

Boosting

Victor Kitov

Motivation for ensembles

- Consider M classifiers $f_1(x), \dots, f_M(x)$, performing binary classification.
- Let ξ_1, \dots, ξ_M denote indicators of mistakes by f_1, \dots, f_M on particular observation x
- Suppose ξ_1, \dots, ξ_M are independent binomial variables with $P(\xi_i = 1) = p$
- Then $\mathbb{E}\xi_i = p$, $\text{Var}[\xi_i] = p(1 - p)$
- Consider $F(x)$ be aggregating classifier, assigning x to the class with maximum votes among $f_1(x), \dots, f_M(x)$.
- Consider

$$\eta = \frac{\xi_1 + \dots + \xi_M}{M}$$

- Probability of mistake = probability that majority of ξ_1, \dots, ξ_M are ones = $P(\eta > 0.5)$.
- $P(\eta > 0.5) \rightarrow 0$ as $M \rightarrow \infty$ because $\mathbb{E}\eta = p$, $\text{Var}[\eta] = \frac{p(1-p)}{M}$.

Linear ensembles

Linear ensemble:

$$F(x) = f_0(x) + c_1 h_1(x) + \dots + c_M h_M(x)$$

Regression: $\hat{y}(x) = F(x)$

Binary classification: $\text{score}(y|x) = F(x)$, $\hat{y}(x) = \text{sign } F(x)$

- Notation: $h_1(x), \dots, h_M(x)$ are called *base learners*, *weak learners*, *base models*.
- Too expensive to optimize $f_0(x), h_1(x), \dots, h_M(x)$ and c_1, \dots, c_M jointly for large M .
- Idea: optimize $f_0(x)$ and then each pair $(h_m(x), c_m)$ greedily.
- After ensemble is built we can fine-tune c_1, \dots, c_M by fitting features $f_0(x), h_1(x), \dots, h_M(x)$ with linear regression/classifier.

Forward stagewise additive modeling (FSAM)

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; loss function $\mathcal{L}(f, y)$, general form of “base learner” $h(x|\gamma)$ (dependent from parameter γ) and the number M of successive additive approximations.

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

$$(c_m, h_m) = \arg \min_{h, c} \sum_{i=1}^N \mathcal{L}(f_{m-1}(x_i) + ch(x_i), y_i)$$

- 2 set

$$f_m(x) = f_{m-1}(x) + c_m h_m(x)$$

Output: approximation function

$$f_M(x) = f_0(x) + \sum_{m=1}^M c_m h_m(x)$$

Comments on FSAM

- Number of steps M should be determined by performance on validation set.
- Step 1 need not be solved accurately, since its mistakes are expected to be corrected by future base learners.
 - we can take $f_0(x) = \arg \min_{\beta \in \mathbb{R}} \sum_{i=1}^N \mathcal{L}(\beta, y_i)$ or simply $f_0(x) \equiv 0$.
- By similar reasoning there is no need to solve 2.1 accurately
 - typically very simple base learners are used such as trees of depth=1,2,3.
- For some loss functions, such as $\mathcal{L}(y, f(x)) = e^{-yf(x)}$ we can solve FSAM explicitly.
- For general loss functions gradient boosting scheme should be used.

Table of Contents

- 1 Adaboost
- 2 Gradient boosting
- 3 Boosting extensions

Adaboost (discrete version): assumptions

- binary classification task $y \in \{+1, -1\}$
- family of base classifiers $h(x) = h(x|\gamma)$ where γ is some fitted parametrization.
- $h(x) \in \{+1, -1\}$
- classification is performed with
$$\hat{y} = \text{sign}\{f_0(x) + c_1 f_1(x) + \dots + c_M f_M(x)\}$$
- optimized loss is $\mathcal{L}(y, f(x)) = e^{-yf(x)}$
- FSAM is applied

Adaboost (discrete version): algorithm

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; number of additive weak classifiers M , a family of weak classifiers $h(x) \in \{+1, -1\}$, trainable on weighted datasets.

- 1 Initialize observation weights $w_i = 1/n$, $i = 1, 2, \dots, n$.
- 2 for $m = 1, 2, \dots, M$:
 - 1 fit $h^m(x)$ to training data using weights w_i
 - 2 compute weighted misclassification rate:

$$E_m = \frac{\sum_{i=1}^N w_i \mathbb{I}[h^m(x) \neq y_i]}{\sum_{i=1}^N w_i}$$

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

$$w_i \leftarrow w_i e^{2c_m}, i \in \{i : h^m(x_i) \neq y_i\}$$

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

Table of Contents

- 1 Adaboost
- 2 Gradient boosting
- 3 Boosting extensions

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

$$F(w) \rightarrow \min_w, \quad w \in \mathbb{R}^N$$

Gradient descend algorithm:

INPUT:

η -parameter, controlling the speed of convergence

M -number of iterations

ALGORITHM:

initialize w

for $m = 1, 2, \dots, M$:

$$\Delta w = \frac{\partial F(w)}{\partial w}$$

$$w = w - \eta \Delta w$$

Modified gradient descent algorithm

INPUT:

M -number of iterations

ALGORITHM:

initialize w

for $m = 1, 2, \dots, M$:

$$\Delta w = \frac{\partial F(w)}{\partial w}$$

$$c^* = \arg \min_c F(w - c\Delta w)$$

$$w = w - c^* \Delta w$$

Gradient boosting

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

Gradient boosting

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; 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_i, y_i) , $i = 1, 2, \dots, N$; 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 $m = 1, 2, \dots, M$:

Gradient boosting

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

- ❶ Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
- ❷ For each step $m = 1, 2, \dots, M$:
 - ❶ calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y_i)}{\partial r} \Big|_{r=f^{m-1}(x_i)}$

Gradient boosting

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

- ❶ Fit initial approximation $f_0(x)$ (might be taken $f_0(x) \equiv 0$)
- ❷ For each step $m = 1, 2, \dots, M$:
 - ❶ calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y_i)}{\partial r} \Big|_{r=f^{m-1}(x_i)}$
 - ❷ fit h_m to $\{(x_i, z_i)\}_{i=1}^N$, for example by solving

$$\sum_{n=1}^N (h_m(x_n) - z_n)^2 \rightarrow \min_{h_m}$$

Gradient boosting

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; 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 $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y_i)}{\partial r} \Big|_{r=f^{m-1}(x_i)}$
 - ② fit h_m to $\{(x_i, z_i)\}_{i=1}^N$, for example by solving

$$\sum_{n=1}^N (h_m(x_n) - z_n)^2 \rightarrow \min_{h_m}$$

- ③ solve univariate optimization problem:

$$\sum_{i=1}^N \mathcal{L}(f_{m-1}(x_i) + c_m h_m(x_i), y_i) \rightarrow \min_{c_m \in \mathbb{R}_+}$$

Gradient boosting

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; 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 $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y_i)}{\partial r} \Big|_{r=f^{m-1}(x_i)}$
 - ② fit h_m to $\{(x_i, z_i)\}_{i=1}^N$, for example by solving

$$\sum_{n=1}^N (h_m(x_n) - z_n)^2 \rightarrow \min_{h_m}$$

- ③ solve univariate optimization problem:

$$\sum_{i=1}^N \mathcal{L}(f_{m-1}(x_i) + c_m h_m(x_i), y_i) \rightarrow \min_{c_m \in \mathbb{R}_+}$$

- ④ set $f_m(x) = f_{m-1}(x) + c_m h_m(x)$

Gradient boosting

Input: training dataset (x_i, y_i) , $i = 1, 2, \dots, N$; 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 $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y_i)}{\partial r} \Big|_{r=f^{m-1}(x_i)}$
 - ② fit h_m to $\{(x_i, z_i)\}_{i=1}^N$, for example by solving

$$\sum_{n=1}^N (h_m(x_n) - z_n)^2 \rightarrow \min_{h_m}$$

- ③ solve univariate optimization problem:

$$\sum_{i=1}^N \mathcal{L}(f_{m-1}(x_i) + c_m h_m(x_i), y_i) \rightarrow \min_{c_m \in \mathbb{R}_+}$$

- ④ set $f_m(x) = f_{m-1}(x) + c_m h_m(x)$

Output: approximation function $f_M(x) = f_0(x) + \sum_{m=1}^M c_m h_m(x)$

Gradient boosting: examples

In gradient boosting

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

Specific cases:

- $\mathcal{L} = \frac{1}{2} (r - y)^2 \Rightarrow -\frac{\partial \mathcal{L}}{\partial r} = -(r - y) = (y - r)$
 - $h_m(x)$ is fitted to compensate regression errors $(y - f_{m-1}(x))$
- $\mathcal{L} = [-ry]_+ \Rightarrow -\frac{\partial \mathcal{L}}{\partial r} = \begin{cases} 0, & ry > 0 \\ y, & ry < 0 \end{cases}$
 - $h_m(x)$ is fitted to $y \mathbb{I}[f(x)y < 0]$

Table of Contents

- 1 Adaboost
- 2 Gradient boosting
- 3 Boosting extensions

Gradient boosting of trees

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

- 1 Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$

Gradient boosting of trees

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

- 1 Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$

- 2 For each step $m = 1, 2, \dots, M$:

Gradient boosting of trees

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

- 1 Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$

- 2 For each step $m = 1, 2, \dots, M$:

- 1 calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f^{m-1}(x)}$

Gradient boosting of trees

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

- ① Fit constant initial approximation $f_0(x)$:
$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$
- ② For each step $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f^{m-1}(x)}$
 - ② fit regression tree h^m on $\{(x_i, z_i)\}_{i=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.

Gradient boosting of trees

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

- ① Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$
- ② For each step $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f_{m-1}(x)}$
 - ② fit regression tree h^m on $\{(x_i, z_i)\}_{i=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
 - ③ for each terminal region R_j^m , $j = 1, 2, \dots, J_m$ solve univariate optimization problem:

$$\gamma_j^m = \arg \min_{\gamma} \sum_{x_i \in R_j^m} \mathcal{L}(f_{m-1}(x_i) + \gamma, y_i)$$

Gradient boosting of trees

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

- ① Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$
- ② For each step $m = 1, 2, \dots, M$:
 - ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f_{m-1}(x)}$
 - ② fit regression tree h^m on $\{(x_i, z_i)\}_{i=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
 - ③ for each terminal region R_j^m , $j = 1, 2, \dots, J_m$ solve univariate optimization problem:

$$\gamma_j^m = \arg \min_{\gamma} \sum_{x_i \in R_j^m} \mathcal{L}(f_{m-1}(x_i) + \gamma, y_i)$$

- 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_i, y_i) , $i = 1, 2, \dots, N$; loss function $\mathcal{L}(f, y)$ and the number M of successive additive approximations.

- ① Fit constant initial approximation $f_0(x)$:

$$f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N \mathcal{L}(\gamma, y_i)$$

- ② For each step $m = 1, 2, \dots, M$:

- ① calculate derivatives $z_i = -\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f_{m-1}(x)}$
- ② fit regression tree h^m on $\{(x_i, z_i)\}_{i=1}^N$ with some loss function, get leaf regions $\{R_j^m\}_{j=1}^{J_m}$.
- ③ for each terminal region R_j^m , $j = 1, 2, \dots, J_m$ solve univariate optimization problem:

$$\gamma_j^m = \arg \min_{\gamma} \sum_{x_i \in R_j^m} \mathcal{L}(f_{m-1}(x_i) + \gamma, y_i)$$

- 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 $h^m(x)$.
- 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 leaves J
 - interaction between no more than $J - 1$ terms
 - usually $4 \leq J \leq 8$
- M controls underfitting-overfitting tradeoff and selected using validation set

Shrinkage & subsampling

- Shrinkage of general GB, step (d):

$$f_m(x) = f_{m-1}(x) + \nu c_m h_m(x)$$

- Shrinkage of trees GB, step (d):

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}[x \in R_{jm}]$$

- Comments:

- $\nu \in (0, 1]$
- $\nu \downarrow \implies M \uparrow$

- Subsampling

- increases speed of fitting
- may increase accuracy

Linear loss function approximation

Consider sample (x, y) .

$$\mathcal{L}(f(x) + h(x), y) \approx \mathcal{L}(f(x), y) + h(x) \left. \frac{\partial \mathcal{L}(r, y)}{\partial r} \right|_{r=f(x)}$$

$\Rightarrow h(x)$ should be fitted to $-\left. \frac{\partial \mathcal{L}(r, y)}{\partial r} \right|_{r=f(x)}$.

Newton method of optimization

- Suppose we want $F(w) \rightarrow \min_w$
- Let $w^* = \arg \min_w F(w)$
- Then $F'(w^*) = \mathbf{0}$
- Taylor expansion of $F'(w)$ around w to w^* :

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

- It follows that

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

- Iterative scheme for minimization:

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

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

Quadratic loss function approximation

$$\begin{aligned} \mathcal{L}(f(x) + h(x), y) &\approx \\ \mathcal{L}(f(x), y) + h(x) \frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f(x)} + \frac{1}{2} (h(x))^2 \frac{\partial^2 \mathcal{L}(r, y)}{\partial r^2} \Big|_{r=f(x)} &= \\ \frac{1}{2} \frac{\partial^2 \mathcal{L}(r, y)}{\partial r^2} \Big|_{r=f(x)} \left(h(x) + \frac{\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f(x)}}{\frac{\partial^2 \mathcal{L}(r, y)}{\partial r^2} \Big|_{r=f(x)}} \right)^2 + \text{const}(h(x)) \end{aligned}$$

$\Rightarrow h(x)$ should be fitted to $-\frac{\partial \mathcal{L}(r, y)}{\partial r} \Big|_{r=f(x)} / \frac{\partial^2 \mathcal{L}(r, y)}{\partial r^2} \Big|_{r=f(x)}$ with
weight $\frac{\partial^2 \mathcal{L}(r, y)}{\partial r^2} \Big|_{r=f(x)}$

Case of $C \geq 3$ classes

- Can fit C independent boostings $\{f_y(x)\}_{y=1}^C$ (one vs. all scheme)
 - $\hat{y}(x) = \arg \max_y f_y(x)$
- Alternatively can optimize multivariate $\mathcal{L}(f(x), y) = -\ln p(y|x)$
 - using linear or quadratic approximation
 - for quadratic approximation need to invert $\left. \frac{\partial^2}{\partial r^2} F(r, y) \right|_{r=f(x)}$.
Can use diagonal approximation.

Types of boosting

- Loss function F :
 - $F(|f(x) - y|)$ - regression
 - $-\ln p(y|x)$ or $F(y \cdot \text{score}(y = +1|x))$ - binary classification
 - $-\ln p(y|x)$ - multiclass classification
- Optimization
 - analytical (AdaBoost)
 - gradient based
 - based on quadratic approximation
- Base learners
 - continuous
 - discrete
- Classification
 - binary
 - multiclass
- Extensions: shrinkage, subsampling