Le boosting appliqué au problème de la classification binaire

PRADIER Antoine

ENSAE Paristech, 3A Data Science*

Introduction

Le boosting est l'une des techniques d'apprentissage les plus puissantes introduites dans les deux dernières décennies. Bien qu'il ait été conçu à l'origine pour des problèmes de classification, il peut être étendu à la régression, mais nous ne l'étudierons pas ici.
L'idée générale du boosting est une procédure qui combine les résultats de classifieurs dits "faibles", *i.e* légèrement meilleurs qu'un choix aléatoire (un arbre de décision par exemple) pour produire un classifieur efficace. De ce point de vue, la technique du boosting ressemble fortement à celle du bagging. Cependant, ll diffère nettement sur la façon de construire la famille de classifieurs qui est dans le cas du boosting récurrente : chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées ou mal prédites. De façon intuitive, cet algorithme concentre ses efforts sur les observations les plus difficiles à prédire tandis que l'agrégation de l'ensemble des modèles réduit le risque de sur-ajustement.

Les algorithmes de boosting diffèrent selon plusieurs critères :

- la façon de pondérer c'est-à-dire de renforcer l'importance des observations mal estimées lors de l'itération précédente
- l'utilisation éventuelle de la randomisation
- la fonction de perte, qui peut être choisie plus ou moins robuste aux "outliers", pour mesurer l'erreur d'ajustement

Nous commencons par une présentation des principes mathématiques du boosting, avant de l'appliquer à des données simulées, puis à un problème de classification binaire. Nous tâcherons de présenter les résultats de différentes variantes du boosting, AdaBoost et Gradient Boosting.

1 Idée générale pour la classification binaire

Comme dit précédemment, le boosting est une méthode d'ensemble learning, dont l'idée générale est de combiner plusieurs classifieurs dits "faibles" afin d'obtenir un classifieur efficace.

Dans le cadre de la présentation du boosting appliqué à la classification binaire, nous utiliserons les notations suivantes :

- $X \in \mathbb{R}^{n \times d}$ représente l'ensemble des variables explicatives observées.
- $y_1, \cdots, y_n \in \{-1, 1\}$ sont les labels associés aux variables explicatives.
- $\mathcal{L}$ désigne la fonction de perte.
- $G \subset \mathcal{F}(X, \{-1, 1\}^n)$ est l'ensemble des classifieurs que l'on cherche à agréger où $\mathcal{F}(E, F)$ désigne l'ensemble des fonctions de E dans F.

Ainsi, en considérant que l'agrégation de classifieurs faibles notés $g_1, \cdots, g_M \in G$ se fait via une somme pondérée de ces derniers, le classifieur obtenu par boosting s'écrit :

$$g_{\text{boost}}(X) = \sum_{m=1}^{M} \alpha_m g_m(X)$$

* Cours de Techniques avancées d'apprentissage donné par Stéphan Clémençon

On cherche donc des fonctions $g_1, \cdots, g_M$ dans G et des réels $\alpha_1, \cdots, \alpha_m$ tels que $g_{\text{boost}}(X)$ minimise le risque empirique sous la fonction de perte $\mathcal{L}$:

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i, g_{\text{boost}}(X_i))$$

Après avoir présenté le principe général du boosting pour la classification binaire, nous nous penchons sur différentes variantes de ce modèle, en commençant par l'algorithme AdaBoost.

2 Algorithme AdaBoost

2.1 Description de l'algorithme

L'algorithme AdaBoost (ADAptive BOOSTing) est historiquement le premier algorithme conçu, par Freund Schapire en 1997, à partir du principe général décrit dans la section précédente.

Les poids de chaque observation sont initialisés à $\frac{1}{n}$ pour l'estimation du premier modèle puis évoluent à chaque itération donc pour chaque nouvelle estimation. L'importance d'une observation (ou le poids associé à cette observation) w_i est inchangée si celle-ci est bien classée, elle croît sinon proportionnellement à la qualité d'ajustement du modèle mesurée par α_m à l'itération m.

L'algortihme AdaBoost s'écrit donc de la manière suivante :

Algorithme : AdaBoost.M1

Entrée : (X, y) ensemble des observations labellisées, M nombre d'itérations.
Initialisation : Pour chaque observation, on initialise les poids $w_i = \frac{1}{n}$.
Itérations : Pour m variant de 1 à M :

1) Entraîner l'estimateur faible g_m par les données de l'échantillon X pondéré par chaque w_i, $i = 1, \cdots, n$
2) Calculer le taux d'erreur $e_m = \dfrac{\sum_{i=1}^{n} w_i \mathbf{1}_{g_m(X_i \neq y_i)}}{\sum_{i=1}^{n} w_i}$
3) Calculer le défaut d'ajustement du modèle : $\alpha_m = \log((1 - e_m)/e_m)$
4) Réajuster les poids associés aux observations :

$$w_i = w_i \exp(\alpha_m \mathbf{1}_{g_m(X_i \neq y_i)}) \text{ pour } i = 1, \cdots, n$$

Sortie : $g_{\text{AdaBoost}}(X) = \text{sgn}\left(\sum_{m=1}^{M} \alpha_m g_m(X)\right)$

2.2 Justification théorique de l'algorithme

Le bon comportement du boosting par rapport à d'autres techniques de discrimination est difficile à expliquer ou justifier par des arguments théoriques. Néanmoins, Hastie et col. [2007] présentent le boosting dans le cas binaire comme l'équivalent d'un modèle additif construit pas-à-pas (Forward Stagewise Additive Modeling) avec une fonction de perte exponentielle : $\mathcal{L}(y, f(x)) = \exp(-yf(x))$.

Nous allons démontrer cette équivalence en introduisant dans un premier temps les modèles additifs.

Forward Stagewise Additive Modeling

Nous introduisons la forme générale d'un modèle additif :

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

où $b(x;\gamma_m)$ représente le m-ième classifieur faible de X dans $\{-1,1\}$.

La procédure inhérente à ce type de modèle est la suivante :

Algorithme : Forward Stagewise Additive Modeling

Entrée : x ensemble des observations.
Initialisation : $f_0(x) = 0$.
Itérations : Pour m variant de 1 à M :
1) Calculer $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \mathcal{L}(y_i, f_{m-1}(x_i)) + \beta b(x_i, \gamma)$
2) Calculer $f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$

Relation entre le Forward Stagewise Additive Modeling et AdaBoost
À partir de l'expression des paramètres (β_m, γ_m) décrite dans l'algorithme précédent, et étant donné que dans le cas d'AdaBoost, $b(x_i;\gamma_i) = g(x_i)$, avec une fonction de perte exponentielle, on peut écrire à chaque étape m :

$$\begin{aligned}(\beta_m, g_m) &= \arg\min \sum_{i=1}^{n} \exp\left(-y_i[f_{m-1}(x_i) + \beta g(x_i)]\right) \\ &= \arg\min \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i g(x_i)) \text{ avec } w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))\end{aligned}$$

À partir de là, on peut voir que quelque soit la valeur de β, on doit nécessairement avoir la valeur de g_m suivante :

$$g_m = \arg\min \sum_{i=1}^{n} w_i^{(m)} \mathbf{1}(y_i \neq g(x_i))$$

Le calcul de β_m nécessite un peu plus de calcul :

$$\begin{aligned}(\beta_m) &= \arg\min \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i g_m(x_i)) \\ &= \arg\min e^{-\beta} \sum_{y_i = g_m(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq g_m(x_i)} w_i^{(m)} \\ &= \arg\min (e^{\beta} - e^{-\beta}) \cdot \sum_{i=1}^{n} w_i^{(m)} \mathbf{1}(y_i \neq g_m(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}\end{aligned}$$

Pour obtenir une formule explicite de β_m, il suffit maintenant d'annuler la dérivée de l'expression de droite en β, et ainsi obtenir :

$$e^{2\beta_m} = \frac{\sum_{i=1}^{n} w_i^{(m)}}{\sum_{i=1}^{n} w_i^{(m)} \mathbf{1}(y_i \neq g_m(x_i))} - 1$$

En posant $e_m = \frac{\sum_{i=1}^{n} w_i^{(m)}}{\sum_{i=1}^{n} w_i^{(m)} \mathbf{1}(y_i \neq g_m(x_i))}$, obtient : $\beta_m = \frac{1}{2} \log\left(\frac{1-e_m}{e_m}\right)$
Nous obtenons ainsi un résultat très similaire à celui que nous avions posé pour α_m dans AdaBoost, à savoir $\alpha_m = 2\beta_m$, et le taux d'erreur pondéré est en outre identique.

Afin de terminer la preuve de l'analogie entre les deux algorithmes, il ne reste plus qu'à s'intéresser à la mise à jour des poids w_i.
De la relation de mise à jour du modèle $f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$, les poids à la nouvelle itération sont :

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \cdot \exp\left(-\beta_m y_i g_m(x_i)\right) \\ &= w_i^{(m)} \cdot \exp(\alpha_m \mathbf{1}(y_i \neq g_m(x_i))) \cdot \exp(-\beta_m) \end{aligned}$$

Comme $\exp(-\beta_m)$ est indépendant des observations, on peut s'en affranchir et retomber sur la même mise à jour que celle décrite dans l'algorithme AdaBoost.

3 Variantes d'algorithmes de boosting

Après avoir présenté le principe général du boosting, la version initiale de l'algorithme AdaBoost et sa validation théorique, nous nous intéressons dans cette partie aux algorithmes plus récents et plus couramment utilisés de nos jours.

3.1 Gradient Boosting Machine

L'idée du gradient boosting est née de l'observation par Leo Breiman que le boosting pouvait être interprété comme un algorithme d'optimisation en considérant une fonction de perte appropriée. Le principe de base est le même que pour AdaBoost, construire une séquence de modèles de sorte que chaque étape, chaque modèle ajouté à la combinaison, apparaisse comme un pas vers une meilleure solution. La principale innovation est que ce pas est franchi dans la direction du gradient négatif de la fonction de perte.

En utilisant les mêmes notations que précédemment, l'algorithme de Gradient Boosting est le suivant :

Algorithme : Gradient Boosting

Entrée : (X, y) ensemble des observations labellisées, M nombre d'itérations.
Initialisation : $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma)$
Itérations : Pour m variant de 1 à M :

1) Calculer les *pseudo-résidus* $r_{i,m} = -\left[\dfrac{\partial \mathcal{L}(y_i, f(X_i))}{\partial f(X_i)}\right]_{F(X)=F_{m-1}(X)}$ pour $i = 1, \cdots, n$
2) Entraîner le classifieur $g_m(X)$ sur l'ensemble des pseudo-résidus, *i.e.* sur $\{(X_i, r_{i,m})\}_{i=1,\cdots,n}$
3) Calculer $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, f_{m-1}(X_i) + \gamma g_m(X_i))$
4) Mise à jour du modèle : $f_m(X) = f_{m-1}(X) + \gamma_m g_m(X)$

Sortie : $F_M(X)$

Cet algorithme est généralement utilisé avec des arbres de régression. Le pseudo-code présenté ci-dessus vaut pour des problèmes de régression comme des problèmes de classification. Néanmoins, il est à noter que pour l'utiliser dans le cas de la classification binaire, il est nécessaire de définir une fonction de perte adaptée à ce type de problème comme la déviance binaire (ou *cross-entropy*).
Dans le cas de la classification, pour une classe donnée, le gradient correspond à l'écart entre l'indicatrice correspondante et la probabilité d'appartenance à cette classe.

3.2 Extreme Gradient Boosting : XGBoost

L'algorithme XGBoost, abréviation de "Extreme Gradient Boosting", a été introduit par Chen en 2016. Depuis sa publication, XGBoost est devenu l'un des algorithmes d'apprentissage les plus populaires. Dans cette sous-section, nous donnons simplement les idées principales de cette méthode de boosting.

L'idée générale est la même que celle de l'algorithme de *gradient boosting*, à chaque itération on cherche à améliorer le modèle en entraînant un classifieur faible dans la direction opposée du gradient.

Les principales différences avec l'algorithme précédent sont les suivantes :

• Ajout d'un terme de régularisation dans la fonction de perte : permet de limiter l'ajustement de l'arbre (classifieur faible) ajouté à chaque étape et contribue à éviter un sur-ajustement.
• Approximation du gradient : à chaque itération, approcher le gradient par un développement de Taylor au second ordre.
• Recherche des divisions dans les arbres de régression : découper les variables quantitatives en classes en utilisant les quantiles de la distribution comme bornes.

3.3 Optimisation des paramètres

Pour chacun des algorithmes ci-dessus, on dispose d'un certain nombre de paramètres sur lesquels on peut jouer pour obtenir de meilleurs résultats de prédiction. L'un d'entre eux est le *shrinkage* ou encore *learning rate*.

Shrinkage : Le *shrinkage* est un réel compris dans $[0, 1]$ qui permet de pénaliser l'ajout d'un nouveau modèle dans l'agrégation et ralentit la convergence. En général, plus sa valeur est faible plus la prédiction est bonne mais plus longue est la convergence. De façon générale, les algorithmes de boosting sont des algorithme qui peuvent effectivement converger exactement, donc éventuellement tendre vers une situation de sur-apprentissage. En pratique, cette convergence peut être rendue suffisamment lente pour être mieux contrôlée.

On dispose en outre d'autres paramètres comme la profondeur maximale des arbres (classifieurs faibles) ou encore le nombre de classifieurs considérés.
Dans le cas de l'algorithme XGBoost, l'ajout d'un terme de pénalisation dans la fonction de perte augmente le nombre de paramètres à notre disposition pour améliorer le modèle ou optimiser les temps de calcul.

4 Implémentations et résultats

Pour comparer les performances des algorithmes de boosting, nous utilisons dans un premier temps des données simulées, puis dans un second temps des données réelles issues du package sklearn. Plus précisément, nous allons comparer la précision des classifieurs en fonction du nombre de classifieurs faibles considérés. Les algorithmes étudiés sont les suivants : **AdaBoostClassifier** du package sklearn, **GradientBoostingClassifier** du package sklearn et une version *from scratch* d'AdaBoost disponible dans le repository GitHub.

4.1 Données simulées

Les données simulées sont générés à partir d'une loi normale pour 5000 observations et 30 features.

Les algorithmes de boosting sont plus efficaces qu'une simple régression logistique sur ces données.

4.2 Données réelles

Dans cette sous-section, on utilisera deux bases de données : la première est celle (bien connue) de la détection du cancer du sein inclue dans sklearn. Pour ce dataset, on sait déjà que les algorithmes classiques tels que la régression logistique fonctionnent bien. Il convient donc de tester l'efficacité du boosting sur une autre base de données : un recensement des accidents de la route aux USA sur une période, la variable cible étant la gravité de l'accident (modalité 1 si l'accident est grave, 0 sinon).

Données relatives au cancer du sein Pour ces données, on obtient des scores de précision des modèles très élevés, même pour un nombre restreint de classifieurs faibles pris en compte. Plus présicément, nous obtenons moins de 2% d'erreur pour l'algorithme AdaBoost (5 classifieurs faible, learning rate égal à 1) et moins de 3% d'erreur pour Gradient Boosting (mêmes paramètres).

Données des accidents de la route Pour ce dataset, la régression logistique donne une précision de classfication environ égale à 0.81. En faisant varier les paramètres des modèles de Boosting, nous obtenons les figures 1 et 2 ci-dessous :

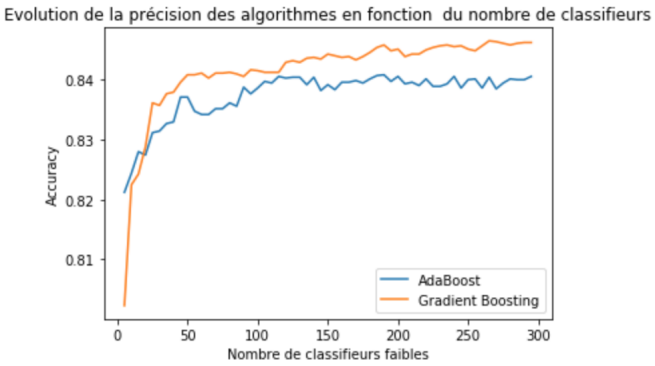

Fig. 1. Évolution de l'accuracy des modèles en fonction du nombre de classifieurs faibles

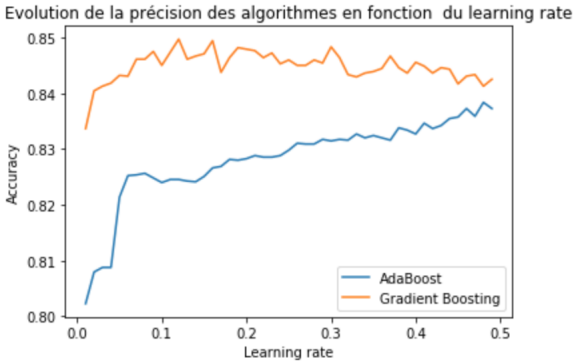

Fig. 2. Évolution de l'accuracy des modèles en fonction du shrinkage à nombre de classifieurs fixé

Comme on peut le constater ci-dessus, sur ces données les algorithmes de Boosting sont plus efficaces que la régression logistique.

Conclusion

Cette étude sur le boosting et différentes variantes des algorithmes qui en découle permet de rendre compte de l'efficacité d'une combinaison de mécanismes simples. Les analogies et similitudes entre domaines mathématiques ou algorithmiques (modèle additif, descente de gradient) sont une clé de l'innovation scientifique.

Bibliographie

- Chapitre 10 de "The Elements of Statistical Learning", T. Hastie, R. Tibshirani et J. Friedman, Springer 2007.
- Friedman, J., Hastie, T., Tibshirani, R. (2000). Additive logistic regression : a statistical view of boosting. The Annals of Statistics, 28(2), 337-407.
- Friedman, J. H. (2001). Greedy function approximation : a gradient boosting machine. Annals of statistics, 1189-1232.
- XGBoost: A Scalable Tree Boosting System, Tianqi Chen Carlos Guestrin, 2016.
-