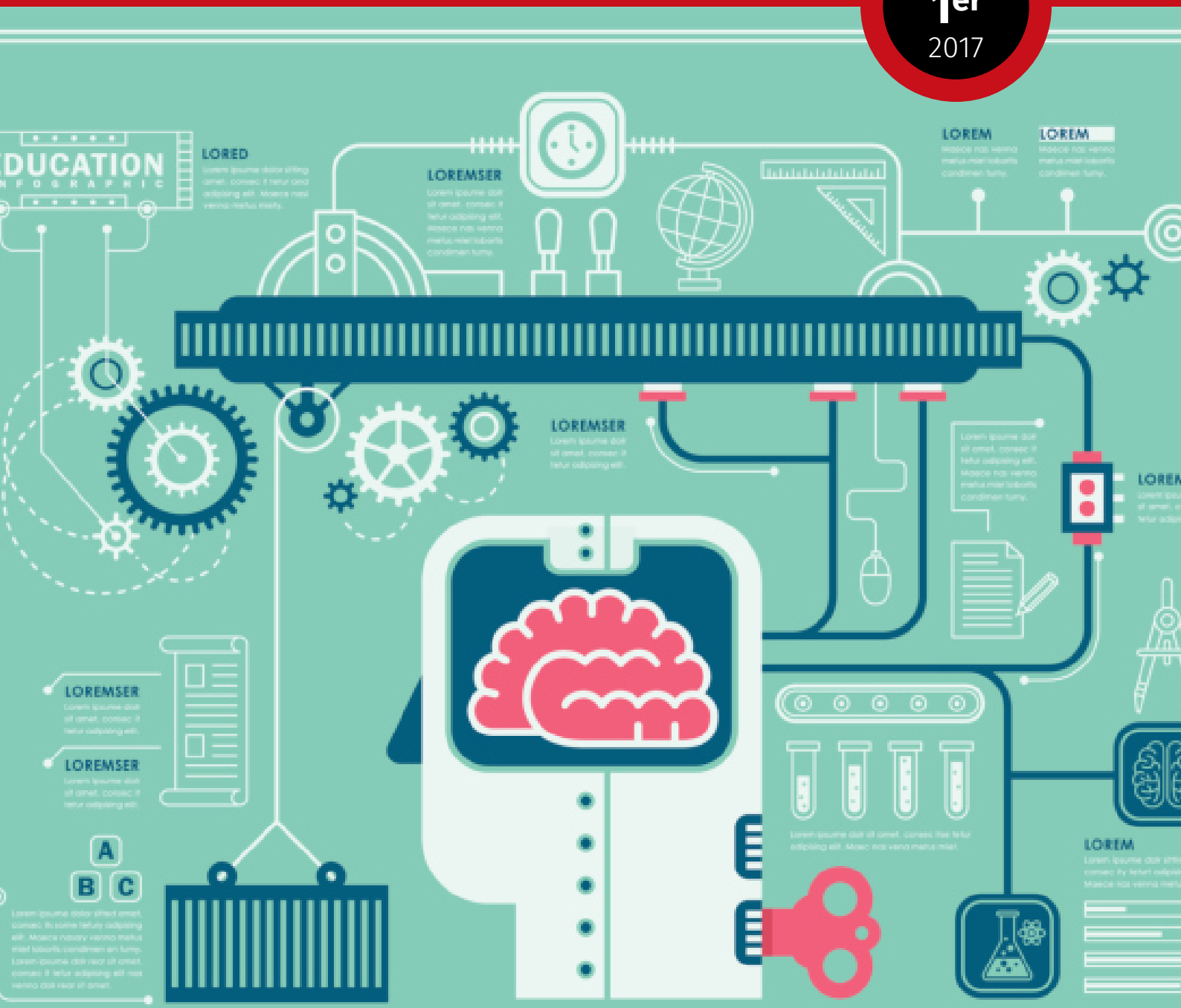
XGBoost
Origine et applications

Principe de la méthode

Mars
1er
2017



1. XGBoost en deux mots

L'objectif de cet article est de présenter XGBoost. Mais qu'est-ce que XGBoost ? Il s'agit d'une méthode de machine learning apparue il y a trois ans et dérivant des méthodes dites de boosting. Cette méthode est présentée par ses créateurs comme étant :

- Flexible Prise en compte de plusieurs thématiques de machine learning.
- Portable Utilisable sous toutes les plateformes (Windows, Linux, MAC)
- Multi-langages L'algorithme a été porté en Python, JAVA (Spark), C++,...
- Distribuée Depuis deux ans, l'algorithme est utilisable avec Hadoop et Spark.
- Perfomante Cet algorithme est donné pour être plus rapide que les algorithmes de sa famille et fournir de même meilleurs résultats.

Mais avant toute chose, que veut exactement dire l'acronyme « XGBoost » ?

E**X**treme **G**radiant **Boost**ing

Ainsi, la méthode XGBoost s'organise autours de trois points essentiels :

- *Boosting* Il s'agit d'une famille d'algorithme utilisés initiallement en apprentissage supervisé. Le principe du boosting sera détaillé en Section 2.2.
- *Gradient Boosting* Il s'agit d'une version du boosting dans laquelle l'objectif sera d'optimiser une fonctione faisant apparaître des gradients. Les détails de cette idée seront détaillées en Section 2.5.
- *« Extreme »* Ce qualificatif signifie que la recherche de performances est poussée au maximum pour cette méthode, comme nous l'étudierons en Section 3.2.

Ainsi, nous allons dans un premier temps présenter les grandes lignes théoriques derrière cet algorithme avant de partir dans l'étude des implémentations et des applications de XGBoost.

En bref

2. Le boosting

2.1 • Qu'est-ce que le boosting ?

Le boosting est une méthode de Machine Learning apparue à la fin des années 1980 et ayant évoluée au fil du temps en plusieurs version, les principales étant AdaBoost ou encore les GBM (*Gradient Boosted Models*)[1].

Le succès de ces méthodes provient de leur manière originale d'utiliser des algorithmes déjà existant au sein d'une stratégie adaptative. Cette stratégie leur permet alors de convertir un ensemble de règles et modèles peu performant en les combinant pour obtenir de (très) bonnes prédictions. L'idée principale est en effet d'ajouter de nouveaux modèles au fur et à mesure, mais de réaliser ces ajouts en accord avec un critère donné. En ce sens, cette famille de méthodes se différencie des Random Forest qui vont elles miser sur l'aléatoire pour moyenner l'erreur.

[1] Historique du Boosting

1989
Proposition de la méthode et premier algorithme par R. SCHAPIRE

1996
Première implémentation d'AdaBoost par Y. FREUND et R. SCHAPIRE

1999
Apparition du Boosting de gradient (GBM) par L. BREIMAN et J. FREIDMAN

2014
Apparition et implémentation de XGBoost par T. CHEN

Cet aspect fondamental du boosting permet ainsi une forte réduction du biais et de la variance de l'estimation, mais surtout garanti une convergence rapide. La contrepartie se fait au niveau de la sensibilité au bruit comme nous le verrons dans la description des algorithmes.

2.2 • Premier algorithme

Pour commencer, nous allons présenter l'idée originale de l'algorithme de Boosting présenté par R. SCHAPIRE en 1989. Comme nous l'avons précisé, l'idée est de construire de « mauvais » classifieurs au fur et à mesure qui combiné fourniront un excellent classifieur.

Pour nos exemples, nous allons considérer la situation de la Figure 1. Le problème a traiter ici est un problème de classification supervisé, c'est-à-dire que l'on désire entraîner un classifieur pour pouvoir séparer les carrés des cercles.

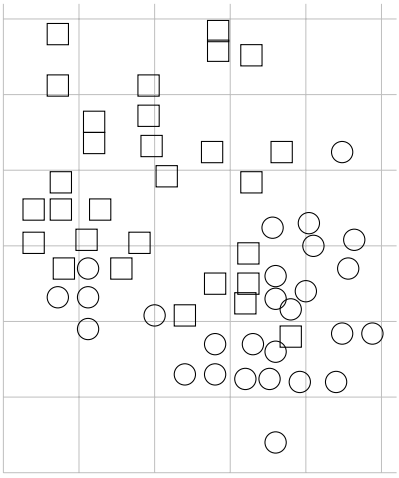
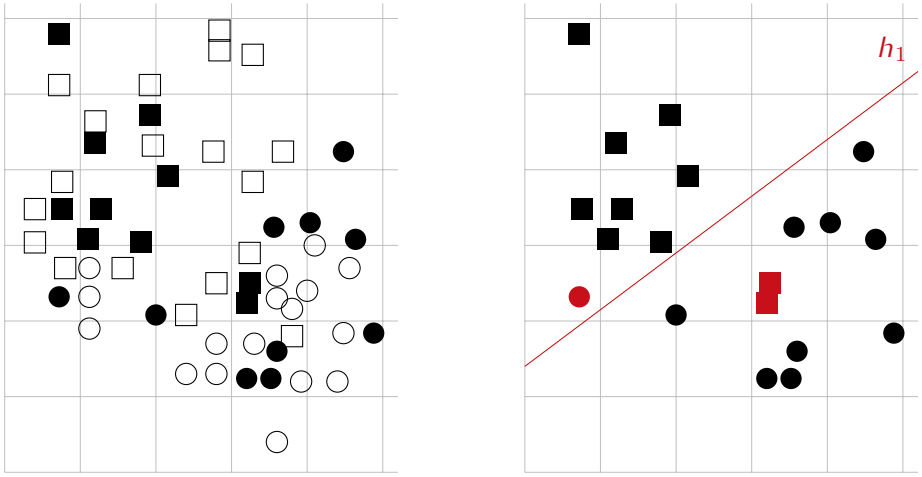


FIGURE 1 • Deux ensembles à séparer par le modèle que l'on entraîne (d'après [**?**])

Pour commencer, l'idée est d'entraîner un premier classifieur. Cependant, pour retirer une partie du biais des données, ce dernier ne sera entraînée que sur une partie des données (choisie arbitrairement). Nous retrouvons ainsi en Figure **??** l'ensemble d'apprentissage qui est en rouge, et en Figure **??** le classifieur appris sur ces données, notés h_1.



(a) Données d'apprentissage choisies aléatoirement (■ ●)

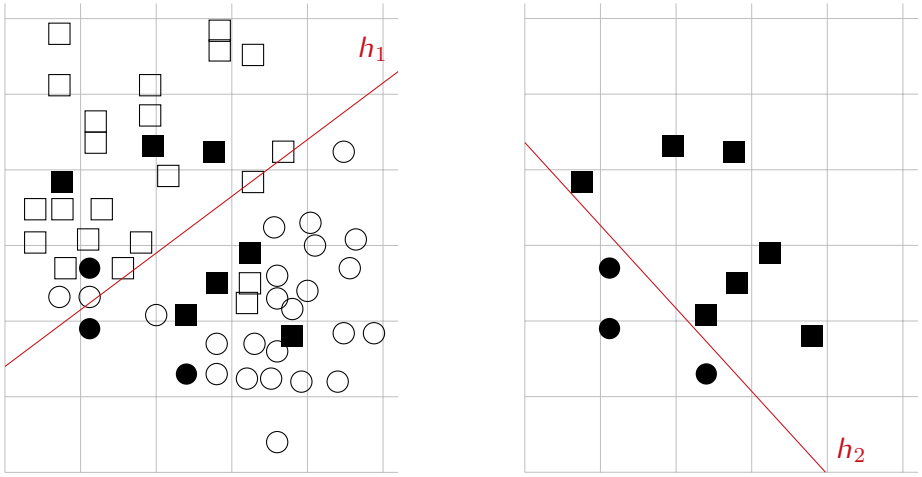
(b) Classifieur h_1 obtenu par entraînement sur l'ensemble ■ ●

FIGURE 2 • Premier modèle appris sur les données (d'après [**?**])

Sans surprise, ce classifieur n'est pas parfait et réalise un certain nombre d'erreurs (indiquées en rouge sur la Figure **??**). Toute l'idée du Boosting

est alors de créer de nouveaux modèles en changeant l'ensemble d'apprentissage de manière « intelligente ».

Dans notre cas, nous allons choisir un nouvel ensemble d'apprentissage qui regroupe des éléments n'étant pas dans l'ensemble initial et pour lesquels les prévisions de h_1 sont équiprobablement correctes et incorrectes. Cet ensemble est représenté sur la Figure **??**. On apprend alors à partir de cet ensemble un nouveau classifieur h_2 présenté à la Figure **??**.



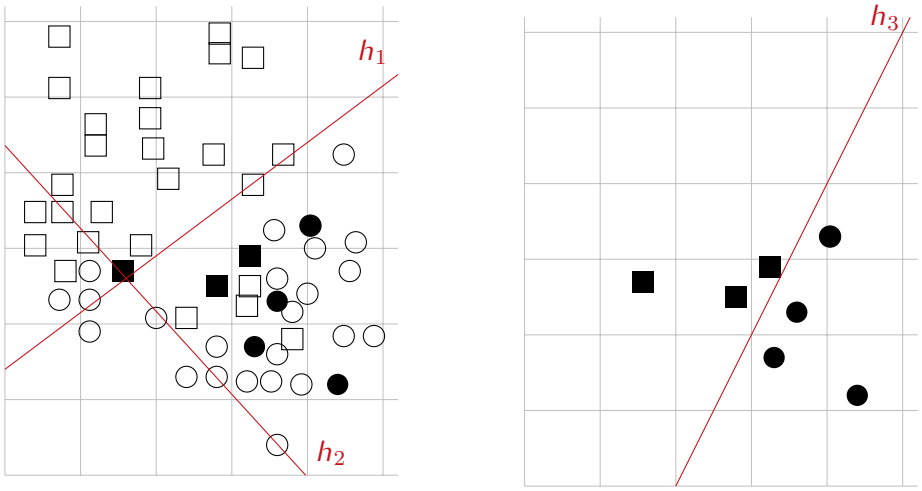
(a) Données d'apprentissage pour le deuxième classifieur (■ ●)

(b) Classifieur h_2 obtenu par entraînement sur l'ensemble ■ ●

FIGURE 3 • Deuxième modèle appris sur les données (d'après [**?**])

On remarque qu'ici le modèle h_2 ne fait pas d'erreurs sur son ensemble d'apprentissage (qui est cependant peu fourni), ceci est un cas particulier et n'est pas automatique. À cette étape, nous disposons donc de deux modèles, le second permettant de « rattraper » des erreurs du premier.

Nous continuons alors sur le même principe, en entraînant un troisième classifieur. Son ensemble d'apprentissage sera choisit dans les points n'ayant pas encore servis à l'apprentissage, et pour lesquels les deux autres classifieurs sont en désaccord, voir la Figure **??**. Le troisième classifieur, noté h_3 est représenté à la Figure **??**.



(a) Données d'apprentissage pour le deuxième classifieur (■ ●)

(b) Classifieur h_2 obtenu par entraînement sur l'ensemble ■ ●

FIGURE 4 • Troisième modèle appris sur les données (d'après [**?**])

À la fin de ce processus (qui aurait pu durer plus longtemps si nous

l'avions désiré), nous obtenons trois classifieurs. Comme cela a été vu, ces derniers ont été créés de manière à ce qu'ils corrigent leurs erreurs entre eux, ce paramètre étant pris en compte par la construction des ensembles d'apprentissage. Ainsi, l'utilisation de tous ces classifieurs revient pour chaque point à réaliser une prédiction avec tous les classifieurs et d'attribuer la classe ayant le plus de vote. On moyenne ainsi les erreurs des classifieurs entre eux et on réduit ainsi le biais de la prédiction. Cette décision finale est illustrée à la Figure 5, où les erreurs sont précisées en rouge.

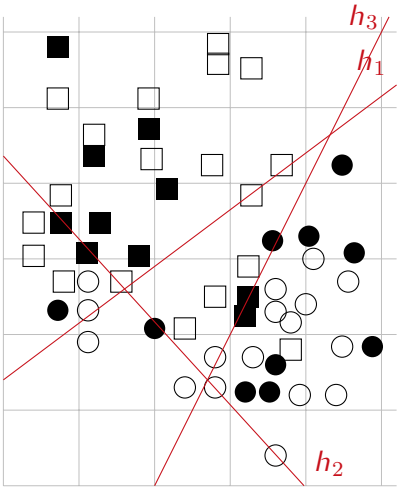


FIGURE 5 • Prévision finale conjointe des trois classifieurs entraînés par Boosting (d'après [**?**])

Cette première version de l'algorithme de Boosting a subit de nombreuses améliorations jusqu'à aujourd'hui, les plus notables étant :

- *Adaboost* Cette version de l'algorithme change la manière de construire l'ensemble d'apprentissage. Au lieu de prendre des sous-ensembles d'apprentissage, cette méthode propose de pondérer très fortement les éléments mal classés, et dans le même temps de pondérer faiblement les éléments bien classés. L'idée est alors qu'à chaque étape on va principalement apprendre sur les données en erreurs, et contrebalancer les votes précédents. Cette méthode va donc se focaliser sur les cas « difficiles » de l'ensemble d'apprentissage. Si cette solution permet de converger très rapidement et avec grande précision, on voit également qu'elle va s'efforcer au maximum de bien classer tous les éléments, y compris ceux étant abhérents. Ainsi, cette méthode va augmenter l'improtance des exemples mal-classés, isolés (outliers) ou encore mal-enregistrés.
- *Gradient Boosting Machine (GBM)* Une version un peu remaniée de ces derniers sera abordée à la Section 2.5.

2.3 • Perte et complexité

Avant d'aborder la question des GBM, nous allons nous intéresser à deux grands concepts importants en machine learning et qui sont tout particulièrement gardés à l'œil dans le cas de XGBoost et du boosting de manière générale.

Nous avons vu que l'idée du Boosting est d'adapter l'ensemble d'apprentissage à chaque étape pour chercher des classifieurs qui vont se compléter, ie. moyenner leurs erreurs. Il s'agit en fait de « coller aux données » du mieux que possible (mais sans sur-apprendre !). Nous verrons que cet aspect est lié à la notion de perte.

De même, nous avons vu que l'idée n'est pas d'apprendre des modèles intermédiaires très puissants, mais que l'ensemble soit performant. Ainsi, nous verrons que le fait de prendre des modèles simples est lié à la notion de complexité.

2.4 • Fonction de perte

Avant d'aller plus loin, regardons formellement quelle est l'idée théorique derrière le Boosting. Ce dernier cherche en fait à optimiser (minimiser) une fonction objectif. Cette fonction peut s'écrire en deux termes :

$$\text{objectif}(\Theta) = \mathcal{L}(\Theta) + \Omega(\Theta) \tag{1}$$

La relation (1) fait apparaître deux termes :

- $\mathcal{L}(\Theta)$ La fonction de perte
- $\Omega(\Theta)$ La fonction de complexité

Ce sont ces deux fonctions (et leur intérêt !) qui vons nous intéresser par la suite. Pour cela, nous allons considérer la situation cobaye de la Figure 6.

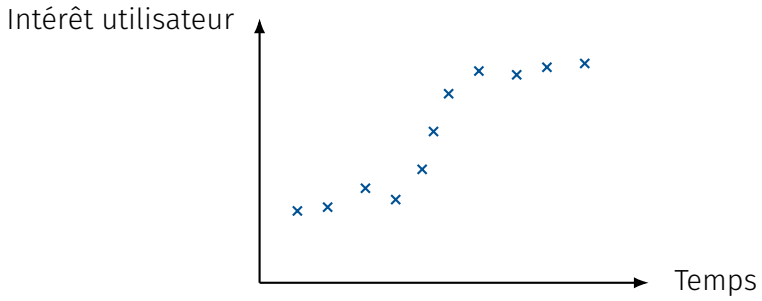


FIGURE 6 • Situation de test pour comprendre les notions de perte et complexité (d'après [**?**])

2.4.1 *Notion de perte*

2.4.2 *Notion de complexité*

2.5 • Gradient Boosting

3. Plus que du boosting

3.1 • Importance des variables

3.2 • Performances

4. Mise en œuvre

4.1 • XGBoost

4.2 • Les paramètres

4.2.1 *Paramètres génériques*

4.2.2 *Paramètres de Boosting*

4.2.3 *Paramètres d'apprentissage*