world_model

April 21, 2025

1 World Model

1.1 Imports

[2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor # Good non-linear choice
# Or use MLPRegressor for a neural network:
# from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
import joblib # To save the trained model
import os
```

1.2 Load Dataset

[6]:
```
def load_data(filepath="../dataset/dataset_limited.txt"):
    """Loads the dataset using pandas."""
    try:
        # Assuming the first row is the header
        df = pd.read_csv(filepath)
        print(f"Dataset loaded successfully. Shape: {df.shape}")
        # Optional: Drop rows with NaN values if any occur
        df = df.dropna()
        print(f"Shape after dropping NaNs: {df.shape}")
        return df
    except FileNotFoundError:
        print(f"Error: Dataset file not found at {filepath}")
        return None
    except Exception as e:
        print(f"Error loading dataset: {e}")
        return None
```

[ ]:
```
# 1. Load the dataset
dataframe = load_data()
```

```
Dataset loaded successfully. Shape: (768, 14)
Shape after dropping NaNs: (768, 14)
```

[8]:
```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   distance_red_init     768 non-null    float64
 1   angle_red_init        768 non-null    float64
 2   distance_green_init   768 non-null    float64
 3   angle_green_init      768 non-null    float64
 4   distance_blue_init    768 non-null    float64
 5   angle_blue_init       768 non-null    float64
 6   rSpeed                768 non-null    int64
 7   lSpeed                768 non-null    int64
 8   distance_red_final    768 non-null    float64
 9   angle_red_final       768 non-null    float64
 10  distance_green_final  768 non-null    float64
 11  angle_green_final     768 non-null    float64
 12  distance_blue_final   768 non-null    float64
 13  angle_blue_final      768 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 84.1 KB
```

1.3 Preprocess Dataset

[9]:
```
def prepare_data(df):
    """Separates features (X) and target variables (Y)."""
    # Input Features: initial state (6) + action (2) = 8 features
    X = df.iloc[:, :8].values
    # Target Variables: final state (6) = 6 features
    Y = df.iloc[:, 8:].values
    print(f"Features (X) shape: {X.shape}")
    print(f"Targets (Y) shape: {Y.shape}")
    return X, Y
```

[10]:
```
def split_data(X, Y, test_size=0.2, random_state=42):
    """Splits data into training and testing sets."""
    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=test_size, random_state=random_state
    )
    print(f"Training set size: {X_train.shape[0]} samples")
    print(f"Testing set size: {X_test.shape[0]} samples")
    return X_train, X_test, Y_train, Y_test
```

[11]:
```
# 2. Prepare Data
X, Y = prepare_data(dataframe)
```

```
Features (X) shape: (768, 8)
Targets (Y) shape: (768, 6)
```

[12]:
```python
def scale_features(X_train, X_test):
    """Scales input features using StandardScaler."""
    scaler = StandardScaler()
    # Fit scaler ONLY on training data
    X_train_scaled = scaler.fit_transform(X_train)
    # Transform both train and test data
    X_test_scaled = scaler.transform(X_test)
    print("Features scaled.")
    return X_train_scaled, X_test_scaled, scaler # Return scaler to save it
```

[13]:
```python
# 3. Split Data
X_train, X_test, Y_train, Y_test = split_data(X, Y)

# 4. Scale Features (Important!)
X_train_scaled, X_test_scaled, scaler = scale_features(X_train, X_test)
```

```
Training set size: 614 samples
Testing set size: 154 samples
Features scaled.
```

1.4 Train model

[14]:
```python
def train_model(X_train, Y_train):
    """Trains a regression model."""
    print("Training RandomForestRegressor model...")
    # Example using RandomForestRegressor
    # n_estimators: number of trees; random_state: reproducibility; n_jobs=-1: use all cores
    model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1, oob_score=True)

    # Example using MLPRegressor (Neural Network)
    # hidden_layer_sizes: tuple defining network structure (e.g., 2 hidden layers)
    # max_iter: maximum number of training iterations
    # alpha: L2 regularization term
    # learning_rate_init: initial learning rate
    # early_stopping: stop training if validation score doesn't improve
    # verbose: print progress
    # model = MLPRegressor(hidden_layer_sizes=(64, 32), activation='relu', solver='adam',
    #                      max_iter=500, random_state=42, early_stopping=True,
    #                      learning_rate_init=0.001, verbose=True)

    model.fit(X_train, Y_train)
```

```
    print("Model training complete.")
    # For RandomForest, you can check the Out-of-Bag score as a quick estimate␣
 ↪of performance
    if isinstance(model, RandomForestRegressor) and model.oob_score_:
         print(f"Model OOB score: {model.oob_score_:.4f}")
    return model
```

[15]:
```
# 5. Train Model (using scaled data)
world_model = train_model(X_train_scaled, Y_train)
```

```
Training RandomForestRegressor model…
Model training complete.
Model OOB score: 0.9579
```

[16]:
```
def evaluate_model(model, X_test, Y_test):
    """Evaluates the model using MAE and MSE."""
    Y_pred = model.predict(X_test)

    mae = mean_absolute_error(Y_test, Y_pred)
    mse = mean_squared_error(Y_test, Y_pred)
    rmse = np.sqrt(mse) # Root Mean Squared Error

    print("\n--- Model Evaluation ---")
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"Mean Squared Error (MSE):  {mse:.4f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

    # Optional: Print metrics per output feature
    print("\nMAE per output feature:")
    output_features = [
        'dist_red_final', 'angle_red_final', 'dist_green_final',
        'angle_green_final', 'dist_blue_final', 'angle_blue_final'
    ]
    for i, name in enumerate(output_features):
         mae_feature = mean_absolute_error(Y_test[:, i], Y_pred[:, i])
         print(f"  {name}: {mae_feature:.4f}")

    return mae, mse
```

1.5 Evaluate

[17]:
```
# 6. Evaluate Model (using scaled test data)
evaluate_model(world_model, X_test_scaled, Y_test)
```

```
--- Model Evaluation ---
Mean Absolute Error (MAE): 22.3253
Mean Squared Error (MSE):  1497.2351
```

```
Root Mean Squared Error (RMSE): 38.6941

MAE per output feature:
  dist_red_final: 39.9570
  angle_red_final: 3.9896
  dist_green_final: 38.2818
  angle_green_final: 7.6712
  dist_blue_final: 40.9996
  angle_blue_final: 3.0525
```

[17]:
```
(22.325294401606275, 1497.2351473844744)
```

1.6 Save model

[22]:
```
def save_model_and_scaler(model, scaler, model_filename="world_model.joblib",␣
↪scaler_filename="scaler.joblib"):
    """Saves the trained model and scaler to disk."""
    try:
        # Ensure the directory exists
        model_dir = "../src/models"
        os.makedirs(model_dir, exist_ok=True)

        model_path = os.path.join(model_dir, model_filename)
        scaler_path = os.path.join(model_dir, scaler_filename)

        joblib.dump(model, model_path)
        joblib.dump(scaler, scaler_path)
        print(f"Model saved to {model_path}")
        print(f"Scaler saved to {scaler_path}")
    except Exception as e:
        print(f"Error saving model/scaler: {e}")
```

[23]:
```
# 7. Save Model and Scaler
save_model_and_scaler(world_model, scaler)
```

```
Model saved to ../src/models/world_model.joblib
Scaler saved to ../src/models/scaler.joblib
```

[24]:
```
# Example prediction (how you'd use it later)
print("\n--- Example Prediction ---")
# Take the first sample from the original test set
sample_X = X_test[0].reshape(1, -1)
sample_Y_actual = Y_test[0]
# Scale the sample using the *saved* scaler
sample_X_scaled = scaler.transform(sample_X)
# Predict using the trained model
sample_Y_pred = world_model.predict(sample_X_scaled)
```

```
print(f"Input State + Action: {sample_X[0]}")
print(f"Actual Final State:   {sample_Y_actual}")
print(f"Predicted Final State:{sample_Y_pred[0]}")
```

```
--- Example Prediction ---
Input State + Action: [ 215.59621861 -116.56767775 1378.20905064 -172.5025863
1075.89103207
   72.16203157   30.            2.          ]
Actual Final State:   [ 156.33665338  -97.02055051 1296.9236501  -173.7067791
1137.12597569
   69.03333994]
Predicted Final State:[ 227.18284769  -88.00610187 1266.99770549 -161.55511559
1065.22886657
   69.61030654]
```