Présentation de l'article *An efficient boosting algorithm for combining preferences* (Y. Freund et al., 1998)

François Rousseau Jérémie Decock

13 octobre 2010

1 Résumé

Nous ferons ici une synthèse des travaux de Freund et Schapire sur l'algorithme Rankboost présentés dans l'article *An efficient boosting algorithm for combining preferences* [FISS98]. Dans un souci de clarté, nous présenterons au préalable les principes généraux des méthodes de boosting ainsi que l'algorithme de référence *Adaboost*.

2 Introduction

Face à un problème de classification complexe, il est difficile de construire un classifieur qui soit à la fois efficace sur les données d'apprentissage et capable de généraliser correctement sur de nouvelles données. Il est en revanche plus facile de construire un classifieur « faible », raisonnablement efficace (ie. classant un peu mieux que le hasard), sur un sous-ensemble des données. Le boosting consiste à résoudre un problème complexe en combinant intelligemment plusieurs classifieurs faibles (Fig. 1).

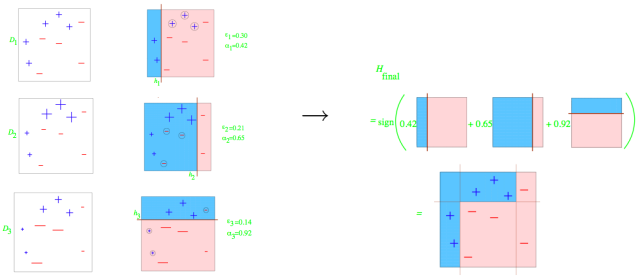

Fig. 1: Illustration du fonctionnement d'un algorithme de boosting sur un problème de classification binaire

Le premier algorithme de boosting a été développé par SCHAPIRE en 90 [Sch90] pour répondre à une question de KEARNS [Kea88] : est-il possible de rendre aussi bon que souhaité un algorithme d'apprentissage « faible » ? SCHAPIRE montra qu'il est toujours possible d'améliorer la performance d'un algorithme faible en l'entraînant sur un ensemble d'exemples bien choisis. Fort de ces résultats, il a poursuivi ses travaux avec FREUND et mis au point en 1995 un algorithme de classification binaire, devenu la référence des algorithmes de boosting : AdaBoost [FS95]. Cet algorithme a servi de base à beaucoup d'autres, parmi lesquels RankBoost [FISS98], un algorithme utilisé pour résoudre des problèmes de Ranking.

3 AdaBoost

Adaboost est l'algorithme le plus populaire de la famille du boosting. Il est essentiellement utilisé pour résoudre des problèmes de classification binaire. Trois idées fondamentales sont à la base d'Adaboost :

1. l'utilisation d'un comité d'experts spécialisés (hypothèses faibles h_t) que l'on fait voter pour atteindre une décision H
2. la pondération adaptative α_t des votes en fonction du taux d'erreur de chaque expert
3. la modification de la distribution D des exemples disponibles pour entraîner chaque expert, en surpondérant au fur et à mesure les exemples mal classés aux itérations précédentes

Algorithme 1 AdaBoost

Entrées: un ensemble d'apprentissage $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$
avec $y_i \in \{+1, -1\}, i = 1, \ldots, m$,
le nombre d'itérations T,
la distribution initiale $D : D(i) = \frac{1}{m}, i = 1, \ldots, m$,
un algorithme d'apprentissage faible *WeakLearn*
Sorties: l'hypothèse finale $H(\boldsymbol{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$
Initialiser la distribution $D_1 \leftarrow D$
pour $t = 1$ to T **faire**
Tirer un échantillon d'apprentissage $\mathcal{S}_t$ dans $\mathcal{S}$ selon D_t
Trouver une *hypothèse faible* h_t qui minimise l'erreur ϵ_t sur $\mathcal{S}_t$
Calculer le poids α_t de h_t : typiquement $\alpha_t \leftarrow \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$
pour tout $i = 1, \ldots, m$ **faire**
Mettre à jour la distribution de probabilité de l'exemple $\boldsymbol{x}_i$:
$D_{t+1}(i) \leftarrow \frac{D_t(i)\, e^{-\alpha_t y_i h_t(\boldsymbol{x}_i)}}{Z_t}$, avec Z_t un facteur de normalisation
fin pour
$t \leftarrow t + 1$
fin pour

4 RankBoost

L'algorithme Rankboost – qui est l'objet de l'article étudié ici – ne traite pas les problèmes de classification comme Adaboost mais les problèmes de *ranking*. Dans ce type de problèmes, on cherche à ordonner un ensemble d'instances $\mathcal{X} = \{x_0, \ldots, x_m\}$ les unes par rapport aux autres en leur donnant un *rang*. C'est la fonction $H : \mathcal{X} \to \mathbb{R}$ qui définit le rang de chaque instances. Cette fonction est apprise à partir d'un ensemble de *caractéristiques* $\mathcal{F} = \{f_0, \ldots, f_n\}$ avec $f_i(x_0) > f_i(x_1)$ si « la caractéristique i préfère x_0 à x_1 » et $f_i(x) = \phi$ si x ne peut pas être classé selon f_i. Nous verrons par la suite que ces fonctions servent de base à l'algorithme d'apprentissage faible générant les *hypothèses faibles* h_t.

La relation d'ordre à approximer est définie par la fonction de *feedback* $\Phi : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. $\Phi(x_0, x_1)$ traduit le degré suivant lequel l'instance x_1 est préféré à x_0 :

$$\begin{cases} \Phi(x_0, x_1) > 0 & \text{si} \quad x_1 \succ x_0 \\ \Phi(x_0, x_1) < 0 & \text{si} \quad x_0 \succ x_1 \\ \Phi(x_0, x_1) = 0 & \text{si} \quad x_0 \sim x_1 \end{cases}$$

avec $\Phi(x, x) = 0 \ \forall x \in \mathcal{X}$ et $\Phi(x_0, x_1) = -\Phi(x_1, x_0) \ \forall x_0, x_1 \in \mathcal{X}^2$ (la transitivité n'est pas assurée).

La fonction de feedback Φ et les caractéristiques f_i sont données en entrée de l'algorithme Rankboost et servent à construire l'hypothèse finale H.

Les couples d'instances sont pondérés par une distribution de probabilité où tous les poids négatifs sont annulés et donc ignorés (la matrice Φ est antisymétrique et les poids négatifs n'apportent aucune information supplémentaire) :

$$D(x_i, x_j) = c.max(0, \Phi(x_i, x_j))$$

avec c une constante positive de normalisation telle que

$$\sum_{x_0, x_1} D(x_0, x_1) = 1$$

Comme pour AdaBoost, la distribution D permet de se concentrer sur les paires d'instances mal ordonnées par les hypothèses construites au cours des itérations précédentes.

Le but de l'algorithme Rankboost est de trouver une hypothèse forte H qui minimise le critère *rloss* suivant D

$$\text{rloss}_D(H) = \sum_{x_0, x_1} D(x_0, x_1)[H(x_1) \leq H(x_0)]$$

où $[H(x_1) \leq H(x_0)]$ vaut 1 si $H(x_1) \leq H(x_0)$ est bien prédit et 0 sinon.

Algorithme 2 RankBoost

Entrées:
la fonction de feedback Φ,
un ensemble de *caractéristiques* $\mathcal{F} = \{f_0, \ldots, f_n\}$,
le nombre d'itérations T,
la distribution initiale D,
un algorithme d'apprentissage faible *WeakLearn*

Sorties: l'hypothèse finale $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$

```
Initialiser la distribution D_1 ← D
pour t = 1 to T faire
    Apprendre une hypothèse faible h_t : X → R avec WeakLearn
    Calculer le poids α_t ∈ R de h_t
    pour tout i = 1, ..., m faire
        pour tout j = 1, ..., m faire
            Mettre à jour la distribution de probabilité du couple (x_i, x_j) :
            D_{t+1}(x_i, x_j) ← D_t(x_i,x_j) exp(α_t(h_t(x_i)−h_t(x_j))) / Z_t
            avec Z_t un facteur de normalisation
        fin pour
    fin pour
    t ← t + 1
fin pour
```

Pour minimiser le critère *rloss*, nous devons construire à chaque itération une hypothèse h_t de sorte à minimiser

$$Z_t = \sum_{x_0, x_1} D_t(x_0, x_1) \exp(\alpha_t(h_t(x_0) - h_t(x_1)))$$

ce qui revient à maximiser la valeur $|r_t|$ avec

$$r_t = \sum_{x_0, x_1} D_t(x_0, x_1)(h_t(x_0) - h_t(x_1))$$

et à définir α_t tel que $\alpha_t = \frac{1}{2} \ln\left(\frac{1+r_t}{1-r_t}\right)$ si $h_t : \mathcal{X} \to [0, 1]$.

La méthode la plus simple et la plus évidente pour construire les hypothèses faibles h_t serait de les définir comme des *caractéristiques* f_i, avec une valeur par défaut q_{def} attribuée aux instances non évaluées. Mais combiner plusieurs *caractéristiques* peut s'avérer difficile, car toutes n'utilisent pas forcément la même échelle de notes. Afin de tenir compte uniquement de la relation d'ordre, les hypothèses faibles sont définies de la façon suivante :

$$h(x) = \begin{cases} 1 & \text{si} \quad f_i(x) > \theta \\ 0 & \text{si} \quad f_i(x) \leq \theta \\ q_{\text{def}} & \text{si} \quad f_i(x) = \phi \end{cases} \quad \text{avec } \theta \in \mathbb{R} \text{ et } q_{\text{def}} \in \{0, 1\}$$

Ainsi, à chaque itération t de l'algorithme, l'hypothèse faible h_t est construite en choisissant une *caractéristique* dans $\mathcal{F}$, un seuil θ et une valeur par défaut q_{def} de sorte à maximiser r_t.

5 Résultats

Les performances de Rankboost sont évaluées à travers deux problèmes. Pour des raisons de place, nous ne détaillerons que le second : la création d'un système de recommandation collaboratif de films pour un *utilisateur cible*, Bob.

Le principe est le suivant :
- le système demande à Bob d'évaluer une liste de films qu'il a déjà vu ;
- les notes attribuées par les autres utilisateurs de la base sont examinées et utilisées pour retourner une liste de nouveaux films supposés plaire à Bob ;
- on s'intéresse aux utilisateurs dont les préférences sont similaires à celles de Bob et on combine leurs recommandations sur les films qu'il n'a pas encore vu.

Les données utilisées sont tirées de la base *EachMovie*[1] de *Digital Equipement Corporation*.

Dans cet expérience, les *instances* sont des films et $\mathcal{X}$ représente l'ensemble de tous les films connus par le système.

Les *caractéristiques* $f_i(x)$ correspondent aux notes accordées par l'utilisateur i sur le film x ($f_i(x) = \phi$ si le film x n'a pas été évalué par l'utilisateur i). Ainsi, $f_i(x_1) > f_i(x_0)$ si l'utilisateur i a préféré le film x_1 au film x_0. Les notes sont prises dans l'ensemble $R = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

La fonction de *feedback* Φ traduit les préférences de Bob. $\Phi(x_0, x_1) = 1$ si il préfère le film x_1 au film x_0 et -1 dans le cas contraire. Si Bob n'est pas capable de discriminer les deux films (ie. si il ne les a pas tous vus ou si il n'a pas de préférence) alors $\Phi(x_0, x_1) = 0$.

L'hypothèse forte $H : \mathcal{X} \to \mathbb{R}$ attribue un rang aux films. Elle est censé représenter les préférences de Bob sur tous les films, y compris (et surtout) sur ceux qu'il n'a pas vu. $H(x_1) > H(x_0)$ si il est supposé préférer x_1 à x_0.

Les performances de Rankboost sur cette expérience sont mesurées à deux autres algorithmes (l'algorithme du *plus proche voisin* et un algorithme de régression), suivant quatre critères d'évaluation (voir l'article pour plus de détails). Rankboost obtient les meilleurs résultats sur ces quatre critères.

[1] `http://www.research.digital.com/SRC/eachmovie/`

6 Conclusion et critiques

Le boosting est une méthode populaire qui permet d'améliorer sensiblement les performances dès lors que l'hypothèse faible est bien choisie. Il est d'autant plus intéressant qu'il offre des garanties théoriques sur l'erreur en généralisation et qu'il ne nécessite pas de connaissance à priori sur l'algorithme d'apprentissage « faible » utilisé.

Une des principales faiblesses du boosting est sa sensibilité au bruit car les exemples aberrants (*outliners*) obtiennent un poids exponentiellement grand lors de l'apprentissage. Les performances peuvent être grandement affectées lorsque de nombreux exemples sont bruités. Bien utilisé, ce phénomène peut toutefois devenir une force car il permet de détecter facilement les exemples aberrants.

Malgré les qualités discernables de Rankboost, on peut émettre quelques réserves quand à l'évaluation des résultats obtenus. On peut notamment reprocher à Freund et Schapire le choix non justifié et à priori arbitraire de certaines valeurs utilisées pour la mesure de performance (comme le seuil *truly top-rated instances*). Le choix d'algorithmes très simplifiés pour positionner Rankboost face à l'existant mérite également d'être souligné (le *plus proche voisin* au lieu de des *K plus proches voisins*).

Références

[FISS98] Y. Freund, R.D. Iyer, R.E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 170–178. Morgan Kaufmann Publishers Inc., 1998.

[FS95] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[Kea88] M. Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript, December*, 1988.

[Sch90] R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2) :197–227, 1990.