1. Model Selection

a) Cross Validation

- Split the data set $\mathcal{D}$ into K subsets $\mathcal{D}^{(i)}$ (called folds)
- For $i = 1, \dots, K$, train $h^{(i)}$ using all data but the i-th fold ($\mathcal{D} \backslash \mathcal{D}^{(i)}$).
- Cross-validation error by averaging all validation errors $\hat{\epsilon}_{\mathcal{D}^{(i)}}(h^{(i)})$

2. KNN

a) 误差界估计 $\lim_{n\to\infty} \epsilon_{\text{KNN}}(n) \le 2\epsilon^*(1-\epsilon^*)$

b) Normalization

- Mean $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$
- Variance $\sigma_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ij}-\mu_j)^2}$
- Normalize the feature into a new one:

$$\hat{x}_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sigma_j}$$

c) Distance

- **Cosine Distance:** $\cos(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\|\boldsymbol{x}_i\| \|\boldsymbol{x}_j\|}$
- **Minkowski Distance:** $D_p(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt[p]{\sum_{k=1}^{d} |x_{ik} - x_{jk}|^p}$
- **Mahalanobis Distance:** $D_M(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}} \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)}$

d) Weighted KNN

- Distance-weighted Classification:

$$\hat{y} = \text{argmax}_{c \in \mathcal{Y}} \sum_{\boldsymbol{x}' \in \text{KNN}_c(\boldsymbol{x})} \frac{1}{D(\boldsymbol{x}, \boldsymbol{x}')^2}$$

- Distance-weighted Regression:

$$\hat{y} = \sum_{\boldsymbol{x}' \in \text{KNN}(\boldsymbol{x})} \frac{1}{D(\boldsymbol{x}, \boldsymbol{x}')^2} y'$$

e) Nearest Centroid Classifier

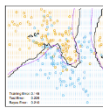
- Given dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$ with class label $y_i \in \mathcal{Y}$:
  - Compute per-class centroids $\boldsymbol{\mu}_c = \frac{1}{|N_c|}\sum_{y_i = c} \boldsymbol{x}_i$.
  - Prediction function: $\hat{y} = \text{argmin}_{c \in \mathcal{Y}} \|\boldsymbol{x} - \boldsymbol{\mu}_c\|$
- $O(Nd)$ training and $O(Cd)$ testing.
  - Faster!
- Gaussian assumption for each class.
  - Cannot tackle complex distributions.
  - Very good for Few-Shot Learning (FSL).

f) Complexity: n examples, d dimensions

- For naïve KNN:
  - $O(1)$ for training.
  - $O(nd)$ to find k closest examples.
- For KNN with k-d Tree:
  - $O(dn \log n)$ for training (build k-d tree)
  - $O(2^d \log n)$ on average when query.
  - Use k-d tree only when $n \gg 2^d$.

g) Summary

- **When to use:**
  - Few attributes per instance (expensive computation)
  - Lots of training data (curse of dimensionality)
- **Advantages:**
  - Agnostically learn complex target functions
  - Do not lose information (store original data)
  - Data number can be very large (big pro!)
  - Class number can be very large (biggest pro!)
  - All other ML algorithms may fail here!
- **Disadvantages:**
  - Slow at inference time (acceleration a must)
  - Ineffective in high dimensions (curse of dimensionality)
  - Fooled easily by irrelevant attributes (feature engineering crucial)



3. Linear Regression

a) Analytic Sol

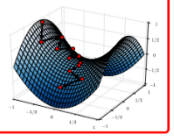
- Computing the gradient of $\hat{\epsilon}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ and setting it to zero yields the optimal parameter $\boldsymbol{w}^*$:

$$\hat{\epsilon}(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$$
$$\nabla_{\boldsymbol{w}} \hat{\epsilon}(\boldsymbol{w}) = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = \boldsymbol{0}$$
$$\Rightarrow \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$
$$\Rightarrow \boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$ Normal Equation

In Matrix Cookbook:
- $\|A\|_2^2 = \text{tr}(A^T A)$
- $\frac{\partial \text{tr}(A^T B)}{\partial A} = B$

b) Optimization: SGD

- In each iteration t ($\le T$):
  - Randomly sample a minibatch of $m \ll n$ points $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$
  - Set $J^t(\boldsymbol{w}^t) = \frac{1}{m}\sum_{i=1}^m \ell(\boldsymbol{w}^t; \boldsymbol{x}_i, y_i)$
  - Compute gradient on minibatch: $\Delta^t = \nabla_{\boldsymbol{w}} J^t(\boldsymbol{w}^t)$
  - Update parameters with learning rate η: $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta\Delta^t$



- For linear regression: $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - 2\eta\boldsymbol{X}_m^T(\boldsymbol{X}_m\boldsymbol{w}^t - \boldsymbol{y})$
  - Computational complexity $O(dmT)$. What if large dimension d?
- Theoretically, convergence rate is $O(1/\sqrt{T})$ under convex condition.

c) Regularization

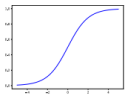
- Norm-regularization will make the hypothesis smooth at any point.

4. Linear Classification

a) Logistic Regression

- To map the output of $h(\boldsymbol{x})$ into $[0,1]$, we use sigmoid function:

$$\sigma(t) = \frac{1}{1+e^{-t}}$$



  - Sigmoid maps $\mathbb{R}$ to $[0,1]$.
  - $t \to +\infty, \sigma(t) \to 1$.
- Set $\sigma(h(\boldsymbol{x})) = p(y=1|\boldsymbol{x})$ as the probability to label $\boldsymbol{x}$ as $y = 1$.

$$\hat{\epsilon}(w) = -\sum_{i=1}^{n}\{y_i \log \sigma(h_w(\boldsymbol{x})) + (1-y_i)\log[1-\sigma(h_w(\boldsymbol{x}))]\} + \lambda\sum_{j=1}^{d} w_j^2$$

b) SoftMax Regression

- Softmax function normalizes multiple outputs in a probability vector:

$$p(y = c|\boldsymbol{x}; \boldsymbol{W}) = \frac{\exp(\boldsymbol{w}_c^T\boldsymbol{x})}{\sum_{r=1}^{C}\exp(\boldsymbol{w}_r^T\boldsymbol{x})}$$ Sum over all classes

Loss for each data point $(\boldsymbol{x}_i, y_i)$:

$$\ell(h(\boldsymbol{x}_i), y_i) = \begin{cases} -\log\frac{\exp(\boldsymbol{w}_1^T\boldsymbol{x}_i)}{\sum_{r=1}^{C}\exp(\boldsymbol{w}_r^T\boldsymbol{x}_i)}, y_i = 1 \\ -\log\frac{\exp(\boldsymbol{w}_2^T\boldsymbol{x}_i)}{\sum_{r=1}^{C}\exp(\boldsymbol{w}_r^T\boldsymbol{x}_i)}, y_i = 2 \\ \vdots \\ -\log\frac{\exp(\boldsymbol{w}_C^T\boldsymbol{x}_i)}{\sum_{r=1}^{C}\exp(\boldsymbol{w}_r^T\boldsymbol{x}_i)}, y_i = C \end{cases}$$

Equivalent to maximum log-likelihood estimation

5. SVM

a) SM SVM

Original form

$$\min_{w,b,\xi} \frac{1}{2}\|\boldsymbol{w}\|_2^2$$
$$\text{s.t. } y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \ge 1 - \xi_i$$
$$\xi_i \ge 0, \sum_{i=1}^{n}\xi_i \le n', 1 \le i \le n$$

Lagrangian equivalent form

$$\min_{w,b,\xi} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \ge 1 - \xi_i$$
$$\xi_i \ge 0, 1 \le i \le n$$

b) Dual Problem

Primal Problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$$
$$\text{s.t. } g_j(\boldsymbol{x}) \le 0 \text{ for } j = 1, \dots, J$$
$$h_k(\boldsymbol{x}) = 0 \text{ for } k = 1, \dots, K$$

- With Lagrange multipliers $\boldsymbol{\lambda}, \boldsymbol{\mu}$, the Lagrangian function is defined as

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{j=1}^{J}\lambda_j g_j(\boldsymbol{x}) + \sum_{k=1}^{K}\mu_k h_k(\boldsymbol{x})$$

KKT conditions

- KKT conditions: for $1 \le j \le J$
  - Primal feasibility: $g_j(\boldsymbol{x}) \le 0, h_k(\boldsymbol{x}) = 0$
  - Dual feasibility: $\lambda_j \ge 0$
  - Complementary slackness: $\lambda_j g_j(\boldsymbol{x}) = 0$
  - The gradient of the Lagrangian function w.r.t. $\boldsymbol{x}$ vanishes to $\boldsymbol{0}$:

$$\nabla_x f(\boldsymbol{x}) + \sum_{j=1}^{J}\lambda_j\nabla_x g_j(\boldsymbol{x}) + \sum_{k=1}^{K}\mu_k\nabla_x h_k(\boldsymbol{x}) = \boldsymbol{0}$$

- If $\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*$ satisfy KKT for a convex problem, then $\boldsymbol{x}^*$ is an optimum: $f(\boldsymbol{x}^*) = L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$
  - $\boldsymbol{x}^*$ is optimal if and only if there exist $\boldsymbol{\lambda}, \boldsymbol{\mu}$ satisfying KKT conditions.

Dual Problem

- Define the primal objective as

$$\Pi(\boldsymbol{x}) = \max_{\boldsymbol{\lambda}\in\mathbb{R}^J, \boldsymbol{\mu}\in\mathbb{R}^K} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

- Define the dual objective as

$$\Gamma(\boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq \min_{\boldsymbol{x}\in\mathbb{R}^d} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

- Weak duality: $\max_{\boldsymbol{\lambda}\in\mathbb{R}^J, \boldsymbol{\mu}\in\mathbb{R}^K} \Gamma(\boldsymbol{\lambda}, \boldsymbol{\mu}) \le \min_{\boldsymbol{x}\in\mathbb{R}^d} \Pi(\boldsymbol{x})$

Minmax theorem: $\max_x \min_y f(x,y) \le \min_y \max_x f(x,y)$

- Dual Problem:

$$\max_{\boldsymbol{\lambda}\in\mathbb{R}^J, \boldsymbol{\mu}\in\mathbb{R}^K} \Gamma(\boldsymbol{\lambda}, \boldsymbol{\mu})$$
$$\text{s.t. } \lambda_j \ge 0 \text{ for } j = 1, \dots, J$$

Soft-Margin SVM Dual Problem

- Lagrangian function (with $2n$ inequality constraints):

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu})$$
$$= \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b)) - \sum_{i=1}^{n}\mu_i\xi_i$$
$$\alpha_i \ge 0, \mu_i \ge 0, i = 1, \dots, n$$

  - After solving for $\boldsymbol{\alpha}$, we can solve for $\boldsymbol{w} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i$.

$-\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{0} \Rightarrow \boldsymbol{w} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i$

$-\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0$

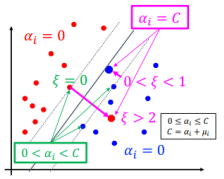
$-\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \mu_i, i = 1, \dots, n$

- Dual Problem of Soft-SVM:

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$$
$$\text{s.t. } \sum_{i=1}^{n}\alpha_i y_i = 0$$
$$0 \le \alpha_i \le C, 1 \le i \le n.$$

- KKT Conditions for Soft-SVM:

$$\begin{cases} \alpha_i \ge 0, \quad \mu_i \ge 0 \\ y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 + \xi_i \ge 0 \\ \alpha_i(y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 + \xi_i) = 0 \\ \xi_i \ge 0, \quad \mu_i\xi_i = 0 \end{cases}$$



- $\alpha_i(y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 + \xi_i) = 0$
  - $\alpha_i \ne 0$: Support vectors (SVs)
    - All data points that satisfy: $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) = 1 - \xi_i \le 1$.
  - $\boldsymbol{w} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i$ is a linear combination of support vectors.

c) Primal Problem: SGD

- Theorem: Soft-SVM is equivalent to Regularized Risk Minimization:

$$\min_{w,b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b)\}$$

- $\ell_{0/1} = 1[t \le 0]$
- $\ell_{\text{hinge}} = \max\{1-t, 0\}$
- $\ell_{\exp} = \exp(-t)$
- $\ell_{\text{logistic}} = \log(1 + e^{-t})$
- where $t = y(\boldsymbol{w}\cdot\boldsymbol{x} + b)$

$$\min_{w,b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b)\}$$

- SGD for Soft-SVM:
  - In each iteration t ($\le T$):
    - Choose i uniformly at random from $1, \dots, n$ (not in mini-batches)
    - The subgradient $\boldsymbol{g}_i^t = \begin{cases} \boldsymbol{w}^t & \text{if } y_i(\boldsymbol{w}^t\cdot\boldsymbol{x}_i + b) \ge 1 \\ \boldsymbol{w}^t - Cy_i\boldsymbol{x}_i & \text{otherwise} \end{cases}$
    - Update $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta\boldsymbol{g}^t$
  - Output $\bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^T \boldsymbol{w}^t$

SGD for convex optimization
Convergence rate: $O(\frac{1}{\sqrt{T}})$ (14.3)

d) Kernel SVM

- Commonly used kernel functions for vector data:

Linear	$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 \cdot \boldsymbol{x}_2$
Polynomial	$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2)^d$
Polynomial (with constant terms)	$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^d$
RBF (Gaussian kernel)	$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{2\sigma^2}\right)$
Hyperbolic Tangent	$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(\beta\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + c)$

- From kernel functions k_1, k_2, we can construct new kernel functions:
  - $k'(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1 \otimes k_2(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2)k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$
  - For any function $g: \mathcal{X} \to \mathbb{R}$, $k'(\boldsymbol{x}_1, \boldsymbol{x}_2) = g(\boldsymbol{x}_1)k_1(\boldsymbol{x}_1, \boldsymbol{x}_2)g(\boldsymbol{x}_2)$
- Theorem (Mercer): If $k(\cdot,\cdot)$ is a symmetric function on space $\mathcal{X}\times\mathcal{X}$, then k is a kernel function $\Leftrightarrow$ For any input set $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m)$,

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_1, \boldsymbol{x}_m) & \cdots & k(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{pmatrix}$$

The kernel matrix is semi-definite ($\boldsymbol{K} \succcurlyeq 0$, i.e. $\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \ge 0, \forall \boldsymbol{x}$).

- For kernel functions $k_1, k_2, \dots, k_s$, and $\gamma_1, \gamma_2, \dots, \gamma_s > 0$:
  - $\sum_{i=1}^s \gamma_s k_s$ is also (multi-)kernel function, because $\sum_{i=1}^s \gamma_s \boldsymbol{K}_s \ge 0$.
- Dual Problem of Soft-SVM:

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$
$$\text{s.t. } \sum_{i=1}^{n}\alpha_i y_i = 0$$
$$0 \le \alpha_i \le C, 1 \le i \le n.$$

SS

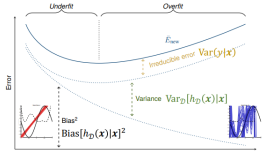
  - After solving for $\boldsymbol{\alpha}$, we can solve for $\boldsymbol{w} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$.
  - Testing: $f(\boldsymbol{x}) = \boldsymbol{w}\cdot\boldsymbol{\Phi}(\boldsymbol{x}) + b = \sum_{i=1}^n \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b$.
  - During training and testing, only need to compute $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

SS

6. Learning Theory

a) Bias-Variance Decomposition

- Theorem: $\mathcal{E}_{L2}(\boldsymbol{x}) = \text{Var}(y|\boldsymbol{x}) + \text{Bias}[h_{\mathcal{D}}(\boldsymbol{x})|\boldsymbol{x}]^2 + \text{Var}_{\mathcal{D}}[h_{\mathcal{D}}(\boldsymbol{x})|\boldsymbol{x}]$.



b) Approximation Error and Estimation Error

- General equality: given a target function f, for any $h \in \mathcal{H}$,

$$\mathcal{E}(h) - \mathcal{E}^*(f) = \underbrace{[\mathcal{E}(h) - \mathcal{E}(h^*)]}_{\text{estimation}} + \underbrace{[\mathcal{E}(h^*) - \mathcal{E}^*(f)]}_{\text{approximation}}$$

c) PAC Learning

- A hypothesis space $\mathcal{H}$ is PAC-learnable if there exists an algorithm $\mathcal{A}$ and a polynomial function poly(), such that for any $\varepsilon > 0, \delta > 0$, for all distributions D on $\mathcal{X}$ and for any target hypothesis $h \in \mathcal{H}$, the following holds for sample complexity $n \geq \text{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}, |\mathcal{H}|)$:

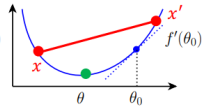
$$P_{\mathcal{D}_n \sim D^n}\left[\mathcal{E}(h_{\mathcal{D}_n}) - \min_{h \in \mathcal{H}} \mathcal{E}(h) \geq \varepsilon\right] \leq \delta$$

$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \mathcal{E}(h)$ — Approximately correct — Probably correct

d) Probability Tools

Basic

- Union bound: $P(A \vee B) \leq P(A) + P(B)$.
- Inversion: if $P(X \geq \epsilon) \leq f(\epsilon)$, then for any $\delta > 0$, with probability at least $1 - \delta$, $X \leq f^{-1}(\delta)$.
  - Useful to interchange ϵ and δ: $P\left(\left|\hat{\mathcal{E}}_{\mathcal{D}_n}(h) - \mathcal{E}(h)\right| \geq \varepsilon\right) \leq \delta$.
- Jensen' inequality: if f is convex, then $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$.
  - Convex function: f is convex if $\forall \lambda \in [0,1], x, x' \in \text{dom}(f)$:
    - $f(\lambda x + (1-\lambda)x')$ $\leq \lambda f(x) + (1-\lambda) f(x')$
    - $f(x') \geq f(x) + g^T(x' - x)$
      - g: Subgradient.



Concentration Inequalities

Markov's Inequality

Theorem: If Z is a nonnegative random variable, then for any $\varepsilon > 0$:

$$P(Z \geq \varepsilon) \leq \frac{\mathbb{E}[Z]}{\varepsilon}$$

Proof:

Chebyshev's Inequality

$$P(V \geq \varepsilon) \leq P(\phi(V) \geq \phi(\varepsilon)) \leq \frac{\mathbb{E}[\phi(V)]}{\phi(\varepsilon)}$$

Chernoff Bound

- Theorem: If we take $\phi(t) = e^{\lambda t}$, we have for any $\lambda > 0$:

$$P(V \geq \varepsilon) \leq \frac{\mathbb{E}\left[e^{\lambda V}\right]}{e^{\lambda \varepsilon}}$$

- If we take $V = Z - \mathbb{E}[Z]$, we have:

$$P(Z - \mathbb{E}[Z] \geq \varepsilon) \leq \frac{\mathbb{E}\left[e^{\lambda(Z - \mathbb{E}[Z])}\right]}{e^{\lambda \varepsilon}}$$

- If we set Z as the sum of i.i.d. random variables: $Z = \sum_{i=1}^n X_i$

$$P\left(\sum_{i=1}^n (X_i - \mathbb{E}X_i) \geq \varepsilon\right) \leq \frac{\mathbb{E}\left[e^{\lambda \sum_{i=1}^n (X_i - \mathbb{E}X_i)}\right]}{e^{\lambda \varepsilon}} = \frac{\prod_{i=1}^n \mathbb{E}\left[e^{\lambda (X_i - \mathbb{E}X_i)}\right]}{e^{\lambda \varepsilon}}$$

Hoeffding's Lemma

- Lemma: Let V be a bounded random variable, $\mathbb{E}[V] = 0$ and $a \leq V \leq b$ with $b > a$. Then for any $\lambda > 0$:

$$\mathbb{E}\left[e^{\lambda V}\right] \leq e^{\frac{\lambda^2 (b-a)^2}{8}}$$

- Plug in the optimal $\lambda = 4\varepsilon / \sum_{i=1}^n (b_i - a_i)^2 \Rightarrow$ Hoeffding's inequality:

$$P\left(\sum_{i=1}^n (X_i - \mathbb{E}X_i) \geq \varepsilon\right) \leq \frac{\prod_{i=1}^n \mathbb{E}\left[e^{\lambda (X_i - \mathbb{E}X_i)}\right]}{e^{\lambda \varepsilon}} \leq e^{\sum_{i=1}^n \frac{\lambda^2 (b_i - a_i)^2}{8} - \lambda \varepsilon} \overset{\min}{\Longrightarrow} \exp\left[-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right]$$

Chernoff Bound — Hoeffding's Lemma — Optimal λ

- If we replace X_i with $-X_i \in [-b_i, -a_i]$, we have

$$P\left(\sum_{i=1}^n (X_i - \mathbb{E}X_i) \leq -\varepsilon\right) \leq \exp\left[-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right]$$

- Theorem (Hoeffding's Inequality): for $a_i \leq X_i \leq b_i$, finite-sample bound

$$P\left(\left|\sum_{i=1}^n (X_i - \mathbb{E}X_i)\right| \geq \varepsilon\right) \leq 2\exp\left[-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right]$$

McDiarmid's Inequality

- Theorem: Let $X_1, X_2, \ldots, X_n \in \mathcal{X}$ be independent random variables and $f: \mathcal{X}^n \to \mathbb{R}$ be a function that is stable to input change:

$$\sup_{x_1, x_2, \ldots, x_n, x_i'} |f(x_1, \ldots, x_i, \ldots, x_n) - f(x_1, \ldots, x_i', \ldots, x_n)| \leq c_i$$

for all $i \in [1, n]$. Then, for all $\varepsilon > 0$: finite-sample bound!

$$P(|f(X_1, \ldots, X_n) - \mathbb{E}f(X_1, \ldots, X_n)| \geq \varepsilon) \leq 2\exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

e) Generalization Bound: Finite Hypothesis Space

- Theorem: Let $\mathcal{H}$ be a finite hypothesis space, $|\mathcal{H}| < \infty$, then for any $\delta > 0$, with probability at least $1 - \delta$,

$$\forall h \in \mathcal{H}, \quad \mathcal{E}(h) \leq \hat{\mathcal{E}}_{\mathcal{D}_n}(h) + \sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}$$

f) Rademacher Complexity

- Let $\mathcal{G}$ be a family of general functions mapping from $\mathcal{Z}$ to $[0,1]$.
- Let σ_i be i.i.d. uniform random variables (i.e. Rademacher variable):

$$P(\sigma_i = 1) = 1/2, P(\sigma_i = -1) = 1/2$$

- Empirical Rademacher Complexity of $\mathcal{G}$ on a size-n sample set $\mathcal{S}_n = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n\}$: Average loss on all dichotomies

$$\hat{\mathcal{R}}_{\mathcal{S}_n}(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{g \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^n \sigma_i g(\boldsymbol{z}_i)\right]$$

Peter Bartl

- (Expected) Rademacher Complexity:

$$\mathcal{R}_n(\mathcal{G}) = \mathbb{E}_{\mathcal{S}_n \sim D^n} \hat{\mathcal{R}}_{\mathcal{S}_n}(\mathcal{G}) = \mathbb{E}_{\mathcal{S}_n \sim D^n} \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{g \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^n \sigma_i g(\boldsymbol{z}_i)\right]$$

- Theorem: Rademacher complexity will decrease when n increases:

$$\mathcal{R}_{n+1}(\mathcal{G}) \leq \mathcal{R}_n(\mathcal{G})$$

- Theorem: Let $\mathcal{H}$ be a family of binary classifiers taking values in $\{-1, +1\}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the generalization bound holds for all $h \in \mathcal{H}$:

$$\mathcal{E}_D(h) \leq \hat{\mathcal{E}}_{\mathcal{D}_n}(h) + \mathcal{R}_n(\mathcal{H}) + \sqrt{\frac{\log(1/\delta)}{2n}}$$

$$\mathcal{E}_D(h) \leq \hat{\mathcal{E}}_{\mathcal{D}_n}(h) + \hat{\mathcal{R}}_n(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2n}}$$

g) VC dim

- VC-dimension of a hypothesis space $\mathcal{H}$ is defined by

$$\text{VCdim}(\mathcal{H}) = \max\{n : \Pi_{\mathcal{H}}(n) = 2^n\}$$

  - VC-dimension is essentially the size of the largest set that can be fully shattered by $\mathcal{H}$.
    - No need to shatter every sample set of size n.
  - Lower bound: Build a sample of size n that can be shattered.
  - Upper bound: Prove all samples of size $n+1$ cannot be shattered.

- Let $\mathcal{H}$ be a hypothesis set with $\text{VCdim}(\mathcal{H}) = d$
  - If $d = \infty, \Pi_{\mathcal{H}}(n) \leq 2^n$.
  - If $d < \infty, \Pi_{\mathcal{H}}(n) \leq \left(\frac{en}{d}\right)^d = O(n^d)$. — $d < \infty$ is PAC learnable!

Vladimir Vapnik
Statistical learning theory

- Theorem: Let $\mathcal{H}$ be a family of functions taking values in $\{-1, +1\}$ with VC-dimension d, then for any $\delta > 0$, with probability at least $1 - \delta$, for all $h \in \mathcal{H}$:

$$\mathcal{E}_D(h) \leq \hat{\mathcal{E}}_{\mathcal{D}_n}(h) + \sqrt{\frac{2d \log\frac{en}{d}}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- The general form: $\mathcal{E}_D(h) \leq \hat{\mathcal{E}}_{\mathcal{D}_n}(h) + O\left(\sqrt{\frac{\log(n/d)}{(n/d)}}\right)$.

7. Decision Tree

- Misclassification error:

$$\text{Err}(\mathcal{D}) = 1 - \max_{1 \leq k \leq K}\left(\frac{|\mathcal{C}_k|}{|\mathcal{D}|}\right)$$

- Entropy (used in ID3 and C4.5):

$$H(\mathcal{D}) = -\sum_{k=1}^K \frac{|\mathcal{C}_k|}{|\mathcal{D}|} \log\frac{|\mathcal{C}_k|}{|\mathcal{D}|}$$

- Gini index (used in CART):

$$\text{Gini}(\mathcal{D}) = 1 - \sum_{k=1}^K \left(\frac{|\mathcal{C}_k|}{|\mathcal{D}|}\right)^2$$

a) ID3

Class label

ID3(*Examples*, *Target_attribute*, *Attributes*)
- create a *Root* node for the tree; assign all *Examples* to *Root*;

Stop Criteria
- if all *Examples* are positive, return the single-node tree *Root*, with label=+;
- if all *Examples* are negative, return the single-node tree *Root*, with label=−;
- if *Attributes* is empty, return the single-node tree *Root*, with label = the most common value of *Target_attribute* in *Examples*;
- otherwise // Main loop:

  $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*; — $O(dn)$ in each layer
  the decision attribute for *Root* $\leftarrow A$;
  for each possible value v_i of A
    add a new tree branch below *Root*, corresponding to the test $A = v_i$;
    let $Examples_{v_i}$ be the subset of *Examples* that have the value v_i for A;
    if $Examples_{v_i}$ is empty
      below this new branch add a leaf node with label = the most common value of *Target_attribute* in *Examples*;
    else
      below this new branch add the subtree — depth $\min(d, \log n)$
      ID3($Examples_{v_i}$, *Target_attribute*, *Attributes*\{A});
- return *Root*;

* The *best* attribute is the one with the highest *information gain*.

b) C4.5

IG Rate

- By penalizing multivalued rate, IG is improved to Gain Ratio (GR):

$$\text{GR} = \frac{\text{Information Gain}}{\text{multivalued rate}}$$

- C4.5 Algorithm measures multivalued rate by Intrinsic Value (IV):

$$\text{IV}(f) = -\sum_{i=1}^{|V|} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \log\frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

  - When $\mathcal{D}$ is split by the feature f to $\mathcal{D}_i$ if feature value is i.
  - Which is the entropy of the value probability of the feature f.

Attribute with costs

- C4.5 introduces a new criterion that could penalize the cost:

$$\frac{(\text{Gain Ratio})^2}{\text{Cost}}$$

Missing Value

- Step 1: When computing IG, for a feature with missing values:
  - Compute the ratio of samples with missing value: $|\bar{\mathcal{D}}|/|\mathcal{D}| = \rho$.
  - Compute the IG on $\mathcal{D}\backslash\bar{\mathcal{D}}$ (all samples with values):
  - The IG for this feature is computed by: $(1 - \rho) \cdot$ IG.
    - Then compute Gain Ratio based on this new Information Gain.

Continuous Variables

- If $x_{1j}, \ldots, x_{nj}$ are the sorted values of the jth feature for n instances.
  - We only need to check split points between adjacent values
  - Traditionally take split points halfway between adjacent values:

$$s_j \in \left\{\frac{1}{2}(x_{rj} + x_{(r+1)j}) \middle| r = 1, \ldots, n-1\right\}$$

- Check performance of $n-1$ splits and find the one with highest IG.

Pruning

- Pre-Pruning
  - If the accuracy grows no more than eta or even decreases: Stop the split
  - May cause underfitting
- Post-Pruning
  - Avoid underfitting

- The cost complexity criterion with parameter α for tree T is

$$C_\alpha(T) = \hat{\mathcal{E}}(T) + \alpha|T| = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T|$$

$$= -\sum_{t=1}^{|T|} \sum_{k=1}^K N_{tk} \log\frac{N_{tk}}{N_t} + \alpha|T|$$

  - $|T|$: #leaf nodes, N_{tk} is the #examples of class k in leaf node t
  - $\hat{\mathcal{E}}(T)$ is the empirical error of the tree on training data.
- Cost Complexity Pruning: (Given tree T_0 and parameter α)
  - Compute the empirical entropy $H_t(T)$ of each node
  - Recursively shrink from leaf nodes to internal nodes
    - If $C_\alpha(T_B) \leq C_\alpha(T_A)$: prune leaf A and use parent B as new leaf

c) CART

Classification Tree

- If $\mathcal{D}$ is split into $\mathcal{D}_1$ and $\mathcal{D}_2$ by whether attribute $A(x) = a$:

$$\mathcal{D}_1 = \{(x, y) \in \mathcal{D} : A(x) = a\}, \mathcal{D}_2 = \mathcal{D} - \mathcal{D}_1$$

- Then under the condition of attribute A, the Gini index of $\mathcal{D}$ is

$$\text{Gini}(\mathcal{D}, A) = \frac{|\mathcal{D}_1|}{|\mathcal{D}|}\text{Gini}(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{|\mathcal{D}|}\text{Gini}(\mathcal{D}_2)$$

- Gini index $\text{Gini}(\mathcal{D}, A)$ is used to evaluate node split in CART, which is a binary tree where each node is split into $A(x) = a$ and $A(x) \neq a$

Regression Tree

- For each splitting variable j and splitting point s
  - Assume that the region R is split by j and s to R_1 and R_2.

$$\hat{v}_1(j, s) = \text{avg}(y_i | \boldsymbol{x}_i \in R_1(j, s))$$

$$\hat{v}_2(j, s) = \text{avg}(y_i | \boldsymbol{x}_i \in R_2(j, s))$$

- Find j, s that minimize the loss

$$\ell(j, s) = \underbrace{\sum_{i: \boldsymbol{x}_i \in R_1(j,s)} (y_i - \hat{v}_1(j, s))^2}_{\text{L2 loss for samples in } R_1(j,s)} + \underbrace{\sum_{i: \boldsymbol{x}_i \in R_2(j,s)} (y_i - \hat{v}_2(j, s))^2}_{\text{L2 loss for samples in } R_2(j,s)}$$

- As we did in classification, search on all (j, s)'s and find a best one.

8. Random Forest

a) Bagging

- Bagging is a general technique to reduce the variance of an estimator:
  - Draw B bootstrap samples $\mathcal{D}_n^1, \ldots, \mathcal{D}_n^B$ from original data $\mathcal{D}_n$.
  - Let $h_{\mathcal{D}_n^1}, \ldots, h_{\mathcal{D}_n^B}$ be the prediction functions for each sample.
  - The bagged prediction function is a combination of these functions:

$$h_{\text{bag}}(x) = \text{Combine}\left(h_{\mathcal{D}_n^1}(\boldsymbol{x}), \ldots, h_{\mathcal{D}_n^B}(\boldsymbol{x})\right)$$

- How can we combine:
  - Binary/multiclass predictions, Class probability predictions } Vote — $\text{Var}(h_{\text{bag}}) \leq \text{Var}(h_{\mathcal{D}_n})$
  - Prediction functions for regression } Average

Leo Breiman

- For each training point $\boldsymbol{x}_i$, let

$$S_i = \{b | \mathcal{D}_n^b \text{ does not contain } \boldsymbol{x}_i\}$$

- The OOB prediction on $\boldsymbol{x}_i$ is

$$h_{\text{OOB}}(\boldsymbol{x}_i) = \frac{1}{|S_i|} \sum_{b \in S_i} h_{\mathcal{D}_n^b}(\boldsymbol{x}_i)$$

b) Breiman Algorithm

Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Feature subset size K.

Process:
1. $N \leftarrow$ create a tree node based on D;
2. **if** *all instances in the same class* **then return** N
3. $\mathcal{F} \leftarrow$ the set of features that can be split further;
4. **if** $\mathcal{F}$ *is empty* **then return** N
5. $\tilde{\mathcal{F}} \leftarrow$ select K features from $\mathcal{F}$ randomly;
6. $N.f \leftarrow$ the feature which has the best split point in $\tilde{\mathcal{F}}$;
7. $N.p \leftarrow$ the best split point on $N.f$;
8. $D_l \leftarrow$ subset of D with values on $N.f$ smaller than $N.p$;
9. $D_r \leftarrow$ subset of D with values on $N.f$ no smaller than $N.p$;
10. $N_l \leftarrow$ call the process with parameters (D_l, K);
11. $N_r \leftarrow$ call the process with parameters (D_r, K); $K \approx \sqrt{d}$
12. **return** N

Randomized features

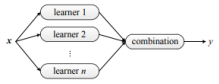
Output: A random decision tree

each tree

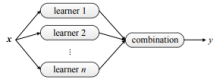
9. Boosting

a) Ensamble

- **Parallel ensemble**
  - Each model is built independently
  - Bagging and Random Forest
  - Idea: Combine different strong models to avoid overfitting



- **Sequential ensemble**
  - Models are built sequentially
  - Boosting and XGBoost
  - Idea: Combine many weak models to avoid underfitting.
  - Try to add new models that do well where previous models lack



b) Ada Boosting

$H \subseteq \{-1,+1\}^X$.

(Freund and Schapire, 1997)

Gödel prize (2003)

ADABOOST$(S=((x_1,y_1),\dots,(x_m,y_m)))$, $\boldsymbol{m=n}$

1 **for** $i \leftarrow 1$ **to** m **do**
2 $\quad D_1(i) \leftarrow \frac{1}{m}$
3 **for** $t \leftarrow 1$ **to** T **do**
4 $\quad h_t \leftarrow$ base classifier in H with small error $\epsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i]$
5 $\quad \alpha_t \leftarrow \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
6 $\quad Z_t \leftarrow 2[\epsilon_t(1-\epsilon_t)]^{\frac{1}{2}}$ ▷ normalization factor
7 $\quad$ **for** $i \leftarrow 1$ **to** m **do**
8 $\quad\quad D_{t+1}(i) \leftarrow \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ (Weight of instance i at iteration $t+1$)
9 $\quad f_t \leftarrow \sum_{s=1}^{t}\alpha_s h_s$
10 **return** $h = \operatorname{sgn}(f_T)$

Why?

Boosting ≻ Bagging ≻ Tree

- **Proof:** We minimize $F(\bar{\boldsymbol{\alpha}})$ by Coordinate Descent (CD).
- Note that $F(\bar{\boldsymbol{\alpha}}_{t-1}+\eta\boldsymbol{e}_k) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i\sum_{j=1}^{N}\bar{\alpha}_{t-1,j}h_j(x_i)-\eta y_i h_k(x_i)}$
- **Direction:** for each unit vector $\boldsymbol{e}_k$ with best directional derivative:

$$\begin{aligned}F'(\bar{\boldsymbol{\alpha}}_{t-1},\boldsymbol{e}_k) &= \lim_{\eta\to 0}\frac{F(\bar{\boldsymbol{\alpha}}_{t-1}+\eta\boldsymbol{e}_k)-F(\bar{\boldsymbol{\alpha}}_{t-1})}{\eta}\\ &= -\frac{1}{n}\sum_{i=1}^{n} y_i h_k(\boldsymbol{x}_i)e^{-y_i\sum_{j=1}^{N}\bar{\alpha}_{t-1,j}h_j(x_i)}\\ &= -\frac{1}{n}\sum_{i=1}^{n} y_i h_k(\boldsymbol{x}_i)\bar{D}_t(i)\bar{Z}_t\\ &= -\left[\sum_{i=1}^{n}\bar{D}_t(i)\mathbf{1}[y_i h_k(\boldsymbol{x}_i)=+1]-\sum_{i=1}^{n}\bar{D}_t(i)\mathbf{1}[y_i h_k(\boldsymbol{x}_i)=-1]\right]\frac{\bar{Z}_t}{n}\\ &= -[(1-\bar{\epsilon}_{t,k})-\bar{\epsilon}_{t,k}]\frac{\bar{Z}_t}{n} = [2\bar{\epsilon}_{t,k}-1]\frac{\bar{Z}_t}{n}\end{aligned}$$

Maximum descent direction k is the one minimizing $\bar{\epsilon}_{t,k}$ over distribution $\bar{D}_t$

- **Step size:** η is chosen to minimize $F(\bar{\boldsymbol{\alpha}}_{t-1}+\eta\boldsymbol{e}_k)$:

$$\begin{aligned}&\frac{\partial F(\bar{\boldsymbol{\alpha}}_{t-1}+\eta\boldsymbol{e}_k)}{\partial\eta} = 0 \Leftrightarrow -\frac{1}{n}\sum_{i=1}^{n} y_i h_k(\boldsymbol{x}_i)e^{-y_i\sum_{j=1}^{N}\bar{\alpha}_{t-1,j}h_j(x_i)-\eta y_i h_k(x_i)} = 0\\ &\Leftrightarrow \sum_{i=1}^{n} y_i h_k(\boldsymbol{x}_i)\bar{D}_t(i)\bar{Z}_t e^{-\eta y_i h_k(x_i)} = 0\\ &\Leftrightarrow \sum_{i=1}^{n}\bar{D}_t(i)y_i h_k(\boldsymbol{x}_i)e^{-\eta y_i h_k(x_i)} = 0\\ &\Leftrightarrow \sum_{i=1}^{n}\bar{D}_t(i)\mathbf{1}_{[y_i h_k(x_i)=+1]}e^{-\eta} - \sum_{i=1}^{n}\bar{D}_t(i)\mathbf{1}_{[y_i h_k(x_i)=-1]}e^{\eta} = 0\\ &\Leftrightarrow [(1-\bar{\epsilon}_{t,k})e^{-\eta}-\bar{\epsilon}_{t,k}e^{\eta}] = 0\\ &\Leftrightarrow \eta = \frac{1}{2}\log\frac{1-\bar{\epsilon}_{t,k}}{\bar{\epsilon}_{t,k}}\end{aligned}$$

Step size η of CD is weight α_t to classifier h_t in the tth iteration

- **Base learners:**
  - Decision trees, quite often just decision stumps.
  - Stumps are simple, weak learners:
    - Think the XOR example.
- **Boosting stumps:**
  - Data in $\mathbb{R}^d$.
  - Associate a stump to each feature.
  - Pre-sort each feature: $O(dn\log n)$
  - At each round, find best feature and threshold.
  - Total complexity with T stumps: $O(dn\log n + ndT)$

- **Strengths:**
  - Simple: straightforward implementation.
  - Efficient: complexity $O(ndT)$ for stumps.
  - Theoretical guarantees: aim to maximize margin but not achieved.
- **Outliers:**
  - AdaBoost assigns larger weights to harder examples.
  - Detecting mislabeled examples and dealing with noisy data.
- **Weaknesses:**
  - Determining T, the number of rounds of boosting is nontrivial.
  - Need to determine base learners: risk of overfitting or low margin.

- Theorem: The empirical error of the classifier h output by AdaBoost verifies:

$$\hat{\mathcal{E}}(h) \le \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right]$$

Furthermore, if for all $t\in[T], \gamma \le \left(\frac{1}{2}-\epsilon_t\right)$, then

$$\hat{\mathcal{E}}(h) \le \exp[-2\gamma^2 T]$$

- Additive model $f(\boldsymbol{x}) = \sum_{t=1}^{T}\alpha_t h_t(\boldsymbol{x})$ and $h(\boldsymbol{x}) = \operatorname{sgn}(f(\boldsymbol{x}))$.
  - ϵ_t is the error of each base learner h_t.
- Note: γ does not need to be known in advance: Adaptive Boosting.
- Proof: As we defined and derived previously, $D_{t+1}(i) = \frac{1}{n}\frac{e^{-y_i f_t(x_i)}}{\prod_{s=1}^{t} Z_s}$.

$$\begin{aligned}\hat{\mathcal{E}}(h) &\le \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}[y_i f(\boldsymbol{x}_i)\le 0] \le \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)}\\ &\le \frac{1}{n}\sum_{i=1}^{n}\left[n\prod_{t=1}^{T}Z_t\right]D_{T+1}(i) = \prod_{t=1}^{T}Z_t\end{aligned}$$

- Now recall that Z_t is a normalization factor:

$$\begin{aligned}Z_t &= \sum_{i=1}^{n} D_t(i)e^{-\alpha_t y_i h_t(x_i)}\\ &= \sum_{i:y_i h_t(x_i)\ge 0} D_t(i)e^{-\alpha_t} + \sum_{i:y_i h_t(x_i)<0} D_t(i)e^{\alpha_t}\\ &= (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}\end{aligned}$$

- Proof: We have $\hat{\mathcal{E}}(h) \le \prod_{t=1}^{T} Z_t$ and $Z_t = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$.
- Derivative: $\frac{\partial Z_t}{\partial \alpha_t} = -(1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 0 \Rightarrow \alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$.
- Compute minimum: $Z_t = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$.

$$\begin{aligned}\hat{\mathcal{E}}(h) &\le \prod_{t=1}^{T} Z_t \overset{\min}{\Longleftrightarrow} \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}\\ &= \prod_{t=1}^{T}\sqrt{1-4\left(\frac{1}{2}-\epsilon_t\right)^2} \qquad 1-x\le e^{-x}\\ &\le \prod_{t=1}^{T}\exp\left[-2\left(\frac{1}{2}-\epsilon_t\right)^2\right] = \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right]\end{aligned}$$

c) Gradient Boosting

- Gradient Boosting:
  - For $u_{ti} = \alpha_t h_t(\boldsymbol{x}_i)$, let the tth base learner $h_t(\boldsymbol{x}_i)$ approximate

$$h_t(\boldsymbol{x}_i) \approx -\nabla_{v_{ti}}\sum_{i=1}^{n}\ell(y_i, v_{ti}) = -\nabla_{v_{ti}}\ell(y_i, v_{ti})$$

  - And α_t be the learning rate η.

Regression

- Gradient boosting applies whenever:
  - Loss function is sub-differentiable w.r.t. training predictions $f(\boldsymbol{x}_i)$.
  - Can do regression on base hypotheses $h_t(\boldsymbol{x}_i)$, e.g. regression tree.

1. 令 $f_0(\mathbf{x}) = 0$。
2. For t=1 to T:
   (a) 计算在各个数据点上的梯度 $\mathbf{g}_t = \left(\frac{\partial}{\partial\widehat{\mathbf{y}}_i}\ell(\mathbf{y}_i,\widehat{\mathbf{y}}_i)|_{\widehat{\mathbf{y}}_i=f_{t-1}(\mathbf{x}_i)}\right)_{i=1}^{n}$。
   (b) 根据 $-\mathbf{g}_t$ 拟合一个回归模型，$h_t = \arg\min_{h\in\mathcal{F}}\sum_{i=1}^{n}(h_t(x_i)+g_{t,i})$
   (c) 选择合适的步长 α_t，最简单的选择是固定步长 $\eta\in(0,1]$。
   (d) 更新模型，$f_t(x) = f_{t-1}(x) + \alpha_t h_t(x)$ _____。

L2 Boosting

$$h_t = \operatorname{argmin}_h\sum_{i=1}^{n}\left(h(x_i)-2(y_i-f_{t-1}(x_i))\right)^2$$

Binomial Boosting

$$h_t = \operatorname{argmin}_h\sum_{i=1}^{n}\left(h(x_i)-\frac{y_i}{1+e^{y_i f_{t-1}(x_i)}}\right)^2$$

10. Reinforcement Learning: Model-Based

a) Mathematical Framework

Bellman Equation

- Similarly decomposed into immediate reward + discounted value.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s]$$
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1}+\gamma q_\pi(S_{t+1},A_{t+1})|S_t=s,A_t=a]$$

$$v_\pi(s) = \sum_{a\in\mathcal{A}}\pi(a|s)\,q_\pi(s,a) \qquad q_\pi(s,a) = R_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a\,v_\pi(s')$$

Optimal

- **Definition:**

The optimal state value function $v_*(s)$ is the maximum state value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

The optimal action value function $q_*(s,a)$ is the maximum action value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- **Theorem:** For any Markov Decision Process (MDP):
  - There exists an optimal policy π_* that is better than or equal to all other policies:

$$\pi_* \ge \pi, \forall\pi$$

  - All optimal policies achieve the optimal state value function:

$$v_{\pi_*}(s) = v_*(s)$$

  - All optimal policies achieve the optimal action value function:

$$q_{\pi_*}(s,a) = q_*(s,a)$$

$$\begin{aligned}v_*(s) &= \max_a q_*(s,a) \quad \text{Proof next slide}\\ q_*(s,a) &= R_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a v_*(s')\\ v_*(s) &= \max_a R_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a v_*(s')\\ q_*(s,a) &= R_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a\max_{a'} q_*(s',a')\end{aligned}$$

b) Policy Iteration

Evaluation

- **Solution:** iterative application of Bellman expectation backup
  - At each iteration $k+1$, for all states $s\in\mathcal{S}$:
  - Update $v_{k+1}(s)$ from $v_k(s')$, where s' is a successor state of s

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}}\pi(a|s)\left(\mathcal{R}_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a v_k(s')\right)$$

Improvement

- Consider a deterministic policy $a = \pi(s)$.
- We can improve the policy by acting greedily

$$\pi'(s) = \operatorname*{argmax}_{a\in\mathcal{A}} q_\pi(s,a)$$

c) Value Iteration

- Iterative method:
  - At each iteration $k+1$, for all states $s\in\mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$, where s' is a successor state of s:

Converge to v_* by Contraction Mapping

$$v_{k+1}(s) = \max_{a\in\mathcal{A}}\mathcal{R}_s^a + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a v_k(s')$$
$$\boldsymbol{v}_{k+1} = \max_{a\in\mathcal{A}}\boldsymbol{\mathcal{R}}^a + \gamma\boldsymbol{\mathcal{P}}^a\boldsymbol{v}_k$$

11. Reinforcement Learning: Model-Free

a) Monte Carlo Method

- First-visit MC prediction, for policy evaluation with $V \approx v_\pi$.
- The first time-step t that state s is visited in an episode:
  - Increment counter $N(s) \leftarrow N(s)+1$
  - Increment total return $G(s) \leftarrow G(s)+G_t$
- Value is estimated by sample average return $V(s) = \frac{G(s)}{N(s)}$
- By law of large numbers: $V(s)\to v_\pi(s)$ as $N(s)\to\infty$
- Every-visit MC update (online): $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

$$\begin{aligned}\mu_k &= \frac{1}{k}\sum_{j=1}^{k}x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1}x_j\right) = \frac{1}{k}(x_k+(k-1)\mu_{k-1})\\ &= \mu_{k-1} + \frac{1}{k}(x_k-\mu_{k-1}) = \mu_{k-1} + \alpha(x_k-\mu_{k-1})\end{aligned}$$

- First-visit MC control (for ϵ-greedy policy), estimates $\pi\approx\pi_*$
- Initialize:
  - $\pi \leftarrow$ an ϵ-soft policy, value $Q(s,a)$, empty list $\text{Returns}(s,a)$, $\forall s,a$
- Repeat (for each episode):
  - Sample an episode following π: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
  - $G \leftarrow 0$
  - Repeat (for each step of episode, $t = T-1, T-2, \dots 0$):
    - $G \leftarrow \gamma G + R_{t+1}$
    - Unless the pair S_t, A_t appears in $S_0, A_0, \dots, S_{t-1}, A_{t-1}$:
      - Append G to $\text{Returns}(S_t, A_t)$
      - $Q(S_t, A_t) \leftarrow \text{Avg}[\text{Returns}(S_t, A_t)]$
      - $A^* \leftarrow \arg\max_a Q(S_t, a)$ Generalized policy iteration (GPI)
      - For all $a\in\mathcal{A}$: $\pi(a|S_t) = \begin{cases}1-\epsilon+\epsilon/|\mathcal{A}| & \text{if } a = A^*\\ \epsilon/|\mathcal{A}| & \text{if } a\neq A^*\end{cases}$

- Theorem (ϵ-greedy policy improvement): For any ϵ-greedy policy π, the ϵ-greedy policy π' wrt q_π is an improvement, $v_{\pi'}(s)\ge v_\pi(s)$.
- Proof:

$$\begin{aligned}q_\pi(s,\pi'(s)) &= \sum_{a\in\mathcal{A}}\pi'(a|s)q_\pi(s,a)\\ &= \frac{\epsilon}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}q_\pi(s,a) + (1-\epsilon)\max_{a\in\mathcal{A}}q_\pi(s,a)\\ &\ge \frac{\epsilon}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}q_\pi(s,a) + (1-\epsilon)\sum_{a\in\mathcal{A}}\frac{\pi(a|s)-\frac{\epsilon}{|\mathcal{A}|}}{1-\epsilon}q_\pi(s,a)\\ &= \sum_{a\in\mathcal{A}}\pi(a|s)q_\pi(s,a) = v_\pi(s)\end{aligned}$$

- By policy improvement theorem: $q_\pi(s,\pi'(s))\ge v_\pi(s) \Rightarrow v_{\pi'}(s)\ge v_\pi(s)$.

b) TD Learning

- TD(0) prediction, for estimating $V \approx v_\pi$
- **Input:** the policy π to be evaluated, step size $\alpha \in [0,1]$
- **Initialize:**
  - Value $V(s), \forall s \in \mathcal{S}$, except that $V(\text{terminal}) = 0$
- **Repeat** (for each episode):
  - Initialize S
  - **Repeat** (for each step of episode):
    - $A \leftarrow$ action given by π for S
    - Take action A, observe R, S'
    - $V(S) \leftarrow V(S) + \alpha\big(R + \gamma V(S') - V(S)\big)$
    - $S \leftarrow S'$
  - **Until** S is terminal

- N-step TD prediction, for estimateing $V \approx v_\pi$
- **Initialize:** Value $V(s), \forall s \in \mathcal{S}$
- **Repeat** (for each episode):
  - Initialize and store $S_0 \neq \text{terminal}, T \leftarrow \infty$ (big cost)
  - **Repeat** (for $t = 0,1,2,\ldots$):
    - If $t < T$:
      - Take an action according to $\pi(\cdot|S_t)$
      - Observe and store next reward R_{t+1} & next state S_{t+1}
      - If S_{t+1} is terminal, then $T \leftarrow t+1$
    - $\tau \leftarrow t - n + 1$ (τ is the time with state value updated)
    - If $\tau \geq 0$:
      - $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      - If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$
      - $V(S_\tau) \leftarrow V(S_\tau) + \alpha\big(G - V(S_\tau)\big)$
  - **Until** $\tau = T - 1$

TD(lambda)

- **Input:** the policy π, discount γ, decay rate λ.
- **Repeat** (for each episode)
  - $E(s) \leftarrow 0$ for all $s \in \mathcal{S}$
  - Initialize S
  - **Repeat** (for each step of episode)
    - Choose $A \sim \pi(\cdot|S)$
    - Take action A, observe R, S'
    - TD error: $\delta = R + \gamma V(S') - V(S)$
    - Eligibility trace: $E(S) \leftarrow E(S) + 1$
    - For all $s \in \mathcal{S}$
      - $V(s) \leftarrow V(s) + \alpha\delta E(s)$
      - Eligibility trace: $E(s) = \gamma\lambda E(s)$
    - $S \leftarrow S'$
  - **Until** S is terminal

- Advantages of Temporal-Difference (TD) over Monte Carlo (MC):
  - Bootstrapping
  - Online, fully incremental updates, efficient
  - Applicable to continuing tasks without episodes

c) Sarsa

- Updating action-value function with Sarsa (S, A, R, S', A'):

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$$

- **Initialize:** $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, $Q(\text{terminal},\cdot) = 0$
- **Repeat** (for each episode):
  - Initialize S
  - Choose A from S using policy derived from Q (e.g. ϵ-greedy)
  - **Repeat** (for each step of episode): On-policy
    - Take action A, observe R, S'
    - Choose A' from S' using policy derived from Q (e.g. ϵ-greedy)
    - $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    - $S \leftarrow S', A \leftarrow A'$
  - **Until** S is terminal

Sarsa(lambda)

- **Initialize:** $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
- **Repeat** (for each episode):
  - $E(s,a) \leftarrow 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  - Initialize S, A
  - **Repeat** (for each step of episode):
    - Take action A, observe R, S'
    - Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    - TD error: $\delta \leftarrow R + \gamma Q(S',A') - Q(S,A)$
    - Eligibility trace: $E(S,A) \leftarrow E(S,A) + 1$
    - For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      - $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
      - Eligibility trace: $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    - $S \leftarrow S', A \leftarrow A'$
  - **Until** S is terminal

d) Q-Learning

- **Initialize:** $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, $Q(\text{terminal},\cdot) = 0$
- **Repeat** (for each episode):
  - Initialize S
  - **Repeat** (for each step of episode):
    - Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    - Take action A, observe R, S'
    - $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    - $S \leftarrow S'$
  - **Until** S is terminal

Q-learning : try to evaluate value function for greedy policy while following e-greedy policy, therefore off-policy
Sarsa : try to evaluate value function for e-greedy policy while following e-greedy policy, therefore on-policy

e) Policy Gradient Method

- Value-based reinforcement learning
  - Approximate value or action-value function by parameter $\boldsymbol{w}$
    - $\hat{v}(s,\boldsymbol{w}) \approx v_\pi(s)$ or $\hat{q}(s,a,\boldsymbol{w}) \approx q_\pi(s,a)$
  - Generate policy from value function by greedy or ϵ-greedy
- Policy-based reinforcement learning
  - Directly parametrize the policy by parameter $\boldsymbol{\theta}$

$$\pi(a|s,\boldsymbol{\theta}) = \mathbb{P}[A_t = a | S_t = s]$$

Exploration!

- Advantages:
  - Better convergence, stochastic policies $\pi(a|s,\boldsymbol{\theta}) \in (0,1)$
  - Effective in high-dimensional or continuous action spaces
- Disadvantages: inefficient and of high variance (can be mitigated)

- **Theorem (policy gradient):** For any differentiable policy $\pi_{\boldsymbol{\theta}}(a|s)$ and $J(\boldsymbol{\theta})$, policy gradient does not involve derivative of state distribution:

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi[q_\pi(S_t,A_t)\nabla \log \pi(A_t|S_t,\boldsymbol{\theta})]$$

f) REINFORCE

- **REINFORCE:** Monte Carlo Policy Gradient
- **Input:**
  - A differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
  - Algorithm parameter: step size $\alpha > 0$
- **Initialize:** policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g. to $\mathbf{0}$)
- **Repeat** (for each episode)
  - Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \sim \pi(\cdot|\cdot,\boldsymbol{\theta})$
  - **Repeat** (for each step of the episode $t = 0,1,\ldots,T-1$)
    - $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \boxed{\nabla \log \pi(A_t|S_t,\boldsymbol{\theta})}$ Eligibility Vector

- **REINFORCE with Baseline:** Monte Carlo Policy Gradient
- **Input:** policy parameterization $\pi(a|s,\boldsymbol{\theta})$, state-value function $\hat{v}(s,\boldsymbol{w})$
  - Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\boldsymbol{w}} > 0$
- **Initialize:** policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, state-value weight $\boldsymbol{w} \in \mathbb{R}^d$
- **Repeat** (for each episode)
  - Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \sim \pi(\cdot|\cdot,\boldsymbol{\theta})$
  - **Repeat** (for each step of the episode $t = 0,1,\ldots,T-1$)
    - $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    - $\delta \leftarrow G - \hat{v}(S_t,\boldsymbol{w})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta\nabla\hat{v}(S_t,\boldsymbol{w})$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\gamma^t\delta\nabla \log \pi(A_t|S_t,\boldsymbol{\theta})$

g) Actor-Critic

- **QAC:** Simple actor-critic algorithm based on action-value critic
- **Input:** policy parameterization $\pi(a|s,\boldsymbol{\theta})$, action-value critic $\hat{q}(s,a,\boldsymbol{w})$
  - Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\boldsymbol{w}} > 0$
- **Initialize:** policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, action-value weight $\boldsymbol{w} \in \mathbb{R}^d$
- **Repeat** (for each episode)
  - Initialize S
  - **Repeat** (for each step of the episode while S is not terminal)
    - $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    - Take action A, observe S', R, sample A'
    - $\delta \leftarrow R + \gamma\hat{q}(S',A',\boldsymbol{w}) - \hat{q}(S,A,\boldsymbol{w})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta\nabla\hat{q}(S,A,\boldsymbol{w})$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\hat{q}(S,A,\boldsymbol{w})\nabla \log \pi(A|S,\boldsymbol{\theta})$
    - $S \leftarrow S', A \leftarrow A'$

- **Input:** policy parameterization $\pi(a|s,\boldsymbol{\theta})$, action-value critic $\hat{q}(s,a,\boldsymbol{w})$
  - Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\boldsymbol{w}} > 0$
- **Initialize:** policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, action-value weight $\boldsymbol{w} \in \mathbb{R}^d$
- **Repeat** (for each episode)
  - Initialize S
  - $\boldsymbol{e}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}, \boldsymbol{e}^{\boldsymbol{w}} \leftarrow \mathbf{0}$ (eligibility traces)
  - **Repeat** (for each step of the episode while S is not terminal)
    - $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    - Take action A, observe S', R, sample A'
    - $\delta \leftarrow R + \gamma\hat{q}(S',A',\boldsymbol{w}) - \hat{q}(S,A,\boldsymbol{w})$
    - $\boldsymbol{e}^{\boldsymbol{w}} \leftarrow \gamma\lambda\boldsymbol{e}^{\boldsymbol{w}} + \nabla\hat{q}(S,A,\boldsymbol{w})$ TD(λ)
    - $\boldsymbol{e}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda\boldsymbol{e}^{\boldsymbol{\theta}} + \nabla \log \pi(A|S,\boldsymbol{\theta})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta\boldsymbol{e}^{\boldsymbol{w}}$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\hat{q}(S,A,\boldsymbol{w})\boldsymbol{e}^{\boldsymbol{\theta}}$ Policy Gradient
    - $S \leftarrow S', A \leftarrow A'$

- Policy gradient methods can follow any **unbiased** ascent direction
  - **Advantage function:** $A_\pi(S_t,A_t) = q_\pi(S_t,A_t) - v_\pi(S_t)$
  - TD error: $\delta = R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(S_t)$

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t,A_t)\, G_t] && \text{REINFORCE} \\
&\propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t,A_t)\, \hat{q}(S_t,A_t,\boldsymbol{w})] && \text{Actor-Critic} \\
&\propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t,A_t)\, \hat{A}(S_t,A_t,\boldsymbol{w})] && \text{Advantage Actor-Critic} \\
&\propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t,A_t)\, \delta] && \text{TD Actor-Critic} \\
&\propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t,A_t)\, \delta e] && \text{TD(}\lambda\text{) Actor-Critic}
\end{aligned}$$

  - Use policy evaluation to update the **critic** (MC or TD prediction)