Deep Reinforcement Learning for Predictive Maintenance

古弘恩 Hantz, 1113534
田雷西 Max, 1113535
范瑞德 Arridson, 1113548

Abstract

This paper explores the use of reinforcement learning (RL) algorithms for predictive maintenance, using a synthetic dataset simulating equipment performance and failures. A custom environment is built to train an agent to make optimal maintenance decisions, with rewards based on avoiding unnecessary actions and addressing failures. This work leverages the Proximal Policy Optimization (PPO) algorithm, evaluates agent performance, and visualizes results to understand decision-making patterns and training dynamics.

1. Introduction

Predictive maintenance aims to minimize equipment downtime and repair costs by addressing potential failures before they occur. Traditional approaches rely on statistical models and machine learning; however, reinforcement learning offers a dynamic and adaptive framework for learning optimal policies in uncertain environments. This work builds a custom RL environment using synthetic equipment data and evaluates the PPO algorithm's performance in such a scenario.

2. Methodology

2.1 Dataset Generation

A synthetic dataset was generated to simulate equipment performance with the following features:

- **Temperature**: Normal distribution with mean = 50, std = 10.
- **Vibration**: Normal distribution with mean = 0.5, std = 0.1.
- **Operational Hours**: Normal distribution with mean = 10, std = 2.
- **Failure Labels**: Binary labels (0 = no failure, 1 = failure) with a 10% probability of failure.

Additional features:

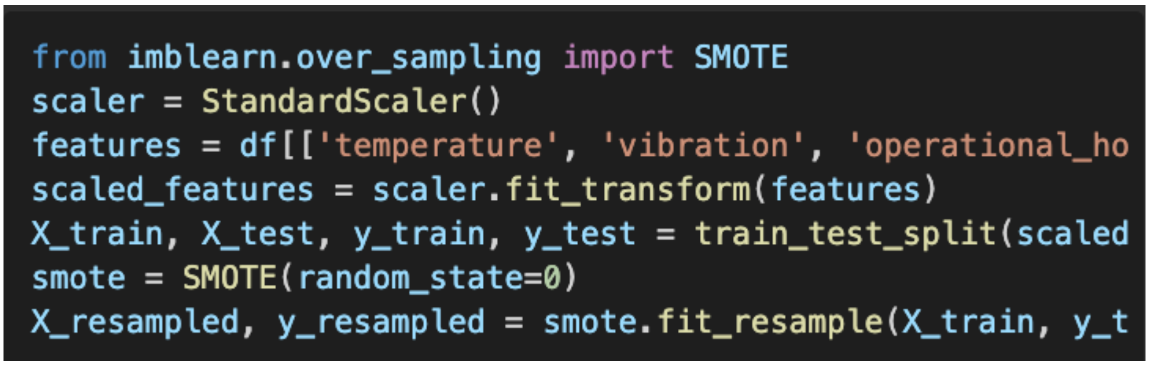
- **Temperature Change**: First-order difference in temperature.
- **Vibration Change**: First-order difference in vibration.

The dataset was divided into training and testing sets, with 80% of the data allocated for training and 20% for testing. For eventual imbalanced dataset—with significantly fewer failure cases (label 1) compared to non-failure cases (label 0)—the Synthetic Minority Over-sampling Technique (SMOTE) was applied to the training data. SMOTE generates synthetic examples for the minority class (failures) by interpolating between existing samples, ensuring equal representation of both classes in the training data. This helps the model to learn more effectively from both failure and non-failure scenarios.

Data Preprocessing

- **Feature Scaling**: StandardScaler normalized continuous features.
- **Balancing**: SMOTE ensured equal representation of failure and non-failure cases in the training data.

Code Snippet:



```
from imblearn.over_sampling import SMOTE
scaler = StandardScaler()
features = df[['temperature', 'vibration', 'operational_ho
scaled_features = scaler.fit_transform(features)
X_train, X_test, y_train, y_test = train_test_split(scaled
smote = SMOTE(random_state=0)
X_resampled, y_resampled = smote.fit_resample(X_train, y_t
```

2.2 Environment Design

A custom environment `EquipmentEnv` was implemented using OpenAI Gym. It simulates the decision-making process for predictive maintenance by interacting with a synthetic dataset. Key components:

Actions

- **0**: No maintenance.
- **1**: Perform maintenance.

States

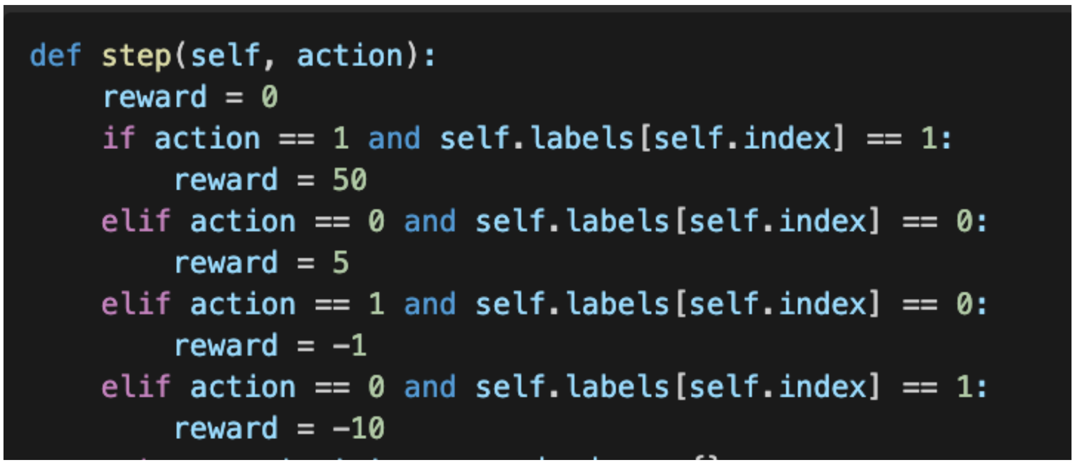
- Continuous features representing equipment metrics (e.g., temperature, vibration).

Reward Structure

- **Correct Maintenance**: +50.
- **Correct No Maintenance**: +5.
- **Unnecessary Maintenance**: -1.
- **Missed Maintenance**: -10.

While this structure is not being claimed to be the best, it was determined to be effective after testing different combinations of rewards and penalties. For the purpose of this study and presentation, we retained this structure to maintain consistency and focus, acknowledging that further tuning or alternative configurations could yield different outcomes. For the purpose of this study and presentation, we adopted this structure to maintain clarity and focus, acknowledging that other configurations may also be viable.

Code Snippet:

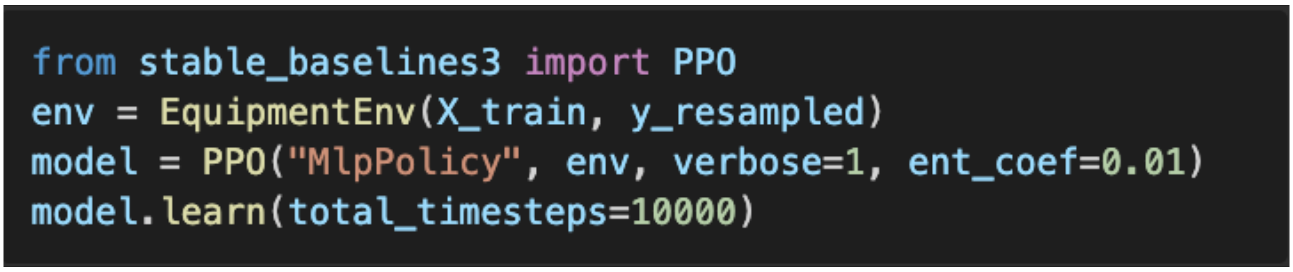


```python
def step(self, action):
    reward = 0
    if action == 1 and self.labels[self.index] == 1:
        reward = 50
    elif action == 0 and self.labels[self.index] == 0:
        reward = 5
    elif action == 1 and self.labels[self.index] == 0:
        reward = -1
    elif action == 0 and self.labels[self.index] == 1:
        reward = -10
```

2.3 Model Training

The Proximal Policy Optimization (PPO) algorithm was used to train the RL agent. The entropy coefficient (ent_coef=0.01) was set to encourage exploration.

Code Snippet:



```python
from stable_baselines3 import PPO
env = EquipmentEnv(X_train, y_resampled)
model = PPO("MlpPolicy", env, verbose=1, ent_coef=0.01)
model.learn(total_timesteps=10000)
```

3. Results and Analysis

3.1 Reward Distribution

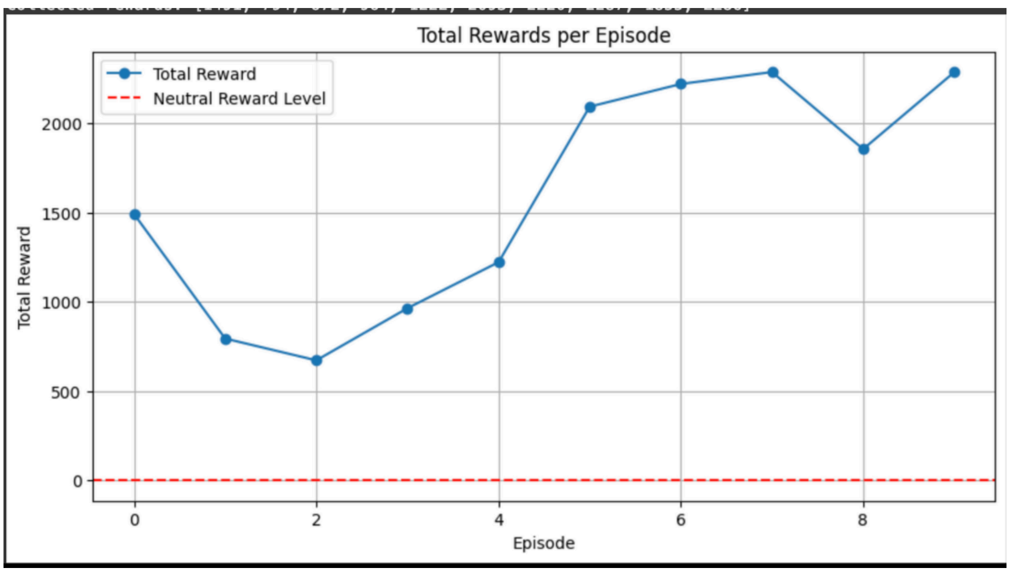
The total rewards per episode were plotted to evaluate the agent's performance. A moving average was applied to smooth fluctuations.

Observations:

1. Rewards showed variability across episodes, with higher rewards in later episodes, indicating learning.
2. The agent avoided unnecessary maintenance while addressing failures effectively.

Graph: Total Rewards Per Episode

- A line plot showed increasing cumulative rewards over time.



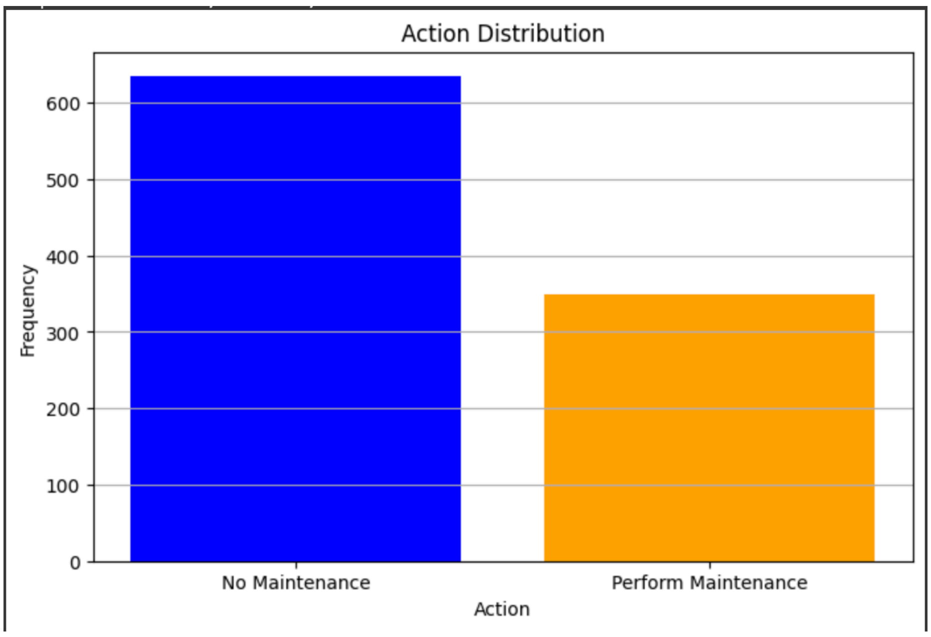
3.2 Action Distribution

The agent’s actions were analyzed:

- **Correct Maintenance**: Frequent when failures occurred.
- **Correct No Maintenance**: Dominant in non-failure cases.

Graph: Action Distribution

- A bar chart revealed balanced decision-making between actions 0 and 1.



4. Discussion

Challenges

1. Reward Balance: Small penalties discouraged unnecessary actions but required careful tuning. Overly harsh penalties for incorrect actions led to overly cautious behavior, while too lenient rewards encouraged inefficient decision-making. The balance was achieved by iterative testing to ensure the agent prioritized critical actions.

2. State Space: Continuous features added complexity, necessitating effective scaling. StandardScaler was used to normalize the features, which helped stabilize the learning process and prevented the agent from being biased toward features with larger ranges.

Recommendations

1. Explore alternative RL algorithms (e.g., DQN) for comparison.
2. Incorporate real-world datasets for practical validation.

5. Conclusion

This study demonstrates the potential of reinforcement learning for predictive maintenance. By designing a custom environment and leveraging PPO, the agent effectively learned to optimize maintenance decisions, balancing costs and equipment performance. Future work will focus on real-world applications and hybrid reward structures.

References

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*.
2. OpenAI Gym Documentation. Retrieved from https://www.gymlibrary.dev/
3. SMOTE: Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Synthetic Minority Over-sampling Technique.