world_model_v5

May 6, 2025

1 World Model

1.1 Imports

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
import joblib
import os
```

[2]:
```
# reproducible results
np.random.seed(42)
```

1.2 Load Dataset

[3]:
```
def load_data(filepath="../dataset/dataset_v3.txt"):
    """Loads the dataset using pandas."""
    try:
        df = pd.read_csv(filepath)
        print(f"Dataset loaded successfully. Shape: {df.shape}")
        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
```

[4]:
```
# 1. Load the dataset
dataframe = load_data()
```

```
Dataset loaded successfully. Shape: (3726, 14)
Shape after dropping NaNs: (3726, 14)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3726 entries, 0 to 3725
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   distance_red_init     3726 non-null   float64
 1   angle_red_init        3726 non-null   float64
 2   distance_green_init   3726 non-null   float64
 3   angle_green_init      3726 non-null   float64
 4   distance_blue_init    3726 non-null   float64
 5   angle_blue_init       3726 non-null   float64
 6   rSpeed                3726 non-null   int64
 7   lSpeed                3726 non-null   int64
 8   distance_red_final    3726 non-null   float64
 9   angle_red_final       3726 non-null   float64
 10  distance_green_final  3726 non-null   float64
 11  angle_green_final     3726 non-null   float64
 12  distance_blue_final   3726 non-null   float64
 13  angle_blue_final      3726 non-null   float64
dtypes: float64(12), int64(2)
memory usage: 407.7 KB
```

1.3 Preprocess Dataset

```
[6]: 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
```

```
[7]: 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
```

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

```
Features (X) shape: (3726, 8)
Targets (Y) shape: (3726, 6)
```

[9]:
```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
```

[10]:
```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: 2980 samples
Testing set size: 746 samples
Features scaled.
```

1.4 Train model

[11]:
```python
def train_ridge_regression(X_train, Y_train):
    """Trains a Ridge Regression model with hyperparameter optimization using␣
 ↪GridSearchCV."""
    print("Training Ridge Regression model with GridSearchCV...")

    # Define the parameter grid for alpha (regularization strength)
    param_grid = {
        'alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],  # Wider range␣
 ↪of regularization strengths
        'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag',␣
 ↪'saga']  # Different solvers for optimization
    }

    # Create the Ridge regression model
    ridge = Ridge()

    # Set up GridSearchCV
    grid_search = GridSearchCV(
```

```
        estimator=ridge,
        param_grid=param_grid,
        scoring='neg_mean_squared_error',  # Optimize for lower MSE
        cv=5,  # 5-fold cross-validation
        n_jobs=-1,  # Use all available CPU cores
        verbose=1
    )

    # Fit GridSearchCV on the training data
    grid_search.fit(X_train, Y_train)

    # Print the best parameters and corresponding score
    print("\nGridSearchCV Complete.")
    print(f"Best parameters found: {grid_search.best_params_}")
    print(f"Best cross-validation score (negative MSE): {grid_search.
↪best_score_:.4f}")

    # Return the best model found by GridSearchCV
    return grid_search.best_estimator_

# Example usage:
world_model = train_ridge_regression(X_train_scaled, Y_train)
```

```
Training Ridge Regression model with GridSearchCV…
Fitting 5 folds for each of 49 candidates, totalling 245 fits

GridSearchCV Complete.
Best parameters found: {'alpha': 0.01, 'solver': 'sag'}
Best cross-validation score (negative MSE): -1470.2011
```

1.5 Evaluate

[12]:
```
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
```

[13]:
```
# 6. Evaluate the Best Model found by GridSearch (using scaled test data) new
evaluate_model(world_model, X_test_scaled, Y_test)
```

```
--- Model Evaluation ---
Mean Absolute Error (MAE): 20.7319
Mean Squared Error (MSE):  1460.9190
Root Mean Squared Error (RMSE): 38.2220

MAE per output feature:
  dist_red_final: 34.2130
  angle_red_final: 8.0814
  dist_green_final: 33.0882
  angle_green_final: 7.7908
  dist_blue_final: 34.2454
  angle_blue_final: 6.9726
```

[13]: (20.73190339516069, 1460.9189664505745)

1.6 Save model

[14]:
```
def save_model_and_scaler(model, scaler, model_filename="world_model_v5.
 ↪joblib", scaler_filename="scaler_v5.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}")
```

[15]:
```python
# 7. Save Model and Scaler
# save_model_and_scaler(world_model, scaler)
```

[16]:
```python
# 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: [680.51968108  89.80900352 315.51674792  90.13283211
632.90729297
   4.12401969  22.          -15.        ]
Actual Final State:   [681.67413708  90.6233093  316.80506478  91.88406189
623.33122452
   4.29052584]
Predicted Final State:[683.13227314  85.61458144 319.99822718  86.15491632
631.71629113
   5.66017759]
```