Problem Set: Python Functions

Say OL

October 06, 2024

Problem Set

1. **Data Normalization**
   Function: `normalize(data: List[float]) -> List[float]`
   Description: Write a function that normalizes a list of numerical data using Min-Max scaling.
   Input: A list of floats (e.g., `[2.0, 4.0, 6.0, 8.0]`).
   Sample Data: `[2.0, 4.0, 6.0, 8.0]`
   Output: A list of normalized floats.

2. **Train-Test Split**
   Function: `train_test_split(data: List[Any], test_size: float = 0.2) -> Tuple[List[Any], List[Any]]`
   Description: Implement a function that splits a dataset into training and testing sets.
   Input: A list of data and a test size.
   Sample Data: `['a', 'b', 'c', 'd', 'e']` with `test_size = 0.2`
   Output: A tuple of training and test sets.

3. **K-Nearest Neighbors Algorithm**
   Function: `k_nearest_neighbors(data: List[Tuple[List[float], str]], new_point: List[float], k: int) -> str`
   Description: Implement KNN classifier.
   Input: A list of tuples with features and class labels, a new point, and an integer k.
   Sample Data: `data = [([1.0, 2.0], 'A'), ([3.0, 4.0], 'B'), ([5.0, 6.0], 'A')]` with `new_point = [2.0, 3.0]` and `k = 2`
   Output: The predicted class label.

4. **Mean Squared Error**
   Function: `mean_squared_error(y_true: List[float], y_pred: List[float]) -> float`
   Description: Calculate mean squared error between actual and predicted values.
   Input: Two lists of floats.
   Sample Data: `y_true = [3.0, -0.5, 2.0, 7.0]` and `y_pred = [2.5, 0.0, 2.0, 8.0]`
   Output: A float representing the mean squared error.

5. **Feature Engineering**
   Function: `one_hot_encode(categories: List[str]) -> List[List[int]]`
   Description: Perform one-hot encoding on a list of categorical variables.
   Input: A list of strings.
   Sample Data: `['red', 'blue', 'green', 'blue']`
   Output: A 2D list of one-hot encoded vectors.

6. **Logistic Regression Sigmoid Function**
   Function: `sigmoid(z: float) -> float`
   Description: Implement the sigmoid function used in logistic regression.
   Input: A float.
   Sample Data: `0.5`
   Output: A float representing the output of the sigmoid function.

7. **Confusion Matrix**
   Function: `confusion_matrix(y_true: List[int], y_pred: List[int]) -> Dict[str, int]`
   Description: Create a confusion matrix as a dictionary.
   Input: Two lists of integers.
   Sample Data: `y_true = [1, 0, 1, 1, 0]` and `y_pred = [1, 0, 0, 1, 1]`
   Output: A dictionary with counts for 'TP', 'TN', 'FP', and 'FN'.

8. **Simple Linear Regression**
   Function: `linear_regression(x: List[float], y: List[float]) -> Tuple[float, float]`
   Description: Implement a function to perform simple linear regression.
   Input: Two lists of floats representing x and y values.
   Sample Data: `x = [1, 2, 3, 4]` and `y = [2, 3, 5, 7]`
   Output: A tuple of slope and intercept.

9. **Variance Calculation**
   Function: `variance(data: List[float]) -> float`
   Description: Calculate the variance of a list of numbers.
   Input: A list of floats.
   Sample Data: `[10.0, 12.0, 23.0, 23.0, 16.0, 23.0, 21.0]`
   Output: A float representing the variance.

10. **Polynomial Regression**
    Function: `polynomial_regression(x: List[float], y: List[float], degree: int) -> Tuple[float, List[float]]`
    Description: Implement a function to perform polynomial regression of a specified degree.
    Input: Two lists of floats and an integer for the degree.
    Sample Data: `x = [1, 2, 3, 4]` and `y = [1, 4, 9, 16]` with `degree = 2`
    Output: A tuple of the intercept and coefficients of the polynomial.