Popular ensemble methods

Victor Kitov

Yandex School of Data Analysis

Table of contents

Bagging

- Random selection of
  - samples (with replacement)
  - features (without replacement)
- During bootstrap approximately $1 - 1/e \approx 2/3$ samples are retained and $1/e \approx 1/3$ samples left out for large training sets.

Random forests

Input: training dataset $TDS = \{(x_i, y_i),\ 1 = 1, 2, ...n\}$; the number of trees B and the size of feature subsets m.

1. for $b = 1, 2, ...B$:
    1. generate random training dataset TDS^b of size n by sampling (x_i, y_i) pairs from TDS with replacement.
    2. build a tree using TDS^b training dataset with feature selection for each node from random subset of features of size m (generated individually for each node).
2. Evaluate the quality by assigning output to x_i, $i = 1, 2, ...n$ using majority vote (classification) or averaging (regression) among trees with $b \in \{b : (x_i, y_i) \notin T^b\}$

Output: B trees. Classification is done using majority vote and regression using averaging of B outputs.

Comments

- Random forests use random selection on both samples and features
- Left out samples are used for evaluation of model performance.
- Less interpretable than individual trees
- Pro: Parallel implementation
- Contra: different trees are not targeted to correct mistakes of each other

1. Popular ensemble methods
   - Bagging and random forest
   - Boosting

Forward stagewise additive modeling

Input: training dataset (x_i, y_i), $i = 1, 2, ...n$; loss function $L(f, y)$, general form of additive classifier $h(x, \gamma)$ (dependent from parameter γ) 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 $m = 1, 2, ...M$:
    1. find next best classifier
    $$(c_m, \gamma_m) = \arg\min \sum_{i=1}^{n} L(f_{m-1}(x_i) + c_m h(x_i, \gamma_m),\ y_i)$$
    2. set
    $$f_m(x) = f_{m-1}(x) + c_m h(x, \gamma_m)$$

Output: approximation function $f^M(x) = f^0(x) + \sum_{j=1}^{M} c_j h(x, \gamma_m)$

Adaboost algorithm is obtained for $L(y, f(x)) = e^{-yf(x)}$

Adaboost (discrete version)

Assumptions: loss function $L(y, f(x)) = e^{-yf(x)}$
Input: training dataset (x_i, y_i), $i = 1, 2, ...n$; number of additive weak classifiers M, a family of weak classifiers $h(x)$, outputting only +1 or -1 (binary classification) and trainable on weighted datasets.

1. Initialize observation weights $w_i = 1/n$, $i = 1, 2, ...n$.
2. for $m = 1, 2, ...M$:
    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. compute $\alpha_m = \ln\left((1 - E_m)/E_m\right)$
    4. increase all weights, where misclassification with $h^m(x)$ was made:
    $$w_i \leftarrow w_i e^{\alpha_m}, \; i \in \{i : h^m(x_i) \neq y_i\}$$

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

Adaboost derivation

Set initial approximation $f^0(x) \equiv 0$.
Apply forward stagewise algorithm for $m = 1, 2, ...M$:

$$\begin{aligned}(c_m, h^m) &= \arg\min_{c_m, h^m} \sum_{i=1}^{n} L(f_{m-1}(x_i) + c_m h^m(x), \, y_i) \\ &= \arg\min_{c_m, h^m} \sum_{i=1}^{n} e^{-y_i f_{m-1}(x_i)} e^{-c_m y_i h^m(x)} \\ &= \arg\min_{c_m, h^m} \sum_{i=1}^{n} w_i^m e^{-c_m y_i h^m(x_i)}, \quad w_i^m = e^{-y_i f_{m-1}(x_i)}\end{aligned}$$

Since $c_m \geq 0$ and $y_i h^m(x_i) \in \{-1, +1\}$ minimum with respect to $h^m(x)$ is attained at

$$h^m(x_i) = \arg\min_h \sum_{i=1}^{n} w_i^m \mathbb{I}[h(x_i) \neq y_i]$$

Adaboost derivation

Denote $F(c_m) = \sum_{i=1}^n w_i^m \exp(-c_m y_i h^m(x_i))$. Then

$$\frac{\partial F(c_m)}{\partial c_m} = -\sum_{i=1}^n w_i^m e^{-c_m y_i h^m(x_i)} y_i h^m(x_i) = 0$$

$$-\sum_{i:h^m(x_i)=y_i} w_i^m e^{-c_m} + \sum_{i:h^m(x_i)\neq y_i} w_i^m e^{c_m} = 0$$

$$e^{2c_m} = \frac{\sum_{i:h^m(x_i)=y_i} w_i^m}{\sum_{i:h^m(x_i)\neq y_i} w_i^m}$$

$$c_m = \frac{1}{2}\ln\frac{\left(\sum_{i:h^m(x_i)=y_i} w_i^m\right) / \left(\sum_{i=1}^n w_i^m\right)}{\left(\sum_{i:h^m(x_i)\neq y_i} w_i^m\right) / \left(\sum_{i=1}^n w_i^m\right)} = \frac{1}{2}\ln\frac{1-E_m}{E_m} = \frac{1}{2}\alpha_m,$$

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

Adaboost derivation

Weights recalculation:

$$w_i^{m+1} \overset{df}{=} e^{-y_i f_m(x_i)} = e^{-y_i f_{m-1}(x_i)} e^{-y_i c_m h^m(x_i)}$$

Noting that $-y_i h^m(x_i) = 2\mathbb{I}[h^m(x_i) \neq y_i] - 1$, we can rewrite:

$$\begin{aligned} w_i^{m+1} &= e^{-y_i f_{m-1}(x_i)} e^{c_m(2\mathbb{I}[h^m(x_i)\neq y_i]-1)} = \\ &= w_i^m e^{2c_m \mathbb{I}[h^m(x_i)\neq y_i]} e^{-c_m} \propto w_i^m e^{2c_m \mathbb{I}[h^m(x_i)\neq y_i]} \end{aligned}$$

On the last step we used the property that classification result is not affected by multiplication of all weights by constant.

Gradient boosting

- For general loss function L forward stagewise algorithm can be solved explicitly in rare cases. In general gradient boosting is applied.
- Gradient boosting is analogous to steepest descent:
    - function approximation is composed of sums of approximations, each of which approximates $\partial L/\partial f$.

Gradient boosting

Input: training dataset (x_i, y_i), $i = 1, 2, ...n$; loss function $L(f, y)$ 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 derivatives $z_i = -\frac{\partial L(r,y)}{\partial r}\big|_{r=f^{m-1}(x)}$
    2. train additive approximation with classifier h^m on (x_i, z_i), $i = 1, 2, ...n$ with some loss function $\sum_{i=1}^n \mathcal{L}\left(h^m(x_i),\, z_i\right)$
    3. solve univariate optimization problem:

$$\sum_{i=1}^{n} L\left(f^{m-1}(x_i) + c_m h^m(x_i), y_i\right) \to \min_{c_m \in \mathbb{R}_+}$$

    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 of trees

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

1. Fit constant initial approximation $f^0(x)$:
   $f^0(x) = \arg\min_\gamma \sum_{i=1}^n L(\gamma, y_i)$
2. For each step $m = 1, 2, ...M$:
   1. calculate derivatives $z_i = -\frac{\partial L(r,y)}{\partial r}\big|_{r=f^{m-1}(x)}$
   2. train regression tree h^m on (x_i, z_i), $i = 1, 2, ...n$ with some loss function $\sum_{i=1}^n \mathcal{L}\left(h^m(x_i), z_i\right)$ and extract terminal regions R_{jm}, $j = 1, 2, ...J_m$.
   3. for each terminal region R_{jm}, $j = 1, 2, ...J_m$ solve univariate optimization problem:
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(f^{m-1}(x_i) + \gamma, y_i)$$
   4. update $f^m(x) = f^{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}[x \in R_{jm}]$

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_{jm}, not globally for the whole classifier $h^m(x)$.
- This is done to increase accuracy: forward stagewise algorithm cannot be applied to find R_{jm}, but it can be applied to find γ_{jm}, because second task is solvable for arbitrary L.

Loss function selection

- Usually $\mathcal{L}(h,z)=(h-z)^2$, though using absolute loss or Huber function loss makes procedure more robust to outliers.
- $L(\widehat{y},y)$ specification:
    - in regression $L(\widehat{y},y)$ is set to regression loss function:
        - squared deviation, absolute deviation, Huber loss
        - when $L(\widehat{y},y)=(\widehat{y}-y)^2$ method is called *L2Boost*.
    - in classification $L(f,y)$ is set to margin loss function:
        - exponential $L(f,y)=e^{-fy}$ or log-loss $L(f,y)=\ln(1+e^{-fy})$.
        - log-loss optimization with approximate solution from single step of Newton-Rhapson method is called *LogitBoost*
        - log-loss optimization yields not only classes, but also class probabilities.