

















































In this unit we will develop the gradient boosted regression tree algorithm, widely acknowledged as one of the most robust and accurate "off-the-shelf" machine learning approaches in industrial applications and online competitions (e.g. www.kaggle.com)

1 Motivation

10.7 "Off-the-Shelf" Procedures for Data Mining 351

TABLE 10.1. Some characteristics of different learning methods. Key:  = good,  = fair, and  = poor.

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type					
Handling of missing values					
Robustness to outliers in input space					
Insensitive to monotone transformations of inputs					
Computational scalability (large N)					
Ability to deal with irrelevant inputs					
Ability to extract linear combinations of features					
Interpretability					
Predictive power					

The figure above (taken from [2]) considers the strengths and weaknesses of several algorithmic approaches to "off-the-shelf" data-mining applications that are prevalent in industry.

2 Supervised Learning and Regression

In the supervised learning paradigm we have labeled data and seek to fit a function to the data such that the function is able to predict the labels of unseen examples within a reasonable error.

More formally, we are given a dataset of N tuples:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$$

where each $\mathbf{x}_i \in \mathbb{R}^d$ is called the *feature vector* representing the particular example indexed by i . $y_i \in \mathbb{R}$ is called the *label* and is continuous in regression problems.

To make this formalization concrete, let us consider a dataset representing housing prices. For each house, we aim to predict the house's recent sale price, which will be y_i in this scenario. We also know a number of attributes such as the area of the lot, the neighborhood it is located in, the number of bathrooms, etc... Each such piece of information can be coded as one or more real numbers in order to construct the feature vector \mathbf{x}_i , quantitatively representing

the house. The discipline of constructing the feature representation which gives the most utility for our supervised learning problem is sometimes known as *feature engineering*. Note that y_i depends on our objective, in this case predicting sales price, if we were trying to predict lot size, then sales price could be included in the feature representation \mathbf{x}_i .

In general, parametric regression proceeds by defining a family of functions which take in a feature vector \mathbf{x}_i and output a predicted label \hat{y}_i , where each member of the (possibly infinite) family can be identified by a setting of the function parameters, collectively denoted θ . Further, we make exact the sense in which we wish our choice of function (equivalently parameters) to “explain” the mapping between the features and the labels. This is typically done by describing a loss-function which maps a particular choice of function to a number indicating the loss (or cost) incurred by this choice. With such a function in hand we can compare between two (or more) choices of function and prefer the choice with smaller loss.

Again to make these ideas concrete lets consider the particular example of linear regression. In this model we consider all functions of the form:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

here our parameters are simply the vector of weights $w \in \mathbb{R}^d$, where d denotes the dimension of \mathbf{x} (“number of features”) clearly there are infinite choices for the vector \mathbf{w} and thus the family of linear functions is infinite.

To choose the “best” function/parameters we have to make exact the notion of “best”. This is done using the loss function. A (very) typical loss function used in regression settings is the “sum of squared errors” defined as:

$$L(\mathbf{w}) = \sum_i (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 = \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

Note, that this function has an implicit dependence on the training dataset \mathcal{D} , but as we usually compare functions given a fixed training set this variable is less important and thus its impact is hidden in the above notation. Formalizing the loss function allows us to compare between different settings of our parameters quantitatively. The loss function also sometimes implies an algorithm for choosing the best setting of the parameters.

3 CART - Classification And Regression Trees

The CART algorithm and model is model to fit the regression function above using a tree model, which is another name for a piecewise constant function over variable size regions of the feature space:

$$f_{\theta}(\mathbf{x}) = \sum_{m=1}^M c_m \mathbb{I}[\mathbf{x} \in \mathcal{R}_m]$$

where each of $\mathcal{R}_1, \dots, \mathcal{R}_M$ defines a disjoint region in the feature space \mathcal{X} , such that $\bigcup_{m=1}^M \mathcal{R}_m = \mathcal{X}$. c_m denotes the constant associated with each region, respectively. Thus our model parameters collectively are $\theta = \{\{c_m\}_{m=1}^M, \{\mathcal{R}_m\}_{m=1}^M\}$.

Further, the CART model assumes a sum of squared error loss as above:

$$L(\theta) = \sum_i (y_i - f_{\theta}(\mathbf{x}_i))^2$$

Given a fixed set of regions $\mathcal{R}_1, \dots, \mathcal{R}_M$, the loss function above is straightforward to maximize analytically with respect to c_m :

$$\hat{c}_m = \arg \max_{c_m} L(\theta) = \frac{1}{\sum_i \mathbb{I}[\mathbf{x}_i \in \mathcal{R}_m]} \sum_i y_i \cdot \mathbb{I}[\mathbf{x}_i \in \mathcal{R}_m]$$

That is, each c_m is set to the average label of the examples that happen to fall in the corresponding region.

However, optimizing the loss with respect to the choice of regions is not as simple. Because general partitions of the feature space are difficult to describe mathematically, we will restrict ourselves to recursive binary partitions of the input space. This still enables us to consider a large class of regression functions. For this reason these models are referred to as trees, these recursive binary decisions can be visualized in a tree structure.

However, even with these restrictions it is still infeasible to find the optimal division into regions. As such the CART algorithm applies a greedy approach. To describe this approach let us define the following notation:

$$\mathcal{R}_{\text{lt}}(j, s) = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid x_{ij} \leq s\}$$

$$\mathcal{R}_{\text{gt}}(j, s) = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid x_{ij} > s\}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is a feature vector in the training dataset \mathcal{D} and $x_{ij} \in \mathbb{R}$ denotes the value of the j -th feature of the i -th example.

Now, let us consider the cost of a particular single split, parameterized by a choice of j and s .

$$L(j, s, c_{\text{lt}}, c_{\text{gt}}) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{lt}}(j, s)} (y_i - c_{\text{lt}})^2 + \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{gt}}(j, s)} (y_i - c_{\text{gt}})^2$$

where c_{lt} and c_{gt} are the constants corresponding to regions $\mathcal{R}_{\text{lt}}(j, s)$ and $\mathcal{R}_{\text{gt}}(j, s)$. The minimum with respect to this parametrization can be written in the nested form:

$$\min L(j, s, c_{\text{lt}}, c_{\text{gt}}) = \min_{j, s} \left[\min_{c_{\text{lt}}} \sum_{\mathbf{x}_i \in \mathcal{R}_{\text{lt}}(j, s)} (y_i - c_{\text{lt}})^2 + \min_{c_{\text{gt}}} \sum_{\mathbf{x}_i \in \mathcal{R}_{\text{gt}}(j, s)} (y_i - c_{\text{gt}})^2 \right]$$

as we have seen before the inner minimizations can be solved analytically given a fixed setting of j and s . The outer minimization is solved exhaustively, by scanning through all possible values of j and s . The optimization on j is bounded as it takes on integer values between 1 and d . The optimization on s , which can take on real values, is also bounded, because for any finite dataset only the candidate values $\{x_{ij} \mid \mathbf{x}_i \in \mathcal{D}\}$ need be considered.

Thus, we can optimize precisely for a given split. After finding the optimal split, the training data is partitioned into two parts according to discovered split, and the process is repeated for each part. This is repeated until a stopping criteria is reached. The stopping criteria may depend on the number of datapoints in a particular node, the decrease in the error, a “maximum depth” hyper-parameter, or all of the above.

3.1 Pruning the Tree

In order to avoid over-complicated trees that are likely to overfit the data, one reasonable strategy is to first grow a tree to a large size with node size or maximum depth stopping criteria (or both). This is followed by a pruning stage which removes unhelpful splits in the tree. Note, that this

is preferable to a greedy strategy when choosing the splits because sometimes an individual split may not give much improvement, but its chaining with another split will yield increased performance.

Letting the component of squared error for a particular leaf node of tree T be denoted:

$$Q_m(T) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_m} (y_i - c_m)^2$$

where c_m is the constant fit to the region by tree T . We can define the complexity criterion of a tree as:

$$C_\alpha(T) = \sum_{m=1}^{|T|} Q_m(T) + \alpha|T|$$

$\alpha \geq 0$ is a hyperparameter governing the penalty given to large trees. With this criteria in hand pruning seeks the subtree T_α that minimizes $C_\alpha(T)$. For a given α this tree can be found by iteratively collapsing the internal node that gives the smallest decrease in total square error. This strategy, called *weakest link pruning* creates a sequence of sub-trees that must contain T_α .

3.2 Pseudocode for CART

Algorithm 1 gives the pseudo-code for CART with a maximum depth and a maximum node size stopping criteria

4 Boosting

The idea of boosting was introduced circa 1997 [1] for classification problems. It is a meta-learning algorithm that uses the combination of a number of “weak-learners” to improve the overall learning. Each of these learners, in turn, optimizes by giving additional weight to examples that are previously misclassified.

Since its inception, Boosting has been extended to work for regression and in general can be formalized in the following way:

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

where $\phi_m : \mathcal{X} \rightarrow \mathbb{R}$ is a basis function with parameters γ_m (These are the “weak learners” in the original boosting terminology), and $\beta_m \in \mathbb{R}$ is are “expansion coefficients”. That is, weights that determine the relative influence of the basis function prediction on the overall prediction.

This form, called an additive model, is not unique to boosting and shows up in many other areas of machine learning (e.g. neural nets) and signal processing (e.g. wavelets) (with a different family of basis functions chosen each time).

An idea that is unique to boosting is the approach to fitting this type of mode, a forward-stage-wise approach. That is, rather than simultaneously find the best value for all model parameters, we first set the value of some parameters, those corresponding to the first basis function, then iteratively proceed, each time considering the parameter settings from previous iterations.

More formally, the parameters for an additive model can be denoted as $\theta = \left\{ \{\beta_m\}_{m=1}^M, \{\gamma_m\}_{m=1}^M \right\}$ and fitting such a model for regression amounts to minimizing the sum of squares loss:

Algorithm 1 CART Algorithm for fitting regression trees

Input: \mathcal{D} - Training data of \mathbf{x}_i, y_i pairs

MaxDepth - the maximum tree depth

MinNodeSize - the minimum number of training examples in a leaf node

Output: setting of $\theta = \{\{c_m\}_{m=1}^M, \{\mathcal{R}_m\}_{m=1}^M\}$. that maximizes $L(\theta)$

//Initialize partition data structure

 $d_0 \leftarrow \{\mathcal{D}\}$ $d_{\text{leaves}} \leftarrow \{\}$

//Initialize decision rule data structure

 $\Omega(\mathcal{D})[\mathbf{x}] \leftarrow 1$ **for** $k = 0, \dots, \text{MaxDepth} - 1$ **do** $d_{k+1} \leftarrow \{\}$ **for** $\mathcal{P} \in d_k$ **do** $j, s \leftarrow \text{GETOPTIMALPARTITION}(\mathcal{P})$ $\mathcal{P}_L \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{P} \mid x_{ij} \leq s\}$ $\mathcal{P}_R \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{P} \mid x_{ij} > s\}$ **if** $|\mathcal{P}_L| \geq \text{MinNodeSize} \ \&\& \ |\mathcal{P}_R| \geq \text{MinNodeSize}$ **then** $d_{k+1} \leftarrow d_{k+1} \cup \mathcal{P}_L \cup \mathcal{P}_R$ $g(\mathbf{x}) = \mathbb{I}[x_j \leq s]$ $\Omega(\mathcal{P}_L)[\mathbf{x}] \leftarrow \Omega(\mathcal{P})[\mathbf{x}] \cdot g(\mathbf{x})$ $\Omega(\mathcal{P}_R)[\mathbf{x}] \leftarrow \Omega(\mathcal{P})[\mathbf{x}] \cdot (1 - g(\mathbf{x}))$ **else** $d_{\text{leaves}} \leftarrow d_{\text{leaves}} \cup \mathcal{P}$ **end if****end for** $M \leftarrow |d_{\text{MaxDepth}}| + |d_{\text{leaves}}|$ $m \leftarrow 1$ **for** $\mathcal{P} \in d_{\text{MaxDepth}} \cup d_{\text{leaves}}$ **do** $c_m \leftarrow \frac{1}{|\mathcal{P}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{P}} y_i$ $\mathcal{R}_m \leftarrow \{\mathbf{x} \in \mathbb{R} \mid \Omega(\mathcal{P})[\mathbf{x}] = 1\}$ $m \leftarrow m + 1$ **end for****end for****procedure** GETOPTIMALPARTITION(\mathcal{P})**for** $j = 1 \dots d$ **do** $\mathcal{S}_j \leftarrow \{x_{ij} \mid \mathbf{x}_i \in \mathcal{P}\}$ $\text{minSoFar} \leftarrow \infty$ **for** $s \in \mathcal{S}_j$ **do** $\mathcal{R}_{\text{lt}}(j, s) \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{P} \mid x_{ij} \leq s\}$ $\mathcal{R}_{\text{gt}}(j, s) \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{P} \mid x_{ij} > s\}$ $c_{\text{lt}} \leftarrow \frac{1}{|\mathcal{R}_{\text{lt}}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{lt}}} y_i$ $c_{\text{gt}} \leftarrow \frac{1}{|\mathcal{R}_{\text{gt}}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{gt}}} y_i$ $\ell(j, s) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{lt}}(j, s)} (y_i - c_{\text{lt}})^2 + \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{\text{gt}}(j, s)} (y_i - c_{\text{gt}})^2$ **if** $\ell(j, s) < \text{minSoFar}$ **then** $\text{minSoFar} \leftarrow \ell(j, s)$ $j^* \leftarrow j$ $s^* \leftarrow s$ **end if****end for****end for****return** j^*, s^* **end procedure**

$$L(\theta) = \sum_i \ell(y_i, f(\mathbf{x}_i)) = \sum_i \ell\left(y_i, \left(\sum_{m=1}^M \beta_m \phi(\mathbf{x}_i; \gamma_m)\right)\right) \quad (1)$$

where $\ell(x, y)$ denotes the per-example loss function.

This minimization with respect to all parameters in θ is intractable computationally in general (in particular for cases where the loss and/or basis functions are not smooth in their parameters).

However, if we adopt a greedy strategy of seeking the optimal settings for a single basis function then we have some idea how to proceed with “Forward Stagewise Modeling”:

```

init  $f_0(\mathbf{x}) \leftarrow 0$ 
for  $m = 1, \dots, M$  do
  (a)  $\{\beta_m, \gamma_m\} = \arg \min_{\{\beta, \gamma\}} \sum_i \ell(y_i, (f_{m-1}(\mathbf{x}_i) + \beta \cdot \phi(\mathbf{x}_i; \gamma)))$ 
  (b)  $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + \beta_m \cdot \phi(\mathbf{x}_i; \gamma_m)$ 
end for

```

the optimization in step (a) depends on the choices of ℓ and the basis function ϕ . Consider the reasonable choice for regression $\ell(x, y) = (x - y)^2$, this choice yields:

$$\begin{aligned} \ell(y_i, (f_{m-1}(\mathbf{x}_i) + \beta \cdot \phi(\mathbf{x}_i; \gamma))) &= (y_i - (f_{m-1}(\mathbf{x}_i) + \beta \cdot \phi(\mathbf{x}_i; \gamma)))^2 \\ &= ((y_i - f_{m-1}(\mathbf{x}_i)) - \beta \cdot \phi(\mathbf{x}_i; \gamma))^2 \\ &= (r_{im} - \beta \cdot \phi(\mathbf{x}_i; \gamma))^2 \end{aligned}$$

where $r_{im} = y_i - f_{m-1}(\mathbf{x}_i)$ is called the *residual* for each example at round m , that is the part of the label that is unexplained by the model. From the above analysis we can see that the above is equivalent to running a separate regression (using square loss) at each iteration, with the residuals r_{im} , serving the “label” role of the y_i .

This is very intuitive. At each round of the stage-wise modeling we try to fit the “unexplained” part of the model with additional parameters.

5 Gradient Boosting

Stagewise modeling actually has a deep connection to another iterative process - Gradient descent. To see this consider the vector:

$$\mathbf{f} = \begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_N) \end{bmatrix}$$

If we consider the loss function in Equation 1 as a function of \mathbf{f} then we can differentiate with respect to f and obtain:

$$\mathbf{g} = \nabla_{\mathbf{f}} L = \begin{bmatrix} \frac{\partial \ell(y_1, f(\mathbf{x}_1))}{\partial f(\mathbf{x}_1)} \\ \frac{\partial \ell(y_2, f(\mathbf{x}_2))}{\partial f(\mathbf{x}_2)} \\ \vdots \\ \frac{\partial \ell(y_N, f(\mathbf{x}_N))}{\partial f(\mathbf{x}_N)} \end{bmatrix}$$

We can then minimize the loss with respect to 'parameter vector' \mathbf{f} using gradient descent as follows:

$$\mathbf{f}_t \leftarrow \mathbf{f}_{t-1} - \eta \nabla_{\mathbf{f}} L(\mathbf{f}_{t-1})$$

starting off at an initial value \mathbf{f}_0 and running the above rule for M iterations we obtain that:

$$\mathbf{f}_M = \sum_{m=0} -\eta \cdot \mathbf{g}_m$$

where $\mathbf{g}_0 = \mathbf{0}$ and $\mathbf{g}_m = \nabla_{\mathbf{f}} L(\mathbf{f}_{m-1})$,

This exposition gives us insight into additive modeling, we see that the additive approximates the numerical optimization of the gradient descent approach by approximating the vector \mathbf{g}_m with a parametric function. Note that, no matter what the choice of loss function \mathbf{g}_m will be a continuous vector in \mathbb{R}^N . Thus, in an additive model, we fit a function mapping \mathbf{x}_i to the i -th entry of this vector g_{mi} using supervised learning. The finite gradient vector at each of the data points is trivial to compute for any choice of loss ℓ , however, we are not only interested in making predictions on the training set. Our ultimate goal is to enable generalization, that is, have a prediction $f_m(\mathbf{x})$ for any value of $\mathbf{x} \in \mathbb{R}^d$. Further, even if the original problem is a classification problem (i.e. loss is log-loss or exponential loss) the gradient vector is always continuous and thus we can fit it using regression.

For the case of the squared-loss $\ell(x, y) = (x - y)^2$, each entry of the gradient vector is given by $\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} = (y_i - f(\mathbf{x}_i))$ and thus the gradient vector is given by:

$$\mathbf{g}_m = \left[\begin{array}{c} \frac{\partial \ell(y_1, f(\mathbf{x}_1))}{\partial f(\mathbf{x}_1)} \\ \frac{\partial \ell(y_2, f(\mathbf{x}_2))}{\partial f(\mathbf{x}_2)} \\ \vdots \\ \frac{\partial \ell(y_N, f(\mathbf{x}_N))}{\partial f(\mathbf{x}_N)} \end{array} \right] \bigg|_{\mathbf{f}_{m-1}} = \left[\begin{array}{c} (y_1 - f_{m-1}(\mathbf{x}_1)) \\ (y_2 - f_{m-1}(\mathbf{x}_2)) \\ \vdots \\ (y_N - f_{m-1}(\mathbf{x}_N)) \end{array} \right] = \left[\begin{array}{c} r_{1m} \\ r_{2m} \\ \vdots \\ r_{Nm} \end{array} \right]$$

thus we see we are fitting the same set of residuals as we saw in the forward stagewise modeling phase.

Although generic gradient descent considers a single step-size η a slightly more sophisticated algorithm called steepest descent uses a different optimal step-size at each iteration, that is once that optimization direction \mathbf{g}_m is set we choose the step-size η_m by solving the following optimization:

$$\eta_m = \arg \min_{\eta} \ell(\mathbf{f}_{m-1} - \eta_m \mathbf{g}_m)$$

This is analogous to choosing the "expansion coefficients" β_m in the additive model presented above. Because, this is a one-dimensional optimization, efficient numerical methods exist to efficiently optimize this quantity (e.g. newton-rhapon).

The pseudo code for a generic gradient boosting algorithm is thus given below in Algorithm 2.

6 Boosting Using Trees

In Algorithm 2 we saw a generic form of gradient boosting for unspecified basis functions, ϕ and per-example loss function ℓ . The choice of these determines how to compute the "gradient vector" \mathbf{g}_m as well as how to carry out the optimization for γ_m and β_m .

Algorithm 2 Generic gradient boosting algorithm

Input: \mathcal{D} - Training data of N \mathbf{x}_i, y_i pairs $\ell(x, y)$ - the per entry loss function $\phi(\mathbf{x}; \gamma)$ - the parametric form of the basis function M - the number of basis functions to fit**Output:** $\theta = \left\{ \{\beta_m\}_{m=1}^M, \{\gamma_m\}_{m=1}^M \right\}$. that (approximately) minimizes $L(\theta)$ (Equation 1) $f_0(\mathbf{x}) \leftarrow 0$ **for** $m = 1, \dots, M$ **do****for** $i = 1, \dots, N$ **do** $g_{im} \leftarrow \left[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f_{m-1}(\mathbf{x}_i)}$ **end for** $\gamma_m \leftarrow \arg \min_{\gamma} \sum_{i=1}^N \ell(g_{im}, \phi(\mathbf{x}_i; \gamma))$ $\beta_m \leftarrow \arg \min_{\beta} \sum_{i=1}^N \ell(y_i, f_{m-1}(\mathbf{x}_i) - \beta \cdot \phi(\mathbf{x}_i; \gamma_m))$ $f_m(\mathbf{x}) \leftarrow f_m(\mathbf{x}) - \beta_m \cdot \phi(\mathbf{x}; \gamma_m)$ **end for**

Choosing the loss to be squared error, $\ell(x, y) = (x - y)^2$ and the basis functions to be regression trees gives the popular Gradient Boosted Regression Trees (GBRT) (also known as MART or GBM) algorithm.

Specifically, our m -th basis function will have the form:

$$\phi(\mathbf{x}; \gamma_m) = \sum_{j=1}^J c_{jm} \mathbb{I}[\mathbf{x} \in \mathcal{R}_{jm}]$$

where $\gamma_m = \left\{ \{c_{jm}\}_{j=1}^J, \{\mathcal{R}_{jm}\}_{j=1}^J \right\}$ denotes the parameters of the m -th basis function.

6.1 Computing the gradient vector

Under the squared loss function the gradient vector elements at each iteration m are simply the (negative) residuals:

$$g_{im} \leftarrow \left[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f_{m-1}(\mathbf{x}_i)} = -(y_i - f_{m-1}(\mathbf{x}_i))$$

6.2 Finding the basis function parameters

Finding a the basis function parameters that fit the residuals is done by solving the minimization:

$$\gamma_m \leftarrow \arg \min_{\gamma} \sum_{i=1}^N \ell(g_{im}, \phi(\mathbf{x}_i; \gamma))$$

In the case of squared loss and tree basis functions, this is solved exactly by a call to the CART algorithm discussed previously. In the case of other losses, we can still call CART and approximate the gradient function using a tree.

6.3 Finding the expansion coefficients

Once we have fit the basis function in iteration m we find the ‘expansion coefficient’ β_m by solving the minimization:

$$\beta_m \leftarrow \arg \min_{\beta} \sum_{i=1}^N \ell(y_i, f_{m-1}(\mathbf{x}_i) - \beta \cdot \phi(\mathbf{x}_i; \gamma_m))$$

In the specific case of squared loss the optimization objective becomes:

$$\sum_{i=1}^N (y_i - f_{m-1}(\mathbf{x}_i) + \beta \cdot \phi(\mathbf{x}_i; \gamma_m))^2$$

taking the derivative of this expression with respect to β we obtain:

$$- \sum_{i=1}^N (y_i - f_{m-1}(\mathbf{x}_i) + \beta \cdot \phi(\mathbf{x}_i; \gamma_m)) \phi(\mathbf{x}_i; \gamma_m)$$

setting to 0 and solving for β allows us to obtain the analytical minimizer

$$\beta_m = \frac{\sum_{i=1}^N (y_i - f_{m-1}(\mathbf{x}_i)) \phi(\mathbf{x}_i; \gamma_m)}{\sum_{i=1}^N \phi(\mathbf{x}_i; \gamma_m)^2}$$

7 An Algorithm For Fitting Boosted Tree Models for Regression

The pseudo-code for the GBRT algorithm is given in algorithm 3.

Algorithm 3 Gradient Boosted Regression Trees (GBRT)

Input:

\mathcal{D} - Training data of N \mathbf{x}_i, y_i pairs

M - the number of basis functions to fit

3: J - the number of leaves in each basis tree

Output: $\theta = \left\{ \{\beta_m\}_{m=1}^M, \left\{ \{c_{jm}\}_{j=1}^J, \{\mathcal{R}_{jm}\}_{j=1}^J \right\}_{m=1}^M \right\}$. that minimizes $L(\theta)$

$f_0(\mathbf{x}) \leftarrow 0$

6: **for** $m = 1, \dots, M$ **do**

for $i = 1, \dots, N$ **do**

$g_{im} \leftarrow -(y_i - f_{m-1}(\mathbf{x}_i))$

9: **end for**

$\{c_{jm}\}_{j=1}^J, \{\mathcal{R}_{jm}\}_{j=1}^J \leftarrow \text{CART} \left(\mathcal{D} = \{\mathbf{x}_i, g_{im}\}_{i=1}^N, \text{MaxDepth} = \log_2(J) \right)$

$\beta_m \leftarrow \frac{\sum_{i=1}^N (-g_{im}) \phi(\mathbf{x}_i; \gamma_m)}{\sum_{i=1}^N \phi(\mathbf{x}_i; \gamma_m)^2}$

12: $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + \beta_m \cdot \phi(\mathbf{x}; \gamma_m)$

end for

8 Additional Considerations

1. Modeling interactions between variables - If we suspect that our label y_i depends on \mathbf{x}_i through interactions of the features (e.g. House Size increases value but only in a good neighborhood), then we require a tree of depth d , where d is the number of variables in the interaction ($d = \log_2(J)$). The real size of interactions is unknown but tends to be low in many real-world datasets. Thus we can often get away with $d = 2$ or even $d = 1$ (decision stumps).
2. Regularization via shrinkage- One way to add regularization to boosted tree models is to add a hyperparameter $\nu \in [0, 1]$ which controls weight decay for trees added in later iterations. The update for each iteration (Line 12 of Algorithm 3) now becomes:

$$f_m(\mathbf{x}) \leftarrow f_m(\mathbf{x}) - \nu \cdot \beta_m \cdot \phi(\mathbf{x}; \gamma_m)$$

This is similar in concept to a decaying learning rate. For the same number of basis functions M a model with a lower ν will achieve larger training error, so the hyper-params M and ν exist at a tradeoff.

3. Sub-sampling - As written each basis tree in the M loop considers all data points. If N is large this could cause significant computational overhead. One approach is to subsample a portion of the data η (Another Hyperparameter) at each iteration and fit the basis function using only this data (similar to a mini-batch). This has the benefit of reducing computational overhead and potentially gives improvement due to averaging (similar to bagging).

9 Feature Importance

One of the advantages of decision/regression trees is that they create explainable (which are also straightforward to visualize). One way to quantify the importance of a particular feature with index k is as follows. Let $t = 1, \dots, J - 1$ be an index over the internal nodes of the tree. The quantity \hat{i}_t^2 measures the improvement split at node t improvement in squared training error over a constant fit, letting $v(t)$ denote the feature chosen at node t , $\mathcal{R}_{bef}, c_{bef}$ denote the set of labeled examples in the relevant region before the split and the best constant (the mean) fit to this region, \mathcal{R}_L, c_L \mathcal{R}_R, c_R are the left and right regions and constants, respectively, after the split:

$$\hat{i}_t^2 = \left(\sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_L} (y_i - c_L)^2 + \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_R} (y_i - c_R)^2 \right) - \left(\sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{bef}} (y_i - c_{bef})^2 \right)$$

if this quantity is large it means that the split at node t gave significant improvement. Since each feature can appear in multiple splits, we quantify the importance of a feature k for tree T as:

$$\mathcal{I}_k^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbb{I}[v(t) == k]$$

where $\mathbb{I}[\cdot]$ denotes the indicator function.

When we do boosting using trees we have to quantify the importance of a particular feature across M trees. One way to achieve this is to take a simple average over the feature importance as follows:

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

References

- [1] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi, editor, *Computational Learning Theory*, 1995.
- [2] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. <https://web.stanford.edu/~hastie/Papers/ESLII.pdf>, 2009.