AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Choose an item.

Assignment Cover Sheet

Assignment Title:	Final Assignment		
Assignment No:	01	Date of Submission:	Click here to enter a date.
Course Title:	MACHINE LEARNING [C]		
Course Code:	Click here to enter text.	Section:	C
Semester:	Spring 2024-2025	Course Teacher:	TAIMAN ARHAM SIDDIQUE

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaborationhas been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.
7. I/we understand thatPlagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a formofcheatingandisaveryseriousacademicoffencethatmayleadtoexpulsionfromtheUniversity. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of them arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

* *Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group Name/No.:

No	Name	ID	Program	Signature
1	Tarikul Islam Nishat	21-44632-1	CSE	
2			Choose an item.	
3			Choose an item.	
4			Choose an item.	
5			Choose an item.	
6			Choose an item.	
7			Choose an item.	
8			Choose an item.	
9			Choose an item.	
10			Choose an item.	

Faculty use only

FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

Q1) Explain how locally weighted regression differs from linear regression, including their formulas. What is an advantage of locally weighted regression over linear regression?

Ans: Locally Weighted Regression (LWR) and Linear Regression (LR) are both statistical methods used to model relationships between variables, but they differ fundamentally in how they fit data.

Linear Regression (LR)

Linear regression is a parametric approach that models the relationship between one or more independent variables (predictors) and a dependent variable (response) by fitting a linear equation to observed data.

The formula for a linear regression model can be expressed as:

$y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots+\beta_nx_n+\epsilon$

where:

- y is the dependent variable,
- $\beta_0, \beta_1\ldots, \beta_n\beta_0, \beta_1\ldots, \beta_n$ are the coefficients of the model,
- $x_1, x_2\ldots, x_nx_1, x_2\ldots, xn$ are the independent variables,
- ϵ is the error term.

The coefficients (β) are typically estimated using least squares, which minimizes the sum of the squared differences between the observed values and the values predicted by the linear model.

Locally Weighted Regression (LWR)

Locally Weighted Regression, also known as LOESS or LOWESS (Locally Weighted Scatterplot Smoothing), is a non-parametric method that builds multiple regression models in the vicinity of each point of interest. LWR does not assume a global form for the model; instead, it fits simple models to localized subsets of the data to make predictions at specific points. The typical formula involves weighting the nearby observations more heavily than those further away. The prediction y^ at a point x is given by:

$y\text{^}(x)=\theta(x)^Tx$

Where $\theta(x)$is calculated for each x using:

$\theta(x)=(X^TW(x)X)^{-1}X^TW(x)\mathrm{y}$

- X is the matrix of input features,
- $W(x)$ is a diagonal matrix of weights, with each weight inversely related to the distance between x and the point where the prediction is made,
- y is the vector of observed responses.

Advantage of Locally Weighted Regression over Linear Regression:

Flexibility in Modeling Non-linear Relationships: One of the main advantages of LWR over traditional linear regression is its ability to model non-linear relationships locally. While linear regression fits a single linear equation to all data points, LWR adapts to changes in the form of the relationship locally. This makes LWR particularly useful for data with non-linear patterns or varying trends across the domain, as it can fit complex, localized patterns without requiring a globally accurate model form. This flexibility leads to potentially better fit and predictions, especially in the presence of non-linear relationships.

Q2) Given you want to apply a model to predict whether a patient has malignant or benign tumor, where model output y = 1 means malignant and y = 0 means benign. Explain how the binary logistic regression model is used to train on patient data and then predict tumor of a new patient. Include formulas and learning algorithm used in your answer.

Ans: Binary logistic regression is a statistical method used to model binary outcome variables. In the context of predicting whether a tumor is malignant or benign, logistic regression can be a suitable model due to its ability to provide probabilities for binary outcomes. Here's a step-by-step explanation of how logistic regression is applied:

__**Model Structure**__

The logistic regression model predicts the probability that the dependent variable belongs to a particular category. For tumor classification:

- **Dependent Variable y**: Binary, where y=1 indicates malignant, and y=0indicates benign.
- **Independent Variables X**: These could be features such as age, tumor size, cell type, uniformity of cell shape, etc.

__**Logistic Function**__

The logistic regression model uses the logistic function to map predicted values to probabilities. The model is formulated as:

$P(y{=}1|X) = \frac{1}{1+e-z}$

where $z=\beta_0+\beta_1x_1+\beta_2x_2+...+\beta_nx_n$is the linear combination of features:

- $\beta_0, \beta_1..., \beta_n$ are the parameters of the model.
- $x_1, x_2..., x_n$ are the input features.
- e is the base of the natural logarithm.

Learning Algorithm

The parameters β of the model are typically estimated using maximum likelihood estimation (MLE). The goal is to find the set of parameters that maximizes the likelihood function, given the observed data. This is equivalent to minimizing the logistic loss function over the dataset.

Likelihood Function

The likelihood function for a binary logistic model given *mm* observations is:

$L(\beta)=\prod_{i=1}^{m}[P(y_i=1|x_i)]^{y_i}[1-P(y_i=1|x_i)]^{1-y_i}$

Taking the logarithm (for computational ease), the function to maximize becomes:

$\ell(\beta)=\sum_{i=1}^{m}[y_i\log(P(y_i=1|x_i)) +(1-y_i)\log(1-P(y_i=1|x_i))]$

This is typically maximized using numerical optimization techniques such as gradient descent.

Predicting New Cases

Once the model is trained, predicting the malignancy of a new patient's tumor involves:

1. Collecting the input features *x* of the new patient.
2. Calculating the logistic function using the estimated parameters:

   $P(y=1|x_{new}) =\frac{1}{1+e-(\beta 0+\beta 1xnew,1+\cdots+\beta nxnew)}$

3. Interpreting the probability $P(y=1|x_{new})$:
   - If $P(y=1|x_{new})$ $P(y=1|x_{new})$ is greater than a chosen threshold (commonly 0.5), the tumor is predicted to be malignant ($y=1$).
   - Otherwise, it is predicted to be benign ($y=0$).

Q3.a) Given the output, y(n), of 3 training items of softmax regression are represented by the following one-hot vectors where y ϵ {1,2,3}: y 1 = [1 0 0], y2 = [0 1 0] and y3 = [0 0 1]. Write the expanded form of the softmax cost function J(w) for these 3 items, and the softmax output function f(x;w).

Ans: The softmax regression model, also known as multinomial logistic regression, generalizes logistic regression to multiple classes. It's a popular method for categorical classification where classes are mutually exclusive.

Softmax Output Function $f(x;W)$

The softmax function provides the probability distribution of the event over 'K' different classes. In this case, since y can take values in {1, 2, 3}, we have three classes (K = 3).

For a given input vector *xx* and a weight matrix *W*, the softmax function for a class k is computed as follows:

$p\,(y=k|x;\,W)=\frac{e^{wTKz}}{\sum je^{wTKz}}$

Where:

- wk is the weight vector corresponding to class k.
- wj is the weight vector for any class j from 1 to 3.
- x is the input feature vector.

Expanded Form of the Softmax Cost Function *J(W)*

The softmax regression cost function $J(W)$ is typically defined using the cross-entropy loss between the predicted probabilities and the actual class labels in a one-hot encoded format. For three training items and three classes, the cost function $J(W)$ can be written as:

$J(W)=-[y1\log f(x1;W)+y2\log f(x2;W)+y3\log f(x3;W)]$

Where:

- yi is the one-hot encoded true label vector for the i-th training example.
- $f\ (xi;\ W)$is the softmax output vector for the i-th training example.

This expression using the one-hot encoded vectors provided:

$y_1=[1,0,0]$:

$y_1\log f(x_1;W)$
$=1\cdot\log p(y=1|x_1;W)+0\cdot\log p(y=2|x1;W)+0\cdot\log p(y=3|x_1;W)=\log p(y=1|x1;W$

$y_2=[0,1,0]$:
$y_2\log f(x_2;W)=0\cdot\log p(y=1|x_2;W)+1\cdot\log p(y=2|x_2;W)+0\cdot\log p(y=3|x_2;W)=\log p(y=2|x_2$
$;W)$

$y_3=[0,0,1]$:

$y_3\log f(x_3;W)=0\cdot\log p(y=1|x_3;W)+0\cdot\log p(y=2|x_3;W)+1\cdot\log p(y=3|x_3;W)=\log p(y=3|x_3$
$;W)$

Hence, the expanded form of the cost function $J(W)$is:

$J(W)=-[\log p(y=1|x1;W)+\log p(y=2|x2;W)+\log p(y=3|x3;W)]$

This expression sums the negative logarithms of the predicted probabilities of the true classes across all training examples, making it a measure of how well the probability distributions predicted by the model align with the actual distributions of the labels. The optimization of WW seeks to minimize $J(W)$, effectively pushing $p(y=k|xi;W)p(y=k|xi;\ W)$ towards 1 for the correct class kk and towards 0 for all incorrect classes.

b) What is the relationship between softmax and binary logistic regression?

Ans:

Binary Logistic Regression

Binary logistic regression is a type of logistic regression that is used specifically for binary classification problems. It predicts the probability that an observation belongs to one of two classes (usually labeled as 0 and 1). The formula for binary logistic regression predicting the probability of class 1 can be expressed as:

$P\ (y=1|x;\ \beta)=\frac{1}{1+e-BTz}$

where β represents the model coefficients, including the intercept, and xx represents the feature vector.

Softmax Regression

Softmax regression, also known as multinomial logistic regression, generalizes binary logistic regression to multi-class problems (more than two classes). Instead of predicting a binary outcome, it predicts the probability of each class over a set of K possible classes. The probability that a given observation belongs to class k is given by:

$p\ (y=k|x;\ W)=\frac{e^{wTKz}}{\sum je^{wTKz}}$

Here, W is the matrix of weights, and wk represents the weight vector for class k.

Relationship and Connection

- **Generalization**: Softmax regression can be seen as a generalization of binary logistic regression to multiple classes. When K=2 (two classes), the softmax regression model reduces to a binary logistic regression model. In this special case, if you let class 1 be represented by $w1$ and class 2 by $w2$, the softmax formula simplifies to:

- $P(y=1|x;W)=\frac{e^{wTKz}}{\sum je^{wTKz}}=\frac{1}{e-(w1-w2)Tx1}$

- **Model Complexity**: In binary logistic regression, there is only one set of coefficients (besides the intercept), as there is only one logistic function modeling the probability of one class versus another. In softmax regression, there is a separate coefficient vector for each class.

- **Interpretability**: In both models, the coefficients β or wk provide the relative contribution of each feature towards the prediction for a class. In binary logistic regression, the interpretation is more straightforward since it directly models the log-odds of one class against another. In softmax regression, each weight vector wk models the log-odds of class k against all other classes combined, but the interpretation is conditioned on the presence of other classes.

- **Usage Context**: Binary logistic regression is used when the outcomes are dichotomous (two possible outcomes). Softmax regression is used when the outcomes can take on more than two categories, making it suitable for broader applications in multi-class classification scenarios.

Q4) What is the penalty term of ridge/L2 regularization and how does it reduce overfitting?

Ans: Ridge regression, also known as L2 regularization, is a technique used to analyze multiple regression data that suffer from multicollinearity (when independent variables are highly correlated). When multicollinearity occurs, least squares estimates are unbiased, but their variances are large, which might lead to large errors in prediction. Here's a detailed look at the penalty term in ridge regression and how it helps reduce overfitting:

Penalty Term in Ridge Regression

The penalty term in ridge regression is added to the cost function of ordinary least squares (OLS) regression. This term penalizes the size of the coefficients, which helps to control their growth. The modified cost function with the L2 penalty term is:

$J(\beta)=\sum^{i=1}{}_{n}(\mathrm{y}_{\mathrm{i}}-\beta^{T}x_{i})^{2}+\lambda\sum^{j=1}{}_{p}\beta_{j}^{2}$

Here:

- yi are the observed values.
- β is the vector of coefficients including the intercept.
- xi are the feature vectors.
- λ is a non-negative regularization parameter that controls the strength of the penalty.
- βj^2 represents the squared magnitude of each coefficient.

L2 Regularization Reduces Overfitting

1. **Shrinking Coefficients**: The key idea behind ridge regression is to introduce the L2 penalty (the sum of the squares of the coefficients) into the loss function. This penalty term causes the coefficients of the least important variables to shrink towards zero. It's important to note that unlike Lasso (L1 regularization), which can shrink coefficients to zero, ridge tends to decrease the magnitude of the coefficients but not zero them out completely.

2. **Bias-Variance Trade-Off**: By introducing the regularization term, you're essentially increasing the bias of the model (as the model will no longer fit the training data as perfectly). However, in return, you reduce the variance of the model predictions, leading to less overfitting. This trade-off is beneficial when the goal is to improve the model's performance on new, unseen data.

3. **Handling Multicollinearity**: In scenarios with high multicollinearity, small changes in the input data can lead to large changes in the model, resulting in high variance. Ridge regression stabilizes the coefficient estimates by dampening the impact of collinear predictors.

4. **Improving Generalization**: The regularization term $\lambda\lambda$ helps in controlling the model's complexity, ensuring that it performs well on unseen data by not catching noise and random fluctuations in the training data. The choice of $\lambda\lambda$ can significantly influence the effectiveness of the regularization, with larger values increasing the amount of shrinkage applied to the coefficients.

5. **Geometric Interpretation**: From a geometric perspective, ridge regression can be seen as constraining the coefficients by penalizing their Euclidean norm. This constraint is akin to fitting the regression within a sphere of radius determined by $\lambda\lambda$. It forces the optimization process to not only focus on minimizing the training error but also on keeping the model coefficients within a predefined limit.

Q5.a) Write the pseudocode/steps of applying Policy iteration to solve an MDP, including the equations.

Ans: Policy iteration is a method used in reinforcement learning to find an optimal policy for a Markov Decision Process (MDP) by iteratively improving the policy based on the evaluation of the current policy. Here's a step-by-step breakdown of the policy iteration algorithm, including pseudocode and relevant equations.

Definitions:

- States (S): The set of all possible states.
- Actions (A): The set of actions available in each state.
- Policy (π): A mapping from states to actions, $\pi(s)$ denotes the action taken in state s.
- Reward (R): Rsa is the reward received after taking action a in state s.
- Transition probabilities (P): $Pss'a$ is the probability of transitioning from state s to state s' after taking action a.
- Discount factor (γ): A factor between 0 and 1 that discounts future rewards.

Policy Iteration Algorithm:

1. **Initialization**:
   a. Initialize the policy π arbitrarily. For all states $s \in S$, $\pi(s)$ can be any action $a \in A(s)$.
   b. Initialize $V(s)$ to zero for all states ss (optional starting point).

2. **Policy Evaluation** (Estimate V^{π}):

   Repeat until V converges

   - For each state $s \in S$:

     $$V_{\text{new}}(s) = R_s^{\pi(s)} + \gamma \sum_{s' \in S} P_{ss'}^{\pi(s)} V(s')$$

   - $V(s) \leftarrow V_{\text{new}}(s)$

3. **Policy Improvement** (Generate $\pi'\pi'$ better than $\pi\pi$):
   - Policy-stable ← true
   - For each state $s \in S$:
     - old-action $\leftarrow \pi(s)$
     - $\pi(s) \leftarrow \text{argmax}_{a \in A(s)}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s'))$
     - If old-action $\neq \pi(s)$, then policy-stable ← false
4. **Convergence Check**:
   - If policy-stable, stop and return V and π.
   - Otherwise, go to Step 2 (Policy Evaluation).

Policy iteration Evaluation Explanation:

Policy Evaluation step computes the state-value function v^{π} for a fixed policy π. This step iteratively updates the value of each state to reflect the expected return starting from that state and following the current policy. The update uses the Bellman equation for deterministic policies.

Policy Improvement step updates the policy based on the current value function. For each state, it chooses the action that maximizes the expected return according to the current value function estimate. This step may change the policy by choosing different actions than the current policy.

The algorithm alternates between these two steps until the policy remains unchanged in the policy improvement step, indicating convergence to an optimal policy.

b) What is the advantage of using an exploration-based policy like ϵ-greedy, to solve an MDP?

Ans: Using an exploration-based policy like ϵ-greedy to solve a Markov Decision Process (MDP) addresses a fundamental challenge in reinforcement learning: the balance between exploration and exploitation. This strategy is particularly advantageous in scenarios where the agent needs to learn a policy from interactions with the environment, without prior knowledge of the environment's dynamics. Here are some key advantages of using ϵ-greedy in such contexts:

1. Balance Between Exploration and Exploitation

- **Exploitation**: The agent exploits its current knowledge to choose actions that it believes yield the highest reward based on the current value function or Q-values. This is essential for maximizing the rewards in the short term.
- **Exploration**: The agent explores by occasionally choosing random actions (with probability ϵ), not necessarily known to maximize the reward. Exploration is crucial for discovering potentially better actions that might not have been taken frequently under a purely greedy policy.

2. Discovery of Optimal Policy

- The ϵ-greedy approach ensures that no action is left untried for an extended period, and every state-action pair is visited infinitely often in the limit of infinite exploration. This property is crucial for guaranteeing that the learning algorithm will converge to an optimal policy, assuming sufficient time and proper learning rate settings.

3. Prevention of Early Convergence to Suboptimal Policies

- Without adequate exploration, an agent might converge prematurely to a suboptimal policy, especially in complex environments where certain actions may initially appear suboptimal but are actually optimal in the long run. ϵ-greedy helps in avoiding local optima by ensuring that the agent periodically steps out of the current policy to test other possibilities.

4. Adaptability and Robustness in Dynamic Environments

- In environments where the dynamics might change over time, a purely greedy strategy might fail if it sticks to a policy based on outdated information. An ϵ-greedy policy, by continually incorporating random actions, may adapt more effectively to such changes.

5. Simple and Easy to Implement

- The ϵ-greedy policy is straightforward to implement compared to more complex exploration strategies like those using Upper Confidence Bounds (UCB) or Thompson Sampling. It requires minimal changes to a greedy algorithm, making it accessible for beginners and practical for initial experiments in complex environments.

Q6.a) What makes Q-learning an off-policy algorithm?

Ans: Q-learning is classified as an off-policy algorithm in reinforcement learning primarily due to its ability to learn the optimal policy independently of the agent's actions and the policy being followed during learning. This distinctive attribute arises from the way Q-learning updates its Q-values and makes decisions during the training phase.

Definition and Key Characteristics:

Off-policy Learning: In off-policy learning, the strategy (or policy) that the learning algorithm follows to select actions (called the behavior policy) is different from the policy it evaluates and improves (called the target policy). Essentially, the learning process can evaluate and improve a policy that is different from the one it executes, allowing it to learn about optimal decisions without needing to perform those decisions.

How Q-learning Implements Off-policy Learning

1. Q-value Updates: Q-learning updates its Q-values using the Bellman equation as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha(r_{t+1}+\gamma \max_a Q(s_{t+1},a)-Q(s_t,a_t))$$

Here:

- $st$and $atat$ are the current state and action.
- r_{t+1} is the reward received after taking action $atat$ in state s_t
- s_{t+1} is the new state after action at.
- $\max_a Q(st+1,a)$ is the maximum Q-value for the next state across all possible actions, which represents the optimal future value.
- α is the learning rate.
- γ is the discount factor.

The critical aspect here is the term $\max_a Q(s_{t+1},a)\max aQ(st+1,a)$. Q-learning always updates its Q-values using the maximum Q-value of the next state, regardless of the policy being followed to choose the actual next action. This means it evaluates the best possible future rewards independently of the action chosen by the current policy, thus learning the optimal policy.

Behavior vs. Target Policy:

- **Behavior Policy**: Can be exploratory, such as ϵ-greedy, which ensures all state-action pairs are visited sufficiently to learn accurate Q-values. This policy may not always choose the best-known action but will explore the environment.
- **Target Policy**: This is the greedy policy derived from the current Q-values, which always selects the action with the highest Q-value in any state (i.e., $\text{argmax}_a Q(s,a)$.Q-learning's updates aim to optimize this policy, even though it is not necessarily followed during exploration.

Advantages of Off-policy Learning:

Flexibility in Exploration: Because learning is separated from the policy being improved, Q-learning can explore the environment freely without the risk of suboptimal exploratory actions degrading the quality of the learned policy.

Robustness: This separation allows Q-learning to learn from transitions gathered from any source, including previous versions of policies or even policies derived from other agents (in multi-agent systems).

b) What is the difference between on-policy and off-policy algorithms?

Ans: On-policy and off-policy algorithms are two broad categories of reinforcement learning methods that differ primarily in how they manage the relationship between the policy used to make decisions and the policy being learned or evaluated. Understanding these differences is crucial for selecting the right approach based on the specific requirements and constraints of a given reinforcement learning problem.

On-policy Algorithms:

On-policy algorithms learn the value of the policy being followed by the agent, meaning that the learning process evaluates and improves the same policy that dictates the agent's actions. Here's a breakdown of on-policy characteristics:

- **Learning and Exploration Integration**: The policy used to make decisions (the behavior policy) is the same policy that is evaluated and improved. This means that the exploration strategy must be integrated within the policy itself.
- **Examples**: The most common on-policy algorithm is SARSA (State-Action-Reward-State-Action), where updates to the policy are based on the action taken according to the current policy.
- **Update Rule** (using SARSA as an example): The update formula for SARSA is typically:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_t+1, a_{t+1}) - Q(s_t, a_t)]$$

where a_{t+1} is the action taken at the next step, chosen according to the current policy (reflecting on-policy learning).

Off-policy Algorithms:

Off-policy algorithms learn the value of an optimal policy independently of the agent's actions, using data collected from a different policy. This allows the algorithm to evaluate and improve a potentially different policy from the one being executed. Here's more about off-policy methods:

- **Separation of Exploration and Exploitation**: The behavior policy, which is used for exploring the environment and collecting data, is distinct from the target policy that is being evaluated and optimized. This allows the target policy to be fully greedy with respect to the learned values.
- **Examples**: Q-learning is a quintessential example of an off-policy algorithm. It updates its estimates based on the maximum reward achievable from the next state, regardless of the action actually taken based on the behavior policy.
- **Update Rule** (using Q-learning as an example): The update formula for Q-learning is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where $\max_a Q(s_{t+1},a) \max_a Q(s_{t+1},a)$ reflects the optimal future value independent of the action chosen by the behavior policy.

<u>**Key Differences:**</u>

1. **Policy Convergence**:
    - **On-policy**: The policy converges based on the trade-off between exploration and exploitation as integrated within the same policy. This often requires methods like $\epsilon\epsilon$-greedy within the policy to ensure adequate exploration.
    - **Off-policy**: Allows the learning of an optimal policy that may be completely deterministic and greedy, using exploration handled by a separate policy.

2. **Data Efficiency and Reusability**:
    - **On-policy**: Can be less data-efficient as it must always explore using the same policy that is being evaluated. This sometimes leads to slower convergence as the policy must be cautious not to exploit poor estimates too quickly.
    - **Off-policy**: Can be more data-efficient and capable of learning from historical data generated by a different policy, even old or suboptimal data, as it separates the data collection from policy optimization.
3. **Robustness and Practicality**:
    - **On-policy**: Generally simpler and with potentially fewer hyperparameters (e.g., no need to manage two distinct policies), but might struggle in environments where exploration can lead to dangerous or costly states.
    - **Off-policy**: More complex in implementation but offers greater flexibility and robustness, especially in environments where safe exploration is a challenge or in scenarios involving multi-agent learning or learning from demonstrations.