Boosting

Victor Kitov

v.v.kitov@yandex.ru

Linear ensembles

Linear ensemble:

$$F_M(x) = f_0(x) - c_1 f_1(x) + ... - c_M f_M(x)$$

Regression: $\widehat{y}(x) = F_M(x)$
Binary classification: $score(y|x) = F_M(x)$, $\widehat{y}(x) = \text{sign}\, F_M(x)$

- Notation: $f_1(x), ...f_M(x)$ are called *base learners, weak learners, base models*.
- Too expensive to optimize $f_0(x), f_1(x), ...f_M(x)$ and $c_1, ...c_M$ jointly for large M.
- Idea: optimize $f_0(x)$ and then each pair $(f_m(x),\ c_m)$ step-by-step.

Forward stagewise additive modeling (FSAM)

Input:

- training dataset (x_n, y_n), $n = 1, 2, ...N$
- loss function $\mathcal{L}(f, y)$
- parametric form of base learner $f_\theta(x)$
- the number of base learners M.

Output: approximation function $F_M(x) = f_0(x) - \sum_{m=1}^{M} c_m f_m(x)$

Forward stagewise additive modeling (FSAM)

1. Fit initial approximation $f_0(x) = \arg\min_f \sum_{n=1}^{N} \mathcal{L}(f(x_n), y_n)$
2. For $m = 1, 2, ...M$:
    - find next best classifier

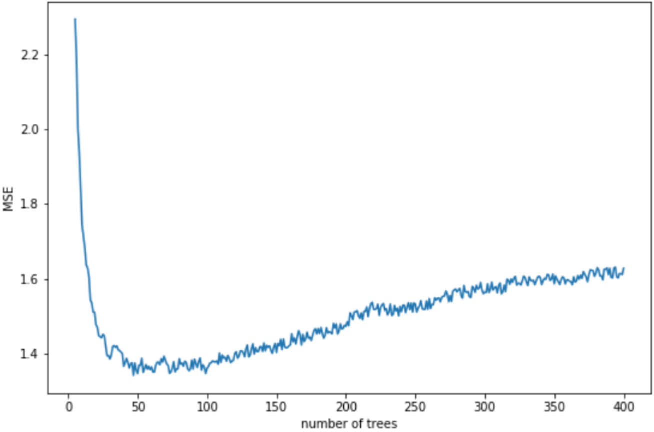
$$(c_m, f_m) := \arg\min_{f,c} \sum_{n=1}^{N} \mathcal{L}(F_{m-1}(x_n) - cf(x_n),\ y_n)$$

    - reevaluate ensemble

$$F_m(x) := F_{m-1}(x) - c_m f_m(x)$$

Dependency on M

Boosting overfits for high M:



Comments

- M should be determined by performance on validation set.
- Each step should be coarse to leave room for future base learners improvement:
    - initial approximation may be zero or constant
    - optimization can be coarse (just few steps)
    - base learner should be simple
        - such as trees of depth=1,2,3.
- For some loss functions (see Adaboost) we can solve minimization explicitly.
- For general loss functions gradient boosting should be used.

Table of Contents

Adaboost (discrete version)

Assumptions:

- binary classification task $y \in \{+1, -1\}$
- $f_m(x) \in \{+1, -1\}$, trainable in weighted dataset.
- classification is performed with $\widehat{y} = sign\{f_0(x) + c_1 f_1(x) + ... + c_M f_M(x)\}$
- optimized loss is $\mathcal{L}(F(x), y) = e^{-yF(x)}$

Optimization in FSAM can be solved explicitly!

Adaboost (discrete version): algorithm

Input:

- training dataset (x_n, y_n), $n = 1, 2, ...N$
- number of additive weak classifiers M
- a family of weak classifiers $f_m(x) \in \{+1, -1\}$
  - should be trainable on weighted datasets.

Output: composite classifier $F_M(x) = \text{sign}\left(\sum_{m=1}^{M} c_m f_m(x)\right)$

Adaboost (discrete version): algorithm

1. Initialize observation weights $w_n = 1/N$, $n = 1, 2, ...N$.
2. for $m = 1, 2, ...M$:
    1. fit $f_m(x)$ to training data using weights w_n
    2. compute weighted misclassification rate:

    $$E_m = \frac{\sum_{n=1}^{N} w_n \mathbb{I}[f_m(x_n) \neq y_n]}{\sum_{n=1}^{N} w_n}$$

    3. if $E_M > 0.5$ or $E_M = 0$: terminate procedure.
    4. compute $c_m = \frac{1}{2} \ln\left((1 - E_m)/E_m\right)$ $\quad E_m < 0.5 => c_m > 0$
    5. increase all weights, where misclassification with $f_m(x)$ was made:

    $$w_n \leftarrow w_n e^{2c_m} = w_n \left(\frac{1 - E_m}{E_m}\right), \text{ for } n : f_m(x_n) \neq y_n$$

Adaboost derivation

Set $F_0(x) \equiv 0$.
Apply FSAM for $m = 1, 2, ...M$:

$$
\begin{aligned}
(c_m, f_m) &= \arg\min_{c_m, f_m} \sum_{n=1}^{N} \mathcal{L}(F_{m-1}(x_n) + c_m f_m(x_n),\ y_n) \\
&= \arg\min_{c_m, f_m} \sum_{n=1}^{N} e^{-y_n F_{m-1}(x_n)} e^{-c_m y_n f_m(x_n)} \\
&= \arg\min_{c_m, f_m} \sum_{i=1}^{N} w_n^m e^{-c_m y_n f_m(x_n)}, \quad w_n^m := e^{-y_n F_{m-1}(x_n)}
\end{aligned}
$$

Adaboost derivation

$$\sum_{n=1}^{N} w_n^m e^{-c_m y_n f_m(x_n)} = \sum_{n: f_m(x_n)=y_n} w_n^m e^{-c_m} + \sum_{n: f_m(x_n)\neq y_n} w_n^m e^{c_m}$$

$$= e^{-c_m} \sum_{n: f_m(x_n)=y_n} w_n^m + e^{c_m} \sum_{n: f_m(x_n)\neq y_n} w_n^m$$

$$= e^{c_m} \sum_{n: f_m(x_n)\neq y_n} w_n^m + e^{-c_m} \sum_{n=1}^{N} w_n^m - e^{-c_m} \sum_{n: f_m(x_n)\neq y_n} w_n^m$$

$$= e^{-c_m} \sum_{n} w_n^m + (e^{c_m} - e^{-c_m}) \sum_{n: f_m(x_n)\neq y_n} w_n^m$$

Since $c_m \geq 0$, $f_m(\cdot)$ should be found from

$$f_m(\cdot) = \arg\min_{f} \sum_{n=1}^{N} w_n^m \mathbb{I}[f(x_n) \neq y_n]$$

Adaboost derivation

Denote $G(c_m) = \sum_{n=1}^{N} w_n^m \exp(-c_m y_n f_m(x_n))$. Then

$$\frac{\partial G(c_m)}{\partial c_m} = -\sum_{n=1}^{N} w_n^m e^{-c_m y_n f_m(x_n)} y_n f_m(x_n) = 0$$

$$-\sum_{n:f_m(x_n)=y_n} w_n^m e^{-c_m} + \sum_{n:f_m(x_n)\neq y_n} w_n^m e^{c_m} = 0$$

$$e^{2c_m} = \frac{\sum_{n:f_m(x_n)=y_n} w_n^m}{\sum_{n:f_m(x_n)\neq y_n} w_n^m}$$

$$c_m = \frac{1}{2}\ln\frac{\left(\sum_{n:f_m(x_n)=y_n} w_n^m\right)/\left(\sum_{n=1}^{N} w_n^m\right)}{\left(\sum_{n:f_m(x_n)\neq y_n} w_n^m\right)/\left(\sum_{n=1}^{N} w_n^m\right)} = \frac{1}{2}\ln\frac{1-E_m}{E_m},$$

$$\text{where } E_m := \frac{\sum_{n=1}^{N} w_n^m \mathbb{I}[f_m(x_n)\neq y_n]}{\sum_{n=1}^{N} w_n^m}$$

Adaboost derivation

Weights recalculation:

$$w_n^{m+1} \stackrel{def}{=} e^{-y_n F_m(x_n)} = e^{-y_n F_{m-1}(x_n)} e^{-y_n c_m f_m(x_n)}$$

Noting that $-y_n f_m(x_n) = 2\mathbb{I}[f_m(x_n) \neq y_n] - 1$, we can rewrite:

$$w_n^{m+1} = e^{-y_n F_{m-1}(x_n)} e^{c_m(2\mathbb{I}[f_m(x_n) \neq y_n]-1)} =$$
$$= w_n^m e^{2c_m \mathbb{I}[f_m(x_n) \neq y_n]} e^{-c_m} \propto w_n^m e^{2c_m \mathbb{I}[f_m(x_n) \neq y_n]}$$

Comments:

- We can remove common constants from weights.
- $w_n^{m+1} = w_n^m$ for correctly classified objects by $f_m(x)$.
- $w_n^{m+1} = w_n^m e^{2c_m}$ for incorrectly classified objects by $f_m(x)$.
    - so later classifiers will pay more attention to them

Table of Contents

Motivation

- Problem: For general loss function L FSAM cannot be solved explicitly
- Analogy with function minimization: when we can't find optimum explicitly we use numerical methods
- Gradient boosting: numerical method for iterative loss minimization

Gradient descent algorithm

$$L(w) \to \min_w, \quad g(w) = \nabla_w L(w), \quad w \in \mathbb{R}^N$$

Gradient descent:

```
initialize w
for m = 1,2,...M:
    g(w) = ∇_w L(w)
    w = w − εg(w)
```

Gradient descent with modified step:

```
initialize w
for m = 1,2,...M:
    g(w) = ∇_w L(w)
    c* = arg min_{c>0} L(w − cg(w))
    w = w − c* Δw
```

Gradient boosting intuition

$$L(F)=\sum_{n=1}^{N}\mathcal{L}(F^n)\to\min_F \qquad F=[F^1,F^2,...F^N]$$

Gradient descent: $\qquad F:=F-c\nabla L(F)$

Pointwise gradient descent: $\quad F^n:=F^n-c\nabla L(F)=F^n-c\nabla\mathcal{L}(F^n)$

We want generalization to new x, so need functional approximation:

$$F(x):=F(x)-cf(x)$$

$$f(x_n)\approx\nabla\mathcal{L}(F(x_n))\quad n=1,2,...N$$

Gradient boosting

- Now consider
$L(f(x_1),...f(x_N)) = \sum_{n=1}^{N} \mathcal{L}(f(x_n), y_n) \to \min_{f(\cdot)}$
- Gradient descent performs pointwise optimization, but we need generalization, so we optimize in space of functions.
- Gradient boosting = modified gradient descent in function space:
    - find gradients: $g(x_n) = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=f^{m-1}(x_n)}$
    - fit base learner $f_m(x)$ to $\{(x_n, g(x_n))\}_{n=1}^{N}$

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, ...M$:

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, ...M$:
    1. calculate gradients: $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, \ldots N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, \ldots M$:
   1. calculate gradients: $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
   2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^N$, for example by solving

$$\sum_{n=1}^{N} (f_m(x_n) - g_n)^2 \to \min_{f_m}$$

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, ...M$:
   1. calculate gradients: $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
   2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^N$, for example by solving
   $$\sum_{n=1}^N (f_m(x_n) - g_n)^2 \to \min_{f_m}$$
   3. solve univariate optimization problem:
   $$c_m = \arg\min_{c>0} \sum_{n=1}^N \mathcal{L}\left(F_{m-1}(x_n) - cf_m(x_n), y_n\right)$$

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, ...M$:
    1. calculate gradients: $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
    2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^N$, for example by solving
    $$\sum_{n=1}^{N} (f_m(x_n) - g_n)^2 \to \min_{f_m}$$
    3. solve univariate optimization problem:
    $$c_m = \arg\min_{c>0} \sum_{n=1}^{N} \mathcal{L}\left(F_{m-1}(x_n) - cf_m(x_n), y_n\right)$$
    4. set $F_m(x) = F_{m-1}(x) - c_m f_m(x)$

Gradient boosting

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
2. For each step $m = 1, 2, ...M$:
    1. calculate gradients: $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
    2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^N$, for example by solving
    $$\sum_{n=1}^{N}(f_m(x_n) - g_n)^2 \to \min_{f_m}$$
    3. solve univariate optimization problem:
    $$c_m = \arg\min_{c>0} \sum_{n=1}^{N} \mathcal{L}\left(F_{m-1}(x_n) - cf_m(x_n), y_n\right)$$
    4. set $F_m(x) = F_{m-1}(x) - c_m f_m(x)$

Output: approximation function $F_M(x) = f_0(x) - \sum_{m=1}^{M} c_m f_m(x)$

Gradient boosting: examples

In gradient boosting

$$\sum_{n=1}^{N}\left(f_m(x_n)-\frac{\partial\mathcal{L}(r,y)}{\partial r}|_{r=F_{m-1}(x_n)}\right)^2\to\min_{f_m}$$

Consider specific cases:

- $\mathcal{L}=\frac{1}{2}\left(r-y\right)^2$
- $\mathcal{L}=[-ry]_+$

Table of Contents

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^{N} \mathcal{L}(\gamma,\ y_n)$

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^N \mathcal{L}(\gamma,\ y_n)$
2. For each step $m = 1, 2, ...M$:

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^{N} \mathcal{L}(\gamma,\ y_n)$
2. For each step $m = 1, 2, ...M$:
   1. calculate gradients $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\big|_{r=F_{m-1}(x_n)}$

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^N \mathcal{L}(\gamma, y_n)$
2. For each step $m = 1, 2, ...M$:
   1. calculate gradients $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
   2. fit regression tree $f_m(\cdot)$ on $\{(x_n, z_n)\}_{n=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^N \mathcal{L}(\gamma,\ y_n)$
2. For each step $m = 1, 2, ...M$:
   1. calculate gradients $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
   2. fit regression tree $f_m(\cdot)$ on $\{(x_n, z_n)\}_{n=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
   3. for each terminal region R_j^m, $j = 1, 2, ...J_m$ solve univariate optimization problem:

$$\gamma_j^m = \arg\min_\gamma \sum_{x_n \in R_j^m} \mathcal{L}(F_{m-1}(x_n) - \gamma,\ y_n)$$

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, ...N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^N \mathcal{L}(\gamma, y_n)$
2. For each step $m = 1, 2, ...M$:
   1. calculate gradients $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\Big|_{r=F_{m-1}(x_n)}$
   2. fit regression tree $f_m(\cdot)$ on $\{(x_n, z_n)\}_{n=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
   3. for each terminal region R_j^m, $j = 1, 2, ...J_m$ solve univariate optimization problem:
   $$\gamma_j^m = \arg\min_\gamma \sum_{x_n \in R_j^m} \mathcal{L}(F_{m-1}(x_n) - \gamma, y_n)$$
   4. update $F_m(x) = F_{m-1}(x) - \sum_{j=1}^{J_m} \gamma_j^m \mathbb{I}[x \in R_j^m]$

Gradient boosting of trees

Input: training dataset (x_n, y_n), $n = 1, 2, \ldots N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

1. Fit initial approximation *with constant*:
   $f_0(x) = \arg\min_\gamma \sum_{n=1}^N \mathcal{L}(\gamma,\, y_n)$
2. For each step $m = 1, 2, \ldots M$:
   1. calculate gradients $g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}|_{r=F_{m-1}(x_n)}$
   2. fit regression tree $f_m(\cdot)$ on $\{(x_n, z_n)\}_{n=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
   3. for each terminal region R_j^m, $j = 1, 2, \ldots J_m$ solve univariate optimization problem:
   $$\gamma_j^m = \arg\min_\gamma \sum_{x_n \in R_j^m} \mathcal{L}(F_{m-1}(x_n) - \gamma,\, y_n)$$
   4. update $F_m(x) = F_{m-1}(x) - \sum_{j=1}^{J_m} \gamma_j^m \mathbb{I}[x \in R_j^m]$

Output: approximation function $F_M(x)$

Modification of boosting for trees

- Compared to first method of gradient boosting, boosting of regression trees finds additive coefficients individually for each terminal region R_j^m, not globally for the whole classifier $f_m(x)$.
- This is done to increase accuracy: forward stagewise algorithm cannot be applied to find R_j^m, but it can be applied to find γ_j^m, because second task is solvable for arbitrary L.
- Max depth K: interaction between K features
- Max leaves K: interaction between no more than $\leq K-1$ features
    - usually $2 \leq K \leq 8$
- M controls underfitting-overfitting trade-off and selected using validation set

Shrinkage & subsampling

- Shrinkage of general GB, step (d):

$$F_m(x) = F_{m-1}(x) - \alpha c_m f_m(x)$$

- Comments:
    - $\alpha \in (0, 1]$
    - $\alpha \downarrow \implies M \uparrow$ $(\alpha M \approx const)$
- Subsampling
    - increases speed of fitting
    - may increase accuracy (diversity of base learners$\uparrow$)

Linear loss function approximation

Consider sample (x, y) and $g(x) = \left.\frac{\partial \mathcal{L}(r,y)}{\partial r}\right|_{r=F(x)}$

$$\mathcal{L}(F(x) + f(x),\ y) \approx \mathcal{L}(F(x), y) + g(x)f(x)$$

=> $f(x)$ should be fitted to $-g(x)$.

Second order approximation (Newton's method)

- Consider general function minimization $L(w) \to \min_w$
- Let $w^* = \arg\min_w L(w)$
- Then $L'(w^*) = \mathbf{0}$
- Taylor expansion of $L'(w)$ around w to w^*:

$$L'(w^*) = 0 = L'(w) + L''(w)(w^* - w) + o(\|w - w^*\|)$$

- It follows that

$$w^* - w = -\left[L''(w)\right]^{-1} L'(w) + o(\|w - w^*\|)$$

- Iterative scheme for minimization:

$$w \leftarrow w - \left[L''(w)\right]^{-1} L'(w)$$

  - it is scaled gradient descent
  - speed of convergence faster (uses quadratic approximation in Taylor expansion)
  - converges in one step for quadratic $L(w)$.

Quadratic loss function approximation

Define $g(x)=\left.\frac{\partial\mathcal{L}(r,y)}{\partial r}\right|_{r=F(x)}$, $h(x)=\left.\frac{\partial^2\mathcal{L}(r,y)}{\partial r^2}\right|_{r=F(x)}$

$$\mathcal{L}(F(x)+f(x),\ y)\approx$$

$$\mathcal{L}(F(x),y)+g(x)f(x)+\frac{1}{2}h(x)\left(f(x)\right)^2=$$

$$\frac{1}{2}h(x)\left(f(x)+\frac{g(x)}{h(x)}\right)^2+const(f(x))$$

So $f(x)$ should be fitted to $-g(x)/h(x)$ with weight $h(x)$.

- $h(x)\ge 0$ around local minimum.

Case $y \in \{1, 2, ...C\}$

One-vs-all, one-vs-one, error-correcting-codes.

Case $y \in \{1, 2, ... C\}$

One-vs-all, one-vs-one, error-correcting-codes.
Alternatively can optimize $\mathcal{L}(F(x), y)$ for $F(x) \in \mathbb{R}^C$

- $F(x) = \{p(y = c|x)\}_{c=1}^{C}$, y - one-hot encoded true class
- $S(F(x), y) = F(x)^T y = p(y = \text{correct class}|x)$ - score on (x, y)
- $g_n = -\frac{\partial \mathcal{S}(r,y)}{\partial r}|_{r=F_{m-1}(x_n)} \in \mathbb{R}^C$
- $\sum_{n=1}^{N}(f_m(x_n) - g_n)^2 \to \min_{f_m}$ yields vector C-dim. regression.
- may use quadratic approximation
    - for efficient inverting of $\left(\frac{\partial^2}{\partial r^2}\mathcal{L}(r, y)\Big|_{r=F(x)}\right)$ may use diagonal approximation.

xgBoost

- One of the most popular algorithms on kaggle.
- Uses decision trees as base learners:
  - $f_m \in \{f(x) = w_{q(x)}\}$,
  - T total number of leaves.
  - $q(x)$ maps $x \in \mathbb{R}^D$ to leaf number
  - $w \in \mathbb{R}^T$ predictions for leaves.

xgBoost

- Loss - 2nd order approximation with with regularization:

$$
\begin{aligned}
\mathcal{L}(f_m) &= \sum_{n=1}^{N} \mathcal{L}(F^{(m-1)}(x_n),\, y_n) \\
&\approx \sum_{n=1}^{N} \left[\mathcal{L}(F^{(m-1)}(x_n),\, y_n) + g_n f_m(x_n) + \frac{1}{2} h_n f_m^2(x_n) \right] \\
&\quad + \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2
\end{aligned}
$$

- Tree impurity function matches original loss $\mathcal{L}(\cdot,\cdot)$.
- Efficiency optimization:
  - feature values may be discretized for speed
  - parallelization over multiple CPU cores and with GPU

Types of boosting

- Loss function $\mathcal{L}$:
    - $\mathcal{L}(|f(x) - y|)$ - regression
    - $F(y \cdot score(y = +1|x))$ - binary classification
    - $\mathcal{L}(F(x), y)$ for $F(x)$, $y \in \mathbb{R}^C$ - multiclass classification
- Optimization
    - analytical (Adaboost)
    - gradient based
    - based on quadratic approximation
- Base learners
    - continious
    - discrete
- Classification
    - binary
    - multiclass
- Extensions: shrinkage, subsampling