Chapter 9
Ensemble Learning

supplementary slides to
Machine Learning Fundamentals
©**Hui Jiang 2020**
published by Cambridge University Press

August 2020

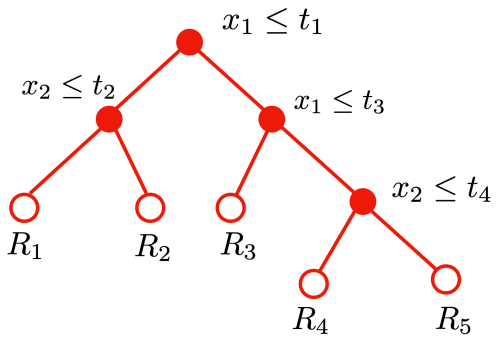
Outline

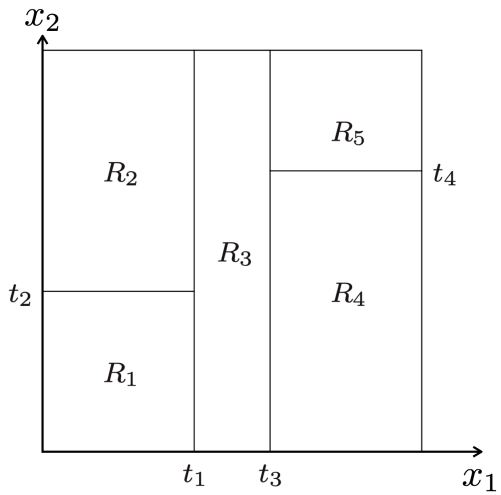
Ensemble Learning

- **ensemble learning**: combine multiple base models that are learned separately for the same task
- how to choose base models?
  - neural networks, linear models, *decision trees*, etc.
- how to learn base models to ensure the diversity?
  - re-sampling the training set, re-weighting training samples, etc.
- how to combine base models optimally?
  - bagging, boosting, stacking

Decision Trees (I)

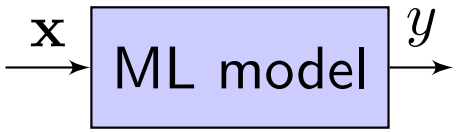
- a popular non-parametric model for regression or classification tasks
- a tree-structured model:
  - each non-terminal node is associated with a binary question regarding an input feature element x_i and a threshold t_j, e.g. $x_i \leq t_j$
  - each leaf node represents a homogeneous region R_l in the input space
- each decision tree represents a particular partition of the input space
- decision trees are a highly interpretable machine learning method





Decision Trees (II)

- fit a simple model to all y values in each region R_l
  - regression: use a constant c_l for each R_l
  - classification: assign all $\mathbf{x}$ in each R_l to one particular class



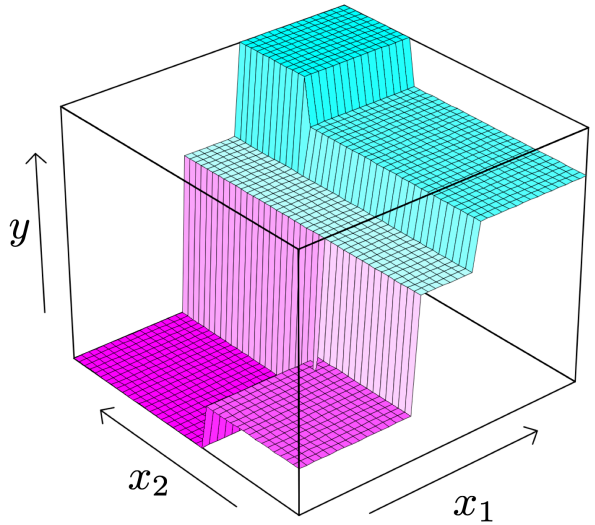
$$y = \bar{f}(\mathbf{x})$$

- approximate the unknown target function by a piece-wise constant function

$$y = f(\mathbf{x}) = \sum_l c_l \, I(\mathbf{x} \in R_l)$$

where

$$I(\mathbf{x} \in R_l) = \begin{cases} 1 & \text{if } \mathbf{x} \in R_l \\ 0 & \text{otherwise} \end{cases}$$



Decision Trees for Regression

- a training set: $\mathcal{D} = \left\{ (\mathbf{x}^{(n)}, y^{(n)}) \,\middle|\, n = 1, 2, \cdots, N \right\}$
- construct the loss functional using a loss function $l(\cdot)$:
  $L(f; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} l\left(y^{(n)}, f(\mathbf{x}^{(n)})\right) = \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - f(\mathbf{x}^{(n)})\right)^2$
- computationally infeasible to find the best partition to minimize the above loss
- use the greedy algorithm to recursively find an optimal split $x_i^* \leq t_j^*$ at a time

$$\left\{x_i^*, t_j^*\right\} = \arg\min_{x_i, t_j} \left[\sum_{\mathbf{x}^{(n)} \in \mathcal{D}_l} \left(y^{(n)} - c_l^*\right)^2 + \sum_{\mathbf{x}^{(n)} \in \mathcal{D}_r} \left(y^{(n)} - c_r^*\right)^2 \right]$$

where
$\mathcal{D}_l = \left\{ (\mathbf{x}^{(n)}, y^{(n)}) \,\middle|\, x_i^{(n)} \leq t_j \right\}$, $\mathcal{D}_r = \left\{ (\mathbf{x}^{(n)}, y^{(n)}) \,\middle|\, x_i^{(n)} > t_j \right\}$,
and c_l^* and c_r^* are the centroids of $\mathcal{D}_l$ and $\mathcal{D}_r$

Decision Trees for Classification

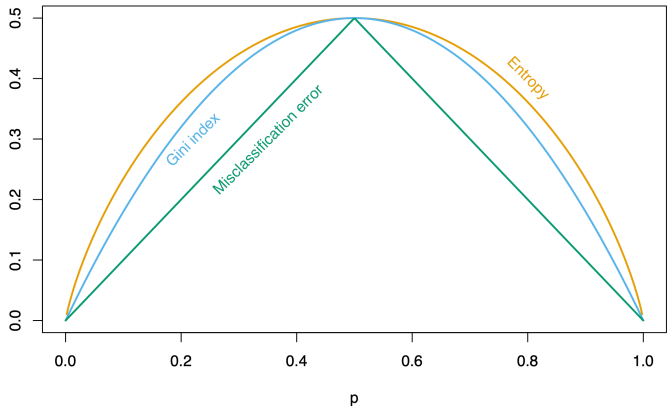
- classification problem involving K classes, i.e. $\{\omega_1, \omega_2, \cdots, \omega_K\}$
- p_{lk} $(k = 1, 2, \cdots, K)$: the portion of class k among all training samples assigned to leaf node l representing R_l

$$p_{lk} = \frac{1}{N_l} \sum_{\mathbf{x}^{(n)} \in R_l} I(y^{(n)} = \omega_k)$$

- all input $\mathbf{x}$ in each region R_l is assigned to the majority class

$$k_l^* = \arg\max_k \; p_{lk}$$

- the criteria for the best split $\{x_i^*, t_j^*\}$



- misclassification error: $\frac{1}{N_l} \sum_{\mathbf{x}^{(n)} \in R_l} I(y^{(n)} \neq \omega_{k_l^*}) = 1 - p_{lk_l^*}$
- Gini index: $1 - \sum_{k=1}^{K} p_{lk}^2$
- entropy: $-\sum_{k=1}^{K} p_{lk} \log(p_{lk})$

Bagging and Random Forests

- **bagging** stands for *bootstrap aggregating*
- bootstrapping (sampling with replacement) a training set into M subsets
- use M bootstrap subsets to independently learn M models
- combine M models by averaging or majority-voting
- **random forests**: use decision trees as base models in bagging
  - row sampling
  - column sampling
  - sub-optimal splitting
- random forests are much more powerful than decision trees

Boosting: Outline

1. Gradient Boosting
2. AdaBoost
3. Gradient Tree Boosting

Boosting

- consider an additive model for ensemble learning

$$F_m(\mathbf{x}) = w_1\, f_1(\mathbf{x}) + w_2\, f_2(\mathbf{x}) + \cdots + w_m\, f_m(\mathbf{x})$$

- each base model $f_m(\mathbf{x}) \in \mathbb{H}$, then $F_m(\mathbf{x}) \in \text{lin}(\mathbb{H}) \supseteq \mathbb{H}$
- ensemble learning: $\Longleftrightarrow$ functional minimization

$$F_m(\mathbf{x}) = \arg \min_{f \in \text{lin}(\mathbb{H})} \sum_{n=1}^{N} l\Big(f(\mathbf{x}_n), y_n\Big)$$

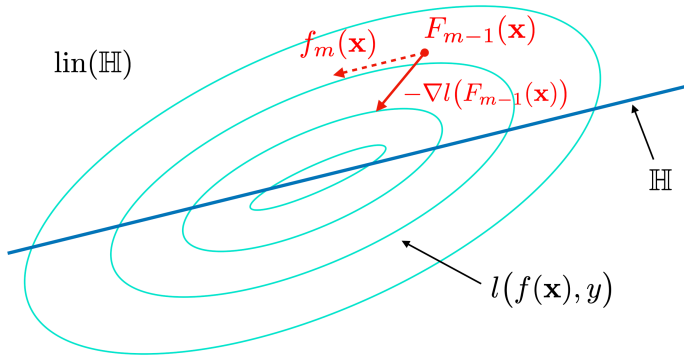
- **boosting**: a sequential learning strategy to add a new base model to improve the current ensemble

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + w_m\, f_m(\mathbf{x})$$

Gradient Boosting

- gradient boosting: estimate the new base model along the direction of the gradient at the current ensemble F_{m-1}:

$$\nabla l\big(F_{m-1}(\mathbf{x})\big) \triangleq \frac{\partial l\big(f(\mathbf{x}), y\big)}{\partial f}\bigg|_{f=F_{m-1}}$$



- project the gradient into $\mathbb{H}$:

$$f_m = \arg\max_{f\in\mathbb{H}} \big\langle f, -\nabla l\big(F_{m-1}(\mathbf{x})\big)\big\rangle$$

$$\langle f, g\rangle \triangleq \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)g(\mathbf{x}_i)$$

- estimate the optimal weight:

$$w_m = \arg\min_w \sum_{n=1}^{N} l\big(F_{m-1}(\mathbf{x}_n) + w\, f_m(\mathbf{x}_n),\, y_n\big)$$

AdaBoost (I)

- apply gradient boosting to binary classification problems
- $\mathbb{H}$: all binary functions, i.e. $\forall f \in \mathbb{H}$, $f(\mathbf{x}) \in \{-1, +1\}$
- the exponential loss function: $l\big(F(\mathbf{x}), y\big) = e^{-yF(\mathbf{x})}$
- given a training set: $\mathcal{D} = \big\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\big\}$, where $\mathbf{x}_n \in \mathbb{R}^d$ and $y_n \in \{-1, +1\}$
- the functional gradient:

$$\nabla l\big(F_{m-1}(\mathbf{x})\big) \triangleq \frac{\partial l\big(f(\mathbf{x}), y\big)}{\partial f}\bigg|_{f=F_{m-1}} = -y\, e^{-yF_{m-1}(\mathbf{x})}$$

- project into $\mathbb{H}$:

$$\begin{aligned} f_m &= \arg\max_{f \in \mathbb{H}} \big\langle f, -\nabla l\big(F_{m-1}(\mathbf{x})\big) \big\rangle \\ &= \arg\max_{f \in \mathbb{H}} \frac{1}{N} \sum_{n=1}^{N} y_n f(\mathbf{x}_n) e^{-y_n F_{m-1}(\mathbf{x}_n)} \end{aligned}$$

AdaBoost (II)

- denote $\alpha_n^{(m)} \triangleq \exp\left(-y_n F_{m-1}\left(\mathbf{x}_n\right)\right)$:

$$\begin{aligned} f_m &= \arg\max_{f\in\mathbb{H}} \left[\sum_{y_n=f(\mathbf{x}_n)} \alpha_n^{(m)} - \sum_{y_n\neq f(\mathbf{x}_n)} \alpha_n^{(m)} \right] \\ &= \arg\max_{f\in\mathbb{H}} \left[\sum_{n=1}^{N} \alpha_n^{(m)} - 2 \sum_{y_n\neq f(\mathbf{x}_n)} \alpha_n^{(m)} \right] \\ &= \arg\min_{f\in\mathbb{H}} \sum_{y_n\neq f(\mathbf{x}_n)} \alpha_n^{(m)} \end{aligned}$$

- normalize all weights as $\bar{\alpha}_n^{(m)} \triangleq \frac{\alpha_n^{(m)}}{\sum_{n=1}^{N} \alpha_n^{(m)}}$, we have

$$f_m = \arg\min_{f\in\mathbb{H}} \sum_{y_n\neq f(\mathbf{x}_n)} \bar{\alpha}_n^{(m)}$$

AdaBoost (III)

- estimate f_m to minimize the weighted classification error:

$$\epsilon_m = \sum_{y_n \neq f_m(\mathbf{x}_n)} \bar{\alpha}_n^{(m)} \qquad (0 \leq \epsilon_m \leq 1)$$

- replace the 0-1 loss function with a weighted loss function, where $\bar{\alpha}_n^{(m)}$ is treated as the loss if $(\mathbf{x}_n, y_n)$ is misclassified
- estimate the optimal weight:

$$w_m = \arg\min_w \sum_{n=1}^{N} e^{-y_n \left(F_{m-1}(\mathbf{x}_n) + w\, f_m(\mathbf{x}_n) \right)}$$

$$\Longrightarrow \quad w_m = \frac{1}{2} \ln \left(\frac{\sum_{y_n = f_m(\mathbf{x}_n)} \bar{\alpha}_n^{(m)}}{\sum_{y_n \neq f_m(\mathbf{x}_n)} \bar{\alpha}_n^{(m)}} \right) = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

AdaBoost (IV)

AdaBoost algorithm

input: $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_n \in \mathbb{R}^d$ and $y_n \in \{-1, +1\}$
output: an ensemble model $F_m(\mathbf{x})$

$m = 1$ and $F_0(\mathbf{x}) = 0$
initialize $\bar{\alpha}_n^{(1)} = \frac{1}{N}$ for all $n = 1, 2, \cdots, N$
while not converged **do**
  learn a binary classifier $f_m(\mathbf{x})$ to minimize $\epsilon_m = \sum_{y_n \neq f_m(\mathbf{x}_n)} \bar{\alpha}_n^{(m)}$
  estimate ensemble weight: $w_m = \frac{1}{2} \ln \left(\frac{1-\epsilon_m}{\epsilon_m} \right)$
  add to ensemble: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + w_m f_m(\mathbf{x})$
  update $\bar{\alpha}_n^{(m+1)} = \frac{\bar{\alpha}_n^{(m)} e^{-y_n w_m f_m(\mathbf{x}_n)}}{\sum_{n=1}^{N} \bar{\alpha}_n^{(m)} e^{-y_n w_m f_m(\mathbf{x}_n)}}$ for all $n = 1, 2, \cdots, N$
  $m = m + 1$
end while

AdaBoost (V)

Theorem

If AdaBoost generates m base models with errors $\epsilon_1, \epsilon_2, \cdots, \epsilon_m$, the error of the ensemble model $F_m(\mathbf{x})$ is bounded as:

$$\varepsilon \leq 2^m \prod_{t=1}^{m} \sqrt{\epsilon_t(1-\epsilon_t)}$$

- combine many weak classifiers towards a strong classifier, i.e. $\varepsilon \to 0$ as $m \to \infty$ if all $\epsilon_t \neq \frac{1}{2}$ (better than random guessing)
- generalize well into unseen samples since it improves the margin distribution of training samples

Gradient Tree Boosting (I)

- apply gradient boosting to regression problems
- $\mathbb{H}$: all decision trees
- use the square error as the loss functional:

$$l\big(f(\mathbf{x}), y\big) = \frac{1}{2}\big(f(\mathbf{x}) - y\big)^2$$

- the functional gradient: $\nabla l\big(F_{m-1}(\mathbf{x})\big) = F_{m-1}(\mathbf{x}) - y$
- given a training set: $\mathcal{D} = \big\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\big\}$
- project it to minimize:

$$\begin{aligned} f_m &= \arg\min_{f \in \mathbb{H}} \big\| f + \nabla l\big(F_{m-1}(\mathbf{x})\big) \big\|^2 \\ &= \arg\min_{f \in \mathbb{H}} \sum_{n=1}^{N} \Big(f(\mathbf{x}_n) - \underbrace{\big(y_n - F_{m-1}(\mathbf{x}_n)\big)}_{\text{residual}} \Big)^2 \end{aligned}$$

Gradient Tree Boosting (II)

- **gradient tree boosting**: build a decision tree f_m to approximate the residuals, i.e. $y_n - F_{m-1}(\mathbf{x}_n)$, for all n

$$y = f_m(\mathbf{x}) = \sum_l c_{ml}\, I(\mathbf{x} \in R_{ml})$$

  where c_{ml} is the mean of all residuals in the region R_{ml}
- a.k.a. gradient boosting machine (GBM), gradient boosted regression tree (GBRT)
- use a pre-set "shrinkage" parameter ν as the weight:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu\, f_m(\mathbf{x})$$

- also applicable to multi-class classification problems

Gradient Tree Boosting (III)

Gradient Tree Boosting

input: $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$
output: an ensemble model $F_m(\mathbf{x})$

fit a regression tree $f_0(\mathbf{x})$ to $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$
$F_0(\mathbf{x}) = \nu\, f_0(\mathbf{x})$
$m = 1$
while not converged **do**
  compute the negative gradients as pseudo outputs:

$$\tilde{y}_n = -\nabla l\big(F_{m-1}(\mathbf{x}_n)\big) \quad \text{for all } n = 1, 2, \cdots, N$$

  fit a regression tree $f_m(\mathbf{x})$ to $\{(\mathbf{x}_1, \tilde{y}_1), \cdots, (\mathbf{x}_N, \tilde{y}_N)\}$
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu f_m(\mathbf{x})$
  $m = m + 1$
end while