

Extreme Gradient Boosting for Regression and Classification

--Summary and Understanding from StatQuest

Created by Ronghzeng (Roger) He

	Regression	Classification	
	Solution, Action	Solution, Action	Understanding
<p>Input data $\{(x_i, y_i)\}_{i=1}^n$, and a differentiable Loss Function $L(y_i, F(x_i))$</p> <p>Similarity function (score) for Decision Tree, with λ the parameter for regularization</p> $SS(y_i, F(x_i) + \gamma) = -2 \times \left[L(y_i, F(x_i) + \gamma) + \frac{1}{2} \lambda \gamma^2 \right]$ <p>Pruning function f_{prune}</p> $2\gamma T - SS(y_i, F(x_i) + \gamma)$	<p>$F(x)$ is a weak model, Loss function uses scaled RRS</p> $L = \frac{1}{2} \sum_i [y_i - F(x_i)]^2$	<p>Loss function uses cross entropy</p> $-\sum_i [p_i \ln(\hat{p}_i) + (1 - p_i) \ln(1 - \hat{p}_i)]$ <p>log(odds): $\ln(odds) = -\ln\left(\frac{1-p}{p}\right)$</p> <p>Probability: $p = \frac{1}{1 + e^{-\log(odds)}}$</p>	<p>Using those forms of functions are for later convenience; For classification, $F(x_i)$ is log(odds), log(odds) calculated by different models can add up together, but probabilities cannot.</p>
Step1: the 0 th model is set to be 0.5, by default $F_0(x) = 0.5$	By default, it's 0.5, but of course you can use a value that minimizes the loss function		
Step2: for m=1 to M Where m denotes m th model	Iterate through weak models one by one, until a max number of models is reached or additional models don't improve the fit.		
<p>Step2.1: compute residuals</p> $\gamma_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x_i)=F_{m-1}(x_i)}$	$\gamma_{im} = y_i - F_{m-1}(x_i)$	$\gamma_{im} \approx p_i - \hat{p}_{m-1,i}$, when γ is small. $\hat{p}_{m-1,i}$ is the predicted probability that i th data point is positive, by model $F_{m-1}(x_i)$.	<p>residuals for each data point, indicating where the previous model creates errors.</p>

Step2.2: Fit a decision tree to γ_{im} values, splitting on features based on similarity score (ss) with optimal output value γ at each decision node. Denotes each leaf by R_{jm} , for all leaves in m^{th} tree.	Similarity score for each decision node $= \frac{[\sum(\gamma_{im} \in R_{jm})]^2}{\sum \delta(x_i \in R_{jm}) + \lambda}$ Min number of values in a node to go on splitting, N_{\min} , is 1	Similarity score for each decision node $= \frac{[\sum(\gamma_{im} \in R_{jm})]^2}{\sum_{x_i \in R_{jm}} [\hat{p}_{m-1,i} \times (1 - \hat{p}_{m-1,i})] + \lambda}$ $N_{\min} = \sum_{x_i \in R_{jm}} [\hat{p}_{m-1,i} \times (1 - \hat{p}_{m-1,i})]$	For each split, always find the feature that increases the similarity, or equivalently decreases the (Loss_function + regularization term)
Pruning: after building a full tree, we can start pruning to keep the balance between tree complexity and (loss + regularization)	<ul style="list-style-type: none"> • T denotes a single tree model complexity, e.g. the total number of leaves in a tree. • For each split of a decision node, the similarity increased by ΔSS and model complexity increased by $2\nu\Delta T$, but we want SS to be large and model complexity small, or equivalently f_{prune} to be small, so there is a tradeoff b/w those two. • If $\Delta SS \geq 2\nu\Delta T$ at a node, then Δf_{prune} will be non-increasing, which is favorable. 		
Step 2.3: For j^{th} leaf in m^{th} tree, compute $\gamma_{jm} =$ $\underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} \left[L(y_i, F_{m-1}(x_i) + \gamma) + \frac{1}{2} \lambda \gamma^2 \right]$	Average of pseudo residuals that end up in the same leaf with regularization $\gamma_{jm} = \frac{\sum(\gamma_{im} \in R_{jm})}{\sum \delta(x_i \in R_{jm}) + \lambda}$	$\gamma_{jm} = \frac{\sum(\gamma_{im} \in R_{jm})}{\sum_{x_i \in R_{jm}} [\hat{p}_{m-1,i} \times (1 - \hat{p}_{m-1,i})] + \lambda}$ $\hat{p}_{m-1,i}$ is the predicted prob by model $F_{m-1}(x_i)$.	To approximate the values in the same leaf by a constant, which is a solution to (Loss + regularization); iterate over all leaves.
Step 2.4: Update model $F_m(x) = F_{m-1}(x) + \eta \cdot \gamma_{jm} \cdot I(x \in R_{jm})$, Where η is a scaling factor for each model, similar to step size	Add up all scaled results from a series of weak models.	Add up all scaled log(odds) from a series of weak models. Still need to convert to probability.	This is why it's called an ensemble method.
Step 3: Output $F_M(x)$	For new input x , use model $F_M(x)$ to predict.	Convert the result from the last step to probability by $p = \frac{1}{1 + e^{-\log(odds)}}$	May need to tune η to find the best converged result.