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 $(f_m(x),\ c_m)$ greedily.

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 $f(x|\gamma)$ (parametrized by γ)
- 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

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

Comments

- M should be determined by performance on validation set.
  - may overfit!
- 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 $y \in \{+1, -1\}$
- $f_m(x) \in \{+1, -1\}$
- 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 $h(x) \in \{+1, -1\}$
  - 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 $w_i = 1/N$, $i = 1, 2, \dots N$.
2. for $m = 1, 2, \dots M$:
    1. fit $f_m(x)$ to training data using weights w_i
    2. compute weighted misclassification rate:
    $$E_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}[f_m(x) \neq y_i]}{\sum_{i=1}^{N} w_i}$$
    3. compute weighting factor $c_m = \frac{1}{2} \ln\left((1 - E_m)/E_m\right)$
    4. if $E_M > 0.5$ or $E_M = 0$: terminate procedure.
    5. increase all weights, where misclassification with $f_m(x)$ was made:
    $$w_i \leftarrow w_i \frac{1 - E_m}{E_m},\ i \in \{i :\ f_m(x_i) \neq y_i\}$$
3. Return classifier $F_M(x) = \text{sign}\left(\sum_{m=1}^{M} c_m f_m(x)\right)$

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 descend algorithm:

- given w move in the direction of steepest descent, given by $\Delta w := -g(w)$, $g(w) = \nabla_w L(w)$

Gradient descent algorithm

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

INPUT:
step size ε
number of iterations M

ALGORITHM:
initialize w
for $m = 1, 2, ...M$:
    $\Delta w = -g(w)$
    $w = w + \varepsilon \Delta w$

Modified gradient descent algorithm

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

INPUT:
number of iterations M

ALGORITHM:
initialize w
for $m = 1, 2, ...M$:
  $\Delta w = -g(w)$
  $c^* = \arg\min_{c>0} L(w + c\Delta w)$
  $w = w + c^* \Delta w$

Gradient boosting

- Now consider $L\left(f(x_1),...f(x_N)\right) = \sum_{n=1}^{N} \mathcal{L}\left(f(x_n), y_n\right) \to \min_{f(\cdot)}$
- 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 $z_n = -g(x_n)$, where $g(x_n) = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\big|_{r=f^{m-1}(x_n)}$
  - fit base learner $f_m(x)$ to $\{(x_n, z_n)\}_{n=1}^{N}$

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 targets $z_n := -g_n$ $\left(g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\Big|_{r=F_{m-1}(x_n)}\right)$

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 targets $z_n := -g_n$ $\quad \left(g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r} \Big|_{r=F_{m-1}(x_n)} \right)$
   2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^{N}$, for example by solving

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

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 targets $z_n := -g_n$ $\left(g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\Big|_{r=F_{m-1}(x_n)} \right)$
    2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^N$, for example by solving
    $$\sum_{n=1}^{N} (f_m(x_n) - z_n)^2 \to \min_{f_m}$$
    3. solve univariate optimization problem:
    $$\sum_{n=1}^{N} \mathcal{L}\left(F_{m-1}(x_n) + c_m f_m(x_n), y_n\right) \to \min_{c_m > 0}$$

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 $m = 1, 2, ...M$:
    1. calculate targets $z_n := -g_n$ $\left(g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r}\Big|_{r=F_{m-1}(x_n)} \right)$
    2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^{N}$, for example by solving
    $$\sum_{n=1}^{N} (f_m(x_n) - z_n)^2 \to \min_{f_m}$$
    3. solve univariate optimization problem:
    $$\sum_{n=1}^{N} \mathcal{L}\left(F_{m-1}(x_n) + c_m f_m(x_n), y_n\right) \to \min_{c_m > 0}$$
    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 targets $z_n := -g_n$ $\quad \left(g_n = \frac{\partial \mathcal{L}(r, y_n)}{\partial r} \Big|_{r = F_{m-1}(x_n)} \right)$
   2. fit $f_m(\cdot)$ to $\{(x_n, z_n)\}_{n=1}^{N}$, for example by solving
   $$\sum_{n=1}^{N} (f_m(x_n) - z_n)^2 \to \min_{f_m}$$
   3. solve univariate optimization problem:
   $$\sum_{n=1}^{N} \mathcal{L}\left(F_{m-1}(x_n) + c_m f_m(x_n), y_n\right) \to \min_{c_m > 0}$$
   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

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

Sample cases:

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

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 (fitting each base model on subset of objects and/or features)
  - increases speed of fitting
  - may increase accuracy due to increased diversity of models

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

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
- $\mathcal{L}(F(x), y) = F(x)^T y = p(y = \text{correct class}|x)$
- $z_n = -\frac{\partial \mathcal{L}(r,y)}{\partial r}|_{r=F_{m-1}(x_n)} \in \mathbb{R}^C$
- $\sum_{n=1}^{N}(f_m(x_n) - z_n)^2 \to \min_{f_m}$ 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