भारतीय सूचना प्रौद्योगिकी संस्थान गुवाहाटी

Indian Institute of Information Technology Guwahati

CS236 - ARTIFICIAL INTELLIGENCE LAB

ASSIGNMENT - 09

Problem Statement

Write a Python program to solve a multivariate linear regression problem using **Particle Swarm Optimization (PSO)**. The goal is to find the optimal weights (coefficients) for a linear model that predicts the median value of owner-occupied homes in the Boston Housing dataset by minimizing the Mean Squared Error (MSE) on the training data.

Background: Multivariate Linear Regression and the Boston Housing Dataset

Multivariate Linear Regression

Multivariate linear regression models the linear relationship between multiple independent variables (features) and a single dependent variable (target). The model is expressed as:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + \epsilon, \tag{1}$$

where y is the dependent variable (predicted value, e.g., median house value), $x_1, x_2, \ldots, x_n$ are the independent variables (features), w_0 is the intercept (bias) term, $w_1, w_2, \ldots, w_n$ are the coefficients (weights) for each feature to be optimized, and ϵ is the error term (difference between actual and predicted values). The objective is to determine the weights $w_0, w_1, \ldots, w_n$ that minimize the Mean Squared Error (MSE) between predicted and actual values.

The Boston Housing Dataset

The Boston Housing dataset contains 506 instances of housing data from Boston suburbs, with 13 features and 1 target variable. It is available in scikit-learn. The features include: **CRIM**, per capita crime rate by town; **ZN**, proportion of residential land zoned for lots over 25,000 sq.ft.; **INDUS**, proportion of non-retail business acres per town; **CHAS**, Charles River dummy variable (1 if tract bounds river; 0 otherwise); **NOX**, nitric oxides concentration (parts per 10 million); **RM**, average number of rooms per dwelling; **AGE**, proportion of owner-occupied units built prior to 1940; **DIS**, weighted distances to five Boston employment centers; **RAD**, index of accessibility to radial highways; **TAX**, full-value property-tax rate per $10,000; **PTRATIO**, pupil-teacher ratio by town; **B** represents the proportion of blacks by town; and **LSTAT**, percentage of lower status of the population. The target variable is **MEDV**, median value of owner-occupied homes in $1000s.

Algorithm Specifications

Implement a Particle Swarm Optimization (PSO) algorithm to optimize the weights of the linear regression model. PSO is a population-based optimization technique inspired by the social behavior of birds flocking. Each particle represents a candidate solution (set of weights) and moves through the search space to find the optimal weights.

1. **Representation:** A particle's position is represented by a 14-dimensional vector $\mathbf{x} = [w_0, w_1, w_2, \ldots, w_{13}]$, where w_0 is the intercept term and w_1 to w_{13} are the coefficients for the 13 features. The prediction for a data point is given by $y_{\text{pred}} = w_0 + \sum_{j=1}^{13} w_j x_j$, where x_j are the feature values.

2. **Fitness Function:** The goal is to minimize the Mean Squared Error (MSE) on the training data, defined as $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_{\text{pred},i})^2$, where N is the number of training data points, y_i is the actual target value for the i-th training point, and $y_{\text{pred},i} = w_0 + w_1 x_{1,i} + \ldots + w_{13} x_{13,i}$ is the predicted value. A lower MSE indicates a better fit.

3. **Constraints:** Weights are bounded such that $-10 \leq w_j \leq 10$ for $j = 0, \ldots, 13$. Use clamping to enforce these bounds: if a weight exceeds 10, set it to 10; if it falls below -10, set it to -10.

4. **Initialization:** Each particle's position is initialized randomly within $[-10, 10]$ for each dimension. Velocities are initialized randomly, e.g., uniformly between $[-1, 1]$, or set to zero.

5. **Velocity and Position Updates:** Each particle's velocity and position are updated using the equations $\mathbf{v}_i^{t+1} = w \cdot \mathbf{v}_i^t + c_1 \cdot \text{rand}() \cdot (\mathbf{p}_{\text{best},i} - \mathbf{x}_i^t) + c_2 \cdot \text{rand}() \cdot (\mathbf{g}_{\text{best}} - \mathbf{x}_i^t)$ and $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}$, where $\mathbf{v}_i^t$ is the velocity of particle i at iteration t (14-dimensional), $\mathbf{x}_i^t$ is the position of particle i at iteration t, w is the inertia weight (controls momentum), c_1 is the cognitive coefficient (pulls particle toward its personal best), c_2 is the social coefficient (pulls particle toward the global best), $\text{rand}()$ is a random number in $[0, 1]$ (different for each term), $\mathbf{p}_{\text{best},i}$ is the best position found by particle i, and $\mathbf{g}_{\text{best}}$ is the best position found by any particle in the swarm.

6. **Data Splitting:** The dataset is split into 80% training and 20% testing sets using `random_state=42` for reproducibility. The training set is used to compute the fitness (MSE) during PSO, while the testing set is used only for the final evaluation of the best weights.

Input Format

Read from standard input a single line: `w c1 c2 num_particles num_iterations` (space-separated), where w is the inertia weight (e.g., 0.7), $c1$ is the cognitive coefficient (e.g., 1.5), $c2$ is the social coefficient (e.g., 1.5), `num_particles` is the number of particles in the swarm (e.g., 50), and `num_iterations` is the number of iterations to run PSO (e.g., 2000).

Output Format

Output to standard output the following: first, the best weights as $[w_0, w_1, \ldots, w_{13}]$ (list of 14 floats, rounded to 3 decimal places); second, the training MSE (float, 2 decimal places); and third, the testing MSE (float, 2 decimal places).

Sample Input

```
0.7 1.5 1.5 50 2000
```

Sample Output

```
Best Weights: [-0.108, 0.046, 0.021, 2.687, -17.796, 3.805, 0.001, -1.476, 0.306, -0.012, -0.95
Training MSE: 22.35
Testing MSE: 28.52
```

Implementation Notes

Data Loading

Two versions for loading the Boston Housing dataset are provided below.

Version 1 (Deprecated)

Using `load_boston` from scikit-learn. This version is available in scikit-learn 1.1 or earlier.

```python
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import numpy as np

boston = load_boston()
X = boston.data  # Features (506 rows, 13 columns)
y = boston.target  # Target (506 rows)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Version 2 (Recommended)

Using `fetch_openml` to load the dataset from OpenML.

```python
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
import numpy as np

boston = fetch_openml(name='boston', version=1, as_frame=False)
X = boston.data  # Features (506 rows, 13 columns)
y = boston.target  # Target (506 rows)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Suggested Structure

```
  # Load and split data
  # Initialize particles: positions, velocities, personal bests
  # Initialize global best
  # For each iteration:
  #     For each particle:
  #         Compute fitness (MSE on training data)
```

```
#         Update personal best if fitness improves
#         Update global best if fitness improves
#         Update velocity
#         Update position (apply clamping)
# Evaluate global best on training and testing data
# Output results
```