Introduction to Boosted Trees

Given $\{(y_i, x_i)\}_i^N$ how can we make new $\hat{y}_i$ predictions? In a linear model, we have $\hat{y}_i = \sum_j w_j x_{ij}$. The parameters are what we learn from the data: $\Theta = \{w_j \mid j = 1, \ldots, d\}$, where $d$ is the number of features.

All objective functions must take the form

$$\text{Obj}(\Theta) = L(\Theta) + \Omega(\Theta)$$

- $L$ is the training loss, measures how well the model fits the data.
- $\Omega$ is regularisation to measure the model's complexity. E.g., $\Omega(\Theta) = \lambda \|w\|^2$.

Optimising $L(\Theta)$ encourages predictive models, while optimising $\Omega(\Theta)$ encourages simple models which may generalise better.

Tree ensembles

- If we have $K$ trees:
  $$y_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

- We can treat the functions as parameters: $\Theta = \{f_1, f_2, \ldots, f_K\}$

- The objective for tree ensembles is:
  $$\text{Obj} = \sum_{i=1}^N L(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- Decision trees typically have heuristics which map to the objective:
  - split by information gain $\rightarrow$ reduce loss
  - prune tree $\rightarrow$ regularisation defined by $n$(nodes)
  - max depth $\rightarrow$ constraint on function space
  - smoothing leaf values $\rightarrow$ L2 reg. on leaf weights

- This objective cannot be optimised with techniques such as SGD, because we are optimising over trees.
Boosting

- In boosting, at each round $t$ we add a new function
  \[
  \hat{y}_i^{(t)} = 0 \\
  \hat{y}_i^{(t)} = f_t(x_i) = \hat{y}_i^{(t−1)} + f_t(x_i) \\
  \hat{y}_i^{(t)} = f_t(x_i) + f_t(x_i) = \hat{y}_i^{(t−1)} + f_t(x_i)
  \]

we need to decide this term at round $t$

\[
\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t−1)} + f_t(x_i)
\]

- The objective at round $t$ is then given by:
  \[
  \text{Obj}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t−1)} + f_t(x_i)) + \Omega(f_t)
  \]

This is only easy to optimize for simple loss functions like square loss.
We can approximate with a second order Taylor expansion:

\[
\text{Obj}^{(t)} \approx \sum_{i=1}^{n} \left[ L(y_i, \hat{y}_i^{(t−1)}) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)
\]

where $g_i = \frac{\partial L(y_i, \hat{y}_i^{(t−1)})}{\partial \hat{y}_i^{(t−1)}}$ and $h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{(t−1)})}{\partial (\hat{y}_i^{(t−1)})^2}$

- We can define a tree as a vector of leaf scores, and a leaf index mapping function that maps an instance to a leaf
  \[
  f_t(x) = \omega_q(x), \quad \omega \in \mathbb{R}^T, \quad q: \mathbb{R}^d \rightarrow \{1,2,...,T\}
  \]

- One possible complexity term is
  \[
  \Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2 \quad \text{L2 norm of weights}
  \]

- The instance set of leaf $j$ is given by $T_j = \{ i | q(x_i) = j \}$
- After removing constants, we can regroup the objective by leaf:
  \[
  \text{Obj}^{(t)} \approx \sum_{j=1}^{n} \left[ g_j \omega_q(x) + \frac{1}{2} h_j \omega_q^2(x) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2
  \]

  \[
  = \sum_{j=1}^{T} \left[ (\sum_{i \in T_j} g_i) \omega_j + \frac{1}{2} (\sum_{i \in T_j} h_i + \lambda) \omega_j^2 \right] + \gamma T
  \]
If we write \( G_j = \sum_{i \in \mathcal{I}_j} g_i \) and \( H_j = \sum_{i \in \mathcal{I}_j} h_i \) such that
\[
\text{Obj}^{(t)} = \frac{1}{T} \sum_{j=1}^{T} \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \delta T
\]
then we can easily optimise this sum of independent quadratics in \( w_j \):
\[
\begin{align*}
    w_j^* &= -\frac{G_j}{H_j + \lambda} \\
    \text{Obj}^{(t)} &= -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \delta T
\end{align*}
\]

In practice, we cannot just enumerate all trees and choose the optimum (as defined by our objective). Instead, we take a greedy approach, choosing a split that maximises the gain:
\[
\text{gain} = \frac{1}{2} \left[ \frac{G_{l}^2}{H_{l} + \lambda} + \frac{G_{r}^2}{H_{r} + \lambda} - \frac{(G_{l} + G_{r})^2}{H_{l} + H_{r} + \lambda} \right] - \gamma
\]
where \( H_{l} \) score of left child, \( H_{r} \) score of right child, \( \gamma \) score if no split.

We can do a left-to-right linear scan to decide on the split:

\[
\begin{array}{cccccc}
\text{gain} & 1 & 0 & 0 & 0 & 0 \\
\text{features} & g_1, h_1, g_2, h_2, g_3, h_3 \end{array}
\]

\(< \text{move a across the sorted instance, calculating gain at each point}.\)

The time complexity is \( O(n d K \log n) \) : we need to sort \( n \) examples for \( d \) features and \( K \) levels.

Categorical variables are easily dealt with by one-hot encoding; XGBoost can manage sparse vectors.

Gain can be negative when training loss reduction < \( \gamma \).
- we can consider pre-stopping if the best split has negative gain
- or grow a tree to max depth, then prune splits with negative gain.