IIIS 2018 Spring: ATCS - Selected Topics in Optimization
Lecture date: Feb 26, 2018
Instructor: Jian Li **Scribe: Chenxing Li**

1 Supervised Learning

A supervised learning is a task of learning a regression function or classification function for a set given a series of training data (x_i, y_i) sampled from the set. x_i is an instance and y_i is the label of this instance. In order to judge the performance of learnt function, we need a series of test data (x_i^*, y_i^*) to judge training data.

In a parametric learning, the number of parameters are fixed. In a non-parametric learning, the number of parameters may grow as training data grows.

1.1 Additive model

An additive model makes classification or regression based on the linear combination of a series of base models. Adaboost, gradient boosting and random forest are the examples of additive model.

An additive model of binary classification is defined following

$$f(x) = \sum_{m=1}^{M} \beta_m G_m(x).$$

Here β_m are the parameters and G_m are binary classifier in which $G_m(x) \in \{-1, 1\}$.

The goal of training algorithm is to minimized the total loss, so the objective function is given by

$$\min_{\beta_m, r_m} \sum_{i=1}^{N} L(y_i, f(x_i)).$$

Here β_m is the parameters of additive model. r_m is the parameters for base classifier G_m. $L(y_i, f(x_i))$ is the loss fucntion given a pair of the true result and preditated result.

In additive model, decision tree is a popular choice for base classfier.

1.2 Forward Stagewise Additive Modeling

The Forward Stagewise Additive Modeling is a greedy algorithm which learns G_m one by one. Suppose we have

$$f_{m-1}(x) = \sum_{i=1}^{m-1} \beta_i G_i(x),$$

we want to learn the $G_m(x)$ and β_m given the minimum loss, a.k.a.

$$(\beta_m, G_m) \leftarrow \arg\min_{\beta_m, G_m} \sum_{i=1}^{N} \exp\left[L(y_i, f_{m-1}(x_i) + \beta_m G_m(x_i))\right].$$

If we use $L(y_i, f(x_i)) = \exp\left[-y_i f(x_i)\right]$ as loss function, the objective function will be

$$\begin{aligned}
&\sum_{i=1}^{N} \left[L(y_i, f_{m-1}(x_i) + \beta_m G_m(x_i))\right] \\
=&\sum_{i=1}^{N} \exp\left[-y_i(f_{m-1}(x_i) + \beta_m G_m(x_i))\right] \\
=&\sum_{i=1}^{N} w_i^{(m)} \exp\left[-\beta_m y_i G_m(x_i)\right]
\end{aligned}$$

Here $w_i^{(m)} = \exp\left[-y_i f_{m-1}(x_i)\right]$ is constant, since β_m and G_m are the only two variables. So we can find the optimal stage solution from

$$(\beta_m, G_m) \leftarrow \arg\min_{\beta_m, G_m} \sum_{i=1}^{N} w_i^{(m)} \exp\left[-\beta_m y_i G_m(x_i)\right].$$

Since

$$\exp\left[-\beta_m y_i G_m(x_i)\right] = \begin{cases} e^{-\beta_m} & y_i = G(x_i) \\ e^{\beta_m} & y_i \neq G(x_i) \end{cases},$$

we have

$$\begin{aligned}
&\sum_{i=1}^{N} w_i^{(m)} \exp\left[-\beta_m y_i G_m(x_i)\right] \\
=&\sum_{i=1}^{N} w_i^{(m)} e^{-\beta_m} + \left(e^{\beta_m} - e^{-\beta_m}\right) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}[y_i \neq G(x_i)].
\end{aligned}$$

For any given β_m, $\sum_{i=1}^{N} w_i^{(m)} \exp\left[-\beta_m y_i G_m(x_i)\right]$ is positive linear correlation with $\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}[y_i \neq G(x_i)]$, we can claim

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}[y_i \neq G(x_i)].$$

Set err_m be the weighted average loss of G_m,

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}[y \neq G(x_i)]}{\sum_{i=1}^{N} w_i^{(m)}}.$$

Apply derivation to the loss function, we can find that $\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$ gives the minimum loss.

1.3 Adaboost Algorithm

Adaboost algorithm is essentially the same as forward stagewise additive modeling with exponential loss function, which is introduced in the previous part.

Given training data $\{(x_i, y_i)\}_{i=1}^N$, adaboost algorithm is defined formally in Algorithm 1

Algorithm 1: An Algorithm

```
Initially, let $w_i = \frac{1}{N}, i \in [N]$;
for $m = 1, \cdots, M$ do
    Fit classifier $G_m$ (with respect to $w_i$);
    Set $\mathrm{err}_m \leftarrow \left(\sum_{i=1}^N w_i \mathbb{I}[y_i \neq G_m(x_i)]\right) / \left(\sum_{i=1}^N w_i\right)$;
    Set $\alpha_m \leftarrow \log(1 - \mathrm{err}_m) - \log(\mathrm{err}_m)$;
    for $i = 1, \cdots, N$ do
        Update $w_i \leftarrow w_i \exp[\alpha_m \cdot \mathbb{I}[y_i \neq G_m(x_i)]]$;
Output classifier $\mathrm{sign}\left(\sum_{m=1}^M \alpha_m G_m(x_i)\right)$;
```

Claim 1 *Let* $z_m = \sum_{i=1}^N w_i^{(m)}$,

$$z_m = 2(1 - err_m) z_{m-1}$$

.

Proof:

$$\begin{aligned} z_m &= \sum_{i=1}^N w_i^{(m)} \\ &= \sum_{i=1}^N w_i^{(m-1)} \exp[\alpha_m \cdot \mathbb{I}[y_i \neq G_m(x_i)]] \\ &= e^{\alpha_m} \cdot \sum_{i \in \{j | y_j \neq G_m(x_j)\}} w_i^{(m-1)} + \sum_{i \in \{j | y_j = G_m(x_j)\}} w_i^{(m-1)} \\ &= \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \cdot \mathrm{err}_m \cdot z_{m-1} + (1 - \mathrm{err}_m) \cdot z_{m-1} \\ &= 2(1 - \mathrm{err}_m) \cdot z_{m-1} \end{aligned}$$

□

Theorem 2

$$\frac{1}{N} \sum_{i=1}^N \mathbb{I}[f(x_i) \neq y_i] \leq \prod_{m=1}^M 2\sqrt{err_m(1 - err_m)}.$$

Proof: Since $y_i, G_m(x_i) \in \{1, -1\}$ for all $i \in [N], m \in [M]$, we have $\mathbb{I}[y_i \neq G_m(x_i)] = \frac{1}{2}(1 - y_i G_m(x_i))$

$$
\begin{aligned}
w_i^{(M)} &= \frac{1}{N} \cdot \exp\left[\sum_{m=1}^{M} \alpha_m \cdot \mathbb{I}[y_i \neq G_m(x_i)]\right] \\
&= \frac{1}{N} \cdot \exp\left[\frac{1}{2} \cdot \sum_{m=1}^{M} \alpha_m \cdot (1 - y_i G_m(x_i))\right] \\
&= \frac{1}{N} \cdot \exp\left[\frac{1}{2} \cdot \sum_{m=1}^{M} \alpha_m\right] \cdot \exp\left[-\frac{1}{2} \cdot y_i \cdot \sum_{m=1}^{t} \alpha_m G_m(x_i)\right] \\
&= \prod_{m=1}^{M} \left[\sqrt{\frac{1-\mathrm{err}_m}{\mathrm{err}_m}}\right] \cdot \frac{1}{N} \cdot \exp\left[-\frac{1}{2} \cdot y_i \cdot \sum_{m=1}^{t} \alpha_m G_m(x_i)\right].
\end{aligned}
$$

Because $\mathbb{I}[f(x_i) \neq y_i] \leq e^{-0.5 y_i f(x_i)}$, we have

$$
\begin{aligned}
&\frac{1}{N} \cdot \sum_{i=1}^{N} \mathbb{I}[f(x_i) \neq y_i] \\
\leq &\frac{1}{N} \cdot \sum_{i=1}^{N} \exp\left[-\frac{1}{2} \cdot y_i \cdot \sum_{m=1}^{t} \alpha_m G_m(x_i)\right] \\
= &\prod_{m=1}^{M} \sqrt{\frac{\mathrm{err}_m}{1-\mathrm{err}_m}} \cdot \sum_{i=1}^{N} w_i^{(M)} \\
= &\prod_{m=1}^{M} \sqrt{\frac{\mathrm{err}_m}{1-\mathrm{err}_m}} \cdot z_M \\
= &\prod_{m=1}^{M} \sqrt{\frac{\mathrm{err}_m}{1-\mathrm{err}_m}} \cdot \prod_{m=1}^{M} 2(1-\mathrm{err}_m) \\
= &\prod_{m=1}^{M} 2\sqrt{\mathrm{err}_m(1-\mathrm{err}_m)}.
\end{aligned}
$$

□

2 Gradient Boosting

2.1 Ensemble of regression tree

Suppose there are K *regression tree*, and f_k is the function defined by the k-th tree. The prediction is given by

$$
\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F},
$$

where $\mathcal{F}$ is the space of functions containing all regression trees.

In training one regression tree, we use piecewise step function to approach the real function. And the training loss function is defined by the error between real function and piecewise step function.

In order to avoid overfitting, we need to balance the training loss and the complexity of function. So the loss function is defined by

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k),$$

where $l(y_i, \hat{y}_i)$ is the loss for each prediction and $\Omega(\cdot)$ is complexity function of each tree. We usually use the depth or the number of leaves to present the complexity of a regression tree.

Loss function has several popular choices

- Square loss: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$, which results in common gradient boosted machine.
- Logistic loss: $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i)\ln(1 + e^{\hat{y}_i})$, which results in LogitBoost

Here is also an example for tree complexity definition.

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} w_j^2.$$

Note: T is the number of leaves, w_j is the L2 norm of leaf scores.

2.2 Additive Training

In additive training, the prediction started with $\hat{y}_i^{(0)} = 0$ for all the testing element x_i. In round t, we update y_i by

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i).$$

In each round, we need to choose round function f_t according to objective function, which is given by

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const.$$

If we use square loss, then the objective function will given by

$$Obj^{(t)} = \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + const.$$

For general loss function, we take Taylor expansion of the loss function, recall

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2,$$

we get

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const.$$

Remove the terms independent with f_t, we get

$$
\begin{aligned}
Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) + \Omega(f_t) \right] \\
&= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{n} \left[\left(\sum_{i \in I_j} g_i w_j \right) + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \\
&= \sum_{j=1}^{n} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T
\end{aligned}
$$

Here $q(x)$ returns the leaf index the sample x fall into, $I_j = \{i | q(x_i) = j\}$, $G_j = \sum_{i \in I_j} g_i, H_j = \sum_{i \in I_j} h_i$.

The optimal solution is

$$
w_j^* = -\frac{G_j}{H_j + \lambda} \quad , Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda}
$$

Now we know that the optimal solution for the weight of each leaf given a tree in additive training model. We also need to decide how to grow the tree.

Here is a greedy algorithm for growing the tree:

- Start with the root
- Calculate the gain of each split according

$$
Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma.
$$

- Choose the best split or stop (if gain¡0).

If there are too many split points, it is expensive to try all the splits, so it is necessary to reduce the number of potential split points, such as sampling or use quantiles. XGBoost and LightGBM used more involved quantiles.

References

[1] Freund Y, Schapire R E. A decision-theoretic generalization of on-line learning and an application to boosting[J]. Journal of computer and system sciences, 1997, 55(1): 119-139.

[2] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016: 785-794.