

BOOSTED TREES

Evgeny Burnaev

Skoltech, Moscow, Russia

- 1 GRADIENT BOOSTING
- 2 REGRESSION TREES AND ENSEMBLES
- 3 GRADIENT BOOSTING DECISION TREES

- 1 GRADIENT BOOSTING
- 2 REGRESSION TREES AND ENSEMBLES
- 3 GRADIENT BOOSTING DECISION TREES

FUNCTION ESTIMATION

- **Output (or response):** a random variable y
- **Input (or explanatory):** a set of random variables $\mathbf{x} = (x_1, \dots, x_N)$
- **Goal:** using a training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of known $(\mathbf{x}_i, y_i) \sim D$ values obtain an estimate $\hat{F}(\mathbf{x})$ of the function $F^*(\mathbf{x})$ mapping \mathbf{x} to y
- $F^*(\mathbf{x})$ minimizes the expected value of some specified loss function $L(y, F(\mathbf{x}))$

$$F^* = \arg \inf_F \mathbb{E}_D L(y, F(\mathbf{x}))$$

NUMERICAL OPTIMIZATION IN FUNCTION SPACE I

- We consider a non-parametric approach
- We apply numerical optimization in function space
- We
 - consider $F(\mathbf{x})$ evaluated at each point \mathbf{x} as a parameter and
 - seek to minimize

$$\Phi(F) = \mathbb{E}_{y,\mathbf{x}} L(y, F(\mathbf{x})) = \mathbb{E}_{\mathbf{x}} [\mathbb{E}_y (L(y, F(\mathbf{x}))) | \mathbf{x}]$$

at each individual \mathbf{x} , directly with respect to $F(\mathbf{x})$

- Numerical optimization: we approximate $F^*(\mathbf{x})$ by

$$F_T(\mathbf{x}) = \sum_{t=0}^T h_t(\mathbf{x}),$$

where $h_0(\mathbf{x})$ is an initial guess, and $\{h_t(\mathbf{x})\}_{t=1}^T$ are incremental functions (steps or boosts) defined by the optimization method

NUMERICAL OPTIMIZATION IN FUNCTION SPACE II

- Steepest-descent:

$$h_t(\mathbf{x}) = -\rho_t g_t(\mathbf{x})$$

with

$$g_t(\mathbf{x}) = \left[\frac{\partial \Phi(F(\mathbf{x}))}{\partial F(\mathbf{x})} \right]_{F(\mathbf{x})=F_{t-1}(\mathbf{x})}$$

and

$$F_{t-1}(\mathbf{x}) = \sum_{s=0}^{t-1} h_s(\mathbf{x})$$

- The multiplier ρ_m is given by the line search

$$\rho_t = \arg \min_{\rho} \mathbb{E}_D L(y, F_{t-1}(\mathbf{x}) - \rho g_t(\mathbf{x}))$$

FINITE DATA I

- **Nonparametric** approach breaks down when the joint distribution is estimated by a **finite data sample**
- Strength must be borrowed from nearby data points by imposing **smoothness** on the solution
- Assume a **parameterized** form and do parameterized optimization to minimize the corresponding data based estimate of the expected loss

$$\{\beta_t^*, \mathbf{w}_t^*\}_{t=1}^T = \arg \min_{\{\beta_t, \mathbf{w}_t\}_{t=1}^T} \sum_{i=1}^m L \left(y_i, \sum_{t=1}^T \beta_t h(\mathbf{x}_i; \mathbf{w}_t) \right)$$

FINITE DATA II

- In situation where this is infeasible one can try a **greedy stagewise** approach
- For $t = 1, 2, \dots, T$

$$\{\beta_t, \mathbf{w}_t\} = \arg \min_{\{\beta, \mathbf{w}\}} \sum_{i=1}^m L(y_i, F_{t-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{w})),$$

and then

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \beta_t h(\mathbf{x}; \mathbf{w}_t)$$

FINITE DATA III

- In signal processing this stagewise strategy is called **matching pursuit**
 - $L(y, F)$ is a squared-loss function
 - $\{h(\mathbf{x}; \mathbf{w}_t)\}_{t=1}^T$ are called basis functions (atoms), usually taken from wavelet-like dictionary
- In machine learning this stagewise strategy is called **boosting**
 - $y \in \{-1, +1\}$
 - $L(y, F)$ is either
 - A) an exponential loss criterion e^{-yF} or
 - B) negative binomial loglikelihood $\log(1 + e^{-2yF})$
 - $h(\mathbf{x}; \mathbf{w})$ is called a **weak learner** or **base learner**, and usually is a classification tree

COMMENTS I

- Suppose for a particular loss $L(y, F)$ and base learner $h(\mathbf{x}; \mathbf{w})$ the solution $\{\beta_t, \mathbf{w}_t\}_{t=1}^T$ is difficult to obtain
- Given any approximator $F_{t-1}(\mathbf{x})$, the function $\beta_t h(\mathbf{x}; \mathbf{w}_t)$ can be viewed as the **best greedy step** toward the **data-based estimate** of $F^*(\mathbf{x})$, under the constraint that the step direction $h(\mathbf{x}; \mathbf{w}_t)$ is a member of the parameterized class of functions
- The data-based analogue of the unconstrained negative gradient:

$$-g_t(\mathbf{x}_i) = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{t-1}(\mathbf{x})}$$

gives the best steepest-descent step direction in the m -dimensional data space at $F_{t-1}(\mathbf{x})$

COMMENTS II

- The gradient $-g_t(\mathbf{x})$ is defined only for the data points $\{(y_i, \mathbf{x}_i)\}_{i=1}^m$ and cannot be generalized to other \mathbf{x} -values
- One possibility for generalization is to choose that member of the parameterized class $h(\mathbf{x}; \mathbf{w}_t)$ that produces $\mathbf{h}_t = \{h(\mathbf{x}_i; \mathbf{w}_t)\}_{i=1}^m$ most parallel to $-\mathbf{g}_t = \{-g_t(\mathbf{x}_i)\}_{i=1}^m \in \mathbb{R}^m$
- It can be obtained as the solution

$$\mathbf{w}_t = \arg \min_{\beta, \mathbf{w}} \sum_{i=1}^m (-g_t(\mathbf{x}_i) - \beta h(\mathbf{x}_i; \mathbf{w}))^2$$

COMMENTS III

- This constrained negative gradient is used in place of the unconstrained one in the steepest-descent strategy. Specifically, the line search is performed

$$\rho_t = \arg \min_{\rho} \sum_{i=1}^m L(y_i, F_{t-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{w}_t)),$$

and the approximation is updated

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t h(\mathbf{x}; \mathbf{w}_t)$$

COMMENTS IV

- **Advantage:** replace the difficult function minimization problem (β_t, \mathbf{w}_t) by least-squares function minimization, followed by only a single parameter optimization based on the original criterion
- For any $h(\mathbf{x}; \mathbf{w})$ for which a **feasible least-squares** algorithm exists for solving above formula, one can use this approach to minimize any **differentiable loss** $L(y, F)$ in conjunction with **forward stage-wise additive modeling**

GENERIC ALGORITHM USING STEEPEST DESCENT

Algorithm 1:

1. $F_0(\mathbf{x}) = \arg \min_{\rho} \sum_{i=1}^m L(y_i, \rho)$
2. For $t = 1$ to T do
 - $\tilde{y}_i = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{t-1}(\mathbf{x})}, i \in [1, m]$
 - $\mathbf{w}_t = \arg \min_{\mathbf{w}, \beta} \sum_{i=1}^m [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{w})]^2$
 - $\rho_t = \arg \min_{\rho} \sum_{i=1}^m L(y_i, F_{t-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{w}_t))$
 - $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t h(\mathbf{x}; \mathbf{w}_t)$

LEAST-SQUARES (LS) REGRESSION

Loss Function: $L(y, F) = \frac{1}{2}(y - F)^2$

Algorithm 2:

1. $F_0(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^m y_i$
2. For $t = 1$ to T do
 - $\tilde{y}_i = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{t-1}(\mathbf{x})} = y_i - F_{t-1}(\mathbf{x}_i),$
 $i \in [1, m]$
 - $(\rho_t, \mathbf{w}_t) = \arg \min_{\mathbf{w}, \rho} \sum_{i=1}^m [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{w})]^2$
 - $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t h(\mathbf{x}; \mathbf{w}_t)$

LEAST ABSOLUTE DEVIATION (LAD) REGRESSION I

- Loss Function: $L(y, F) = |y - F|$,

$$\tilde{y}_i = \text{sign}(y_i - F_{t-1})$$

- Consider the special case where each base learner is a J -terminal node regression tree. Each regression tree has the additive form

$$h(\mathbf{x}; \mathbf{w} = \{a_j, R_j\}_{j=1}^J) = \sum_{j=1}^J a_j 1(\mathbf{x} \in R_j)$$

- $\{R_j\}_{j=1}^J$ are disjoint regions that collectively cover the space of all values of the predictor variable \mathbf{x}

LEAST ABSOLUTE DEVIATION (LAD) REGRESSION II

Algorithm 2:

1. $F_0(\mathbf{x}) = \text{median}\{y_i, i \in [1, m]\}$
2. For $t = 1$ to T do
 - $\tilde{y}_i = \text{sign}(y_i - F_{t-1}(\mathbf{x}_i)), i \in [1, m]$
 - $\{R_{j,t}\}_{j=1}^J = J - \text{terminal node tree}(\{\tilde{y}_i, \mathbf{x}_i\}_{i=1}^m)$
 - $\gamma_{j,t} = \text{median}\{y_i - F_{t-1}(\mathbf{x}_i), \mathbf{x}_i \in R_{j,t}\}, j \in [1, J]$
 - $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \sum_{j=1}^J \gamma_{j,t} 1(\mathbf{x} \in R_{j,t})$

LEAST ABSOLUTE DEVIATION (LAD) REGRESSION III

The LAD algorithm is highly robust:

- Trees use only order information on the individual variables x_k
- Pseudo-responses \tilde{y}_i have only two values, $\tilde{y}_i \in \{-1, +1\}$
- Terminal node updates are based on medians

OTHER REGRESSION TECHNIQUES

- M -regression
- Two-class logistic regression and classification
- Multiclass logistic regression and classification

REGULARIZATION I

- Fitting the **training data** too closely can be counterproductive
- Reducing the expected loss on the training data beyond some point causes the **population-based** loss to **stop decreasing** and often to **start increasing**
- **Regularization** methods attempt to prevent **overfitting** by constraining the fitting procedure

REGULARIZATION II

- For additive expansions a natural regularization parameter is the number of components T
- Controlling the value of T regulates the degree to which expected loss on the training data can be minimized
- It has often been found that regularization through **shrinkage** provides superior results

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \nu \cdot \rho_t h(\mathbf{x}; \mathbf{w}_t)$$

REGULARIZATION III

- **Decreasing** the value of ν **increases** the best value for T
- We could tune parameters according to applications

ADVANTAGES

- All TreeBoost procedures are **invariant** under all **strictly monotone** transformations of the individual input variables. For example, using x_k , $\log x_k$, e^{x_k} , x_k^δ
- Eliminate the **sensitivity** to long-tailed distributions and outliers
- Trees tend to be **robust** against the **addition of irrelevant** input variables

COMPARISON WITH SINGLE TREE MODELS

- Disadvantage of **single tree models**:
 - Inaccurate for smaller trees
 - Unstable for larger trees, involve high-order interactions
- Mitigated by boosting:
 - Produce piecewise constant approximations, but the granularity is much finer
 - Enhance stability by using **small trees** and averaging over many of them

SCALABILITY

- After sorting the input variables, the computation of the regression TreeBoost procedures scales **linearly** with the number of observations m , the number of input variables N and the number of iterations T
- The classification algorithm scales **linearly** with the number of classes
- More data become available after modeling is complete, boosting can be **continued on the new data** starting from the previous solution
- Boosting on successive subsets of data can also be used when there is insufficient random access main memory to store the entire data set

- 1 GRADIENT BOOSTING
- 2 REGRESSION TREES AND ENSEMBLES
- 3 GRADIENT BOOSTING DECISION TREES

NOTATIONS

- Objective function

$$R(h) = L(S, \mathbf{w}) + \lambda \Omega(\mathbf{w}),$$

- $L = \sum_{i=1}^m (y_i - \hat{y}_i)^2$ is a training loss, depending on parameters \mathbf{w} of the model $h = h(\mathbf{x}; \mathbf{w})$ and training sample S
- $\Omega(\mathbf{w})$ is a regularization, which measures complexity of the model h
- Loss on training data: $L = \sum_{i=1}^m l(y_i, \hat{y}_i)$
 - Squared loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
 - Logistic loss in case $y_i \in \{0, 1\}$
 $l(y_i, \hat{y}_i) = y_i \log(1 + e^{-\hat{y}_i}) + (1 - y_i) \log(1 + e^{\hat{y}_i})$
- Regularization:
 - L_2 -norm: $\Omega(\mathbf{w}) = \|\mathbf{w}\|^2$
 - L_1 -norm (lasso): $\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$

EXAMPLES

- Ridge Regression: $\sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2$
- Lasso: $\sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_1$
- Two-class logistic regression:

$$\sum_{i=1}^m y_i \log(1 + e^{-\mathbf{w} \cdot \mathbf{x}_i}) + (1 - y_i) \log(1 + e^{\mathbf{w} \cdot \mathbf{x}_i}) + \lambda \|\mathbf{w}\|^2$$

OBJECTIVE AND BIAS-VARIANCE TRADE-OFF

- Objective function

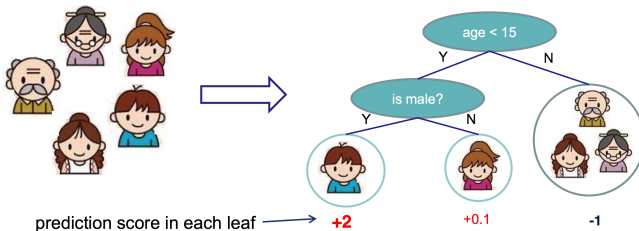
$$R(h) = L(S, \mathbf{w}) + \lambda \Omega(\mathbf{w}),$$

- Optimizing training loss $L(S, \mathbf{w})$ encourages **predictive** models
 - Fitting well on training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization $\Omega(\mathbf{w})$ encourages **simple** models
 - Simpler models tends to have smaller variance in future predictions, making prediction **stable**

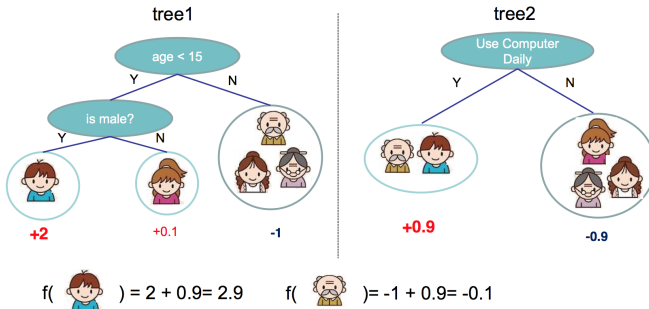
CLASSIFICATION AND REGRESSION TREES (CART)

- Classification and Regression Trees:
 - Decision rules
 - Contains one score in each leaf value

Input: age, gender, occupation,... \Rightarrow Does the person like computer games?



TREE ENSEMBLES



Prediction is a sum of scores predicted by each of the tree

TREE ENSEMBLES I

- Very widely used, look for GBM, random forest...
 - Almost half of data mining competitions are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization
- Learn higher order interaction between features
- Can be scalable, and are used in Industry

TREE ENSEMBLES II

- Model: we have T trees

$$F_T(\mathbf{x}) = \sum_{t=1}^T h_t(\mathbf{x}), \quad h_t(\mathbf{x}) \in \mathcal{F},$$

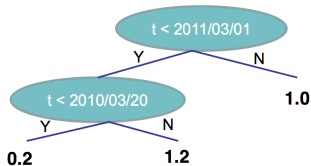
where \mathcal{F} is a space of functions, containing all regression trees

- Parameters: structure of each tree, and the score in the leaf

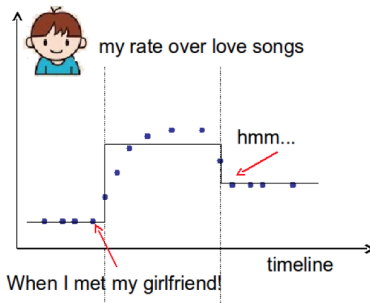
LEARNING A TREE ON A SINGLE VARIABLE

- How can we learn functions?
- Define objective (loss, regularization), and optimize it
- Example:
 - Consider regression tree on single input t (time)
 - We want to predict whether a person like romantic music at time t

The model is a regression tree that splits on time

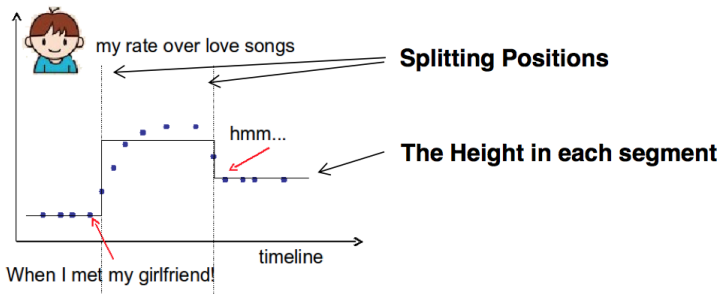


Piecewise step function over time



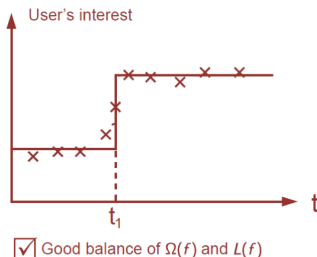
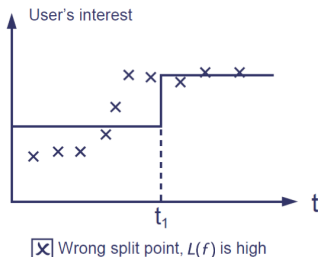
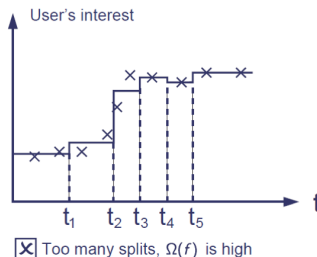
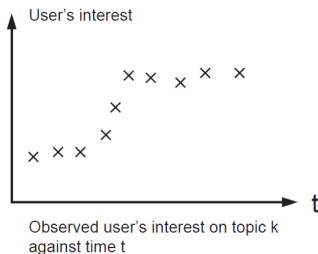
LEARNING A STEP FUNCTION I

- Things we need to learn



- Objective for single variable regression tree (step functions)
 - Training Loss: How will the function fit on the points?
 - Regularization: How do we define complexity of the function?
E.g. number of splitting points, L_2 -norm of the height in each segment, ...

LEARNING A STEP FUNCTION II



OBJECTIVE FOR TREE ENSEMBLES

- Assume that we construct T trees

$$F_T(\mathbf{x}) = \sum_{t=1}^T h_t(\mathbf{x}), \quad h_t \in \mathcal{F}$$

- Objective

$$R(\mathbf{w}) = \sum_{i=1}^m l(y_i, \hat{y}_i) + \sum_{t=1}^T \lambda_t \Omega(h_t)$$

- Possible ways to define Ω
 - Number of nodes in the tree, depth
 - L_2 -norm of the leaf weight
 - ...

THEORY VS. HEURISTICS

- Decision Trees algorithms contain a lot of heuristics:
 - Split by information gain
 - Prune the tree
 - Maximum depth
 - Smooth leaf values
- Most heuristics map well to objectives
 - Information gain \rightarrow training loss
 - Pruning \rightarrow regularization defined by number of nodes
 - Max depth \rightarrow constraint on the function space
 - Smoothing leaf values $\rightarrow L_2$ -norm regularization on leaf weights
- Decision Trees can be used for Classification, Regression, Ranking, ... We just need to define appropriate objective function

- 1 GRADIENT BOOSTING
- 2 REGRESSION TREES AND ENSEMBLES
- 3 GRADIENT BOOSTING DECISION TREES**

LEARNING PROCESS

- Objective: $R(\mathbf{w}) = \sum_{i=1}^m l(y_i, \hat{y}_i) + \sum_{t=1}^T \lambda_t \Omega(h_t)$
- We can not use methods such as SGD, to find F_T (since parameters are trees, instead of just numerical vectors) → **Additive Training (Boosting)**
 - Start from constant prediction, add a new function each time

$$\hat{y}^{(0)} = h_0(\mathbf{x}) = 0$$

$$\hat{y}^{(1)} = h_1(\mathbf{x}) = \hat{y}^{(0)} + h_1(\mathbf{x})$$

$$\hat{y}^{(2)} = h_1(\mathbf{x}) + h_2(\mathbf{x}) = \hat{y}^{(1)} + h_2(\mathbf{x})$$

...

$$\hat{y}^{(t)} = \sum_{s=1}^t h_s(\mathbf{x}) = \hat{y}^{(t-1)} + h_t(\mathbf{x})$$

ADDITIVE TRAINING

- Iterative Objective Optimization
- At round t we need to tune $h_t(\mathbf{x})$

$$\begin{aligned}
 R^{(t)} &= \sum_{i=1}^m l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^t \lambda_i \Omega(h_i) \\
 &= \sum_{i=1}^m l\left(y_i, \hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \lambda_t \Omega(h_t) + \text{const}
 \end{aligned}$$

- Consider squared loss

$$\begin{aligned}
 R^{(t)} &= \sum_{i=1}^m \left(y_i - \left(\hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i) \right) \right)^2 + \lambda_t \Omega(h_t) + \text{const} \\
 &= \sum_{i=1}^m \left[2 \left(\hat{y}_i^{(t-1)} - y_i \right) h_t(\mathbf{x}_i) + h_t^2(\mathbf{x}_i) \right] + \lambda_t \Omega(h_t) + \text{const}
 \end{aligned}$$

TAYLOR EXPANSION APPROXIMATION OF LOSS

- Goal: optimize

$$R^{(t)} = \sum_{i=1}^m l\left(y_i, \hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \lambda_t \Omega(h_t) + \text{const} \rightarrow \text{computationally complicated}$$

- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^m \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \lambda_t \Omega(h_t) + \text{const},$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right), \quad d_i = \partial_{\hat{y}^{(t-1)}}^2 l\left(y_i, \hat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\hat{y}^{(t-1)}} \left(\hat{y}^{(t-1)} - y_i \right)^2 = 2 \left(\hat{y}^{(t-1)} - y_i \right),$$

$$d_i = \partial_{\hat{y}^{(t-1)}}^2 \left(\hat{y}^{(t-1)} - y_i \right)^2 = 2$$

OUR NEW GOAL

• Objective:

$$\sum_{i=1}^m \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \lambda_t \Omega(h_t), \text{ where}$$
$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad d_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

- Approximation of the objective depends on $l(\cdot, \cdot)$ only through g_i and d_i . Thus using it we can
 - Computationally efficiently control growing of trees
 - Easily introduce different loss functions $l(\cdot, \cdot)$ inside boosted trees implementation

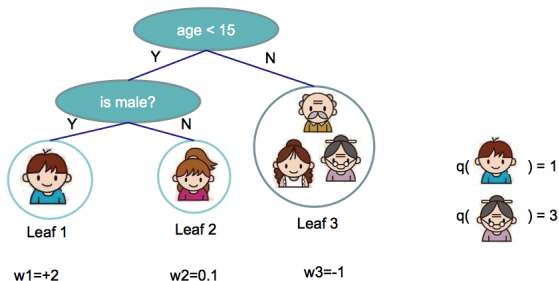
NOTATIONS

- We define a tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$h_t(\mathbf{x}) = w_{q(\mathbf{x})},$$

where

- $\mathbf{w} = (w_1, \dots, w_J) \in \mathbb{R}^J$ are the leaf weights of the tree
- $q : \mathbf{x} \rightarrow \{1, 2, \dots, J\}$ is the structure of the tree



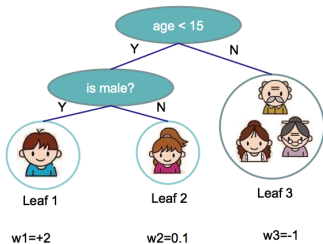
COMPLEXITY OF A TREE

- Define complexity as (this is not the only possible definition)

$$\lambda_t \Omega(h_t) = \gamma J + \frac{1}{2} \lambda \sum_{j=1}^J \omega_j^2,$$

where

- J is a number of leaves in h_t
- L_2 -norm of leaf scores in h_t is used as a regularizer



$$\Omega = \gamma \cdot 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

RE-DEFINE THE OBJECTIVES

- Define the instance set in the leaf j as $I_j = \{i \mid q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$\begin{aligned}
 R^{(t)} &\approx \sum_{i=1}^m \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \lambda_t \Omega(h_t) \\
 &= \sum_{i=1}^m \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^J w_j^2 \\
 &= \sum_{j=1}^J \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J
 \end{aligned}$$

- This is a sum of J independent quadratic functions

EXPLICIT SOLUTION

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^J \left[G_j w_j + \frac{1}{2} (D_j + \lambda) w_j^2 \right] + \gamma J$$






- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

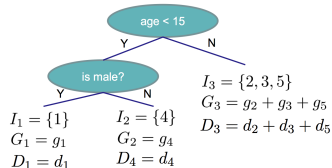
$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \quad R^{(t)} = -\underbrace{\frac{1}{2} \sum_{j=1}^J \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term $(*)$ measure how good a tree structure is

CALCULATION OF THE OBJECTIVE

Instance index gradient statistics

1		g_1, d_1
2		g_2, d_2
3		g_3, d_3
4		g_4, d_4
5		g_5, d_5



$$R^{(t)} = -\frac{1}{2} \sum_j \frac{G_j^2}{D_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

SEARCHING ALGORITHM FOR SINGLE TREE

- Enumerate the possible tree structures by $q(\mathbf{x})$
- Calculate the structure score for the $q(\mathbf{x})$, using the scoring equation

$$R^{(t)} = -\frac{1}{2} \sum_{j=1}^J \frac{G_j^2}{D_j + \lambda} + \gamma J$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{D_j + \lambda}$$

- However, there can be infinite possible tree structures

GREEDY TREE LEARNING

In practice, we grow the tree greedily

- Start from a tree with depth 0
- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

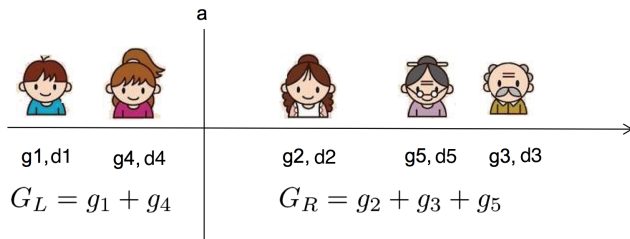
$$\Delta R^{(t)} = \underbrace{\frac{G_L^2}{D_L + \lambda}}_{(1)} + \underbrace{\frac{G_R^2}{D_R + \lambda}}_{(2)} - \underbrace{\frac{(G_L + G_R)^2}{D_L + D_R + \lambda}}_{(3)} - \underbrace{\gamma}_{(4)},$$

where

- (1) and (2) are scores of Left and Right childs,
 - (3) is the score if we do not split
 - (4) is the complexity cost by introducing additional leaf
- How do we find the best split?

EFFICIENT FINDING OF THE BEST SPLIT

- What is the gain of a split rule $x_k < a$? E.g. x_k is an age



- We sum g 's and d 's in each leaf and calculate

$$\Delta R^{(t)} = \frac{G_L^2}{D_L + \lambda} + \frac{G_R^2}{D_R + \lambda} - \frac{(G_L + G_R)^2}{D_L + D_R + \lambda} - \gamma,$$

- Left to right linear scan over sorted instances is enough to decide the best split along the feature x_k

CONSTRUCTION OF SPLITS

- For each node, enumerate over all input features
 - For each feature, sort the instances by feature value
 - Use a linear scan to decide the best split along that feature
 - Take the best split solution along all features
- Time complexity growing a tree of depth M
 - It is $O(mNM \log m)$: on each level we need $O(m \log m)$ time to sort. There are N features, and we need to do it for M levels
 - This can be further optimized (e.g. use approximation or caching the sorted features)
 - The algorithm can be applied to very large datasets

PROCESSING OF CATEGORICAL VARIABLES

- Some tree learning algorithms handle categorical and continuous variables separately
- Using scoring formula we can easily derive a score to split based on categorical variables
- Actually it is not necessary to handle categorical variables separately
 - We can encode categorical variables into numerical vector using one-hot encoding. Allocate a vector with a dimension, equal to the number of categories

$$z_r = \begin{cases} 1, & \text{if the } k\text{-th feature } x_k \text{ value is in category } r \\ 0 & \text{otherwise} \end{cases}$$

- The vector will be sparse if there are lots of categories, the learning is preferred to handle sparse data

PRUNING AND REGULARIZATION

- Recall the gain of split

$$\Delta R = \frac{G_L^2}{D_L + \lambda} + \frac{G_R^2}{D_R + \lambda} - \frac{(G_L + G_R)^2}{D_L + D_R + \lambda} - \gamma,$$

- It can be negative if the training loss reduction is smaller than regularization γ , imposed on the number of leafs
- Trade-off between simplicity and predictiveness
- Pre-stopping
 - Stop split if the best split have negative gain
 - In principle, a split can benefit future splits
- Post-pruning
 - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

BOOSTED TREE ALGORITHM

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad d_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree $h_t(\mathbf{x})$

$$R^{(t)} = -\frac{1}{2} \sum_{j=1}^J \frac{G_j^2}{D_j + \lambda} + \gamma J$$

- Add $h_t(\mathbf{x})$ to the model $\hat{y}^{(t)} = \hat{y}^{(t-1)} + h_t(\mathbf{x})$
 - Usually, instead we calculate $\hat{y}^{(t)} = \hat{y}^{(t-1)} + \nu \cdot h_t(\mathbf{x})$
 - ν is called step-size or shrinkage, usually set around 0.1
 - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting