ML Lec 11

- Dimensionality Reduction

  : learn a mapping $f: \underset{\in \mathbb{R}^d}{X} \to \underset{\in \mathbb{R}^k}{Z}$ from the input feature space to a transformed space, where $k \ll d$

  (the mapping should preserve certain intrinsic structure of the given data)

  1. Why reducing feature dimension

     ⇒ computation time, space requirement ↓

     ⇒ curse of dimensionality

     (feature dimension ↑ ⇒ # of data for nice generalization ↑)

     ⇒ manifold assumption

     (intrinsic dimensionality could be smaller)

     ⇒ visualization for better understanding and analysis

2. Linear Dimensionality Reduction

: Learn a linear transformation $z = W^T x$, where $x \in \mathbb{R}^d$, $z \in \mathbb{R}^k$, $W \in \mathbb{R}^{d \times k}$ and $k \ll d$

✯ Principal Component Analysis (PCA)

: orthogonal projection of data onto lower dimensional linear space

(basis transformation that transforms the data to a new coordinate system)

→ maximizes the variance of the projected data

→ minimizes the reconstruction error

① Standardize the data to have mean 0 and variance 1

② Find the first principal component α_1

$$\Rightarrow \max_{\alpha_1} \; Var(\alpha_1^T x) = \alpha_1^T \underbrace{\left(\frac{1}{n} \sum_{i=1}^{n} x_i x_i^T \right)}_{\text{sample covariance matrix } (= \frac{1}{n} X X^T)} \alpha_1 = \alpha_1^T \Sigma \alpha_1$$

$$\text{s.t. } \alpha_1^T \alpha_1 = 1$$

$$\Rightarrow \max_{\alpha_1} \min_{\lambda_1} \alpha_1^T \Sigma \alpha_1 - \lambda_1 (\alpha_1^T \alpha_1 - 1)$$

$$\rightarrow \frac{\partial}{\partial \lambda_1} \left(\alpha_1^T \Sigma \alpha_1 - \lambda_1 (\alpha_1^T \alpha_1 - 1) \right) = \alpha_1^T \alpha_1 - 1 = 0$$

$$\rightarrow \frac{\partial}{\partial \alpha_1} \left(\alpha_1^T \Sigma \alpha_1 - \lambda_1 (\alpha_1^T \alpha_1 - 1) \right) = 2 (\Sigma \alpha_1 - \lambda_1 \alpha_1) = 0$$

(α_1 : eigenvector of Σ, λ_1 : eigenvalue of Σ)

$$\Rightarrow \max \; \lambda_1 \; (= Var(\alpha_1^T x))$$

$$\text{s.t. } \Sigma \alpha_1 = \lambda_1 \alpha_1, \quad \alpha_1^T \alpha_1 = 1$$

$\therefore$ α_1 : eigenvector that corresponds to the largest eigenvalue λ_1

③ Find the second principal component α_2

$\Rightarrow \max_{\alpha_2} \; Var(\alpha_2^T x) = \alpha_2^T \Sigma \alpha_2$

s.t. $\alpha_2^T \alpha_2 = 1$

$Cov(\alpha_1^T x, \alpha_2^T x) = \alpha_2^T \Sigma \alpha_1 = \lambda_1 \alpha_1^T \alpha_2 = 0$

$\Rightarrow \max_{\alpha_2} \min_{\lambda_2, \mu} \; \alpha_2^T \Sigma \alpha_2 - \lambda_2(\alpha_2^T \alpha_2 - 1) - \mu \alpha_1^T \alpha_2$

$\longrightarrow \frac{\partial}{\partial \lambda_2}(\alpha_2^T \Sigma \alpha_2 - \lambda_2(\alpha_2^T \alpha_2 - 1) - \mu \alpha_1^T \alpha_2) = \alpha_2^T \alpha_2 - 1 = 0$

$\longrightarrow \frac{\partial}{\partial \mu}(\alpha_2^T \Sigma \alpha_2 - \lambda_2(\alpha_2^T \alpha_2 - 1) - \mu \alpha_1^T \alpha_2) = \alpha_1^T \alpha_2 = 0$

$\longrightarrow \frac{\partial}{\partial \alpha_2}(\alpha_2^T \Sigma \alpha_2 - \lambda_2(\alpha_2^T \alpha_2 - 1) - \mu \alpha_1^T \alpha_2) = 2(\Sigma \alpha_2 - \lambda_2 \alpha_2) - \mu \alpha_1 = 0$

(multiplying α_1) $\Rightarrow 2(\alpha_1^T \Sigma \alpha_2 - \lambda_2 \alpha_1^T \alpha_2) - \mu \alpha_1^T \alpha_1 = 2(\lambda_1 \alpha_1^T \alpha_2 - \lambda_2 \alpha_1^T \alpha_2) - \mu$

$\Rightarrow \mu = 0$ so that $\Sigma \alpha_2 = \lambda_2 \alpha_2$

(α_2 : eigenvector of Σ, λ_2 : eigenvalue of Σ)

$\Rightarrow \max \; \lambda_2 \; (= Var(\alpha_2^T x))$

s.t. $\Sigma \alpha_2 = \lambda_2 \alpha_2$, $\alpha_2^T \alpha_2 = 1$, $\alpha_1^T \alpha_2 = 0$

$\therefore$ α_2 : eigenvector that corresponds to the largest (possible) eigenvalue λ_2

★ Eigenvectors with distinct eigenvalues are orthogonal in symmetric matrix

★ Symmetric matrix is diagonalizable $\longrightarrow$ algebraic multiplicity = geometric multiplicity

★ Gram-Schmidt process enables to find orthonormal basis

★ Magnitude of eigenvalues indicates fraction of variances captured

④ Repeat the process for finding d different principal components

⑤ Choose k components using cross-validation

* Simple way

⇒ d-dimensional projection with PCA

: $Z = XA$ where $\begin{cases} X \in \mathbb{R}^{n \times d} \text{ is a data matrix,} \\ A \in \mathbb{R}^{d \times d} \text{ is a collection of eigenvector of } \Sigma \end{cases}$

⇒ k-dimensional projection with PCA

: $Z' = XA'$ where $A' \in \mathbb{R}^{d \times k}$ so that only k eigenvectors are selected

* Eigenface

: Principle components of face images derived by PCA

⇒ face image x can be converted into coordinates of the eigenfaces

$$x \longrightarrow \left((x-\bar{x}) \cdot v_1, (x-\bar{x}) \cdot v_2, \cdots, (x-\bar{x}) \cdot v_k\right)$$

⇒ a new face image can be generated by linear combination of basis faces

⇒ closest image can be found by Nearest Neighbor search on coefficients.

* Problems

⇒ assumes normality of the input space distribution.

⇒ fail if data lies on a complicated manifold.

⇒ Euclidian distance may not be a good measure between points in manifold

(⊕ geodesic distance : the distance of two nodes on a graph)

3. Non-Linear Dimensionality Reduction

: Learn a low-dimensional non-linear manifolds of given high-dimensional data.

☆ Locally Linear Embedding (LLE)

① Select neighbors

② Learn linear weight W that minimizes the reconstruction error by the neighbors given the data points. X

$$\Rightarrow \min_{W} \sum_{i=1}^{n} \left\| x_i - \sum_{j} w_{ij} x_j \right\|_2^2$$

③ Learn low-dimensional vectors y that minimizes the reconstruction error by the neighbors given the reconstruction weight W

$$\Rightarrow \min_{Y} \sum_{i=1}^{n} \left\| Y_i - \sum_{j} w_{ij} Y_j \right\|_2^2$$

ML Lec 12

- Clustering

  : process of organizing unlabeled data points into clusters where similar data items are collected

  1. Usage

     ⇒ similarity / dissimilarity analysis

     ⇒ dimensionality reduction

     ⇒ image segmentation

     ⇒ neural network quantization

  2. Requirement

     ① Algorithm to perform clustering

     ② Number of clusters

        ⇒ determined by the cross-validation and criterion function

     ③ Similarity measure ($S(x_i, x_j)$)

        : large if x_i and x_j is similar (e.g. cosine similarity)

     ④ Dissimilarity measure ($D(x_i, x_j)$)

        : small if x_i and x_j are similar (e.g. euclidian distance)

        → Euclidian distance : $\sqrt{\sum_{k=1}^{d} (x_i^k - x_j^k)^2}$

        → Manhatten distance : $\sum_{k=1}^{d} |x_i^k - x_j^k|$

        → Minkowski distance : $\left(\sum_{k=1}^{d} (x_i^k - x_j^k)^p \right)^{\frac{1}{p}}$

⑤ Criterion function to evaluate the generated clusters

→ intra-cluster cohesion (compactness)

: how near the data points in a cluster are to the cluster centroid.

e.g. Sum of squared errors (SSE) = $\sum_{j=1}^{k} \sum_{x \in D_j} d(x, \mu_j)^2$

D_j: jth cluster; d: Euclidian distance; μ_j: centroid of cluster j

→ inter-cluster separation (isolation)

: how far the cluster centroids are from each other

e.g. expert judgement

→ purity

: ratio of correctly assigned data points when the label of the most frequent class is assigned to each data point in clusters

⇒ $\text{purity}(D, C) = \frac{1}{N} \sum_{j=1}^{k} \max_{i} |D_j \cap C_i|$

D_j: jth cluster; C_i: ith class

☆ k-means clustering

: each data point belongs to one of the k clusters with the nearest mean

$\Rightarrow \min_{S} \sum_{j=1}^{k} \sum_{x \in S_j} \| x - \mu_j \|^2$

① Assign each data point to the closest cluster center

② Recalculate centers as the mean of the points in a cluster.

(most frequent value is used for categorical data)

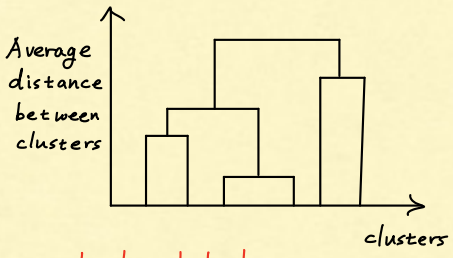
③ Repeat until convergence criterion is met

- no (or minimum) reassignment of data points to different clusters
- no (or minimum) change of centroids
- minimum decrease in the sum of squared errors

* strength
  - simple : easy to understand and implement
  - fast : $O(tkn)$ time complexity
  - general : work well with multiple distance measure

* weakness
  - applicable only when mean is defined
  - k must be specified in advance
  - sensitive to outliers, initial centroids
  - can only create spherical clusters

☆ hierarchical clustering



⇒ dendogram is a visualization tool

① Divisive (top-down) clustering → slower, look global structure

: start with all data points in one cluster, then split into child clusters recursively until one singleton clusters of individual data points remain.

⇒ can use any flat algorithms that produce a fixed number of clusters (e.g. k-means)

② Agglomerative (bottom-up) clustering → faster, look local structure

: start with each example in singleton cluster, then find 2 nearest clusters and merge until there is only one cluster

⇒ four common ways to measure cluster distance

1) Minimum distance : $\min_{x \in D_i, y \in D_j} \|x-y\|$

→ generate minimum spanning tree

→ encourage elongated clusters

→ very sensitive to noise

2) Maximum distance : $\max_{x \in D_i, y \in D_j} \|x-y\|$

→ encourage compact clusters

→ perform bad for elongated clusters

3) Average distance : $\frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \|x-y\|$

4) Mean distance : $\|\mu_i - \mu_j\|$

☆ Spectral clustering

: make use of spectrum of the similarity matrix (Graph Laplacian) to reduce dimensionality before clustering (e.g. k-means)

☆ Max-margin clustering

: find maximum margin hyperplanes through data

☆ Gaussian Mixture Model

: assumes that data points are generated from a mixture of finite number of Gaussians with unknown parameters

⇒ EM (expectation and maximization) algorithm can be utilized

ML Lec 13

- Reinforcement Learning

: learning which actions to take at each state to maximize reward in the long-run

1. Notations

⇒ State (S_t) : representation of the environment

(summarizes the past in a compact way)

⇒ Action (A_t) : action that an agent take which leads to a new state

⇒ Reward (R_t) : numerical value received by performing an action A_t at state S_t

→ Total reward (G_t) $= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

(discounting is used to determine the present value of the future rewards)

($\gamma \to 0$: myopic , $\gamma \to 1$: far-sighted)

⇒ Policy (π_t) : mapping from states to probabilities of selecting each possible action

2. Characteristics

⇒ there is no supervisor, but a reward signal

⇒ feedback is delayed and not instanteous

⇒ time matters (sequential, not i.i.d.)

3. Markov Decision Process (MDP)

: learning task for reinforcement learning that satisfies Markov Property, which consists of the following components

- A set of states S
- A set of actions A
- A transition model $T(s,a,s')$ → known : model-based approach.
- A reward function $R(s,a,s')$ → unknown : model-free approach

⇒ when states or actions are finite ⟶ "finite MDP"

⇒ agent does not know T and R which are estimated by experience

⇒ goal is to find a policy $\pi(s)$ that maximizes total reward.

4. Value functions

⇒ for finite MDP, policies can be partially ordered.

→ $\pi' \succ \pi$ if and only if $v_{\pi'}(s) \geq v_{\pi}(s)$, $\forall s \in S$

⇒ there is always one or more policies that are better than or equal to others which share the same state-value ($v_{\pi^*}(s)$) and action-value function ($q_{\pi^*}(s,a)$)

① State-value function for policy π ($v_\pi(s)$)

: expected return starting from the state s, then following π

→ $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$

② Action-value function for policy π ($q_\pi(s,a)$)

: expected return starting from the state s taking action a, then following π

→ $q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$

☆ Bellman expectation equation

: a relationship between the value of a state and the value of its successor states

⇒ $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$

$= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s'\right]\right]$

$= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right]$

⇒ $q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$

$= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s'\right]\right]$

$= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_\pi(s')\right]$

☆ Bellman optimality equation

$$\Rightarrow V_{\pi^*}(s) = \max_a q_{\pi^*}(s,a)$$

$$= \max_a \mathbb{E}_{\pi^*}\left[G_t \mid S_t = s, A_t = a \right] = \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

$$= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a \right]$$

$$= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \cdot V_{\pi^*}(S_{t+1}) \mid S_t = s, A_t = a \right]$$

$$= \max_a \sum_{s'} p(s' \mid s,a) \left[r(s,a,s') + \gamma \cdot V_{\pi^*}(s') \right]$$

(any policy that is greedy w.r.t. V_{π^*} is an optimal policy)

$$\Rightarrow q_{\pi^*}(s,a) = \mathbb{E}_{\pi^*}\left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

$$= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a \right]$$

$$= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \cdot V_{\pi^*}(S_{t+1}) \mid S_t = s, A_t = a \right]$$

$$= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \cdot \max_{a'} q_{\pi^*}(S_{t+1}, a') \mid S_t = s, A_t = a \right]$$

$$= \sum_{s'} p(s' \mid s,a) \left[r(s,a,s') + \gamma \max_{a'} q_{\pi^*}(s', a') \right]$$

(any action that is greedy w.r.t. q_{π^*} is an optimal action)

(the agent does not have to do one-step ahead search.)

5. Approaches in reinforcement learning

① Model-based approach

⇒ build a model of the environment and plan using the model

② Model-free approach

⇒ Value-based approach

→ estimate the optimal value functions $V_{\pi^*}(s)$, $q_{\pi^*}(s,a)$

⇒ policy-based approach

→ search directly for the optimal policy π^*

ML Lec 14

- Model-based Reinforcement Learning

1. Dynamic programming (DP) → use bootstrap → Does not scale well to large problems.

: technique to solve problems with optimal substructure and overlapping subproblems

- optimal substructure : optimal solution can be decomposed into solutions of subproblems.
- overlapping subproblems : subproblems recur many times where solutions are stored and reused

⇒ Assuming full knowledge of MDP and deterministic π, it can be resolved through DP.

⇒ it considers all successor states (full back-up)

☆ Policy iteration ($\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$)

: iteratively update the optimal policy using DP

① Initialize π randomly

② Repeat until convergence

→ $V_\pi(s) = V_{\pi'}(s)$

→ Repeat until convergence (expensive)

(policy evaluation) ⇒ $V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_\pi(s')\right]$ (full-back-up)

(Bellman expectation equation)

→ $\pi'(s) = \arg\max_a q_\pi(s,a)$

(policy improvement) $= \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_{\pi^*}(s')\right]$ (given)

(+) Converge in a finite number of iteration (often small in practice)

(−) Each iteration requires a full policy evaluation, which is expensive

☆ Value iteration

: iteratively search for optimal value function using DP

① Initialize $V(s) = 0, \forall s$

② Repeat until convergence

(policy evaluation) $\longrightarrow V(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \cdot V(s')\right]$ (full-back-up)

(Bellman optimality equation)

③ After obtaining the optimal value function, find the optimal policy

(policy improvement) $\longrightarrow \pi^*(s) = \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_*(s')\right]$ ($p(s'|s,a)$: given)

(+) Each iteration is computationally efficient

(−) Convergence is only symptotic

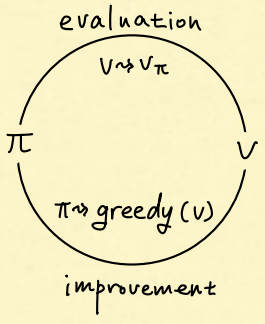
※ With finite number of states, both converge to optimal solution in polynomial time

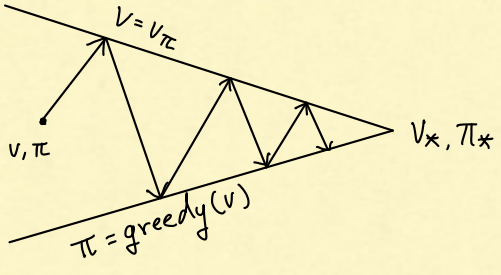
* Generalized Policy iteration (GPI)

: allows the policy evaluation and the policy improvement to interact

⇒ every reinforcement learning can be described as GPI

⇒ if both the evaluation and improvement stabilize, then the value function and the policy must be optimal.





- Value-based Reinforcement Learning

  : optimize the state-value and action-value functions

  (policy is generated directly from these value functions)

  1. Monte Carlo method (on-policy : learn π from episodes of π)

     ⇒ learns from complete sample returns

     → only defined for episodic tasks that have ends

     ⇒ learns directly from experience

     → online : optimality is attainable even without model

     → simulated : no need for a full model

     ✡ Monte Carlo policy iteration ($\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*}$)

     : estimate $V_\pi(s)$ for model-based MC and $q_\pi(s,a)$ for model-free MC of the current behavior policy π, then improve π

     ① Initialize $\pi(s)$, $v(s)$ (or $q(s,a)$), $r(s)$ for all s (empty list)

     ② Repeat until convergence

     → generate an episode using π

     → for each s appearing in the episode

     ⇒ $G \leftarrow$ return following the occurrence of s (followed by a)

     ⇒ Append G to $r(s)$

     ⇒ $v(s)$ (or $q(s,a)$) $\leftarrow$ average ($r(s)$)

     ($\approx v(s) \leftarrow v(s) + \alpha[G - v(s)]$)

     ($\approx q(s,a) \leftarrow q(s,a) + \alpha[G - q(s,a)]$)

     - every-visit MC : consider every time when s is visited
     - first-visit MC : consider the first time when s is visited

→ ε-greedy policy improvement

: simplist idea for ensuring continual exploration

(all m actions have non-zero probability)

$$\Rightarrow \pi(a|s) = \begin{cases} 1-\varepsilon+\frac{\varepsilon}{m} & \text{if } a = \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V(s')\right] \\ & \quad \text{(or } a = \arg\max_a q(s,a)\text{)} \\ \frac{\varepsilon}{m} & \text{otherwise} \end{cases}$$

($p(s'|s,a)$ and $r(s,a,s')$: given)

$\Rightarrow \varepsilon \uparrow \Rightarrow$ more exploration , $\varepsilon \downarrow \Rightarrow$ more exploitation

※ Theorem. ($\varepsilon' < \varepsilon$) → ε would converge to 0 in the end

: For any ε-greedy π, the ε'-greedy policy π' with respect to q_π is an improvement

pf)
$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s)\, q_\pi(s,a) \\ &= \frac{\varepsilon'}{m}\sum_a q_\pi(s,a) + (1-\varepsilon')\cdot \max_a q_\pi(s,a) \\ &\geq \frac{\varepsilon'}{m}\sum_a q_\pi(s,a) + (1-\varepsilon')\cdot \sum_a \frac{\pi(a|s) - \frac{\varepsilon'}{m}}{1-\varepsilon'} q_\pi(s,a) \\ &= \sum_a \pi(a|s)\, q_\pi(s,a) \\ &= V_\pi(s) \end{aligned}$$

($\frac{\pi(a|s) - \frac{\varepsilon'}{m}}{1-\varepsilon'}$: sum to 1)

(+) MC uses return G that is an unbiased estimate of $V_\pi(s)$

(−) MC uses return G that has high variance

→ depends on many actions, transitions, rewards

(−) MC have to wait until the episodes end, which could be very long

2. Temporal difference

⇒ similar to MC, TD does not need a model

→ learn directly from experience

⇒ similar to DP, TD bootstraps

→ learn a guess from a guess

★ Simple Temporal difference (On-policy)

: estimate $V_\pi(s)$ for the current behavior policy π, then improve policy π

① Initialize $V(s)$ for all s and $V(\text{terminal state}) = 0$

② Repeat until convergence

→ initialize s

→ repeat until s is terminal

⇒ choose a from s using policy π

⇒ take action a and observe reward r and next state s'

⇒ $V(s) \leftarrow V(s) + \alpha\,(r + \underbrace{\gamma V(s') - V(s)}_{\text{Temporal difference}})$

⇒ $s \leftarrow s'$

→ ε-greedy policy improvement

(+) TD can be fully incremental

→ learn before knowing the final outcome

→ work in incomplete, non-terminating environments

→ less memory, less peak compntation

→ converges faster than MC

(+) TD uses target $r+\gamma V(s')$ that has low variance

→ depends on one random action, transition, reward

(−) TD uses target $r+\gamma V(s')$ that is a biased estimator of $V_\pi(s)$

☆ SARSA (On-policy)

: estimate $q_\pi(s,a)$ for the current behavior policy π, then improve policy π

① Initialize $q(s,a)$ for all s and a and $q(\text{terminal state}, \cdot) = 0$

② Repeat until convergence

→ initialize s

→ choose a from s using policy derived from $q(s,a)$ (e.g. ε-greedy)

→ repeat until s is terminal

⇒ take action a and observe reward r and next state s'

⇒ choose a' from s' using policy derived from $q(s,a)$ (e.g. ε-greedy)

⇒ $q(s,a) \leftarrow q(s,a) + \alpha\left[r + \gamma\, q(s',a') - q(s,a)\right]$

(on-policy)

⇒ $s \leftarrow s'$, $a \leftarrow a'$

☆ Q-learning (Off-policy : learn π from episodes of π')

: directly approximate $q_{\pi^*}(s,a)$ independent of the policy being followed

① Initialize $q(s,a)$ for all s and a and $q(\text{terminal state}, \cdot) = 0$

② Repeat until convergence

→ initialize s

→ repeat until s is terminal

⇒ choose a from s using policy derived from $q(s,a)$ (e.g. ε-greedy)

⇒ take action a and observe reward r and next state s'

⇒ $q(s,a) \leftarrow q(s,a) + \alpha\left[r + \gamma \max_a q(s',a) - q(s,a)\right]$

(off-policy)

⇒ $s \leftarrow s'$

※ SARSA ⇒ learns longer but safe path

※ Q-Learning ⇒ learns shorter but non-safe path

3. Value function approximation

: represent $v(s)$ or $q(s,a)$ with a parameterized function instead of a table

⇒ no need to explicitly store or learn every value of $v(s)$ or $q(s,a)$

⇒ more compact representations that generalize across $v(s)$ or $q(s,a)$

ex) Linear functions (differentiable), decision trees, nearest neighbors, neural networks (differentiable)

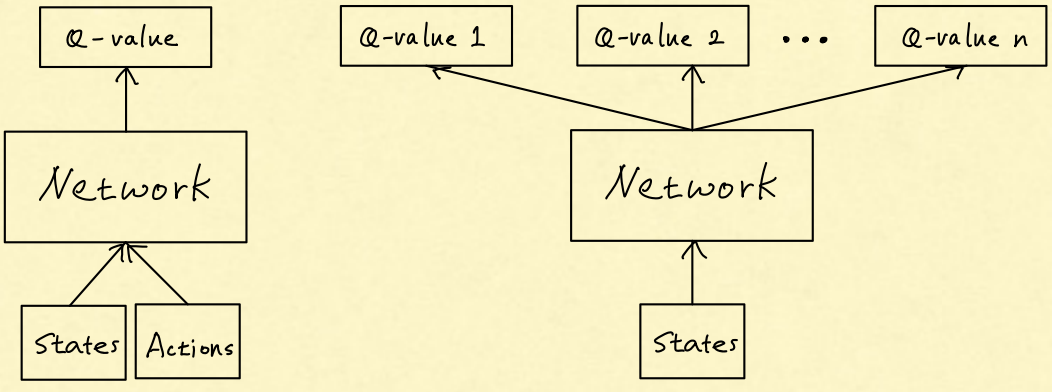
★ Deep Q-learning

: represent value functions by deep Q-network with weight w.

$\left(Q(s,a,w) = q_\pi(s,a)\right)$

⇒ loss function $\left(L(w)\right) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w)\right)^2\right]$

$$\longrightarrow \frac{\partial L(w)}{\partial w} = \mathbb{E}_{s,a,r,s' \sim D}\left[2\left(r + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w)\right)\frac{\partial Q(s,a,w)}{\partial w}\right]$$



⇒ experience replay → break correlations in the states

① store all experiences $\langle s, a, r, s' \rangle$ in a replay memory

② while training, use random minibatches from the replay memory instead of the recent most transition

⇒ freeze target Q-network → avoid oscillation

① compute q-learning target w.r.t. old, fixed parameters w^-

→ $r + \gamma \max_{a'} Q(s', a', w^-)$

② optimize MSE between Q-network and q-learning targets

→ $L(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$

→ periodically update the fixed parameter w^-.

⇒ clip rewards or normalize network adaptively → robust gradients

$[-1, +1]$

→ prevent $Q(s, a, w)$ from getting too large

→ ensures the gradient to be well conditioned

→ can use the same learning rate across multiple games

(cannot tell difference between small and large rewards)

ML Lec 15

- Policy-based Reinforcement Learning

: directly learn the policy by approximating it with a parameterized function.

($\pi_\theta(a|s) = P(a|s,\theta)$ → find the best parameter θ)

⇒ advantages

→ better convergence properties

→ effective in high-dimensional or continuous action space

→ can learn stochastic policies

⇒ disadvantages

→ typically converges to a local rather than global optimum

→ typically evaluating a policy is inefficient and has high variance

1. Policy objective functions

: measure the quality of policy π_θ

⇒ episodic environment

→ start value : $J_1(\theta) = V_{\pi_\theta}(s_1)$

⇒ continuing environment

→ average value : $J_V(\theta) = \sum_s d_{\pi_\theta}(s) V_{\pi_\theta}(s) := \mathbb{E}\left[V_{\pi_\theta}(s)\right]$

→ average reward per time-step : $J_R(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) r(s,a,\cdot) := \mathbb{E}\left[r(s,a,\cdot)\right]$

($d_{\pi_\theta}(s)$: stationary distribution ($\pi P = \pi$) of markov chain for π_θ)

$\pi_\theta \neq \pi$

2. Policy optimization

: find θ that maximizes $J(\theta)$

$\Rightarrow$ both gradient-based and non-gradient based methods can be used

(e.g. (stochastic) gradient ascent, conjugate gradient, quasi Newton, genetic algorithm, ...)

3. policy gradient algorithm

: searches for a local maximum in $J(\theta)$ by ascending the gradients w.r.t. θ

$\Rightarrow$ assume π_θ is differentiable almost everywhere

(e.g. linear function or neural network)

$\Rightarrow \Delta\theta = \alpha \cdot \nabla_\theta J(\theta) = \alpha \cdot \nabla \mathbb{E}\left[V_{\pi_\theta}(s)\right]$ or $\alpha \cdot \nabla \mathbb{E}\left[r(s,a,\cdot)\right]$

$\longrightarrow$ Monte Carlo sampling can be used

4. score function trick

: estimate the gradient with sampling without consideration of $d_{\pi_\theta}(s)$

$$\begin{aligned}
\Rightarrow \nabla_\theta \mathbb{E}\left[r(s,a,\cdot)\right] &= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s)\, r(s,a,\cdot) \\
&= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s)\, r(s,a,\cdot) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \cdot \underbrace{\nabla_\theta \log \pi_\theta(a|s)}_{\text{(score function)}}\, r(s,a,\cdot) \\
&= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s) \cdot r(s,a,\cdot)\right]
\end{aligned}$$

5. Policy gradient theorem

: For any differentiable policy $\pi_\theta(a|s)$ and any policy objective function J, (J_1, $\frac{1}{1-\gamma} J_r$, J_R)

the policy gradient is as follow.

$\longrightarrow \nabla_\theta \mathbb{E}\left[r(s,a,\cdot)\right] = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s)\, q_{\pi_\theta}(s,a)\right]$

☆ Monte Carlo policy gradient method

⇒ assume $J(\theta) = \mathbb{E}[r(s,a,\cdot)]$

→ cannot simply sample R_{t+1} and take gradient

($\because$ R_{t+1} is a scalar and does not depend on θ)

⇒ score function trick can be used

→ $\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t)$

(stochastic policy gradient update)

→ the probability of selecting A_t should go up if R_{t+1} is high

⇒ softmax policy

→ applicable when the action space is discrete and not too large

→ $\pi_\theta(a|s) = \dfrac{\exp(\theta^T \phi(s,a))}{\sum_{k=1}^{m} \exp(\theta^T \phi(s,a_k))}$

→ $\nabla_\theta \log \pi_\theta(a|s) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$

⇒ gaussian policy

→ applicable when the action space is continuous

→ $\pi(a|s) = \mathcal{N}(\mu(s), \sigma^2)$ where $\mu(s) = \phi(s)^T\theta$ (linear function)

(variance can be either fixed or further parameterized)

→ $\nabla_\theta \log \pi_\theta(a|s) = \dfrac{(a-\mu(s))\phi(s)}{\sigma^2}$

⇒ REINFORCE algorithm

: estimate $q_{\pi_\theta}(s,a)$ with the return G_t

(G_t is an unbiased estimator of $v_\pi(s)$)

① Initialize policy parameter θ

② Repeat until convergence

→ generate an episode $\{S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T\} \sim \pi_\theta(\cdot|\cdot)$

→ repeat $t = 0, \cdots, T-1$

⇒ $G_t = \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$

⇒ $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(A_t|S_t)\, G_t$

(-) MC policy gradient uses return G_t that has high variance

☆ Actor-critic method

⇒ actor adjusts parameter θ of $\pi_\theta(a|s)$ by policy gradient algorithm

⇒ critic $Q(s,a,w)$ estimates $q_{\pi_\theta}(s,a)$ using an approximate policy evaluation algorithms (e.g. Monte Carlo method, Temporal difference methods)

→ linear function can be used : $Q(s,a,w) = w^T \phi(s,a)$

⇒ actor-critic algorithm

① Initialize parameter θ, w and S_0

② Repeat $t = 1, \cdots, T$

→ sample $R_t \sim r(s,a,\cdot)$ and $s' \sim p(s'|s,a)$

→ sample $a' \sim \pi_\theta(a'|s')$

(update critic) → $w \leftarrow w + \beta \left(R_t + \gamma Q(s',a',w) - Q(s,a,w)\right) \nabla_w Q(s,a,w)$

(update actor) → $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a|s) \cdot Q(s,a,w)$

→ $s \leftarrow s'$, $a \leftarrow a'$

* Reducing variance of policy gradient $\nabla_\theta J(\theta)$

① substract the baseline function $b(s)$ that only depends on s

→ state value function can be used : $b(s) = V_{\pi_\theta}(s)$

→ this does not change the expectation of $\nabla_\theta J(\theta)$

$$\Rightarrow \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s)\, b(s)\right] = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s)\, b(s)$$

$$= \sum_s d(s)\, b(s)\, \nabla_\theta \underbrace{\sum_a \pi_\theta(a|s)}_{(=1)}$$

$$= 0$$

② critic estimates the advantage function $A(s,a,w) = Q(s,a,w) - V(s,\xi)$

by estimating both $Q(s,a,w)$ and $V(s,\xi)$ through TD-learning

→ $w \leftarrow w + \beta_1 \left(R_t + \gamma Q(s',a',w) - Q(s,a,w)\right) \nabla_w Q(s,a,w)$

→ $\xi \leftarrow \xi + \beta_2 \left(R_t + \gamma V(s',\xi) - V(s,\xi)\right) \nabla_\xi V(s,\xi)$

③ actor updates θ with an unbiased estimate of $A(s,a)$

→ $\delta_{\pi_\theta} = R_t + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$: true TD error → unbiased

(in practice) ($\delta_v = R_t + \gamma V(s',\xi) - V(s,\xi)$: approximate TD error) → biased

→ $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a|s) \cdot \delta_{\pi_\theta}$

(in practice) ($\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) \cdot \delta_v$)

∴ Summary

① Stochastic gradient ascent for θ

⇒ Monte Carlo policy gradient

→ $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$

⇒ Temporal Difference (0) – Actor-critic policy gradient

→ $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) V(s, \xi)$

⇒ Q – Actor-critic policy gradient

→ $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) Q(s, a, w)$

⇒ Advantage – Actor-critic policy gradient

→ $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) A(s, a, w)$

② Value functions estimation ($V(s, \xi)$, $Q(s, a, w)$ or $A(s, a, w)$)

⇒ Monte Carlo method

→ $\Delta\xi = \beta (G_t - V(s_t, \xi)) \nabla_\xi V(s_t, \xi)$

⇒ Temporal Difference (0)

→ $\Delta\xi = \beta (R_t + \gamma V(s', \xi) - V_\xi(s_t, \xi)) \nabla_\xi V(s_t, \xi)$

⇒ SARSA

→ $\Delta w = \beta (R_t + \gamma Q(s', a', w) - Q(s_t, a_t, w)) \nabla_w Q(s_t, a_t, w)$

⇒ Q-learning

→ $\Delta w = \beta (R_t + \gamma \max_{a'} Q(s', a', w) - Q(s_t, a_t, w)) \nabla_w Q(s_t, a_t, w)$

ML Lec 16

- Introduction to Bayesian Machine Learning

1. Uncertainty in Machine Learning

⇒ inherent uncertainty (future forecast)

⇒ insufficient observation

⇒ data ambiguity

(conventionally, point estimate finds a single optimal set of parameters
However, given limited observations, it is difficult to estimate parameters)

2. Bayesian Machine Learning

: average over the uncertain variables and parameters rather than optimize. → more expensive

⇒ apply to both discriminative and generative learning

⇒ eliminate the arbitrary loss function and regularizers

⇒ facilitate the incorporation of prior knowledge

⇒ quantifies the uncertainty in hypothesis and prediction

⇒ assume that the hypotheses and predictions follow some distributions → transparent

⇒ find the most likely distributions given the data using the prior beliefs and Bayes' rule

→ Posterior : $p(\theta|D) = \frac{p(D|\theta) \cdot p(\theta)}{p(D)}$ → full Bayesian.

→ Evidence : $p(D) = \begin{cases} \sum_{\theta} p(D|\theta) \cdot p(\theta) & \text{(discrete } \theta\text{)} \\ \int_{\theta} p(D|\theta) \cdot p(\theta)\, d\theta & \text{(continuous } \theta\text{)} \end{cases}$ ⇒ often intractable to compute

→ Likelihood : $p(D|\theta)$

→ Prior : $p(\theta)$
- objective prior : noninformative prior that attempts to capture ignorance
- subjective prior : prior that capture our beliefs
- empirical prior : prior that is learned from the data

(Wrong prior → worse performance, vague prior would be better)

⇒ conjugate prior yields posterior that is the same family as the prior

(e.g. Beta prior, Bernoulli likelihood → Beta posterior)

→ $p(\theta) = \frac{\theta^{\beta_H - 1}(1-\theta)^{\beta_T - 1}}{B(\beta_H, \beta_T)} \sim \text{Beta}(\beta_H, \beta_T)$ where $B(\beta_H, \beta_T) = \frac{\Gamma(\beta_H)\Gamma(\beta_T)}{\Gamma(\beta_H + \beta_T)}$

→ $p(\theta|D) \propto p(D|\theta) \cdot p(\theta)$

$$\propto \theta^{\alpha_H}(1-\theta)^{\alpha_T} \cdot \theta^{\beta_H - 1}(1-\theta)^{\beta_T - 1} = \theta^{\alpha_H + \beta_H - 1} \cdot (1-\theta)^{\alpha_T + \beta_T - 1}$$

$\therefore p(\theta|D) \sim \text{Beta}(\alpha_H + \beta_H, \alpha_T + \beta_T)$

⇒ prediction forms a distribution, which accounts for uncertainty

→ $p(x^*|D) = \mathbb{E}_{p(\theta|D)}\left[p(x^*|\theta, D)\right] = \int p(x^*|\theta, D)\, p(\theta|D)\, d\theta$

(Exact posterior inference is difficult due to the intractable evidence term)

→ approximate posterior inference can be used

(This can resolve the scaling issues to large datasets)

① Monte-Carlo-Markov-Chain (MCMC) sampling

: generates samples from Markov chain using algorithms that make it converge to the posterior distribution

② Variational Inference

: find parameters for the parametric distribution that fits the posterior distribution

⇒ Maximum A Posteriori (MAP) estimation

: estimates the most probable parameters θ given the data

→ regularization method ⇒ not restricted to Bayesian methods

→ $\hat{\theta}_{MAP} = \arg\max_\theta P(\theta|D) = \arg\max_\theta \frac{P(D|\theta) \cdot P(\theta)}{P(D)} = \arg\max_\theta P(D|\theta)\, p(\theta)$

($\hat{\theta}_{MLE} = \arg\max_\theta P(D|\theta)$)

⇒ PAC-Bayes

: derive generalization error bound to Bayesian methods for theoretical guarantees

(convergence, consistency)

☆ Discrete variables ⟶ $P(X=x \mid \theta) = \theta^x (1-\theta)^{1-x}$, $x \in \{0,1\}$

⇒ data points are i.i.d. $D = \{x_i \mid i = 1, 2, \cdots, n\}$

① Maximum Likelihood Estimation (MLE)

⇒ Likelihood : $P(D|\theta) = \prod_i P(x_i|\theta) = \prod_i \theta^{x_i}(1-\theta)^{1-x_i} = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$ $(\alpha_H + \alpha_T = n)$

⇒ $\hat{\theta}_{MLE} = \arg\max_\theta \log P(D|\theta) = \arg\max_\theta \alpha_H \log\theta + \alpha_T \log(1-\theta)$

$= \dfrac{\alpha_H}{\alpha_H + \alpha_T}$

② Maximum A Posteriori (MAP) estimation

⇒ Prior : $P(\theta) = Beta(\beta_H, \beta_T)$

⇒ Posterior : $p(\theta|D) = \dfrac{\theta^{\alpha_H+\beta_H-1}(1-\theta)^{\alpha_T+\beta_T-1}}{B(\alpha_H+\beta_H, \alpha_T+\beta_T)}$

⇒ $\hat{\theta}_{MAP} = \arg\max_\theta \log P(\theta|D) = \arg\max_\theta (\alpha_H+\beta_H-1)\log\theta + (\alpha_T+\beta_T-1)\log(1-\theta)$

$= \dfrac{\alpha_H+\beta_H-1}{\alpha_H+\beta_H-1+\alpha_T+\beta_T-1}$

⟶ as the number of data points increases, prior is negligible

☆ Continuous variables $\longrightarrow P(X=x\mid\theta)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\quad x\in\mathbb{R}$

⇒ affine transformation of gaussian is another gaussian

$\longrightarrow X\sim N(\mu,\sigma^2)\Rightarrow Y=aX+b\sim N(a\mu+b,a^2\sigma^2)$

⇒ sum of gaussians is another gaussian

$\longrightarrow X\sim N(\mu_X,\sigma_X^2),\ Y\sim N(\mu_Y,\sigma_Y^2)\Rightarrow X+Y\sim N(\mu_X+\mu_Y,\ \sigma_X^2+\sigma_Y^2+2Cov(X,Y))$

⇒ data points are i.i.d. $D=\{x_i\mid i=1,2,\cdots,n\}$

① Maximum Likelihood Estimation (MLE)

⇒ Likelihood : $P(D\mid\mu,\sigma)=\prod_i P(x_i\mid\mu,\sigma)=\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}}\exp\left(-\sum_{i=1}^n\frac{(x_i-\mu)^2}{2\sigma^2}\right)$

⇒ $\hat{\mu}_{MLE}=\arg\max_\mu \log P(D\mid\mu,\sigma)=\arg\max_\mu\left(-\frac{n}{2}\log 2\pi\sigma^2-\sum_{i=1}^n\frac{(x_i-\mu)^2}{2\sigma^2}\right)$

$\longrightarrow \sum_{i=1}^n\frac{x_i-\hat{\mu}_{MLE}}{\hat{\sigma}^2}=0$

$\longrightarrow \hat{\mu}_{MLE}=\frac{1}{n}\sum_{i=1}^n x_i$

⇒ $\hat{\sigma}_{MLE}=\arg\max_\sigma \log P(D\mid\mu,\sigma)=\arg\max_\sigma\left(-\frac{n}{2}\log 2\pi\sigma^2-\sum_{i=1}^n\frac{(x_i-\mu)^2}{2\sigma^2}\right)$

$\longrightarrow -\frac{n}{\hat{\sigma}_{MLE}}+\frac{1}{\hat{\sigma}_{MLE}^3}\sum_{i=1}^n(x_i-\hat{\mu})^2=0$

$\longrightarrow \hat{\sigma}_{MLE}^2=\frac{1}{n}\sum_{i=1}^n(x_i-\hat{\mu})^2$

$\Rightarrow \mathbb{E}\left[\hat{\sigma}_{MLE}^2\right]=\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n(x_i^2-2x_i\hat{\mu}+\hat{\mu}^2)\right]=\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n x_i^2-\hat{\mu}^2\right]$

$=\sigma^2+\mu^2-\left(\frac{\sigma^2}{n}+\mu^2\right)=\frac{n-1}{n}\sigma^2\longrightarrow$ biased

$\Rightarrow \hat{\sigma}^2_{unbiased}=\frac{1}{n-1}\sum_{i=1}^n(x_i-\hat{\mu})^2$

② Maximum A Posteriori (MAP) estimation

⇒ Prior : $\begin{cases} P(\mu)=N(\eta,\lambda) \\ p(\sigma^2)=\text{Wishart distribution}\end{cases}$

ML Lec 17

- Naïve Bayes

1. Bayesian classification

: given an input with features $X_1, X_2, \cdots, X_D$, predict its label Y

$\Rightarrow y = \arg\max_y P(y|x_1, x_2, \cdots, x_D) = \frac{P(x_1, x_2, \cdots, x_D|y) \cdot P(y)}{P(x_1, x_2, \cdots, x_D)}$

$\longrightarrow$ assume features are generated by the label

$\longrightarrow$ $2^D - 1$ parameters are needed to specify $P(X_1, X_2, \cdots, X_D|Y)$ for certain Y

2. Naïve Bayes Classifier

: bayesian classifier based on conditional independence assumption

* X is conditionally independent on Y given Z

$\Leftrightarrow \forall i, j, k,\ p(X = x_i | Y = y_j, Z = z_k) = p(X = x_i | Z = z_k)$

ex) thunder is independent of rain given lightening

* Naïve Bayes assumption

: features are independent given the label

$\Rightarrow p(X_1, X_2, \cdots, X_D|Y) = \prod_d p(X_d|Y)$

$\Rightarrow$ D parameters are needed to specify $P(X_1, X_2, \cdots, X_D|Y)$ for certain Y

* Decision rule

$\Rightarrow y = \arg\max_Y p(x_1, x_2, \cdots, x_D|Y) \cdot p(Y) = \arg\max_Y \prod_d p(x_d|Y)\, p(Y)$

$\longrightarrow$ both $p(x_d|Y)$ and $p(Y)$ can be obtained from the training set

☆ Maximum likelihood estimation (MLE)

① $X \sim \text{Multinomial}(\theta) \longrightarrow p(X=j|\theta) = \theta_j$ where $\sum_{j=1}^{J} \theta_j = 1$

$\Rightarrow$ Likelihood: $p(D|\theta) = \prod_{i=1}^{n} p(X_i|\theta) = \theta_1^{\alpha_1}\theta_2^{\alpha_2}\cdots\theta_J^{\alpha_J}$ $\quad (\alpha_1+\alpha_2+\cdots+\alpha_J = n)$

$\Rightarrow \hat{\theta} = \arg\max_{\theta} \log P(D|\theta) = \alpha_1 \log\theta_1 + \cdots + \alpha_J \log\theta_J$ s.t. $\sum_{j=1}^{J}\theta_j = 1$

$= \arg\max_{\theta}\min_{\lambda} \alpha_1\log\theta_1 + \cdots + \alpha_J\log\theta_J - \lambda\left(\sum_{j=1}^{J}\theta_j - 1\right)$

$\longrightarrow \frac{\alpha_j}{\hat{\theta}_j} - \lambda = 0 \quad \forall j$, $\sum_{j=1}^{J}\hat{\theta}_j = 1$

$\longrightarrow \sum_{j=1}^{J}\alpha_j = \lambda \quad \forall j$

$\longrightarrow \hat{\theta}_j = \frac{\alpha_j}{\sum_{j=1}^{J}\alpha_j} = \frac{\#D(X=i)}{\#D}$

② $X_d|y_k \sim \text{Multinomial}(\theta_{dk}) \longrightarrow p(X_d=j|y_k,\theta) = \theta_{djk}$ where $\sum_{j=1}^{J}\theta_{djk} = 1$

$\Rightarrow$ Likelihood: $p(D(y_k)|\theta) = \prod_{i=1}^{n} p(X_{di}|y_{ki},\theta) = \theta_{d1k}^{\alpha_1}\theta_{d2k}^{\alpha_2}\cdots\theta_{dJk}^{\alpha_J}$ $\quad (\alpha_1+\alpha_2+\cdots+\alpha_J = \#D(y_k))$

$\Rightarrow \hat{\theta}_{dk} = \arg\max_{\theta} \log P(D(y_k)|\theta) = \alpha_1\log\theta_{d1k} + \cdots + \alpha_J\log\theta_{dJk}$ s.t. $\sum_{j=1}^{J}\theta_{djk} = 1$

$= \arg\max_{\theta}\min_{\lambda} \alpha_1\log\theta_{d1k} + \cdots + \alpha_J\log\theta_{dJk} - \lambda\left(\sum_{j=1}^{J}\theta_{djk} - 1\right)$

$\longrightarrow \frac{\alpha_j}{\hat{\theta}_{djk}} - \lambda = 0 \quad \forall j$, $\sum_{j=1}^{J}\hat{\theta}_{djk} = 1$

$\longrightarrow \sum_{j=1}^{J}\alpha_j = \lambda \quad \forall j$

$\longrightarrow \hat{\theta}_{djk} = \frac{\alpha_j}{\sum_{j=1}^{J}\alpha_j} = \frac{\#D(X_d=j, Y=y_k)}{\#D(y_k)}$

③ $Y \sim \text{Multinomial}(\pi) \longrightarrow p(y_k|\pi) = \pi_k$ where $\sum_{k=1}^{K}\pi_k = 1$

$\Rightarrow \hat{\pi}_k = \frac{\#D(y_k)}{\#D}$

☆ Maximum A Posteriori (MAP) estimation

① $X_d | y_k \sim \text{Multinomial}(\theta_{dk}) \longrightarrow P(X_d = j | y_k, \theta) = \theta_{djk}$ where $\sum_{j=1}^{J} \theta_{djk} = 1$

$\Rightarrow$ Prior : $P(\theta) = \text{Dirichlet}(\beta_1, \beta_2, \cdots, \beta_J) = \frac{\theta_1^{\beta_1 - 1} \theta_2^{\beta_2 - 1} \cdots \theta_J^{\beta_J - 1}}{B(\beta_1, \beta_2, \cdots, \beta_J)}$

$\Rightarrow$ Posterior : $P(\theta | D(y_k)) \propto \prod_{i=1}^{n} P(X_{di} | y_{ki}, \theta) P(\theta) = \theta_{d1k}^{\alpha_1 + \beta_1 - 1} \theta_{d2k}^{\alpha_2 + \beta_2 - 1} \cdots \theta_{dJk}^{\alpha_J + \beta_J - 1}$ $(\alpha_1 + \alpha_2 + \cdots + \alpha_J = \#D(y_k))$

$\Rightarrow \hat{\theta}_{dk} = \frac{\alpha_j + \beta_j - 1}{\sum_{j=1}^{J} (\alpha_j + \beta_j - 1)} = \frac{\#D(X_d = j, Y = y_k) + \beta_j - 1}{\#D(y_k) + \sum_{j=1}^{J} (\beta_j - 1)}$

② $Y \sim \text{Multinomial}(\pi) \longrightarrow P(y_k | \pi) = \pi_k$ where $\sum_{k=1}^{K} \pi_k = 1$

$\Rightarrow \hat{\pi}_k = \frac{\#D(y_k) + \gamma_k - 1}{\#D + \sum_{k=1}^{k} (\gamma_k - 1)}$

* Naïve Bayes vs Logistic regression

① Naive Bayes (NB) $\longrightarrow$ generative

$\Rightarrow$ maximize data likelihood $\longrightarrow \log P(D|\theta) = \sum_{i=1}^{n} \log P(x_i, y_i | \theta)$

② Logistic Regression (LR) $\longrightarrow$ discriminative

$\Rightarrow$ maximize conditional data likelihood $\longrightarrow \log P(Y|X, \theta) = \sum_{i=1}^{n} \log P(y_i | x_i, \theta)$

* When model is correct, both produce identical classifier

* When model is incorrect, LR is less biased as conditional independence isn't assumed.

* When # of data points is small, NB is better as it does not overfit