Review on boosting algorithms

Vu Tuan Hung and Do Quoc Khanh

10 mai 2012

1 Introduction

2 Two-class classification

In this first part, we present an overview on boosting methods in the two-class classification framework. From a *training* set $(\mathbf{x}_i, y_i)_{i=1,...,N}$ in which $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, we try to construct a function $F : \mathcal{X} \to \mathcal{Y}$ so that when a new value $\mathbf{x}$ is randomly introduced, we have the highest probability to predict correctly the value y corresponding to this value of $\mathbf{x}$. Formally, we want to minimize the probability :

$$\mathbb{P}_{(\mathbf{x},y)} \left(y \neq F(\mathbf{x}) \right)$$

The variable $\mathbf{x}$ is called explanatory variables ($\mathbf{x}$ may be multi-variational) and y is called response variable. In the two-class classification framework, $\mathcal{Y} = \{-1, 1\}$.

2.1 Boosting and optimization in function space

We exploit the point of view presented in [2] by considering this problem as an estimation and optimization in function space. Indeed, if there exists a function F^* which minimizes the above error :

$$\begin{aligned} F^* &= arg \min_F \mathbb{P}_{(\mathbf{x},y)}(y \neq F(\mathbf{x})) \\ &= arg \min_F \mathbb{E}_{(\mathbf{x},y)} \left[1(y \neq F(\mathbf{x})) \right] \end{aligned}$$

then we are trying to estimate F^* by a function $\hat{F}$ through the training set $(\mathbf{x}_i, y_i)_{i=1,...,N}$.

Base classifiers. An approach frequently employed by classification algorithms is to suppose F^* belongs to a function class parameterized by $\theta \in \Theta$:

$$F^* \in \mathcal{Q} = \{F(., \theta) | \theta \in \Theta\}$$

so that the problem of estimating F^* becomes an optimization of the parameters on Θ :

$$\hat{\theta} = arg \min_{\theta \in \Theta} \mathbb{E}_{(\mathbf{x},y)} \left[1(y \neq F(\mathbf{x}, \theta)) \right]$$

and then we will take $\hat{F} = F(., \hat{\theta}) \in \mathcal{Q}$. For example, with regression tree algorithms, we have :

$$\mathcal{Q} = \left\{ F(x, \theta) = \sum_{k=1}^{K} \lambda_k 1(\mathbf{x} \in R_k) | (\lambda_1,...,\lambda_K) \in \mathbb{R}^K, (R_1,...,R_K) \in \mathcal{P}_{\mathcal{X}} \right\}$$

in which $\theta = (\lambda_{1:K}, R_{1:K})$ and $\mathcal{P}_{\mathcal{X}}$ is the set of all partitions of $\mathcal{X}$ into K disjoint subsets by hyperplans which are orthogonal to axes. Similarly for support vector machines, K disjoint subsets R_1,...,R_K are divided by hyperplans in the reproducing kernel Hilbert space of $\mathcal{X}$ corresponding to some kernel.

We can see that a classifier is characterized by its function sub-space $\mathcal{Q}$ and the corresponding parameter space. Having the base classifiers $\mathcal{Q}_{1:M}$ with parameter spaces $\Theta_{1:M}$, instead of considering each of these classifiers separately, boosting methods consider functions of the following additive form :

$$\hat{F} \in \mathcal{F}_{\mathcal{Q}_1,...,\mathcal{Q}_M} = \left\{ \sum_{m=1}^{M} \beta_m f(.,\theta_m) | \theta_m \in \Theta_m, \forall m = 1,...,M \right\}$$

so that the optimization problem becomes :

$$\left\{\hat{\beta}_{1:M}, \hat{\theta}_{1:M}\right\} = arg \min_{\beta\in\mathbb{R}^M, \theta_{1:M}\in\Theta_{1:M}} \mathbb{E}_{(\mathbf{x},y)} \left[1(y \neq F(\mathbf{x}; \beta_{1:M}, \theta_{1:M}))\right] \tag{1}$$

Friedman, J. and Hastie, T. in [1] explained boosting as a forward stepwise algorithm for resolve the optimization problem (1). Friedman, J. in [2] considered boosting like optimization algorithm in function space. We will try to adopt the latter to explain all mentioned boosting algorithms. Before going into greater details, we remark that as an classification algorithm, the boosting algorithm has its corresponding function subset which is $\mathcal{F} = \mathcal{F}_{\mathcal{Q}_1,...,\mathcal{Q}_M} = \sum_{m=1}^{M} \mathcal{Q}_m$ so much larger than function subset of all base classifiers, explaining the dominating performance of boosting compared to its base classifiers.

Loss function. We remark that the binary loss function $1(y \neq F(\mathbf{x}))$ is not the only function that reflects the difference between y and $F(\mathbf{x})$. In machine learning, one has other loss functions that are continuous, convex and then easier to do the optimization. For the rest of the report, we use in general $L(y, F(\mathbf{x}))$ to indicate this function.

Optimization on training set. One difficulty is that we can not evaluate the distribution of $(\mathbf{x}, y)$ and calculate the expectation in the right hand side formula of (1). Instead, we only want to optimize on the training data, which means the following optimization problem :

$$F^* = arg \min_F \sum_{i=1}^{N} L(y_i, F(x_i))$$

Put $Q(F) = \sum_{i=1}^{N} L(y_i, F(\mathbf{x}_i))$, we remark that $Q(F)$ depends only on N values of the function F at $(\mathbf{x}_1,...,\mathbf{x}_N)$. We denote for each function F in the function space, a corresponding vector $\overline{F} \in \mathbb{R}^N$ so that $\overline{F} = (F(\mathbf{x}_1),...,F(\mathbf{x}_N))$, and we consider the relaxation problem on vector space $\mathbb{R}^N$: $\overline{F^*} = arg \min_{\overline{F}\in\mathbb{R}^N} Q(\overline{F})$. We try to resolve this problem by recursive numerical methods. Suppose that at $m - 1^{th}$ step we obtain a value $\overline{F}_{m-1}$. By numerical methods (Newton-Raphson, algorithm of gradient descent etc.) we find a direction of descent $d_m \in \mathbb{R}^N$ and a coefficient c_m so that if we put $\overline{F}_m = \overline{F}_{m-1} + c_m.d_m$, then $Q(\overline{F}_m) \leq Q(\overline{F}_{m-1})$. But we can not use the direction d_m directly in the original problem with functions F_m because $\overline{F}_m$ identifies the values of functions only at N points. Instead, we have to find a regression function near to the direction d_m ; it means if we use a function subspace $\mathcal{Q}_m$ at m^{th} step, then we have to solve :

$$\{f_m, c\} = arg \min_{f_m\in\mathcal{Q}_m, c\in\mathbb{R}_+} \|d_m - c.\overline{f_m}\|^2$$

in which $\overline{f_m}$ is the vector of values of f_m at $(\mathbf{x}_1,...,\mathbf{x}_N)$. After that, we have to look for a coefficient β_m so that, if F_{m-1} is the function obtained at the precedent step, then $Q(F_m) \leq Q(F_{m-1})$

with $F_m = F_{m-1} + \beta_m f_m$. If we start with $F_0 = 0$ then we obtain at M^{th} the additive form $F_M = \sum_{m=1}^{M} \beta_m f_m \in \mathcal{F} = \sum_{m=1}^{M} \mathcal{Q}_m$. We summary this generic algorithm in the following table.

Algorithm 1 : Generic algorithm of boosting

```
1. Start with $F_0(\mathbf{x}) = 0$.
2. Repeat for $m = 1,2,...,M$ :
    (a) Search for a direction descent $d_m \in \mathbb{R}^N$ by some Newton-like numerical algorithm of optimization in $\mathbb{R}^N$.
    (b) Solve $\{f_m, c\} = arg \min_{f_m \in \mathcal{Q}_m, c \in \mathbb{R}_+} \|d_m - c.\overline{f_m}\|^2$. The least-square criterion is not mandatory.
    (c) Search for a coefficient $\beta_m$ so that $Q(F_{m-1} + \beta_m f_m) \leq Q(F_{m-1})$, a line-search strategy can be used.
    (d) $F_m = F_{m-1} + \beta_m f_m$.
3. Conclude with $F(\mathbf{x})$.
```

In most cases, the direction of descent will be calculated from the gradient of $Q(\overline{F})$ at $\overline{F} = \overline{F}_{m-1}$ according to Newton-like optimization algorithms in $\mathbb{R}^N$:

$$d_m = -\frac{\partial Q}{\partial \overline{F}}(\overline{F}_{m-1}) = -\left(\frac{\partial L}{\partial \overline{F}_1}(y_1, \overline{F}_1), ..., \frac{\partial L}{\partial \overline{F}_N}(y_N, \overline{F}_N)\right)|_{\overline{F}_{1:N} = \overline{F}_{m-1}}$$

After having found f_m, the coefficient β_m is often evaluated by line-search procedure $\beta_m = arg \min_{\beta \in \mathbb{R}} Q(F_{m-1} + \beta f_m)$ in which F_{m-1} and f_m are functions from $\mathcal{X}$ to $\mathcal{Y}$. In the next sections, we will present and explain boosting algorithms following the same paradigm described above.

2.2 $L(y, F(\mathbf{x})) = e^{-yF(\mathbf{x})}$

This loss function is used in some of the most popular boosting algorithms like *Discrete AdaBoost* or *Real AdaBoost*; we use it as an example for our model. Firstly the direction descent at m^{th} step $d_m = \nabla_{\overline{F}} Q(\overline{F})|_{\overline{F} = \overline{F}_{m-1}} = \left(y_i e^{-y_i F_{m-1}(x_i)}\right)_{i=1:N}$. Following 2(b), we look for $f_m \in \mathcal{Q}_m$ and $c > 0$ which minimize :

$$S(f_m, c) = \sum_{i=1}^{N} \left(y_i e^{-y_i F_{m-1}(\mathbf{x}_i)} - c f_m(\mathbf{x}_i)\right)^2 \tag{2}$$

If we precise $\mathcal{Q}_m$ so that $f_m(\mathbf{x}) \in \{-1, 1\}, \forall \mathbf{x}$, we have :

$$S(f_m, c) = Nc^2 + \sum_{i=1}^{N} e^{-2y_i F_{m-1}(\mathbf{x}_i)} - 2c \sum_{i=1}^{N} e^{-y_i F_{m-1}(\mathbf{x}_i)} + c \sum_{i=1}^{N} e^{-y_i F_{m-1}(\mathbf{x}_i)} (y_i - f_m(\mathbf{x}_i))^2$$

so if we put $w_i = e^{-y_i F_{m-1}(\mathbf{x}_i)}$, we have to solve $arg \min_{f_m \in \mathcal{Q}_m} \sum_{i=1}^{N} w_i (y_i - f_m(\mathbf{x}_i))^2$ which is equivalent to classification of $\mathbf{x}_i$ using weights w_i. The line 2(c) is equivalent to :

$$\begin{aligned}
\beta_m &= arg \min_{\beta\in\mathbb{R}} \sum_{i=1}^{N} e^{-y_i(F_{m-1}(\mathbf{x}_i)+\beta f_m(\mathbf{x}_i))} \\
&= arg \min_{\beta\in\mathbb{R}} \sum_{i=1}^{N} w_i e^{-\beta y_i f_m(\mathbf{x}_i)} \\
&= arg \min_{\beta\in\mathbb{R}} \left(e^{-\beta} \times \sum_{y_i=f_m(\mathbf{x}_m)} w_i + e^{\beta} \times \sum_{y_i\neq f_m(\mathbf{x}_i)} w_i \right) \\
&= \frac{1}{2}\log\left(\frac{\sum\limits_{y_i=f_m(\mathbf{x}_i)} w_i}{\sum\limits_{y_i\neq f_m(\mathbf{x}_i)} w_i} \right)
\end{aligned}$$

we obtain Discrete AdaBoost algorithm described in the table below.

```
Discrete AdaBoost
1. Start with weights $w_i = 1/N, i = 1,...,N$.
2. Repeat for $m = 1,2,...,M$ :
(a) Fit the classifier $f_m(x) \in \{-1,1\}$ using weights $w_i$ on the training data.
(b) Compute $err_m = \mathbb{E}_w\left[1(y \neq f_m(\mathbf{x}))\right]$, and $\beta_m = \log\left(\frac{1-err_m}{err_m}\right)$.
(c) Set $w_i \leftarrow w_i \times e^{\beta_m \times 1(y_i\neq f_m(\mathbf{x}_i))}, i = 1,...,N$, and renormalize $w$.
3. Output the classifier $sign\left[\sum_{m=1}^{M} \beta_m f_m(\mathbf{x})\right]$.
```

Now we extend the subset $\mathcal{Q}_m$ to contain f_m of real values and not only the functions with discrete values $\{-1,1\}$. *Real AdaBoost* is a boosting algorithm that uses such base classifiers. In order to understand this algorithm, we note that *Real AdaBoost* does not use gradient as the direction of descent d_m, but find it directly from an optimization problem. The coefficient of descent β_m is taken value 1. Firstly, d_m is calculated by :

$$d_m = arg \min_{d\in\mathbb{R}^N} Q(\overline{F}_{m-1} + d) = arg \min_{d\in\mathbb{R}^N} \sum_{i=1}^{N} e^{-y_i(F_{m-1}(\mathbf{x}_i)+d^{(i)})}$$

In order to calculate exactly d_m as an estimate of some added regression function f_m's values, we remark that in $(\mathbf{x}_i, y_i)$, there may be many $\mathbf{x}_i$ taking a same values, so $(F(\mathbf{x}_i))_{i=1:N}$ and then d_m may not be real vectors in $\mathbb{R}^N$. By simplicity, we parameterize d_m by $\mathbf{x}$ (which are different values in $(\mathbf{x}_i)_{i=1:N}$) and not by $i = 1 : N$. We have :

$$d_m(\mathbf{x}) = arg \min_{d(\mathbf{x})\in\mathbb{R}} \sum_{\mathbf{x}_i=\mathbf{x}} e^{-y_i(F_{m-1}(\mathbf{x})+d(\mathbf{x}))}$$

We imply that $d_m(\mathbf{x}) = \frac{1}{2}\log\left(\frac{\sum\limits_{\mathbf{x}_i=\mathbf{x},y_i=1} w_i}{\sum\limits_{\mathbf{x}_i=\mathbf{x},y_i=-1} w_i} \right)$. We do not use the least-square criterion to find a regression function in $\mathcal{Q}_m$ that approximates d_m (line 2(b) of the Generic Algorithm). Instead, we use a comment that, $d_m(\mathbf{x})$ is in fact an empirical estimation of the quantity $\frac{1}{2}\log\left(\frac{\mathbb{P}_w(y=1|\mathbf{x})}{\mathbb{P}_w(y=-1|\mathbf{x})}\right)$ after having trained $\mathbf{x}$ and y on training set $(\mathbf{x}_i, y_i)_{i=1:N}$ with weights $w_i, i = 1,...,N$. The coefficient β_m is taken 1. We have the following *Real AdaBoost* algorithm.

```
Real AdaBoost
1. Start with weights $w_i = 1/N, i = 1,...,N$.
2. Repeat for $m = 1,2,...,M$ :
    (a) Fit the classifier to obtain a class probability estimate $p_m(\mathbf{x}) = \hat{P}_w(y = 1|\mathbf{x}) \in [0,1]$, using weights $w_i$ on the training data.
    (b) Set $f_m(\mathbf{x}) = \frac{1}{2}\log\frac{p_m(\mathbf{x})}{1-p_m(\mathbf{x})}$.
    (c) Update $w_i \leftarrow w_i e^{-y_i f_m(\mathbf{x}_i)}$ and renormalize.
3. Output the classifier $sign\left[\sum_{m=1}^{M} f_m(\mathbf{x})\right]$.
```

The last algorithm that we want to present in this section is *Gentle AdaBoost*. This algorithm use $\beta_m = 1$ but d_m is not calculated from a direct optimization but from the stepping of Newton-Raphson algorithm. Like in *Real AdaBoost*, we parameterize d_m not by $i = 1 : N$ but by $\mathbf{x}$ which are the differents values in $(\mathbf{x}_i)_{i=1:N}$. We note that the algorithms with such parameterization of d_m is called **population version**. More precisely,

$$
\begin{aligned}
d_m(\mathbf{x}) &= -\left(\frac{\partial^2 Q}{\partial F(\mathbf{x})^2}\right)^{-1}\frac{\partial Q}{\partial F(\mathbf{x})}|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \\
&= \frac{\sum\limits_{\mathbf{x}_i=\mathbf{x}} y_i e^{-y_i F(\mathbf{x})}}{\sum\limits_{\mathbf{x}_i=\mathbf{x}} e^{-y_i F(\mathbf{x})}} = \frac{\sum\limits_{\mathbf{x}_i=\mathbf{x}} w_i y_i}{\sum\limits_{\mathbf{x}_i=\mathbf{x}} w_i}
\end{aligned}
$$

We recognize that $d_m(\mathbf{x})$ is in fact an empirical estimation of the quantity $\mathbb{E}_w(y|\mathbf{x})$; therefore we take $f_m(\mathbf{x})$ to be the regression of y to $\mathbf{x}$ with weights w_i on the training data. We have the following *Gentle AdaBoost* algorithm.

```
Gentle AdaBoost
1. Start with weights $w_i = 1/N, i = 1,...,N$.
2. Repeat for $m = 1,2,...,M$ :
    (a) Fit the regression function $f_m(\mathbf{x})$ by weighted least-squares of $y_i$ to $\mathbf{x}_i$ with weights $w_i$.
    (b) Update $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + f_m(\mathbf{x})$.
    (c) Update $w_i \leftarrow w_i e^{-y_i f_m(\mathbf{x}_i)}$ and renormalize.
3. Output the classifier $sign\left[F(\mathbf{x})\right] = sign\left[\sum_{m=1}^{M} f_m(\mathbf{x})\right]$.
```

Experiments : Here we compare the performance of these three algorithms on a same simulated dataset. The dataset is simulated following the following rules : $\mathbf{x} \in \mathbb{R}^2$ follows uniform distribution in $(0,1)^2$, and $y_i = 2 * 1((\mathbf{x}_i^{(1)} - 0.5)^2 + (\mathbf{x}_i^{(2)} - 0.5)^2 > 1/6) - 1$. The Bayes error is therefore 0. We can see that the *Gentle AdaBoost* obtain the best performance in this experiment. The *Gentle AdaBoost* corresponds to two-degree optimization in function space (Newton-Raphson algorithm) while the *Discrete AdaBoost* corresponds to one-degree method. We can link this result to the dominating convergence rate of two-degree methods in numerical optimization.

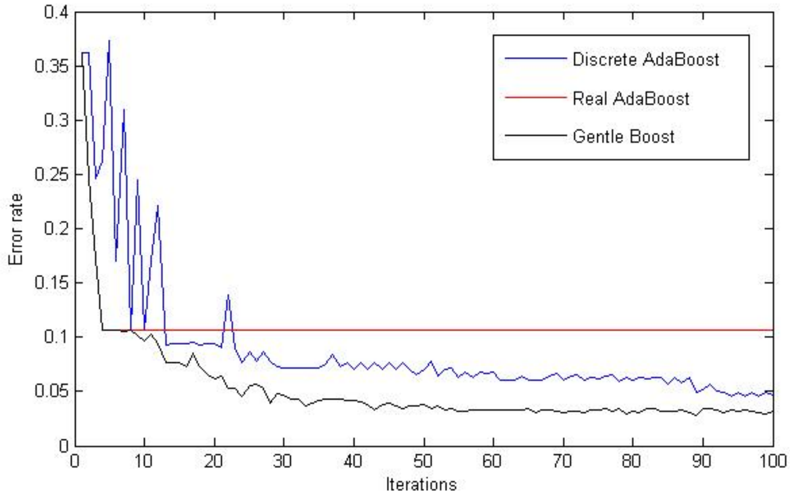

FIGURE 1 – Error rates corresponding to *Discrete AdaBoost*, *Real AdaBoost* and *Gentle AdaBoost*.

2.3 $L(y, F(\mathbf{x})) = (y - F(\mathbf{x}))^2$

2.4 $L(y, F(\mathbf{x})) = |y - F(\mathbf{x})|$

With $L(y, F(\mathbf{x})) = |y - F(\mathbf{x})|$, we choose $d_m^{(i)} = \dfrac{\partial Q}{\partial \overline{F}_i}|_{\overline{F}=\overline{F}_{m-1}} = sign(y_i - F_{m-1}(\mathbf{x}_i)), \forall i = 1 : N$. We can always follow the generic algorithm presented in 2.1 with all kinds of base classifiers. Here we discuss on how to adapt regression tree to our boosting algorithm, because it will be the only base classifier used in our experiments.

Firstly, we remark that the coefficient $c > 0$ in the line 2(b) of our generic algorithm does not change fondamentally the properties of regression trees f_m ; it suffices to get regression tree f_m to approximate the direction d_m. A regression tree f_m divide $\mathcal{X}$-space into K disjoint subspace $R_{m,k}$ so that $\bigcup_{k=1}^{K} R_{m,k} = \mathcal{X}$. For $\mathbf{x}$ belonging to each of these areas, $f_m(\mathbf{x})$ takes a value $\lambda_{m,k}$. We can rewrite $f_m(\mathbf{x}) = \sum_{k=1}^{K} \lambda_{m,k} 1(\mathbf{x} \in R_{m,k})$. The constant β_m in the line 2(c) changes these values $\lambda_{m,k}$ by a same way. Another method which gives a better optimization than line 2(c) is to modify all regression values $\lambda_{m,k}$ in this phase and not only the parameter β_m, that means :

$$\{\lambda_{m,k}\}_{k=1:K} = arg \min_{\lambda \in \mathbb{R}^K} Q(F_{m-1} + \sum_{k=1}^{K} \lambda_k 1(\mathbf{x} \in R_{m,k}))$$

This method is clearly more powerful than optimization only on β_m and is in fact operated separately in each $R_{m,k}$: For each $k = 1 : K$,

$$\lambda_{m,k} = arg \min_{\lambda \in \mathbb{R}} \sum_{\mathbf{x}_i \in R_{m,k}} L(y_i, F_{m-1}(\mathbf{x}_i) + \lambda) \tag{3}$$

Finally, the areas $R_{m,k}$ are divided accordint to the regression in line 2(b), which tries to approximate d_m by f_m (as explained, we have taken $c = 1$).

```
LAD TreeBoost
1. Start with $F_0(\mathbf{x}) = 0$.
2. Repeat for $m = 1,2,...,M$ :
(a) $\overline{y}_i = sign(y_i - F_{m-1}(\mathbf{x}_i)), i = 1 : N$.
(b) $\{R_{m,k}\}_{k=1:K}$ is the $K$-terminal node tree regression of $\overline{y}_i$ on $\mathbf{x}_i, i = 1 : N$.
(c) $\lambda_{m,k} = \text{median}\{y_i - F_{m-1}(\mathbf{x}_i)\}_{i,\mathbf{x}_i \in R_{m,k}}$ for $k = 1 : K$.
(d) $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{k=1}^{K} \lambda_{m,k} 1(\mathbf{x} \in R_{m,k})$.
3. Output the classifier $sign\,[F(\mathbf{x})]$.
```

2.5 M-regression loss function

2.6 $L(y, F(\mathbf{x})) = 2y^* F(\mathbf{x}) - log(1 + e^{2F(\mathbf{x})})$

2.7 $L(y, F(\mathbf{x})) = \log(1 + e^{-2yF(\mathbf{x})})$

3 Multi-class classification and some generalizations

3.1 A traditional approach

3.2 Some generalization of two-class algorithms

3.3 Other generalizations

4 Experiments

4.1 Experiments with simulated data

4.2 Experiments with real data

5 Conclusion

Références

[1] Friedman, J., Hastie, T. & Tibshirani, R. *Additive Logistic Regression : a Statistical View of Boosting*, 2000.

[2] Friedman, J. *Greedy Function Approximation : A Gradient Boosting Machine*, IMS 1999 Reitz Lecture, 2001.

[3] Schapire, R.E. & Singer, Y. *Improved Boosting Algorithms : Using Confidence-rated Predictions*, 1998.