Mock Test Report

Student: Amit Kulkarni

Date: April 01, 2025

15
out of 50

Suggestions to Improve

The student demonstrates some understanding of reinforcement learning and data preprocessing but needs significant improvement in several key areas.

Topic: Reinforcement Learning (Reward Processing) - Review the concept of discounted cumulative rewards and how they are calculated using discount factors. Practice implementing the `discount_rewards` function with various reward sequences and discount factors. Pay close attention to the iterative nature of the calculation, working backwards from the final reward.

Topic: Data Preprocessing for Machine Learning - Research and compare different techniques for handling categorical features with high cardinality (many unique values). Explore methods like one-hot encoding, target encoding, binary encoding, and embedding techniques. Consider the trade-offs of each method in terms of dimensionality and computational cost.

Topic: Handling Imbalanced Datasets - Study various techniques for addressing class imbalance in classification problems. Explore resampling methods (oversampling, undersampling, SMOTE), cost-sensitive learning, and ensemble methods. Compare their effectiveness and understand when each technique is most appropriate.

Topic: TensorFlow/Keras for Deep Learning - Review the basics of TensorFlow/Keras, focusing on building and using sequential models. Practice making predictions using a trained model and understand how to handle input shapes correctly when using `model(obs[np.newaxis])`. Understand the output shape of a Keras model's prediction.

Practice coding the solutions to the problems you missed. Focus on understanding the underlying logic and debugging your code.

Work through additional exercises and examples to reinforce your understanding of the concepts.

Consult relevant documentation and tutorials for clarification on specific functions and libraries.

Correct Answers

Question	Correct Answer

What will be the output of the following code? ```python import numpy as np rewards = [10, 0, -50] discount_factor = 0.8 discounted_rewards = np.array(rewards) for step in range(len(rewards) - 2, -1, -1): discounted_rewards[step] += discounted_rewards[step + 1] * discount_factor print(discounted_rewards) ```	[10, -22, -50]
Which of the following best describes the purpose of the `discount_rewards` function (assuming `rewards` is a list of rewards and `discount_factor` is a float)? ```python def discount_rewards(rewards, discount_factor): discounted = np.array(rewards) for step in range(len(rewards) - 2, -1, -1): discounted[step] += discounted[step + 1] * discount_factor return discounted ```	It discounts future rewards based on the discount factor.
The following code snippet is intended to normalize rewards. What is the flaw in this implementation? ```python import numpy as np def normalize_rewards(rewards): return (rewards - np.mean(rewards)) / np.std(rewards) ```	It doesn't handle the case where the standard deviation is zero.
Given the following code, what will `all_rewards` contain after execution? ```python import numpy as np def play_one_step(env, obs): #Simplified for brevity # ... some code ... return [10, 20, 30] # Example rewards rewards_list = [] for i in range(2): rewards = play_one_step(None, None) #Simulating env rewards_list.append(rewards) all_rewards = rewards_list ```	[[10, 20, 30], [10, 20, 30]]
You are training a model using TensorFlow/Keras. Which callback would be MOST useful for preventing overfitting during training?	EarlyStopping
Which preprocessing technique is best suited for handling categorical features with a large number of unique values?	One-hot encoding
What is the primary purpose of using a discount factor (gamma) in reinforcement learning?	To prioritize immediate rewards over long-term rewards.
Consider the following Python code using scikit-learn. What does the code achieve? ```python from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) ```	It splits the data into training and testing sets.
You're working with a highly imbalanced dataset for a classification task. Which technique would be MOST effective in improving model performance?	Applying resampling techniques (oversampling/undersampling).
What will the following code print? ```python import numpy as np import tensorflow as tf model = tf.keras.Sequential([tf.keras.layers.Dense(1, input_shape=(1,))]) obs = np.array([1.0]) with tf.GradientTape() as tape: prediction = model(obs[np.newaxis]) print(prediction.numpy().shape) ```	(1, 1)