Notation

$L(y, \hat{y})$ – loss function differentiable by $\hat{y}$.
$X, y = \{x_1, \dots, x_N\}, \{y_1, \dots, y_N\}$ – data.
$\mathcal{H}$ – class of base learners (aka weak learners).
We will suppose that $\mathcal{H}$ is closed under multiplication by constants.
We want to build model as a sum of T base learners:

$$F(x) = \sum_{i=0}^{T-1} f_i(x), \quad f_i \in \mathcal{H}$$

We greedily minimize an empirical loss $L(F) = \sum_{i=1}^{N} L(y_i, F(x_i))$

Notation

$F_t(x) = \sum_{i=1}^{t-1} f_i(x)$ – our model at iteration t. Let:

$$g_i = -\frac{\partial}{\partial \hat{y}} L(y_i, \hat{y})|_{\hat{y}=F_t(x_i)} \quad \text{(antigradients)}$$

g_i is a vector if $\hat{y}$ is a vector.
And hopefully we will succeed to talk about second derivatives:

$$h_i = \frac{\partial^2}{\partial^2 \hat{y}} L(y_i, \hat{y})|_{\hat{y}=F_t(x_i)}$$

We assume h_i to be positive (i.e. that L is convex in $\hat{y}$).

Vanilla gradient boosting

Let's add base learner f to F_t.
Let $g = (g_1, \dots, g_N)$ and $f(X) = (f(x_1), \dots, f(X_N))$
Because f is supposed to be small we can use first order loss approximation:

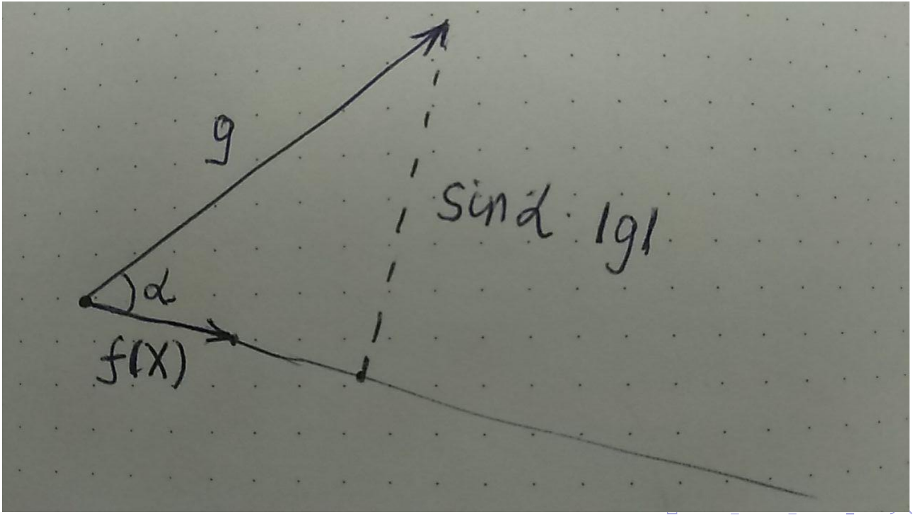
$$L(F_t + f) = \sum_{i=1}^{N} L(y, F_t(x_i) + f(x_i)) = L(F_t) - \langle g, f(X) \rangle + o(\|f(X)\|) =$$

$$L(F_t) - \cos(g, f(X)) \cdot \|g\| \cdot \|f(X)\| + o(\|f(X)\|)$$

So steepest descent achieves f which maximize $\cos(g, f(X))$
Or minimize $\|g - f(X)\|^2 \geqslant |\sin(g, f(X)) \cdot \|g\||^2$

Geometric proof

$$argmax_{f \in \mathcal{H}} \cos(g, f(X)) = argmin_{f \in \mathcal{H}} \|g - f(X)\|^2$$



Pseudocode

see pseudocode here

What to tell:

- Learning rate. It's called shrinkage sometimes (like one other thing).
- Line search (from wiki pseudocode) sucks.
- Gradient boosting is an inferior approximation of simple boosting.
- Greedy trees.
- Other score functions (like 'mae' in sklearn) are strange.

More notation

Tree consists of its structure and leaf values.
Suppose we have some fixed tree structure. Then

- V – space of leaf values.
- $T_{\hat{y}} \approx \mathbb{R}^N$ – space of predictions.
- $A : V \to T_{\hat{y}}$ linear mapping defined by tree structure ($A_{ij} \in \{0, 1\}$ indicates if object i hits leaf j).
- W – matrix of scalar product on $T_{\hat{y}}$ (np.diag(object_weights)).

Suddenly, main optimization formula

Let $g \in \mathbb{R}^n$, A – simmetric, positively definite.

$$\mathrm{argmin}_v \left(-\langle g, v \rangle + \frac{1}{2} v^T A v \right) = A^{-1} g$$

Specifically

$$\mathrm{argmin}_v (Av - b)^T W (Av - b) =$$

$$\mathrm{argmin}_v \left((Av)^T W (Av) - 2b^T W A v + b^T W b \right) =$$

$$\mathrm{argmin}_v \left(-\langle A^T W b, v \rangle + \frac{1}{2} v^T (A^T W A) v \right) = (A^T W A)^{-1} A^T W b$$

Optimal leaf values

Leaf values optimal for tree structure A is given by

$$v^*(A) = \mathrm{argmin}_{v \in V} (Av - g)^T W (Av - g) = (A^T W A)^{-1} A^T W g$$

$A^T WA$ – diagonal matrix with leaf weights
$A^T Wg$ – sums of weighted gradients in leafs
So in the simplest case v^* is a vector of average gradients in leaf.
Let $\mathrm{tree}(A, v)$ be a tree with structure A and values v:

$$v^*(A) \neq \frac{\partial}{\partial v} L(F_t + \mathrm{tree}(A, v))$$

Second order tree scoring

тут очень плохо

let H be a $N \times N$ matrix of weighed second derivatives:

$$H = \begin{pmatrix} w_1 h_1 & 0 & \cdots & 0 \\ 0 & w_2 h_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_N h_N \end{pmatrix}$$

Then second order loss expansion is:

$$L(F_t + f) = \sum_{i=1}^{N} w_i L(y, F_t(x_i) + f(x_i)) =$$

$$L(F_t) - g^T \cdot W \cdot f(X) + \frac{1}{2} f(X)^T \cdot H \cdot f(X) + o(\|f(X)\|^2) =$$

Second order tree scoring

Minimize in with respect to leaf values v, given a structure A:

$$v^*(A) = \mathrm{argmin}_{v \in V} \left[-g^T \cdot W \cdot Av + \frac{1}{2}(Av)^T H(Av) \right] =$$

$$\mathrm{argmin}_{v \in V} \left[-\langle A^T Wg, v \rangle + \frac{1}{2} v^T (A^T HA) v \right] =$$

$$(A^T HA)^{-1} A^T Wg$$

But it's equivalent to calculation of gradient by v with respect to metric H taken from $T_{\hat{y}}$ to V via mapping A.

$A^T HA$ – diagonal matrix with sums of weighted second derivatives in leafs
$A^T Wg$ – sums of weighted gradients in leafs

Pairwise loss

Pairwise loss:

$$\sum_{(w,l)\in pairs} -\log\left[\sigma(F(x_w) - F(x_l))\right]$$

Prediction space is R^N, where N is pair count. mapping A is something like:

$$\begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

columns correspond to tree leafs.
rows correspond to winner-looser pairs.

linear interpretation

- block coordinate descent with respect to metric is taken from prediction space.
- linear model with feature miner.
- leaf value fine tuning.
- But! learning rate schedules don't work.

Bagging

- bootstrap
  - bernoulli
  - poisson (like vanilla bootstrap)
  - Bayesian (reweighting)
  - goss, mvs
- random subspace
- random strength (catboost only)

Additional tricks

- Leaf iterations.
- ctrs
- Ordered boosting?
- h "diagonalization"??

Algorithm complexity

About quantization

- T, F, N – number of trees (aka iterations), features and objects
- Q – number of bins in quantization
- d – tree depth
- p_{obj}, p_{feat} – part of objects/features used in iteration

Time complexity: $O(T \cdot N \cdot F \cdot p_{obj} \cdot p_{feat})$
Memory complexity: $O(N \cdot F + p_{feat} \cdot Q \cdot 2^d)$
About max-ctr-complexity (memory usage for online ctrs and model size)
About leaf iterations slowdown (merely a joke).

Practical recommendations

validation and auto stop (with custom metric).
Model complexity regulation:

- main tool: tree count and learning rate
- tree depth
- oblivious/not-oblivious
- quantization

? rsm / sample rate