Boosting and Additive Trees

Chapter 10 of

Overview

1. Boosting:
    a. Intuition
    b. AdaBoost
    c. Boosting as Forward Stagewise Additive Modeling
    d. Loss functions robustness
2. Boosting trees:
    a. Gradient Boosted Models
    b. Regularization
    c. Interpretation

Boosting

Intuition

Boosting is one of the most powerful learning ideas introduced in last 30 years. It can be used both for classification and regression (we will focus on two-class classification with classes {-1, 1}).

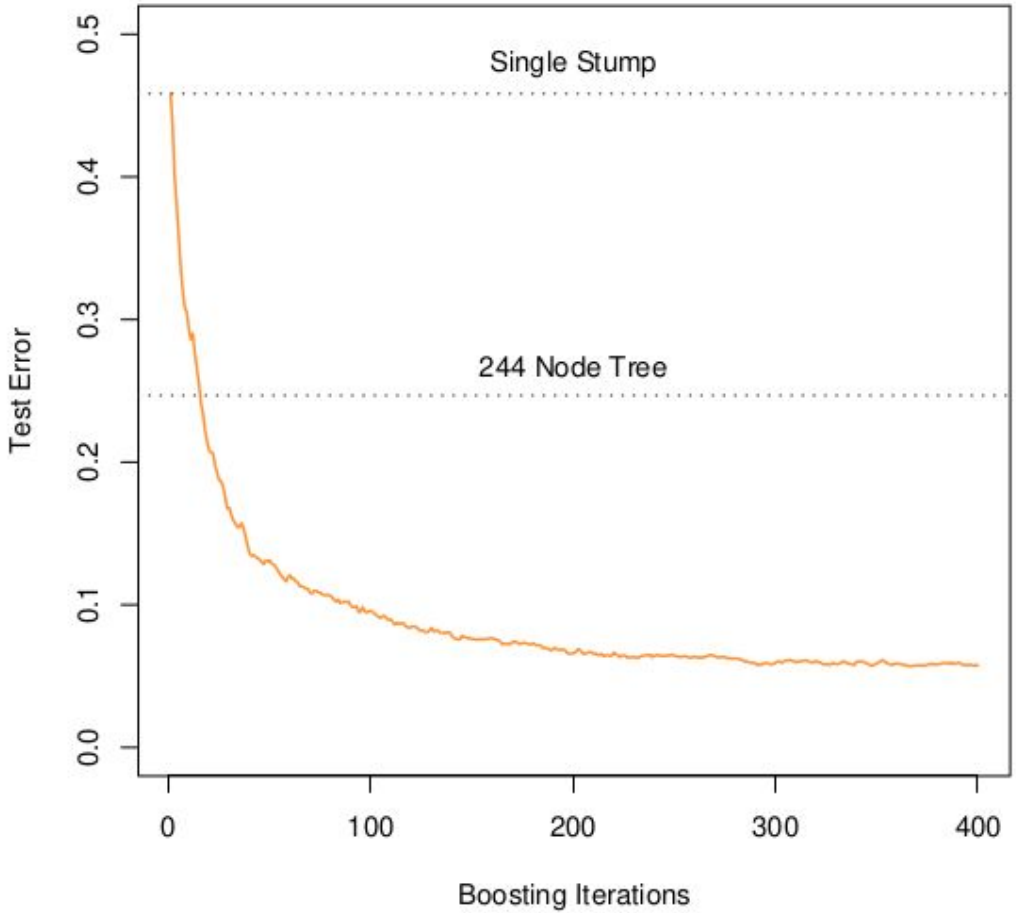
How does it work?:

- Combination of weak classifiers into powerful committee
- Sequential application of weak classificator on repeatedly modified versions of data
- At each step, weight of previously misclassified samples are increased
- Predictions of weak classifiers are combined through majority vote

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

Algorithm 10.1 *AdaBoost.M1.*

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


Single Stump
244 Node Tree
Test Error
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
Boosting Iterations

Boosting as Additive Model

Basis function expansion:

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

Boosting:

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

To train this kind of model, we need to find:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

Problem: This is very computationally intensive!

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

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

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

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

Examples of Forward Stagewise

Squared error loss:

$$L(y, f(x)) = (y - f(x))^2$$

Loss at m-th iteration of Forward Stagewise algorithm:

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned}$$

New term added to the expansion is fitted to current residuals.

Examples of Forward Stagewise

Exponential loss:

$$L(y, f(x)) = \exp(-y\, f(x))$$

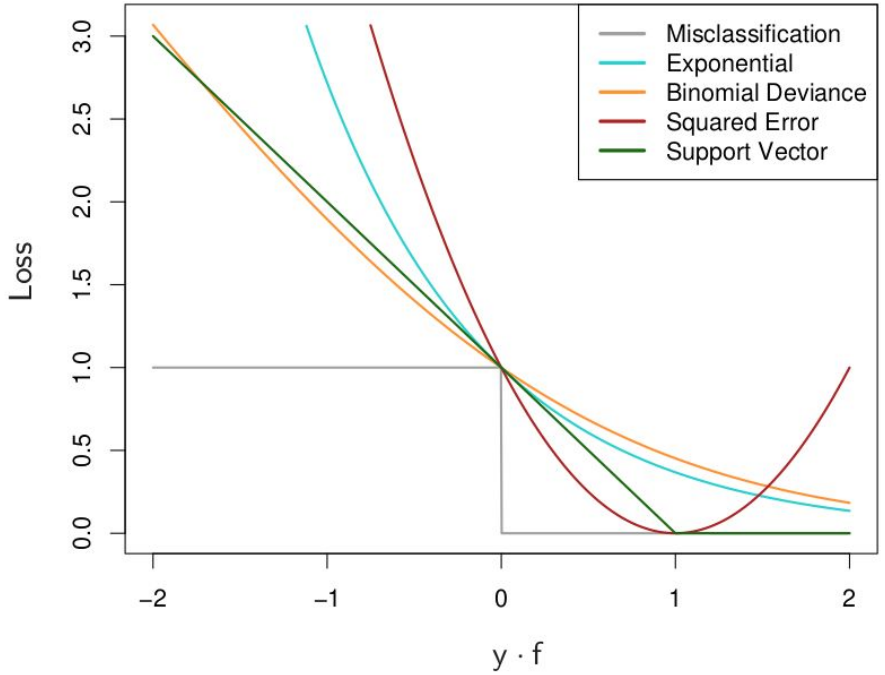
Fact 1: It can be proven that AdaBoost is minimizing the exponential loss criterion via a Forward Stagewise additive modeling approach.

Fact 2: Population minimizer for exponential loss is:

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

Fact 3: Population minimizer for binominal cross-entropy is the same.

Loss functions and robustness



y - classes: {-1, 1}
f - prediction, class prediction is sign(f)
Conclusions:

- Exponential and squared error loss are sensitive to outliers
- Squared error is bad for classification: it penalizes correctly classified examples

Boosting trees

Motivation

- Decision trees have some useful characteristics: handling missing data and mixed data types, ability to deal with irrelevant inputs but have low predictive power
- AdaBoost increases accuracy, but is implementing exponential loss which is not ideal
- We would like to be able to boost decision trees using different loss functions

Decision trees

Decision tree can be expressed as:

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$$

where R_j - region corresponding to leaf j and γ_j - label assigned to leaf j.

While creating such tree, we want to find:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

Given the regions, calculating their optimal labels is easy, but finding these regions is much more difficult (we use greedy top-down strategy).

Boosted trees

Boosted trees are a sum of decision trees:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

We induce them in forward stagewise manner:

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

Finding labels is easy:

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

Finding regions is more difficult than for single tree. **We can do it for exponential and squared-error loss, but we can't do it for more robust loss functions (but we want!).**

How does gradient descent help?

We want to minimize loss with respect to predict function.

Imagine that that function is a simple mapping between inputs and outputs (we lose the constraint that the predictor must be a tree/forest).

$$\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T$$

For a forest:

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m\,, \quad \mathbf{h}_m \in \mathbb{R}^N$$

Solving is easy using gradient descent:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

where ρ_m - step length, $\mathbf{g}_m$ is gradient of loss with respect to $\mathbf{f}_m$

Gradient boosting

We need to deal with the constraint that predictor must be a tree, not a simple mapping.

Hack:

1. Calculate the gradient of loss (it can be any differentiable loss) as in previous slide
2. Induct a tree minimizing square loss between the gradient and the tree's predictions (we can induct a tree for square loss):

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \left(-g_{im} - T(x_i; \Theta)\right)^2$$

This allows us to create boosted forests using any loss function!

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Regularization techniques: sizes of trees

Small trees are desired - they are less prone to overfitting.

We can prune them one by one (poor results) or limit size of all the trees to a constant J (number of leaves).

How to chose J:

- J-1 limits the level of interactions between input features
- too big value leads to overfitting
- sensible values are normally between 4 and 8

Regularization techniques: other

- limiting number of boosting rounds (size of forest): early stopping
- shrinkage (learning rate):

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$$

- subsampling: training each tree on randomly selected η fraction of training data

Often using all of that techniques at once leads to good results, but introduce new hyperparameters to optimize.

Interpretation: relative importance of predictor variables

To compare importance of predictor variables, we can calculate how much of an improvement in loss splits based on them gives us:

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 \, I(v(t) = \ell)$$

For forests we can average:

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m)$$

Interpretation: partial dependence plots

To analyze how value of one predictor value affects prediction, we can calculate partial dependence on that value (or set of values):

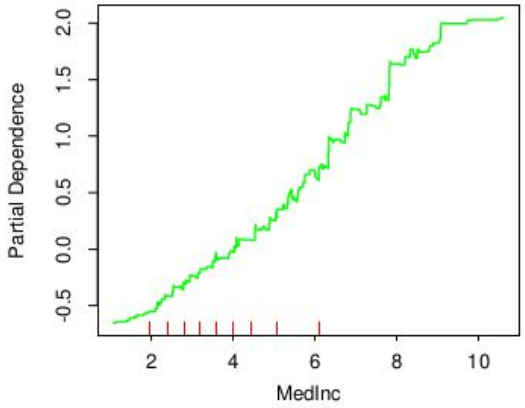
$$f_S(X_S) = \mathrm{E}_{X_C} f(X_S, X_C)$$

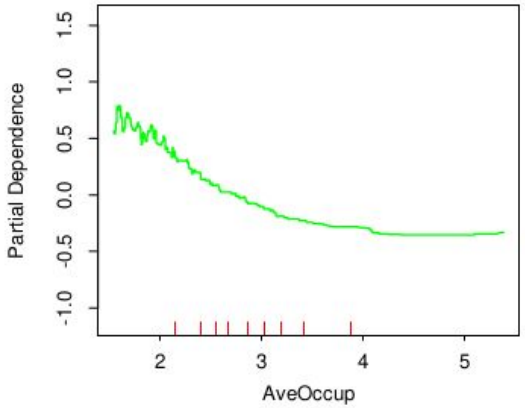
This “averages” the contribution of other values.

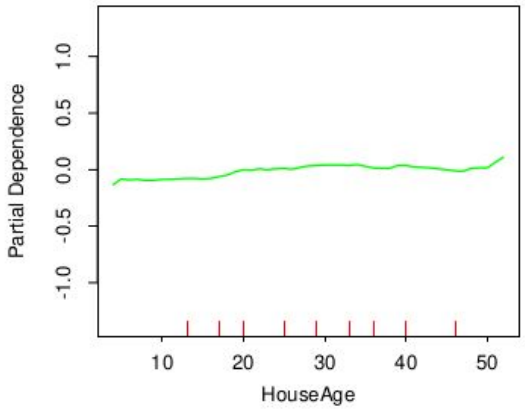
We can also ignore the other values:

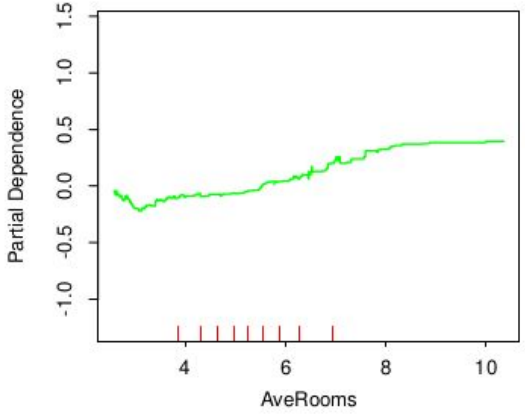
$$\tilde{f}_S(X_S) = \mathrm{E}(f(X_S, X_C)|X_S)$$

These approaches lead to different results, example: $f(a, b) = a^b$


Partial Dependence
-0.5
0.0
0.5
1.0
1.5
2.0
2
4
6
8
10
MedInc


Partial Dependence
-1.0
-0.5
0.0
0.5
1.0
1.5
2
3
4
5
AveOccup


Partial Dependence
-1.0
-0.5
0.0
0.5
1.0
10
20
30
40
50
HouseAge


Partial Dependence
-1.0
-0.5
0.0
0.5
1.0
1.5
4
6
8
10
AveRooms

Thanks for your attention!