Lecture 13: Decision Tree and Ensemble Learning

Shukai Gong

1 Information Theory

Self Information

Self Information measures the amount of information of an r.v.:

$$I(x) = -\log P(x)$$

The higher probability of an event, the less self information it carries. When the base of the logarithm is 2, the unit of self information is bit; when the base is e, the unit is nat.

Entropy

In information theory, entropy is a measure of the uncertainty associated with an r.v. The entropy of an r.v. X is defined as:

$$H(X) = -\sum_{x \in X} P(x) \log P(x) = \mathbb{E}_X[I(x)] = \mathbb{E}_X[-\log P(x)]$$

Specially we define $0 \log 0 = 0$ when $P(x) = 0$.

The larger the entropy, the more information (uncertainty) the r.v. carries. The entropy is maximized when all outcomes are equally likely (**uniform distribution**) and is minimized when the rr.v. is deterministic ($P(x) = 1$).

One major goal of Information Theory is to use minimal bits to encode the information. For a r.v. $X \sim P(X)$, $H(X)$ is the optimal and averaged number of bits needed to encode the information of X.

Joint Entropy

For two r.v.s $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, the joint entropy is defined as:

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log P(x, y) = \mathbb{E}_{(X,Y) \sim P(X,Y)}[-\log P(x, y)]$$

When X and Y are independent,

$$\begin{aligned} H(X, Y) &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x)P(y) \log \left(P(x)P(y)\right) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x)P(y) \log P(x) - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x)P(y) \log P(y) \\ &= -\sum_{x \in \mathcal{X}} P(x) \log P(x) - \sum_{y \in \mathcal{Y}} P(y) \log P(y) = H(X) + H(Y) \end{aligned}$$

Conditional Entropy

Conditional entropy quantifies the uncertaintyof r.v. X given another r.v. Y:

$$H(X|Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log P(x|y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x, y)}{P(y)}$$

It's clear that $H(X|Y) = H(X, Y) - H(Y)$

Mutual Information

Mutual Information measures the amount of information loss of one r.v. when another r.v. is observed:

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)\log\frac{P(x,y)}{P(x)P(y)} = \mathbb{E}_{(X,Y)\sim P(X,Y)}\left[\log\frac{P(x,y)}{P(x)P(y)}\right]$$

Maximizing the amount of information **shared by two r.v.s** is equivalent to maximizing the their mutual information. It's clear that

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$
$$I(X;Y) = I(Y;X)$$

Cross Entropy

Cross entropy measures the average number of bits needed to encode the information of one r.v. $X \sim P$ using the probability distribution of another r.v. $Y \sim Q$:

$$H(P,Q) = -\sum_{x\in\mathcal{X}} P(x)\log Q(x) = \mathbb{E}_{X\sim P}[-\log Q(x)]$$

The closer the two distributions P and Q are, the smaller the cross entropy is. When $P = Q$, the cross entropy is equal to the entropy of P. Note that

$$H(P,Q) = H(P) + \mathrm{KL}(P||Q)$$

KL Divergence / Relative Entropy

Given a true distribution P and its approximate distribution Q, the KL Divergence is defined as

$$\mathrm{KL}(P||Q) = \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)} = \mathbb{E}_{X\sim P}\left[\log\frac{P(x)}{Q(x)}\right]$$

Specially we define $0\log\frac{0}{0} = 0$ and $0\log\frac{0}{q} = 0$.

KL Divergence provides a measure of the difference between two probability distributions P and Q. Note that

$$I(X;Y) = \mathrm{KL}(P(X,Y)||P(X)P(Y))$$

So minimizing KL Divergence works for fitting model, and maximizing KL Divergence ensures that $X \sim P$ and $Y \sim Q$ are heavily dependent.

KL divergence is always non-negative in that

$$-\mathrm{KL}(P||Q) = \int_{x\in\mathcal{X}} P(x)\log\frac{Q(x)}{P(x)}\mathrm{d}x \underbrace{\leq}_{\text{Jensen Inequality}} \log\int_{x\in\mathcal{X}} P(x)\frac{Q(x)}{P(x)}\mathrm{d}x = 0$$

The equality holds if and only if $P = Q$. However, KL Divergence is not symmetric, i.e., $\mathrm{KL}(P||Q) \neq \mathrm{KL}(Q||P)$, and doesn't satisfy the triangle inequality, i.e., $\mathrm{KL}(P||Q) + \mathrm{KL}(Q||R) \geq \mathrm{KL}(P||R)$ doesn't hold, so it's not a true distance metric.

JS Divergence

The Jensen-Shannon Divergence is defined as

$$\mathrm{JS}(P||Q) = \frac{1}{2}\mathrm{KL}\left(P\middle\|\frac{P+Q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(Q\middle\|\frac{P+Q}{2}\right)$$

Wasserstein Distance

Wasserstein Distance is also used to measure the difference between two probability distributions. Given two distributions P and Q, the p-th Wasserstein Distance is defined as

$$W_p(P,Q) = \left(\inf_{\gamma\in\Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}\left[d(x,y)^p\right]\right)^{\frac{1}{p}}$$

where $\Gamma(P,Q)$ is the set of all joint distributions $\gamma(x,y)$ whose marginals are P and Q, and $d(x,y)$ is the distance between x and y.

2 Decision Tree

In general, a decision tree consists of a root node, several internal nodes, and several leaf nodes. Each leaf node corresponds to a class label, and each internal node corresponds to a feature test. The tree is constructed by recursively partitioning the data into subsets based on the results of feature tests, so a path from the root to a leaf represents a decision rule. The goal of decision tree learning is to find a simple and interpretable rule for different features that sufficiently achieves **complicated non-linear classification** and has **good generalization performance**.

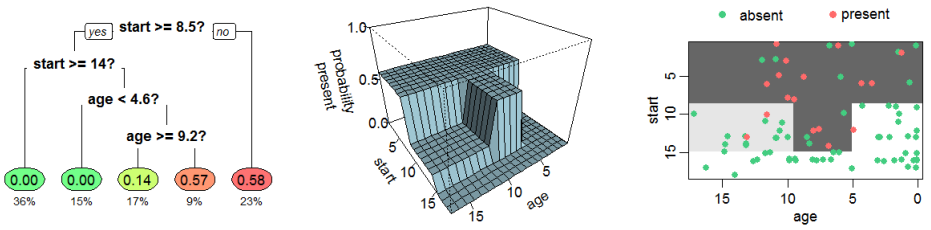


Figure 1: Example: The probability of kyphosis after spinal surgery given the age of the patient and the vertebrae at which the surgery was started.

The construction of a decision tree generally follows a divide-and-conquer strategy: denote

- $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$: The training set. $\boldsymbol{x}_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \mathcal{Y}$ is the label.
- $A = \{a_1, a_2, \ldots, a_d\}$: The Attribute set.

```
def DecisionTreeLearning(D, A):
    generate a node
    if all samples in D have the same label C:
        return a leaf node with the label C
    if A = ∅ or all samples in D have the same feature values on A:
        return a leaf node with the majority label in D
```

```
        else:
            choose the best splitting attribute a* from A
            for each value a*_v of a*:
                add a branch corresponding to a* = a*_v
                D_v = {samples in D with a* = a*_v}
                if D_v is empty:
                    add a leaf node with the majority label in D
                else:

        return the decision tree
```

There are three return conditions in the algorithm:

1. All samples have the same label C, no need for further partition.
2. Current attribute set A is empty or all samples have the same feature values on A, no way for further partition.
3. Current sample set D is empty, no way for further partition.

Clearly, the key of decision tree learning is to `choose the best splitting attribute` a^*. There are several criteria to measure the goodness of a split, but they can be boiled down to one principle: **maximizing the information gain**.

2.1 Iterative Dichotomiser 3 (ID3)

Denote p_k as the proportion of samples in class k in the current sample set D, $k = 1, 2, \ldots, |\mathcal{Y}|$. Suppose discrete attribute a has V values $\{a_1, a_2, \ldots, a_V\}$, and D_v is the subset of samples with $a = a_v$. The information gain of attribute a can be defined as the mutual information between the attribute a and the training set D:

$$\begin{aligned}
\mathrm{Gain}(D, a) = I(D; a) = & \underbrace{H(D)}_{\text{Entropy of parent}} - \underbrace{H(D|a)}_{\text{Weighted sum of children's entropies}} \\
&= H(D) - \sum_{v=1}^{V} P(a_v) H(D_v) \\
&= -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k - \sum_{v=1}^{V} P(a_v) \sum_{k=1}^{|\mathcal{Y}|} -p_k(a_v) \log_2 p_k(a_v) \\
&= -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{k=1}^{|\mathcal{Y}|} -p_k(a_v) \log_2 p_k(a_v)
\end{aligned}$$

and we choose the attribute with the largest information gain as the splitting attribute.

$$a^* = \arg\max_{a \in A} \mathrm{Gain}(D, a)$$

For continous attributes, we can discretize them by **bi-partitioning**. Suppose an attribute a has N values $\{a_1, a_2, \ldots, a_N\}$ on training set D, we can choose the threshold t to split D into $D_t^- = \{(\boldsymbol{x}, y) \in D | a(\boldsymbol{x}) \leq t\}$ and $D_t^+ = \{(\boldsymbol{x}, y) \in D | a(\boldsymbol{x}) > t\}$. Obviously, for arbitrary adjacent attributes a_i and a_{i+1}, any $t \in (a_i, a_{i+1})$ yields the same splitting of D. Specifically, we consider a **candidate set of thresholds:**

$$T_a = \left\{ t = \frac{a_i + a_{i+1}}{2} | i = 1, 2, \ldots, N-1 \right\}$$

and find the best threshold t^* that maximizes the information gain:

$$\text{Gain}(D, a) = \max_{t \in T_a} \text{Gain}(D, a, t) = \max_{t \in T_a} \left(H(D) - \sum_{\theta \in \{+,-\}} \frac{|D_t^\theta|}{|D|} H(D_t^\theta) \right)$$

$$a^* = \arg\max_{a \in A} \text{Gain}(D, a) = \arg\max_{a \in A} \max_{t \in T_a} \text{Gain}(D, a, t)$$

2.2 C4.5

The ID3 algorithm has a problem that it tends to **choose the attribute with more values as the splitting attribute.** To solve this problem, the C4.5 algorithm uses the **information gain ratio** as the criterion to choose the splitting attribute:

$$\text{GainRatio}(D, a) = \frac{\text{Gain}(D, a)}{IV(a)}$$

where $IV(a)$ refers to the intrinsic value of attribute a:

$$IV(a) = \sum_{v=1}^{V} -\frac{|D_v|}{|D|} \log_2 \frac{|D_v|}{|D|}$$

$IV(a)$ grows with the number of values of attribute a, so the information gain ratio can effectively penalize the attribute with more values.

In practice, the C4.5 algorithm uses a **heuristic method** to choose the splitting attribute instead of simply $a^* = \arg\max_{a \in A} \text{GainRatio}(D, a)$. It first chooses the attributes with the over-average information gain ratio, and then chooses the attribute with the largest information gain ratio from them.

2.3 CART (Classification and Regression Trees)

The CART algorithm uses the **Gini index** as the criterion to choose the splitting attribute: the purity of current node D is measured by

$$\text{Gini}(D) = \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2$$

Intuitively, $\text{Gini}(D)$ reflects the probability of two randomly selected samples in D having different labels. The Gini index of attribute a is defined as

$$\text{GiniIndex}(D, a) = \sum_{v=1}^{V} \frac{|D_v|}{|D|} \text{Gini}(D_v)$$

and we choose $a^* = \arg\min_{a \in A} \text{GiniIndex}(D, a)$ as the splitting attribute.

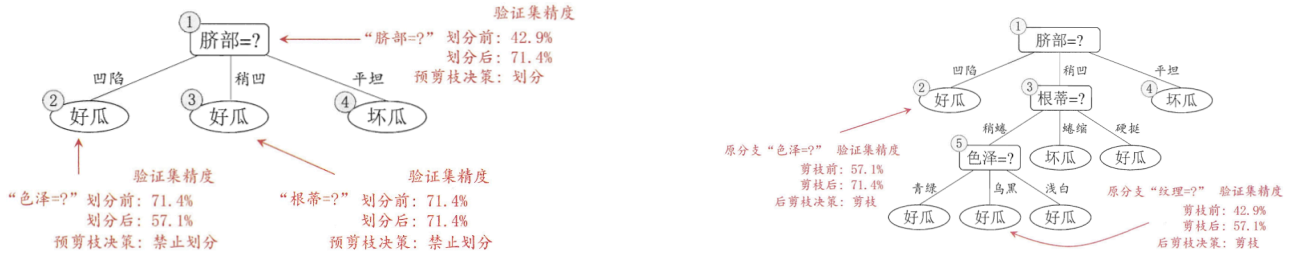
2.4 Pruning

In decision tree learning, the tree may be too complex and overfit the training set. Pruning is a technique to reduce the complexity of the tree and improve the generalization performance. There are two types of pruning:

- **Pre-pruning:** Pruning the tree **during the construction** process. If the splitting of a node doesn't improve the generalization performance (validation set accuracy), the partitioning is stopped and the node is turned into a leaf node.
- **Post-pruning:** Pruning the tree **after the construction** process. The tree is first constructed, and then the nodes are pruned from leaf to root. If replacing a subtree with a leaf node improve the generalization performance (validation set accuracy), the subtree is pruned.

Pros and Cons of pre-pruning and post-pruning:

- Pre-pruning lowers the risk of overfitting and the cost of constructing the tree, but may lead to underfitting.
- Post-pruning generally preserves more branches and has better generalization performance, but is more computationally expensive (Constructing the whole tree and then traversing the tree to prune).



(a) Pre-pruning

(b) Post-pruning

Figure 2: Pruning to avoid overfitting

3 Ensemble Learning

Ensemble learning is a machine learning paradigm where multiple models for the same problem are trained and combined to improve the performance.

3.1 Boosting

Boosting is a family of incremental learning algorithms that iteratively trains a series of weak learners and combines them to form a strong learner.

- A base learner h_t is trained on the training set;
- The distribution of training set $\mathcal{D}_t$ is **re-weighted** to emphasize the samples that are misclassified by the previous base learner;
- A new base learner h_{t+1} is trained on the re-weighted training set;
- The process is repeated T times.

The ensemble model is then constructed as a weighted sum of the base learners: $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$.

3.1.1 AdaBoost

One of the most popular boosting algorithms is AdaBoost (Adaptive Boosting). First we denote

- $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$: The training set. $\boldsymbol{x}_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \{-1, 1\}$ is the label.
- $\mathcal{D}_t = \{w_{t1}, w_{t2}, \ldots, w_{tN}\}$: The distribution of the training set at iteration t, i.e. the weight for each data point.
- $h_t(\boldsymbol{x})$: The base learner at iteration t.
- α_t: The weight of h_t in the ensemble model.

- ϵ_t: The classification error rate of $h_t(\boldsymbol{x})$.
- $H_t(\boldsymbol{x})$: The ensemble model at iteration t.

```
def AdaBoost(D, T):
    Initialize the distribution D_1 = {1/N, ..., 1/N}
    for t=1,2,...,T:
        Train h_t(x) : R^d -> {-1, 1} on D with distribution D_t
        epsilon_t = sum_{i=1}^{N} w_ti I(h_t(x_i) != y_i)
        if epsilon_t > 0.5: break
        alpha_t = (1/2) log((1 - epsilon_t) / epsilon_t)
        Update the distribution: w_{t+1,i} = w_ti exp(-alpha_t y_i h_t(x_i)) / sum_{i=1}^{N} w_ti exp(-alpha_t y_i h_t(x_i))
    return H(x) = sign( sum_{t=1}^{T} alpha_t h_t(x) )
```

There are several noticable facts in the algorithm:

- **Error rate:** Although we allows for weak models in AdaBoost, the classification error rate of each base learner should still be less than 0.5, otherwise the weight α_t for each model would be negative.
- **Weight update:** The denominator $Z_t = \sum_{i=1}^{N} w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$ is a normalization term to ensure that the distribution of weights is a probability distribution. We can further simplify the update of weights as

$$w_{t+1,i} = \frac{w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{\sum_{i=1}^{N} w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))} = \begin{cases} \frac{w_{ti} \exp(-\alpha_t)}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \frac{w_{ti} \exp(\alpha_t)}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases} = \begin{cases} \frac{w_{ti} \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \frac{w_{ti} \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$

  Note that

$$\begin{aligned} Z_t &= \sum_{i=1}^{N} w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i)) = \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} \sum_{y_i = h_t(\boldsymbol{x}_i)} w_{ti} + \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \sum_{y_i \neq h_t(\boldsymbol{x}_i)} w_{ti} \\ &= \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} (1-\epsilon_t) + \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \epsilon_t = 2\sqrt{\epsilon_t (1-\epsilon_t)} \end{aligned}$$

  So $w_{t+1,i} = \begin{cases} \frac{w_{ti}}{2(1-\epsilon_t)} & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \frac{w_{ti}}{2\epsilon_t} & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$
- **Weight assign:** Note that $w_{t+1,i} = \begin{cases} \frac{w_{ti} e^{-\alpha_t}}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \frac{w_{ti} e^{\alpha_t}}{Z_t} & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$, a sample that is misclassified by the base learner h_t will have a **larger weight** in the next iteration.

Now we will introduce the derivation of AdaBoost. The optimzation follows a forward stagewise strategy. Instead of directly optimizing

$$(\{\alpha_t\}^*, \{h_t(\boldsymbol{x})\}^*) = \arg \min_{\{\alpha_t\}, \{h_t(\boldsymbol{x})\}} \sum_{i=1}^{N} \mathcal{L}(y_i, \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}_i))$$

we optimize the following objective function:

$$(\alpha_t^*, h_t^*) = \arg\min_{\alpha_t, h_t} \sum_{i=1}^{N} \mathcal{L}(y_i, H_{t-1}(\boldsymbol{x}_i) + \alpha_t h_t(\boldsymbol{x}_i))$$

and iteratively update the ensemble model $H_t(\boldsymbol{x}) = H_{t-1}(\boldsymbol{x}) + \alpha_t^* h_t^*(\boldsymbol{x})$.

The loss function we adopt is the exponential loss function:

$$\mathcal{L}(y_i, H(\boldsymbol{x}_i)) = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i H(\boldsymbol{x}_i))$$

Why exponential loss function?

The loss function can be written as (population version):

$$\begin{aligned}
\mathcal{L}(y, H(\boldsymbol{x})) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\exp(-yH(\boldsymbol{x}))\right] \\
&= e^{-H(\boldsymbol{x})} P(y=1|\boldsymbol{x}) + e^{H(\boldsymbol{x})} P(y=-1|\boldsymbol{x}) \\
\frac{\partial \mathcal{L}(y, H(\boldsymbol{x}))}{\partial H(\boldsymbol{x})} &= -P(y=1|\boldsymbol{x}) e^{-H(\boldsymbol{x})} + P(y=-1|\boldsymbol{x}) e^{H(\boldsymbol{x})} = 0 \\
\Rightarrow H(\boldsymbol{x}) &= \frac{1}{2} \log \frac{P(y=1|\boldsymbol{x})}{P(y=-1|\boldsymbol{x})}
\end{aligned}$$

therefore

$$\begin{aligned}
\operatorname{sign}(H(\boldsymbol{x})) &= \operatorname{sign}\left(\log \frac{P(y=1|\boldsymbol{x})}{P(y=-1|\boldsymbol{x})}\right) \\
&= \begin{cases} 1 & \text{if } P(y=1|\boldsymbol{x}) > P(y=-1|\boldsymbol{x}) \\ -1 & \text{if } P(y=1|\boldsymbol{x}) < P(y=-1|\boldsymbol{x}) \end{cases} = \arg\max_{y\in\{-1,1\}} P(H(\boldsymbol{x}) = y|\boldsymbol{x})
\end{aligned}$$

meaning that $\operatorname{sign}(H(\boldsymbol{x}))$ is the Bayes optimal classifier. The continous and differentiable properties of $\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} e^{-y_i H(\boldsymbol{x}_i)}$ also justifies the choice of exponential loss function.

In this case

$$\begin{aligned}
(\alpha_t^*, h_t^*) &= \arg\min_{\alpha_t, h_t} \sum_{i=1}^{N} \mathcal{L}(y_i, H_{t-1}(\boldsymbol{x}_i) + \alpha_t h_t(\boldsymbol{x}_i)) \\
&= \arg\min_{\alpha_t, h_t} \sum_{i=1}^{N} \exp\left(-y_i H_{t-1}(\boldsymbol{x}_i) - \alpha_t y_i h_t(\boldsymbol{x}_i)\right) \\
&\triangleq \arg\min_{\alpha_t, h_t} \sum_{i=1}^{N} \widetilde{w_{ti}} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))
\end{aligned}$$

First we solve for h_t^*:

$$h_t^* = \arg\min_{h_t} \sum_{i=1}^{N} \widetilde{w_{ti}} \exp(-\alpha_t^* y_i h_t(\boldsymbol{x}_i)) = \arg\min_{h_t} \sum_{i=1}^{N} \widetilde{w_{ti}} \exp\left(-\alpha_t^* (1 - 2\mathbf{I}(h_t(\boldsymbol{x}_i) \neq y_i))\right)$$

$$\Rightarrow h_t^* = \arg\min_{h_t} \sum_{i=1}^{N} \widetilde{w_{ti}} \mathbf{I}(h_t(\boldsymbol{x}_i) \neq y_i)$$

meaning that h_t^* is the base learner that minimizes the **weighted** misclassification error rate.
Then we solve for α_t^*:

$$\begin{aligned}
\mathcal{L} = \widetilde{w_{ti}} \exp(-\alpha_t y_i h_t^*(\boldsymbol{x}_i)) &= \sum_{y_i = h_t^*(\boldsymbol{x}_i)} \widetilde{w_{ti}} e^{-\alpha_t} + \sum_{y_i \neq h_t^*(\boldsymbol{x}_i)} \widetilde{w_{ti}} e^{\alpha_t} \\
&= \sum_{i=1}^{N} \widetilde{w_{ti}} e^{-\alpha_t} + \sum_{i=1}^{N} \widetilde{w_{ti}} (e^{\alpha_t} - e^{-\alpha_t}) \mathbf{I}(h_t^*(\boldsymbol{x}_i) \neq y_i) \\
\frac{\partial \mathcal{L}}{\partial \alpha_t} &= -e^{-\alpha_t} \sum_{i=1}^{N} \widetilde{w_{ti}} + (e^{\alpha_t} + e^{-\alpha_t}) \sum_{i=1}^{N} \widetilde{w_{ti}} \mathbf{I}(h_t^*(\boldsymbol{x}_i) \neq y_i) = 0 \\
\Rightarrow \frac{e^{-\alpha_t}}{e^{\alpha_t} + e^{-\alpha_t}} &= \frac{\sum_{i=1}^{N} \widetilde{w_{ti}} \mathbf{I}(h_t^*(\boldsymbol{x}_i) \neq y_i)}{\sum_{i=1}^{N} \widetilde{w_{ti}}} = \sum_{i=1}^{N} \widetilde{w_{ti}} \mathbf{I}(h_t^*(\boldsymbol{x}_i) \neq y_i) = \epsilon_t \\
&\Rightarrow \alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}
\end{aligned}$$

A base model with a smaller classification error rate will have a larger weight in the ensemble model.
Last we update the distribution:

$$\begin{aligned}
\widetilde{w_{t+1,i}} &= \exp(-y_i H_t(\boldsymbol{x}_i)) = \exp(-y_i (H_{t-1}(\boldsymbol{x}_i) + \alpha_t h_t^*(\boldsymbol{x}_i))) = \exp(-y_i H_{t-1}(\boldsymbol{x}_i)) \exp(-\alpha_t y_i h_t^*(\boldsymbol{x}_i)) \\
&= \widetilde{w_{ti}} \exp(-\alpha_t y_i h_t^*(\boldsymbol{x}_i)) \\
\Rightarrow w_{t+1,i} \leftarrow \frac{\widetilde{w_{t+1,i}}}{\sum_{i=1}^{N} \widetilde{w_{t+1,i}}} &= \frac{w_{ti} \exp(-\alpha_t y_i h_t^*(\boldsymbol{x}_i))}{\sum_{i=1}^{N} w_{ti} \exp(-\alpha_t y_i h_t^*(\boldsymbol{x}_i))} \qquad \text{(Normalization)}
\end{aligned}$$

Boosting algorithm requires that **the base learners accept weighted samples**. For base learners that don't support weighted samples, we can address this issue by **re-sampling**, i.e., we can sample the training set with replacement according to the distribution $\mathcal{D}_t$ to generate a new training set.

3.2 Bagging

In order to obtain a more generalized ensemble model, each base learner should be as independent as possible. To make the base learners more diverse, one possible approach is to train them on different training sets.

Therefore, **Bagging**(Bootstrap Aggregating) is proposed. Given a training set $D = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, **we can bootstrap T new training sets $D_1, D_2, \ldots, D_T$ by sampling N samples from D with replacement each time.** The probability of a sample not being selected is

$$\left(1 - \frac{1}{N}\right)^N \to \frac{1}{e} \approx 0.368$$

so each new training set contains about 63.2% of the original samples. Then, we train T base learners on $D_1, D_2, \ldots, D_T$ respectively and combine them to form the ensemble model. For classification tasks, Bagging uses **majority voting** to combine the base learners; For regression tasks, Bagging uses **the average of the predictions** of the base learners.

```
def Bagging(D, T):
    for t=1,2,...,T:
        Sample D_t from D with replacement
        Train h_t(x) on D_t
    return H(x) = arg max_{y∈Y} ( Σ_{t=1}^{T} I(h_t(x) = y) )   # Classification Task
```

Moreover, since each base learner uses only 63.2% of the training set, the remaining 36.8% of the training set can be used as a **validation set** to estimate the **out-of-bag** performance of the ensemble model. Let $H^{\text{OOB}}(\boldsymbol{x})$ be the prediction of the ensemble model on the out-of-bag samples

$$H^{\text{OOB}}(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \left(\sum_{t=1}^{T} \mathbf{I}(h_t(\boldsymbol{x}) = y) \mathbf{I}(\boldsymbol{x} \notin D_t) \right)$$

then the error rate of the ensemble model can be estimated by

$$\varepsilon^{\text{OOB}} = \frac{1}{|D|} \sum_{(\boldsymbol{x},y) \in D} \mathbf{I}(H^{\text{OOB}}(\boldsymbol{x}) \neq y)$$

3.3 Random Forest

Random Forest is an extension for Bagging. In Random Forest, each base learner is a decision tree. To make the base learners more diverse, Random Forest introduces a randomization strategy:

- **Random feature selection:** Compared to traditional decision tree training, we first randomly select k features from total d features, and then choose the best splitting attribute from the k features.

Conventionally $k = \log_2 d$ is a good choice. So in Random Forest model, **the randomness not only comes from the bootstrapped training set, but also from the random feature selection**, which makes the base learners more diverse.

3.4 Ensemble Strategies

Suppose the ensemble model contains T base learners $\{h_1, h_2, \ldots, h_T\}$, and $h_t(\boldsymbol{x})$ is the output on the data point $\boldsymbol{x}$.

For numerical output $h_t(\boldsymbol{x}) \in \mathbb{R}$

- Averaging: $H(\boldsymbol{x}) = \sum_{t=1}^{T} w_t h_t(\boldsymbol{x})$

For categorical output $h_t(\boldsymbol{x}) \in \mathcal{Y} = \{c_1, \cdots, c_N\}$, denote the output of base learning h_t on data point $\boldsymbol{x}$ as a vector $h_t(\boldsymbol{x}) = [h_t^1(\boldsymbol{x}), h_t^2(\boldsymbol{x}), \cdots, h_t^N(\boldsymbol{x})]$. Each element $h_t^j(\boldsymbol{x})$ can be a hard label ($h_t^j(\boldsymbol{x}) \in \{0, 1\}$) or a soft label ($h_t^j(\boldsymbol{x}) \in [0, 1]$).

- Majority Voting: vote c_j if it receives more than half of the votes. Reject to predict if there's no "majority".

$$H(\boldsymbol{x}) = \begin{cases} c_j & \text{if } \sum_{t=1}^{T} h_t^j(\boldsymbol{x}) \geq 0.5 \sum_{k=1}^{N} \sum_{t=1}^{T} h_t^k(\boldsymbol{x}) \\ \text{Reject} & \text{otherwise} \end{cases}$$

- Plurality Voting: vote c_j if it receives the most votes $H(\boldsymbol{x}) = c_{\arg\max_j \sum_{t=1}^{T} h_t^j(\boldsymbol{x})}$
- Weighted Voting: vote c_j if it receives the most weighted votes $H(\boldsymbol{x}) = c_{\arg\max_j \sum_{t=1}^{T} w_t h_t^j(\boldsymbol{x})}$

References

- Machine Learning, Zhi-Hua Zhou
- AdaBoost: Algorithm and Derivation