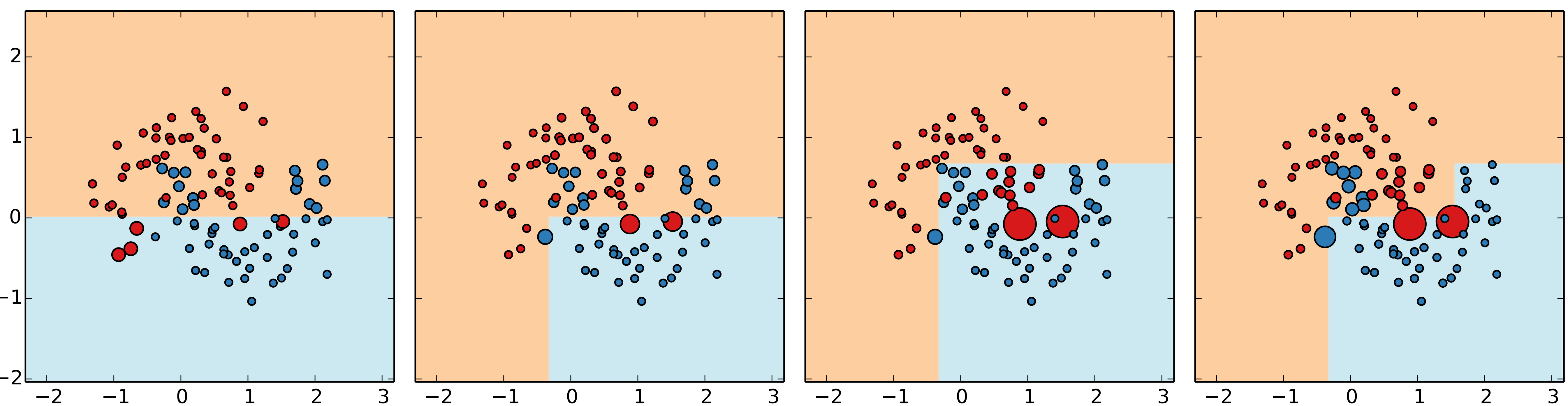
Advanced Data Science part 2

Alexandre Gramfort
Thomas Moreau

Ensemble of experts: Boosting

- Each model is an expert on the errors of its predecessor
- Iteratively re-weights training examples based on errors
- ERM with weights:

$$\arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_i w_i \ell(f(x_i), y_i)$$



AdaBoost [Y. Freund & R. Schapire, 1995]

For binary classification

$\textsc{AdaBoost}(D_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \textsc{Base}(\cdot,\cdot), T)$

$\quad \mathbf{w}^{(1)} \leftarrow (1/n, \ldots, 1/n)$ $\qquad \triangleright$ *initial weights*

$\quad$ **for** $t \leftarrow 1$ **to** T

$\qquad h^{(t)} \leftarrow \textsc{Base}\big(D_n, \mathbf{w}^{(t)}\big)$ $\qquad \triangleright$ *calling the base learner*

$\qquad \gamma^{(t)} \leftarrow \sum_{i=1}^{n} w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i$ $\qquad \triangleright$ $edge = 1 - 2 \times error$

$\qquad \alpha^{(t)} \leftarrow \frac{1}{2} \ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}}\right)$ $\qquad \triangleright$ *coefficient of* $h^{(t)}$

$\qquad$ **for** $i \leftarrow 1$ **to** n $\qquad \triangleright$ *re-weighting the points*

$\qquad\quad$ **if** $h^{(t)}(\mathbf{x}_i) \neq y_i$ **then**

$\qquad\qquad w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{1}{1-\gamma^{(t)}}$

$\qquad\quad$ **else**

$\qquad\qquad w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{1}{1+\gamma^{(t)}}$

$\quad$ **return** $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$

Remark: Freund & Schapire won the Gödel prize 2003

GRADIENT BOOSTING

- Gradient Boosting generalizes adaboost to any arbitrary loss
- a.k.a. GB(R)T, Gradient boosting (regression) trees
- It was originally proposed by [J. Friedman, 1999]
- Variants of the original GBT algorithm are now state-of-the-art models.
- Numerous successes in Kaggle competitions
- State-of-the-art implementations:
  - XGBoost [Chen & Guestrin, Arxiv 2016] (w. Apple, NVidia)
  - LightGBM [Ke et al., Proc. NIPS 2017] (by Microsoft)
  - CatBoost [Prokhorenkova et al. Arxiv 2017] (by Yandex)
  - sklearn.ensemble.HistGradientBoostingClassifier (v0.21)

Tree Boosting in a nutshell

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i),\ f_t \in \mathscr{F}$$ (additive ensemble model)

where $\mathscr{F} = \{f(x) = z^f_{q(x)}\}$ (set of trees with K leafs)

with $q : \mathbb{R}^m \to \{1,\ldots,K\}$ (tree partitioning structure)

and $z^f \in \mathbb{R}^K$ (leaf values)

Each f_t has a different tree structure

Remark: Continuous values even for classifi cation

based on [Chen & Guestrin, Arxiv 2016]

TREE BOOSTING IN A NUTSHELL

(smooth convex loss function)

Objective function:

$$\mathscr{L}(f_1, \ldots, f_T) = \sum_i \ell(y_i, \hat{y}_i) + \sum_t \Omega(f_t)$$

Where: $\Omega(f) = \gamma K + \frac{\lambda}{2} \|z^f\|^2$ (penalize complex trees)

Remark: Original GBT algo. by Friedman had no regularization

Example with MSE / L2 loss:

$$\mathscr{L}(f_1, \ldots, f_T) = \sum_i \|y_i - \hat{y}_i\|^2 + \sum_t \Omega(f_t)$$

Tree Boosting in a nutshell

Model is trained in a sequential / additive manner:

$$\mathscr{L}^{(t)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Recap:

Taylor approximation (order 2): $v(y+\epsilon) \approx v(y) + \epsilon v'(y) + \frac{\epsilon^2}{2} v''(y)$

$$\mathscr{L}^{(t)} \approx \sum_{i=1}^{n} \left[\ell(y_i, \hat{y}_i^{(t-1)}) + f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$$ (gradient)

$$h_i = \partial^2_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$$ (hessian)

Tree Boosting in a nutshell

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[\ell(y_i, \hat{y}_i^{(t-1)}) + f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \Omega(f_t)$$

Example with MSE / L2 loss:

$$\ell(y_i, \hat{y}_i^{(t-1)}) = \|y_i - \hat{y}_i^{(t-1)}\|^2$$

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

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)}) = 2$$

Tree Boosting in a nutshell

$$\mathscr{L}^{(t)} \approx \sum_{i=1}^{n} \left[\ell(y_i, \hat{y}_i^{(t-1)}) + f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \Omega(f_t)$$

Removing constant terms we just need to minimize w.r.t. f_t :

$$\tilde{\mathscr{L}}^{(t)} = \sum_{i=1}^{n} \left[f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \Omega(f_t)$$

Tree Boosting in a nutshell

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \Omega(f_t)$$

which can be rewritten:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[f_t(x_i) g_i + \frac{f_t^2(x_i)}{2} h_i \right] + \gamma K + \frac{\lambda}{2} \sum_{k=1}^{K} (z_k^{f_t})^2$$

$$= \sum_{k=1}^{K} \left[\left(\sum_{i \in I_k} g_i \right) z_k^{f_t} + \frac{1}{2} \left(\lambda + \sum_{i \in I_k} h_i \right) (z_k^{f_t})^2 \right] + \gamma K$$

where: $I_k = \{ i \,|\, q(x_i) = k \}$ (samples in leaf j)

Tree Boosting in a nutshell

Minimizing this:

$$\tilde{\mathscr{L}}^{(t)} = \sum_{k=1}^{K} \left[\left(\sum_{i \in I_k} g_i \right) z_k^{f_t} + \frac{1}{2} \left(\lambda + \sum_{i \in I_k} h_i \right) (z_k^{f_t})^2 \right] + \gamma K$$

leads to: $$z_k^* = - \frac{\sum_{i \in I_k} g_i}{\lambda + \sum_{i \in I_k} h_i}$$

and: $$\tilde{\mathscr{L}}^{(t)} = - \frac{1}{2} \sum_{k=1}^{K} \frac{\left(\sum_{i \in I_k} g_i \right)^2}{\lambda + \sum_{i \in I_k} h_i^2} + \gamma K$$

Tree Boosting in a nutshell

Greedy optimization of:

$$\tilde{\mathscr{L}}^{(t)} = -\frac{1}{2}\sum_{k=1}^{K}\frac{\left(\sum_{i\in I_k} g_i\right)^2}{\lambda + \sum_{i\in I_k} h_i^2} + \gamma K$$

For one leaf splitting it leads to the following criteria that should be <u>maximized</u>:

$$\boxed{\mathscr{C}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\lambda + \sum_{i\in I_L} h_i^2} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\lambda + \sum_{i\in I_R} h_i^2} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\lambda + \sum_{i\in I} h_i^2}\right] - \gamma}$$

It corresponds to the loss reduction by splitting: $I = I_L \cup I_R$

Tree Boosting in a nutshell

Criteria that should be maximized:

$$\mathscr{C}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\lambda + \sum_{i\in I_L} h_i^2} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\lambda + \sum_{i\in I_R} h_i^2} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\lambda + \sum_{i\in I} h_i^2}\right] - \gamma$$

Example with MSE / L2 loss: $\ell_i(y_i, \hat{y}_i) = \|y_i - \hat{y}_i\|^2$

$$\mathscr{C}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} 2(y_i - \hat{y}_i^{(t-1)})\right)^2}{\lambda + \sum_{i\in I_L} 4} + \frac{\left(\sum_{i\in I_R} 2(y_i - \hat{y}_i^{(t-1)})\right)^2}{\lambda + \sum_{i\in I_R} 4} - \frac{\left(\sum_{i\in I} 2(y_i - \hat{y}_i^{(t-1)})\right)^2}{\lambda + \sum_{i\in I} 4}\right] - \gamma$$

Remark: It's close to a variance impurity criterion but not equal

TREE BOOSTING ALGORITHM

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node

Input: d, feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ ***to*** m **do**

  $G_L \leftarrow 0,\ H_L \leftarrow 0$

  for j *in sorted*(I, *by* $\mathbf{x}_{jk}$) **do**

    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$

    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$

    $score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

  end

end

Output: Split with max score

GOING BEYOND:

- Approximate splitting (feature binning)
- Parallel implementation (multi-thread & multi-machine)
- Sparsity aware split finding (think one-hot encoding)
- Cache-aware access
- Monotonic constraints

APPROXIMATE SPLITTING

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**

  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

  Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

Questions?

GBRT Hands on

See: code in `tinygbt.py` folder