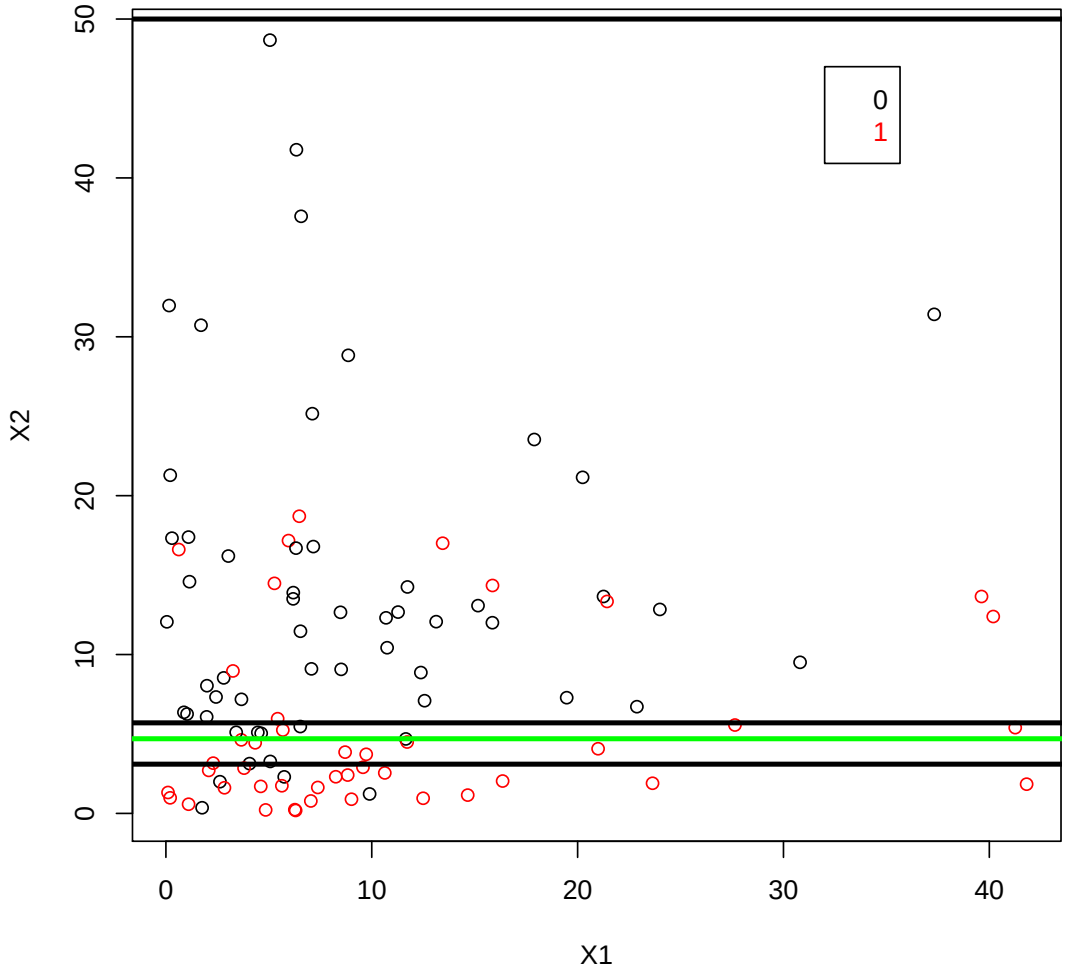
Exemple 1

```
head(data[,c(1,2,3,4,6,8,9,10)])
          X1         X2 Y    R1    R2   R3  pred pred1
1  1.993098  8.039945 0 FALSE FALSE TRUE FALSE  TRUE
2  9.729683  3.721495 1 FALSE  TRUE TRUE  TRUE FALSE
3 11.653844  4.688819 0 FALSE  TRUE TRUE  TRUE FALSE
4  9.895877  1.226427 0  TRUE  TRUE TRUE  TRUE FALSE
5  6.568907 37.584709 0 FALSE FALSE TRUE FALSE  TRUE
6 23.639619  1.899055 1  TRUE  TRUE TRUE  TRUE FALSE

sum(data$pred == data$Y)/length(data$pred)
[1] 0.78

sum(data$pred1 == data$Y)/length(data$pred1)
[1] 0.21
```

Exemple 1



Idées principales

Les approches probabilistes sont basées sur trois concepts :

- faire « voter » des modèles plus ou moins nombreux afin de prendre une décision ;
- pondérer les votes des modèles et adapter cette pondération aux résultats ;
- modifier les observations de la base d'entrainement pour chaque modèle afin de donner davantage d'importance aux observations mal représentées par les précédents modèles.

Algorithme AdaBoost

On s'intéresse à un problème de classification binaire (0 ou 1) à partir d'une base de données $\mathcal{D} = \{Y_i; \mathbf{X}_i\}_{i=1,\ldots,n}$.

1. Toutes les observations reçoivent un poids identique : $p_1(\mathbf{X}_i) = 1/n$, $i = 1, \ldots, n$.

Algorithme AdaBoost (suite)

2. Pour l'étape $t = 1, \ldots, T$,
   - 2a. Piger au hasard (en utilisant les poids $p_t(\mathbf{X}_i)$) une base de données d'entrainement $\mathcal{D}_t$.
   - 2b. Apprendre une règle de classification r_t sur cette base de données.
   - 2c. Calculer l'*erreur apparente*

$$\epsilon_t = p_t\left(r_t(\mathbf{X}_i) \neq Y_i\right) = \sum_{i: r_t(\mathbf{X}_i) \neq Y_i} p_t(i)$$

   pour $\mathcal{D}_t$ et évaluer $\alpha_t = 0.5 \ln((1-\epsilon_t)/\epsilon_t)$.
   - 2d. Mettre à jour les poids :

$$p_{t+1}(\mathbf{X}_i) = \begin{cases} \frac{p_t(\mathbf{X}_i)}{Z_t} e^{-\alpha_t}, & r_t(\mathbf{X}_i) = Y_i \\ \frac{p_t(\mathbf{X}_i)}{Z_t} e^{+\alpha_t}, & r_t(\mathbf{X}_i) \neq Y_i, \end{cases}$$

   où Z_t est une constante de normalisation assurant que $\sum_{i=1}^{n} p_{t+1}(\mathbf{X}_i) = 1$.

Algorithme AdaBoost (suite)

3. La classification finale faite par le modèle est alors

$$R(\mathbf{X}_i) = \begin{cases} 1, & \sum_{t=1}^{T} \alpha_t r_t(\mathbf{X}_i) > 0 \\ -1, & \sum_{t=1}^{T} \alpha_t r_t(\mathbf{X}_i) < 0. \end{cases}$$

Remarques

- L'erreur apparente est calculée en utilisant la distribution avec laquelle le modèle est entrainé.
- Chacune des nouvelles règles r_t est pondérée par un poids α_t qui indique l'importance que cette règle aura dans la décision finale R.
- Si T, le nombre de modèles, est $\ll$ trop $\gg$ grand, alors il y a un risque que l'algorithme se concentre trop vers la fin sur les cas difficiles (potentiellement du bruit) et accorde à ces derniers trop d'importance dans la décision finale R.

Bornes

- L'erreur apparente

$$\epsilon_t = p_t\left(r_t(\mathbf{X}_i) \neq Y_i\right) = \sum_{i:r_t(\mathbf{X}_i)\neq Y_i} p_t(i)$$

peut être réécrite comme

$$\epsilon_t = 0.5 - \gamma_t,$$

où γ_t est l'amélioration apportée à la prédiction par la règle r_t en comparaison avec le hasard (0.5).

Bornes (suite)

- On peut démontrer que l'erreur apparente finale (pour R) a comme borne supérieure

$$\epsilon_R \leq \exp\left(-2\sum_t \gamma_t^2\right) \leq \exp\left(-2T\gamma^2\right),$$

où $\gamma = \min(\gamma_1, \ldots, \gamma_T)$.

- Ainsi, si le *weak learner* fait légèrement mieux que le hasard, $\gamma_t > 0$ et l'erreur apparente finale diminue exponetiellement avec le nombre de modèles (T).

Bornes (suite)

- On peut également démontrer que l'erreur de généralisation, c'est-à-dire l'erreur calculée sur une base de données qui n'a pas été utilisée pour l'entrainement du modèle, a comme borne supérieure

$$\text{Err}_G \leq \epsilon_R + \sqrt{\frac{(T)(d_{\mathcal{H}})}{n}},$$

où $d_{\mathcal{H}}$ est un terme lié à la taille de l'espace d'hypothèses.

- Cette borne indique que si T devient grand, alors le *Boosting* devrait tendre à surapprendre.
- En pratique, on observe rarement ce phénomène.

Généralisation

- L'algorithme AdaBoost peut également s'utiliser pour des problèmes de classification pour un nombre quelconque de classes et pour des problèmes de régression.
- L'algorithme peut aussi être indirectement utilisé pour détecter des observations aberrantes.

Exemple 2 : données initiales

Base de données de taille $n = 10\,000$.

	X1	X2	X3	X4	Y
1	1.05	24.19	0.04	0.21	0
2	19.16	2.42	1.96	0.14	1
3	18.63	4.65	-0.89	0.15	1
4	3.56	2.12	-0.72	0.76	0
5	4.59	3.57	1.18	0.53	0
6	2.68	9.38	-2.19	0.98	0
	...	...	...	...	...

Exemple 2 : étape 1

Y	p1	pred1	p2
0	1e-04	0	4.141366e-05
1	1e-04	1	4.141366e-05
1	1e-04	1	4.141366e-05
0	1e-04	1	2.321244e-04
0	1e-04	1	2.321244e-04
0	1e-04	0	4.141366e-05

Exemple 2 : étape 2

Y	p1	pred1	p2	pred2	p3
0	1e-04	0	4.141366e-05	0	2.009414e-05
1	1e-04	1	4.141366e-05	1	2.009414e-05
1	1e-04	1	4.141366e-05	1	2.009414e-05
0	1e-04	1	2.321244e-04	0	1.126280e-04
0	1e-04	1	2.321244e-04	0	1.126280e-04
0	1e-04	0	4.141366e-05	0	2.009414e-05

Exemple 2 : étape finale

Y	p1	pred1	p2	pred2	...	PRED_FIN
0	1e-04	0	4.141366e-05	0	...	0
1	1e-04	1	4.141366e-05	1	...	1
1	1e-04	1	4.141366e-05	1	...	1
0	1e-04	1	2.321244e-04	0	...	0
0	1e-04	1	2.321244e-04	0	...	0
0	1e-04	0	4.141366e-05	0	...	0

Exemple 2 : taux correction

```
sum(data$Y == data$PRED_FIN)/length(data$Y)
[1] 0.6963
```