Tianqi Chen slides

Introduction to Boosted Trees

Date 30/9/17 No. 1/2

Given $\{(y_i, x_i)\}^N$ 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

must be minimised

$$Obj(\theta) = \underbrace{L(\theta)}_{\text{training loss measures the model's fit}} + \underbrace{\Omega(\theta)}_{\text{regularisation to measure complexity}}$$

training loss measures the model's fit: $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$

regularisation to measure complexity e.g $\Omega(w) = \lambda \|w\|^2$

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

Tree ensembles

- If we have K trees:

$$y_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

space of all trees

- We can treat the functions as parameters: $\theta = \{f_1, f_2, \ldots, f_K\}$
- The objective for tree ensembles is:

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

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

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\vdots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

we need to decide this term at round t

- The objective at round t is then given by:

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

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

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

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

- We can define a tree as a vector of leaf scores, and a leaf index mapping function that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbb{R}^T, \quad q: \mathbb{R}^d \to \{1, 2, \dots, T\}$$

- One possible complexity term is

$$\underbrace{\Omega(f_t) = \gamma T}_{n(\text{leaves})} + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

L2 norm of weights

- The *instance set* of leaf j is given by $I_j = \{ i \mid q(x_i) = j \}$
- After removing constants, we can regroup the objective by leaf:

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$
$$= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$
$$= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

- If we write $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ such that

$$Obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

then we can easily optimise this sum of independent quadratics in w_j:

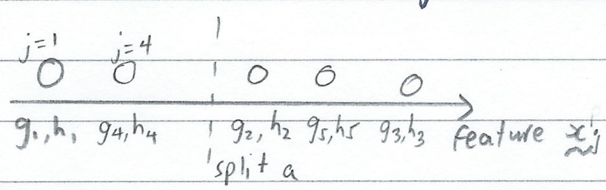
$$w_j^* = -\frac{G_j}{H_j + \lambda} \Rightarrow \boxed{Obj^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T}$$

- 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:

$$gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

score of left child score of right child score if no split

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



← 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 < reg.
  - 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.