Lecture 8 – Boosting

Martin Spindler

2016-04-26

Introduction

- One of the most powerful learning ideas
- Originally for classification problems, but extended to regression settings
- Idea: Combining the output of many "weak" classifiers to produce a powerful committee
- Weak classifier: classifier which error rate is only slightly better than random guessing.
- "best off-the-shelf classifier in the world" (Breiman, 1998)

Boosting for Classification

- Freund and Schapire (1997): AdaBoost.M1
- Output variable $Y \in \{-1, 1\}$, X predictor variable, N observations (training sample)
- $G(x)$: classification rule with values in $\{-1, 1\}$
- Training error rate: $\bar{err} = 1/N \sum_{i=1}^{N} I(y_i \neq G(x_i))$
- Expected error rate (on new observations): $\mathbb{E}_{XY} I(Y \neq G(X))$

Boosting for Classification | Main idea

- Sequentially applying the weak classification algorithm to repeatedly modified versions of the data.
- This produces a sequence of weak classifiers $G_m(x), m = 1, \ldots, M$
- Finally, the predictions of all of them are then combined through a weighted majority vote to produce the final prediction:

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- Weights $\alpha_1, \ldots, \alpha_M$ computed by boosting. Higher influence to the more accurate classifiers in the sequence.

Boosting for Classification | Main idea | Weights

- Boosting applies different weights $w_1, \ldots, w_N$ to the training data at each step.
- Initially, $w_i = 1/N$
- For $m = 2, \ldots, M$ observations weights are individually modified and classifier is reapplied to weighted data.
- At step m those observations that were misclassified by the classifier $G_{m-1}(x)$ at the previous step have their weight increased, the weights for the correctly classified ones are decreased.
- Concentration on the training observations missed in the previous rounds.

Boosting for Classification | Algorithm

1. Initialize the observation weights $w_i = 1/N$
2. For $m = 1$ to M:
    - Fit a classifier $G_m(x)$ to the training data using weights w_i
    - Compute $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
    - Compute $\alpha_m = \log((1 - err_m)/err_m)$
    - Set $w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(x_i))]$
3. Output $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

Boosting as Additive Modelling

- Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions.
- Basis function expansion

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

where β_m are expansion coefficients and $b(x; \gamma)$ simple functions / basis functions parametrized by γ

Boosting as Additive Modelling

- Estimation by solving

$$\min_{\beta_m,\gamma_m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right) (*)$$

- Often hard to solve, but often feasible subproblem

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i; \beta b(x_i;\gamma))$$

Forward Stagewise Additive Modeling

- Idea: Solving (*) by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those already added.
- At each step m: Solve for the optimal β_m and $b(x; \gamma_m)$ given the current expansion $f_{m-1}(x)$; this gives $f_m(x)$ and continue.
- For squared-error loss:

$$L(y_i; f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2$$

Forward Stagewise Additive Modeling | Algorithm

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M:

- Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

- Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

Exponential Loss and AdaBoost

- AdaBoost as stagewise additive modeling using the loss function $L(y, f(x)) = \exp(-yf(x))$
- Solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))]$$

- This can be expressed as

$$(\beta_m, G_m) = \arg\min_{\beta, G} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

with $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$

Exponential Loss and AdaBoost

- Solution can be obtained in two steps.

1. For any value of $\beta > 0$, the solution for $G_m(x)$ is

$$G_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

Exponential Loss and AdaBoost

2. Plugin this G_m into the criterion function and solving for β gives:

$$\beta_m = 1/2 \log \frac{1 - err_m}{err_m},$$

where err_m is the minimized weighted error rate

$$err_m = \frac{\sum_i w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_i w_i^{(m)}}$$

Exponential Loss and AdaBoost

- Update is given by $f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$ and weights $w_i^{(m)} = w_i^{(m)} exp(-\beta_m y_i G_m(x_i))$
- Weights can be rewritten as $w_i^{(m+1)} = w_i^{(m)} exp(\alpha_m I(y_i \neq G_m(x_i))) exp(-\beta_m)$
- $\alpha_m = 2\beta_m$

Remarks on the Loss Function | Classification

- $yf(x)$ is called margin. Goal: maximize margin.
- Classification rule: $G(x) = sign(f(x))$
- Exponential loss:
- Computational easy (simple modular reweighting)
- $f^*(x) = \arg\min_{f(x)} E_{Y|x}(e^{-Yf(x)}) = 1/2 \log \frac{Pr(Y=1|x)}{Pr(Y=-1|x)}$ or equivalently $Pr(y=1|x) = \frac{1}{1+e^{-2f^*(x)}}$
- Hence, AdaBoost estimates one half of the log-odd of $P(Y=1|x)$
- Alternative; binomial negative log-likelihood or deviance (coded $\{0,1\}$) has the same population minimizer.

Remarks on the Loss Function | Regression Case

- Squared error loss: $L(y, f(x)) = (y - f(x))^2$ with population minimizer $f(x) = E[Y|x]$
- Mean absolute loss: $L(y, f(x)) = |y - f(x)|$ with population minimizer $f(x) = median[Y|x]$
- Huber Loss: $L(y, f(x)) = 1(|y - f(x)| \leq \delta)(y - f(x))^2 +$ $+1(|y - f(x)| > \delta)(2\delta|y - f(x)| - \delta^2)$

Boosting Trees

- Trees: Partition of the space of all joint predictors into disjoint regions R_j (terminal nodes of the tree) with

$$x \in R_j \rightarrow f(x) = \gamma_j$$

- Trees can be expressed as $T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$ with parameters $\Theta = \{R_j, \gamma_j\}_1^J$
- Usually estimated via recursive partitioning

Boosting Trees

- Boosted tree model: $f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$
- Forward stagewise procedure solves:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

- Simplification with squared-error loss and two-class classification with exponential loss (specialized tree-growing algorithm)

Numerical Optimization via Gradient Boosting

- Goal: approximate algorithms for solving $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_i L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$
- Loss to predict y using $f(x)$: $L(f) = \sum_i L(y_i; f(x_i))$ (e.g. f as sum of trees)
- $\hat{f} = \arg\min_f L(f)$ (**) where the "parameters" $f \in \mathbb{R}^N$ are the values of the approximating function $f(x_i)$ at each of the N data points x_i.
- Numerical optimization procedures solve (**) as as sum of component vectors $f_M = \sum_{m=0}^{M} h_m, h_m \in \mathbb{R}^N$ and h_0 starting value / initial guess.
- Numerical methods differ in how to specify h_m

Gradient Boosting | Steepest Descent

- $h_m = \rho_m g_m$ (steepest descent) with ρ_m scalar and $g_m \in \mathbb{R}^N$ is the gradient of $L(f)$ evaluated at $f = f_{m-1}$
- $g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}$
- Step length ρ_m solves

$$\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)$$

- Updating: $f_m = f_{m-1} - \rho g_m$
- **Greedy Strategy**

Gradient Boosting

- Gradient is the unconstrained maximal descent direction. Only defined at the training data points x_i, but goal is generalization.
- Idea: Approximate negative gradient by some "model", e.g. tree $T(x; \Theta_m)$ at mth iteration whose predictions t_m are as close as possible to the negative gradient.
- This leads to

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_i (-g_{im} - T(x_i; \Theta))^2.$$

Gradient Tree Boosting Algorithm

1. Initialize $f_0(x) = \arg\min_\gamma \sum_i L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, \ldots, N$ compute

     $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   - Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$.
   - For $j = 1, 2, \ldots, J_m$ compute

     $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$

Remarks

- Shrinkage: $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$
- Size of trees J in each step: important choice, usually $2 < J < 10$ (by experimenting)
- Early stopping required. (When to stop?)
- Penalization / tuning parameters: M, J, ν