Generalized Linear Methods[*]

1 Introduction

In *the Ensemble Methods* the general idea is that *using a combination of several weak learner one could make a better learner.* More formally, assume that we have a set of *weak* learners, $\{g_t(x)\}_{t=1}^{T}$, i.e. we have trained these learners on the training data $\{(x_i, y_i)\}_{i=1}^{n}$, which correspond to the weights $\{w_i\}_{i=1}^{n}$. We can create a *strong* model, as a combination of all these weak learners via creating a linear combination of them. We define the weighted output as $G_T(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(x)\right)$, where the sign(.) is the sign function.

One of the most famous method for such boosting based data is called "AdaBoost". Based on AdaBoost one could develop methods for gradient boosting, especially for trees. Here we first explain the AdaBoost.

2 AdaBoost

Mainly introduced in [1], is considered as one of the most important steps in the Statistical Learning.

One could show that AdaBoost is minimizing an upper bound on the empirical error. To be accurate, the empirical error (for a classification) is defined as the following:

$$\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) = \frac{1}{n} \left|\{i : G(x_i) \neq y_i\}\right|$$

which can be upper-bounded by the multiplication of the normalization constants:

$$\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) \leq \prod_{t=1}^{T} Z_t.$$

[*]Or Boosting Techniques, Ensemble Methods, ...

```
Algorithm 1: Adaboost.
Input: The set of weak learners, ${g_t(x)}_{t=1}^T$
Output: The weights of the generalized learner ${\alpha_t}_{t=1}^T$
Initialize the weights, $w_i^{(1)} = \frac{1}{n},\ i = 1, \dots, n$
for $t = 1$ to $T$ do
    Fit the learner $g_t(x)$ to the training data $\{(x_i, y_i)\}_{i=1}^n$, weighted
    by $\left\{w_i^{(t)}\right\}_{i=1}^n$.
    Choose $\alpha_t \in \mathbb{R}$;
    $w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{\exp[-\alpha_t y_i g_t(x_i)]}{Z_t}$, for $i = 1, \dots, n$, where $Z_t$ is a
    normalization factor.
Return the output $G(x)$.
```

One unanswered question by now is that, how do we choose α_t? One option is to choose it in a greedy fashion to make $Z_t(\alpha)$ at each step. Since $Z_t(\alpha)$ is a convex function, it has a unique minimum. Given that $g_t(x) \in \{\pm 1\}$, the greedy choice of α_t would give the following answer:

$$\epsilon_t \leftarrow \sum_i w_i^{(t)} \mathbb{I}\left(y_i \neq g_t(x_i)\right) \tag{1}$$

$$\alpha_t \leftarrow \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \tag{2}$$

With this choice we can find a guarantee on the empirical error bound of this algorithm, as stated by the next algorithm.

Theorem 1. *The given procedure in Algorithm 1, with the choice of α_t in the Equation 1, if $\epsilon_t \leq \frac{1}{2} - \gamma$ for all t will result in the following bound:*

$$\hat{\mathbb{P}}\left(Y G_T(X) \leq 0\right) \leq \delta \tag{3}$$

for any arbitrary $\gamma > 0$, if T is big enough. More accurately if $T \geq \frac{\ln 1/\delta}{2\gamma^2}$.

Now we will provide the proof of Theorem 1 to the next section. If you don't feel like reading a relatively boring proof, you can skip it to the next section!

We chunk the proof into smaller sections. First we prove the following lemma.

Lemma 1. *The given procedure in Algorithm 1, with the choice of α_t in Equation 1, will result in the following bounds:*

$$\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) = \frac{1}{n}\left|\{i : G(x_i) \neq y_i\}\right| \leq \prod_{t=1}^{T} Z_t. \tag{4}$$

Proof. The equivalent event to $YG_T(X) \leq 0$ is $\exp\left(-YG_T(X)\right) \geq 1$. It is easy to see that:

$$\begin{aligned}\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left\{G(x_i) \neq y_{i=1}\right\} &(5)\\ &\leq \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y_i G(x_i)\right) &(6)\\ &\leq \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t g_t(x_i)\right) &(7)\\ &\leq \frac{1}{n}\sum_{i=1}^{n}\prod_{t=1}^{T} \exp\left(-y_i \alpha_t g_t(x_i)\right) &(8)\end{aligned}$$

Using the updates of $w_i^{(t+1)}$ in the Algorithm 2, we have:

$$\exp\left(-y_i \alpha_t g_t(x_i)\right) = Z_t \frac{w_i^{(t+1)}}{w_i^{(t)}}$$

which would give:

$$\begin{aligned}\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) &\leq \frac{1}{n}\sum_{i=1}^{n}\prod_{t=1}^{T} Z_t \frac{w_i^{(t+1)}}{w_i^{(t)}} &(9)\\ &= \left(\prod_{t=1}^{T} Z_t\right)\frac{1}{n}\sum_{i=1}^{n}\frac{w_i^{(T+1)}}{w_i^{(1)}} &(10)\end{aligned}$$

Since we chose $w_i^{(1)} = 1/n$, and $w_i^{(T+1)}$ is a proper probability distribution,

$$\begin{aligned}\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) &\leq \left(\prod_{t=1}^{T} Z_t\right)\sum_{i=1}^{n} w_i^{(T+1)} &(11)\\ &\leq \prod_{t=1}^{T} Z_t. &(12)\end{aligned}$$

□

Lemma 2. *The given procedure in Algorithm 1, with the choice of α_t in Equation 1, the greedy choice of α_t to minimize Z_t will result in,*

$$\alpha_t \leftarrow \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$$

and

$$Z_t = 2\sqrt{\epsilon_t \left(1-\epsilon_t\right)}$$

where,

$$\epsilon_t \leftarrow \sum_i w_i^{(t)} \mathbb{I}\left(y_i \neq g_t(x_i)\right)$$

Proof. The proof is easy. Let's write the definition of the normalization constant explicitly:

$$\begin{aligned} Z_t = \sum_i e^{-\alpha_t y_i g_t(x_i)} &= \sum_{i: y_i = g_t(x_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{i: y_i \neq g_t(x_i)} w_i^{(t)} e^{\alpha_t} \\ &= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \end{aligned}$$

Z_t is a convex function of α_t and has unique minimum. By talking derivative we can find the minimizer, which is the following:

$$\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$$

By plugging this into the definition of Z_t, we will find its minimum value:

$$Z_t = 2\sqrt{\epsilon_t \left(1-\epsilon_t\right)}$$

□

Lemma 3. *The results of Lemma 1 and Lemma 2, and given that $\epsilon_t \leq \frac{1}{2} - \gamma, \forall t$, the bound on the empirical error is not more than, $\left(1-4\gamma^2\right)^{T/2}$ (where $\gamma \in (0, 0.5)$).*

Proof. By the end of Lemma 1 and Lemma 2, we have proven that:

$$\hat{\mathbb{P}}\left(Y G_T(X) \leq 0\right) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t \left(1-\epsilon_t\right)}$$

Given that $\gamma \in (0, 0.5)$, we can show that,

$$\hat{\mathbb{P}}\left(Y G_T(X) \leq 0\right) \leq \left(1-4\gamma^2\right)^{T/2}$$

□

Now we have everything we needed for the proof of the Theorem 1.

Proof of the Theorem 1. Given the result of the Lemma 3, we have

$$\hat{\mathbb{P}}\left(YG_T(X) \leq 0\right) \leq \left(1 - 4\gamma^2\right)^{T/2}$$

Now define

$$\delta \leq \left(1 - 4\gamma^2\right)^{T/2}$$

which is equivalent to

$$T \geq \frac{\ln 1/\delta}{2\gamma^2}$$

□

2.1 AdaBoost as minimizing a global objective function

One alternate view to what mentioned above is minimizing a global objective function, with coordinate-descent (greedy) updates. It can be shown that the global objective is the following:

$$L = \frac{1}{n}\sum_i e^{-y_i \sum_t \alpha_t f_t(x_i)}$$

when optimizing locally with respect to α_t.

2.2 More general AdaBoost

As shown previously, the standard form of AdaBoost can be interpreted as minimizing an exponential loss function. One can show a general form of AdaBoost, on arbitrary loss function, but due to computational cost these general forms are not very popular.

3 Gradient Boosting

Using the boosting methods, new methods are proposed for Gradient Boosting methods. In fact, the gradient boosting methods, are using both of *boosting* and *gradient* methods, especially gradient descent (in the functional level). Using only gradient methods have cones, e.g. negative effect on generalization error and local optimization. However, in glm (a package in R language) suggests selecting a class of functions that uses the covariate information to approximate the gradient, usually a regression "tree". The algorithm is shown in the Algorithm 3. At each iteration the algorithm determines the gradient, the direction in which it needs to improve the approximation to fit to the data, by selecting from a class of functions. In other words, it selects a function which has the most agreement with the current approximation error.

Just to remind you what problem is formally, we want to find a function F^* such that:

$$G^*(x) = \arg\min_G \mathbb{E}_{x,y}\left[L(y, F(x))\right] = \arg\min_G \int_{x,y} L(y, F(x))\mathbb{P}(x,y)dxdy$$

But in practice the true distribution $\mathbb{P}(x,y)$ is not known, and instead we have samples of it, in the form of $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. Also the set of functions we can choose from is also limited, which we represent with $\mathcal{G}$. Thus the problem is approximated in the following form:

$$\Rightarrow G^*(x) \approx \arg\min_{G \in \mathcal{G}} \sum_{(x_i,y_i)\in\mathcal{D}} L(y_i, F(x_i))$$

Suppose a *good* approximation could be written as a linear combination of some coarse approximations:

$$\sum_{t=1}^{T} \alpha_t g_t(x)$$

Suppose the following is our initial approximation, with a constant function:

$$G_0(x) = \arg\min_{\alpha} \sum_{(x_i,y_i)\in\mathcal{D}} L(y_i, \alpha)$$

Followed by the incremental approximations:

$$G_m(x) = G_{m-1}(x) + \arg\min_{g\in\mathcal{G}} \sum_{(x_i,y_i)\in\mathcal{D}} L(y_i, G_m(x_i) + g(x_i))$$

Since the minimization in the previous equation is done over functions (functional minimization) it is relatively hard to solve. Instead we can approximate it with greedy (functional-)gradient based updates. The negative functional-gradient of the loss function:

$$-\nabla_g L(y_i, G(x) + g(x))$$

is the direction in which loss functions has the most decrease. Thus following updates, appropriate choice of step size will result in reduction:

$$G_m(x) = G_{m-1}(x) - \gamma_m \sum_{x_i \in \mathcal{D}} \nabla_g L(y_i, G_{m-1}(x_i))$$

One possible way to find the step size is via line search:

$$\gamma_m = \arg\min_{\gamma} \sum_{x_i \in \mathcal{D}} L\left(y_i, G_{m-1}(x_i) - \gamma \nabla_g L(y_i, G_{m-1}(x_i))\right)$$

Algorithm 2: Gradient boosting algorithm.

```
Input: The set of weak learners, $\{g_t(x)\}_{t=1}^T$
Output: The weights of the generalized learner $\{\alpha_t\}_{t=1}^T$
Initialize an approximation with a constant
$g_0(x) = \arg\min_\gamma \sum_i L(y_i, \gamma)$.
for $m$ = 1 to $M$ do
    Compute the negative gradient, $-\mathbf{r}_m = (-r_{1m}, -r_{2m}, \dots, -r_{nm})^\top$,
    such that
        $r_{im} = \left. \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \right|_{G(x)=G_{m-1}(x)}$
    Fit the a function $h_m(x)$ on the gradient residuals $\{(x_i, r_{im})\}_{i=1}^n$.
    Find the scaling parameter $\gamma_m$ such that the following objective is
    minimized:
        $\gamma_m = \arg\min_{\gamma} \sum_{x_i \in \mathcal{D}} L\left(y_i, G_{m-1}(x_i) + \gamma h_m(x)\right)$
    Update, $G_m(x) = G_{m-1}(x) + \gamma_m h_m(x)$
Return the output $G_M(x)$.
```

Suppose we want to generalize the Algorithm 2 to trees. The only difference is that, the approximating function $h_m(x)$ is made of a tree, which we can represent with $\sum_j b_j I\left(x \in R_j\right)$, where $I(.)$ is an indicator function which shows whether the input x belongs to a specific region R_j or not, and b_j is the prediction for the values in this region. In the following step, a value of γ_m is estimated via line search. In the [2] it is suggested to use γ value for each region. In other words change

$$\gamma_m = \arg\min_{\gamma} \sum_{x_i \in \mathcal{D}} L\left(y_i, G_{m-1}(x_i) + \gamma h_m(x)\right)$$

to

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, G_{m-1}(x_i) + \gamma b_j\right).$$

Note that, in this line search, the values of $\{b_j\}_{j=1}^{M}$ do not have any effect in the final result; the only things that matter are the set of regions $\{R_j\}_{j=1}^{M}$. Thus we can simplify it, and write it as the following:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, G_{m-1}(x_i) + \gamma\right).$$

The overall algorithm is shown in Algorithm 3.

Algorithm 3: Gradient boosting for trees.

```
Input: The set of weak learners, $\{g_t(x)\}_{t=1}^{T}$
Output: The weights of the generalized learner $\{\alpha_t\}_{t=1}^{T}$
Initialize a single-node tree $G_0(x) = \arg\min_\gamma \sum_i L(y_i, \gamma)$.
for $m = 1$ to $M$ do
    Compute the negative gradient, $-\mathbf{r}_m = (-r_{1m}, -r_{2m}, \ldots, -r_{nm})^\top$,
    such that
        $r_{im} = \left.\frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}\right|_{G(x)=G_{m-1}(x)}$
    Fit the regression to the pseudo-responses (residuals)
    $\{(x_i, r_{im})\}_{i=1}^{n}$, which results in terminal regions,
    $R_{jm},\ j = 1, \ldots, J_m$.
    for $j = 1, 2, 3, \ldots, J_m$ do
        $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, G_{m-1}(x_i) + \gamma\right)$
    Update, $G_m(x) = G_{m-1}(x) + \sum_j \gamma_{jm} I\left(x \in R_{jm}\right)$
Return the output $G_M(x)$.
```

In the glm library of R, given the scenario shown in the Algorithm 3, the shrinkage parameter is the (or learning rate) parameter in gradient updates. So the main effort in variable selection is in selecting are the choice of n.trees and shrinkage parameters.

3.0.1 Regularization

Since the gradient boosting for trees has a big degree of freedom, it is highly prone to overfitting. One possible way to reduce the amount of overfitting is shrinkage in the coefficients. Suppose we have a parameter $\lambda \in (0,1)$, such that:

$$G_m(x) = G_{m-1}(x) + \lambda \gamma_m h_m(x)$$

4 Final notes

Some intuition is from David Forsyth's class at UIUC. Peter Bartlett's class notes provided a very good summary of the main points.

References

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

[2] J. Friedman, T. Hastie, and R. Tibshirani. The elements of statistical learning, 2008.