Data Mining and Machine Learning, Boosting Trees, Apr 6, 2011
Boosting Trees

General Framework for Regression and Classification Trees

- Trees partition the space of all joint predictor variable values into disjoint region R_j.
- A constant γ_j is assigned to each such region and the predictive rule is

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

- Thus a tree can be formally expressed as

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

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. J is usually treated as meta-parameter. The parameters are found by minimizing the empirical risk

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

This is a formidable combinatorial optimization problem, and we usually settle for approximate suboptimal solutions. It is useful to divide the optimization problem into two parts:

  - Finding γ_j given R_j:
  - Finding R_j: This is the difficult part, for which approximate solution are found. Note also that finding the R_j entails estimating the γ_j as well. A typical strategy is to use a greedy, top-down recursive partitioning algorithm to find the R_j. In addition, it is sometimes necessary to approximate the empirical risk by a smoother and more convenient criterion for optimizing the R_j.

The Boosted Tree Model

- The boosted tree model is a sum of such trees,

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

induced in a forward stage manner.

- At each step in the forward stagewise procedure one must solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + T\left(x_i; \Theta_m\right)\right) \tag{2}$$

  for the region set and constants $\Theta_m = \left\{R_{jm}, \gamma_{jm}\right\}_1^{J_m}$ of the next tree, given current model $f_{m-1}\left(x\right)$.

- Given the regions R_{jm}, finding the optimal constants γ_{jm} in each region is typically straightforward:

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}\left(x_i\right) + \gamma_{jm}\right) \tag{3}$$

- For square loss, the solution is no harder than for a single tree. It is simple the regression tree that best predicts the current residuals $y_i - f_{m-1}\left(x_i\right)$, and $\hat{\gamma}_{jm}$ is the mean of these residuals in each corresponding region.

- For two-classification and exponential loss, this stagewise approach gives rise to the AdaBoost method for boosting classification trees.

Numerical Optimization

- Fast approximate algorithms for solving (2) with any differentiable loss criterion can be derived by analogy to numerical optimization. The loss in using $f\left(x\right)$ to predict y on the training data is

$$L\left(f\right) = \sum_{i=1}^{N} L\left(y_i, f\left(x_i\right)\right).$$

- The goal is to minimize $L\left(f\right)$ with respect to f, where here $f\left(x\right)$ is constrained to be a sum of trees (1). Ignoring this constraint, minimizing $L\left(f\right)$ can be viewed as a numerical optimization

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L\left(\mathbf{f}\right)$$

  where $\mathbf{f} \in \mathbf{R}^N$ and $\mathbf{f} = \left\{f\left(x_1\right), f\left(x_2\right), ..., f\left(x_N\right)\right\}$.

- Numerical optimization procedures solve $\mathbf{f}$ as a sum of component vectors

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

  where $\mathbf{f}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the current parameter vector $\mathbf{f}_{m-1}$, which is the sum of the previously induced vectors.

- Steepest Decent

- Steepest descent chooses $\mathbf{f}_m = -\rho_m \mathbf{g}_m$ where ρ_m is a scalar and $\mathbf{g}_m \in \mathbf{R}^N$ is the gradient of $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$. The components of the gradient $\mathbf{g}_m$ are

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

- The step length ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m)$$

- The current solution is then updated

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

and the process repeated at the next iteration.

- Gradient Boosting

- $T(x_i; \Theta_m)$ are analogous to the components of the negative gradient $-g_{im}$. But the tree components $t_m = (T(x_1; \Theta_m), T(x_2; \Theta_m), ..., T(x_N; \Theta_m))$ are not independent.
- Induce a tree $T(x; \Theta_m)$ at the mth iteration whose predictions $\mathbf{t}_m$ are as close as possible to the negative gradient. Using square error to measure closeness, this leads us to

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

That is, one fits the tree T to the negative gradient values by least squares.

- After constructing the tree, the corresponding constants in each region are given by (3)
- This approach is referred to as "multiple additive regression trees" or MART.

- The MART Algorithm (Gradient tree boosting for multiple additive regression trees)

1. Initialize $f_0(x) = \arg\min_r \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M :
   (a) For $i = 1, 2, ..., N$ 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 targets r_{im} giving terminal regions $R_{jm},\ j = 1, 2, ..., J_m$.
   (c) For $j = 1, 2, ..., J_m$ compute

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

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

3. Output $\hat{f}(x) = f_M(x)$.

Right-Sized Trees for Boosting

- The usual tree building algorithm assumes each tree in boosting procedure the last one in the expansion. The result is that trees tend to be much too large. This substantially degrades performance and increases computation.
- The simplest strategy for avoiding this problem is to restrict all trees to be the same size $J_m = J \; \forall m$. Thus J becomes a meta-parameter of the entire boosting procedure, to be adjusted to maximize estimated performance for the data at hand.
- The interaction level of tree-based approximations is limited by the tree size J. Namely, no interaction effects of level greater than $J-1$. Since boosted models are additive in the trees, this limit extends to them as well.
- For many problems in practice, lower-order interaction effects tend to dominate. When this is the case, models that produce strong higher-order interaction effects suffer in accuracy.
- Although in many applications $J = 2$ will be insufficient, it is unlikely that $J > 10$ will be required. Experience so far indicates that $4 \leq J \leq 8$ works well in the context of boosting. (Fig 10.9 and 10.10).

Regularization

- Besides the size of the constituent trees, J, the other meta-parameter of MART procedure is the number of boosting iterations M. Each iteration usually reduces the training risk $L(f_M)$, so that for M large enough this risk can be made arbitrarily small.
- However, fitting the training data too well can lead to overfitting, which degrades the risk on future predictions. Thus there is an optimal number M^* minimizing future risk that is application dependent. A convenient way to estimate M^* is to monitor prediction risk as a function of M on a validation sample.
- Shrinkage: The simplest implementation of shrinkage in the context of boosting is to scale the contribution of each tree by a factor $0 < v < 1$ when it is added to the current approximation.

$$f_m(x) = f_{m-1}(x) + v \sum_{j=1}^{J} \hat{\gamma}_{jm} I(x \in R_{jm}).$$

The parameter v can be regarded as controlling the learning rate of the boosting procedure. Smaller values of v lead to larger values of M. Empirically it has been found that smaller values of v favor better test error, and require correspondingly larger values of M. (Fig 10.11)

- Penalized Regression: Intuition for the success of the shrinkage strategy can be obtained by drawing analogies with penalized linear regression with a large basis expansion.
  - Let $\{T_k\}$ be the set of J-terminal regression trees. The linear model is

$$f(x) = \sum_{k=1}^{K} \alpha_k T_k(x)$$

  - The penalized least squares is required

$$\hat{\alpha}(\lambda) = \arg\min_{\alpha} \left\{ \sum_{i=1}^{N} \left(y_i - \sum_k \alpha_k T_k(x_i) \right)^2 + \lambda J(\alpha) \right\}$$

    where $J(\alpha)$ is a function of the coefficients that generally penalizes large values.
  - Owing to the very large number of basis functions T_k, directly solving with the penalty is not possible. However, a feasible forward stagewise strategy(Alg 10.4, p329) exists that closely approximates the effect of the lasso, and very similar to boosting and forward stagewise algorithm. (Fig 10.12)

Interpretation
Relative Importance of Predictor Variables

- For a single decision tree T, Breiman et al. (1984) proposed

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

  as a measure of relevance for each predictor variable X_l. The sum is over the $J-1$ internal nodes of the tree. At each such node t, one of the input variables $X_{v(t)}$ is used to partition the region associated with that node into two subregions. The particular variable chosen is the one that gives maximal estimated improvement $\hat{\imath}_t^2$ in square error risk over that for a constant fit over the entire region.

- For additive tree expansions (1) it is simply average over the trees

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

  $\mathcal{I}_l^2$ refer to square relevance; the actual relevances are their respective square roots. Since these measures are relative, it is customary to assign the largest 100 and then scale the others accordingly. (Fig 10.6)

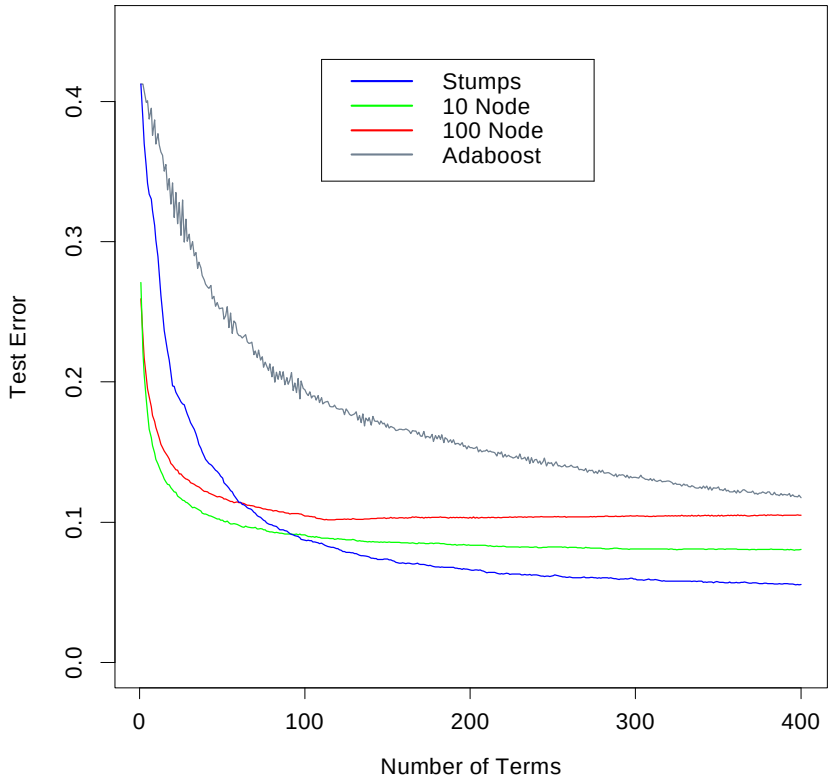

Figure 10.9: *Boosting with different sized trees, applied to the example (10.2) used in Figure 10.2. Since the generative model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost algorithm 10.1.*