Phase 1: AI/ML Model for Predicting Kubernetes Issues

Team Name: DPN Coders

Team Members: Deepashree K

Selvaraju Nikethna Sri

Priyadharshini M

College: PSG College of Technology, Coimbatore, India

Model 1: Kubernetes Performance Metrics Analysis and Anomaly Detection

File Source: GW2.ipynb

Introduction

This report presents an analysis of Kubernetes performance metrics and resource allocation datasets. The primary objectives include data preprocessing, anomaly detection using machine learning models, and time-series forecasting using LSTM neural networks.

Dataset Description

Two datasets were used:

1. Performance Metrics Dataset (df_metrics): Contains various performance indicators such as CPU usage, memory usage, and network bandwidth usage.
2. Resource Allocation Dataset (df_resources): Provides information about allocated resources for Kubernetes pods.

The first few data points in both datasets are shown below.

	timestamp	pod_name	namespace	cpu_allocation_efficiency
0	01/01/2023 00:00	pod_0	dev	0.038162
1	01/01/2023 00:00	pod_1	default	0.500763
2	01/01/2023 00:00	pod_2	kube-system	0.746726
3	01/01/2023 00:00	pod_3	default	0.526692
4	01/01/2023 00:00	pod_4	prod	0.425342

	memory_allocation_efficiency	disk_io	network_latency
0	0.949259	9.993579	13.722542
1	0.048543	935.792442	55.493953
2	0.447345	328.352359	173.910016
3	0.870251	778.297708	67.395729
4	0.885459	711.181295	91.724730

	node_temperature	node_cpu_usage	node_memory_usage	event_type
0	77.619073	93.177619	37.900532	Warning
1	84.182245	61.442289	5.208161	Error
2	21.295244	55.819311	18.335802	Normal
3	85.028829	78.968463	94.619689	Warning
4	29.157695	52.718141	70.770594	Error

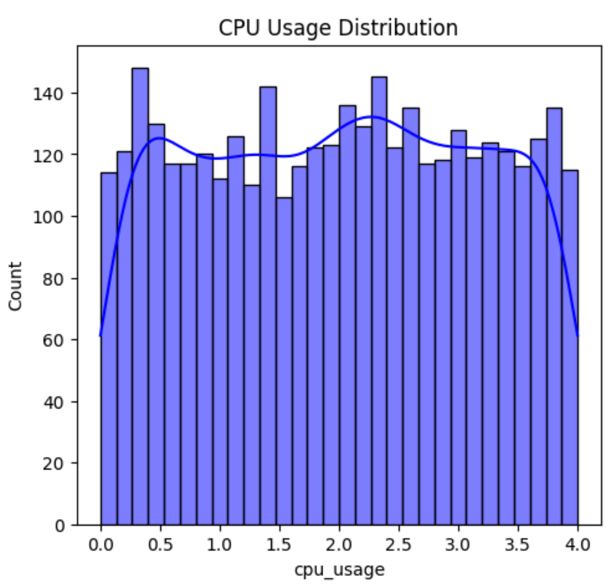
	event_message	scaling_event	pod_lifetime_seconds
0	Killed	False	119648
1	Failed	True	144516
2	Completed	True	68857
3	OOMKilled	True	72080
4	Killed	False	123016

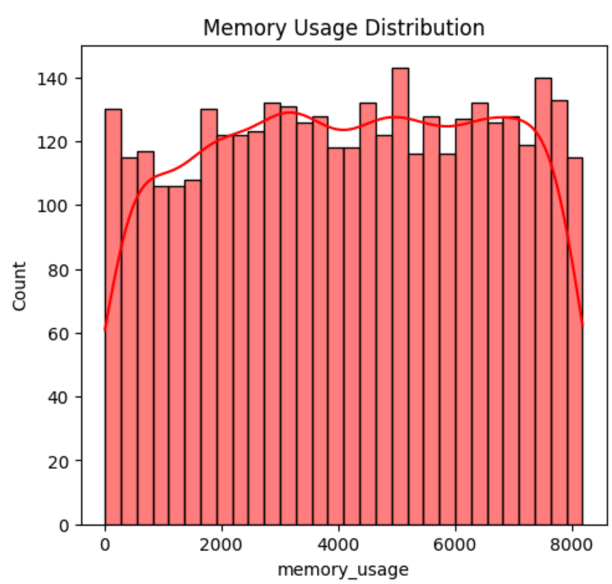
```
  pod_name    namespace  cpu_request  cpu_limit  memory_request  memory_limit  \
0    pod_0          dev     1.569542   3.679152     3174.582783   5134.413852
1    pod_1      default     0.343119   3.722716     3551.459173   3698.349366
2    pod_2  kube-system     0.249271   1.318147     1578.313253   7418.271122
3    pod_3      default     0.311497   2.852595     1392.962372   3628.480705
4    pod_4      default     1.532775   0.521618     2660.192655   5091.497752

   cpu_usage  memory_usage node_name pod_status  restart_count  \
0   3.345496   2135.310365   node_12     Failed              0
1   2.758188   7442.200271   node_18    Unknown              2
2   1.319703   5142.897754    node_7     Failed              2
3   3.752312   2952.449331   node_20     Failed              6
4   0.874224   3382.299355   node_38    Unknown              3

   uptime_seconds deployment_strategy scaling_policy  network_bandwidth_usage
0           76536        RollingUpdate         Manual               459.015733
1           97849        RollingUpdate         Manual               507.770808
2           47370        RollingUpdate         Manual               527.702531
3            5685             Recreate           Auto               473.530315
4            4502             Recreate           Auto               973.928080
```

The datasets were merged based on common columns (pod_name and namespace), and the timestamp column was converted to datetime format for time-series analysis.

Exploratory Data Analysis (EDA)

- Checked for missing values and summary statistics.
- Performed visualizations:
    - Histograms of cpu_usage and memory_usage.
    - Correlation heatmap for numerical features.





Data Preprocessing

- Selected numerical columns (cpu_usage, memory_usage, network_bandwidth_usage).
- Standardized the data using StandardScaler.

Methodology

In this project, we implemented a hybrid approach that combines machine learning-based anomaly detection with deep learning-based time-series forecasting. This methodology enhances the reliability of system resource usage predictions by incorporating anomaly-aware forecasting.

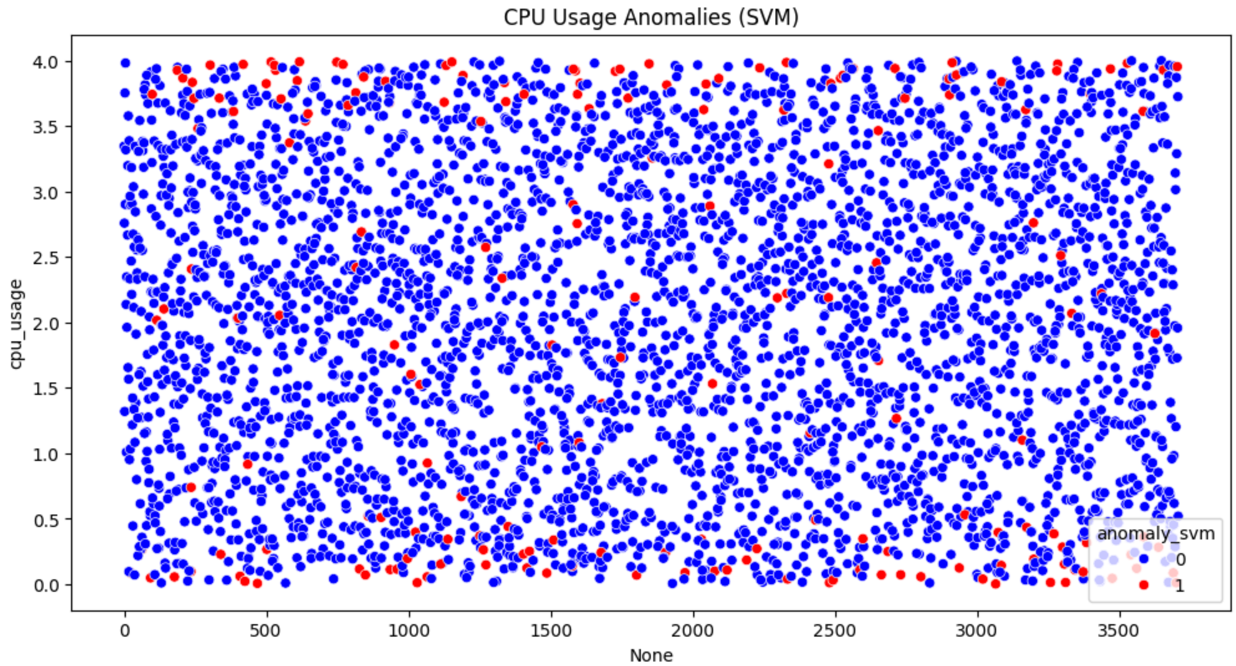
1. Anomaly Detection:

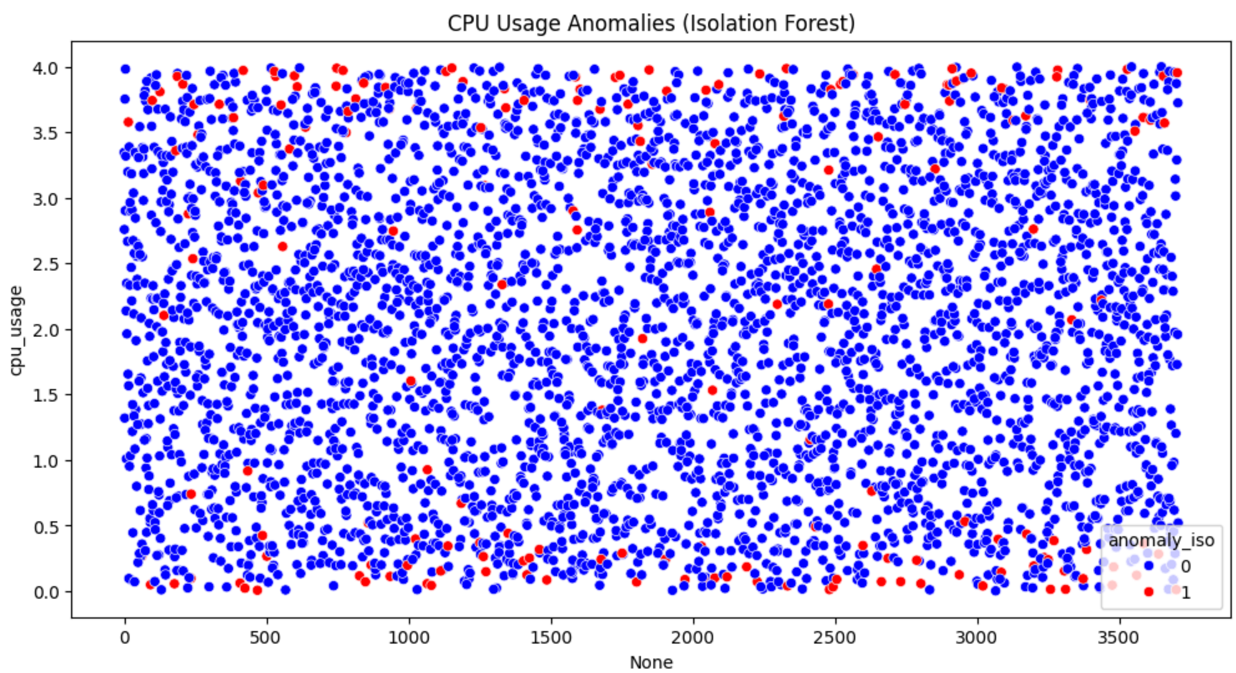
To detect anomalies in CPU, memory, and network bandwidth usage, we used two unsupervised machine learning techniques:

- **One-Class SVM (Support Vector Machine)**: This method models normal behavior and flags deviations as anomalies. We used an RBF kernel and tuned hyperparameters to detect anomalies in system resource usage.

- **Isolation Forest**: This model identifies anomalies by isolating outliers in the dataset. We set a contamination level of 5% to detect unusual resource consumption patterns.

The CPU usage Anomalies detected using both techniques is plotted as shown below.





The anomaly summary is printed as shown below.

```
SVM Anomaly Distribution:
anomaly_svm
0    3521
1     188
Name: count, dtype: int64

Isolation Forest Anomaly Distribution:
anomaly_iso
0    3523
1     186
Name: count, dtype: int64
```

This shows that both techniques detected a similar number of anomalies (~5% of the data), which aligns with the contamination level assumption.

The anomaly scores from both models were combined into a new feature, **anomaly_score**, which was later used as an input feature in time-series forecasting.

2. Time-Series Forecasting with LSTM

To predict future resource usage, we employed an LSTM (Long Short-Term Memory) model. The key steps included:

- **Feature Selection**: The LSTM model was trained using CPU usage, memory usage, network bandwidth usage, and the newly created **anomaly_score** feature.
- **Data Normalization**: MinMaxScaler was used to scale numerical features for better training stability.
- **Sequence Creation**: We created time-step sequences of 50 data points to capture temporal dependencies.
- **LSTM Model Architecture**:
  - Three LSTM layers with ReLU activation
  - Dropout layers to prevent overfitting
  - Adam optimizer with a learning rate of 0.001
  - Mean Squared Error (MSE) as the loss function

The LSTM model was trained for 50 epochs with a batch size of 16 and validated using an 80-20 train-test split.

3. Benefits of the Hybrid Approach

By combining anomaly detection with time-series forecasting, this hybrid approach enhances predictive accuracy and robustness:

- **Improved Forecasting**: The anomaly score helps the LSTM model differentiate between normal fluctuations and abnormal spikes.

- **Anomaly-Aware Predictions**: The forecasting model can learn from past anomalies and adjust future predictions accordingly.

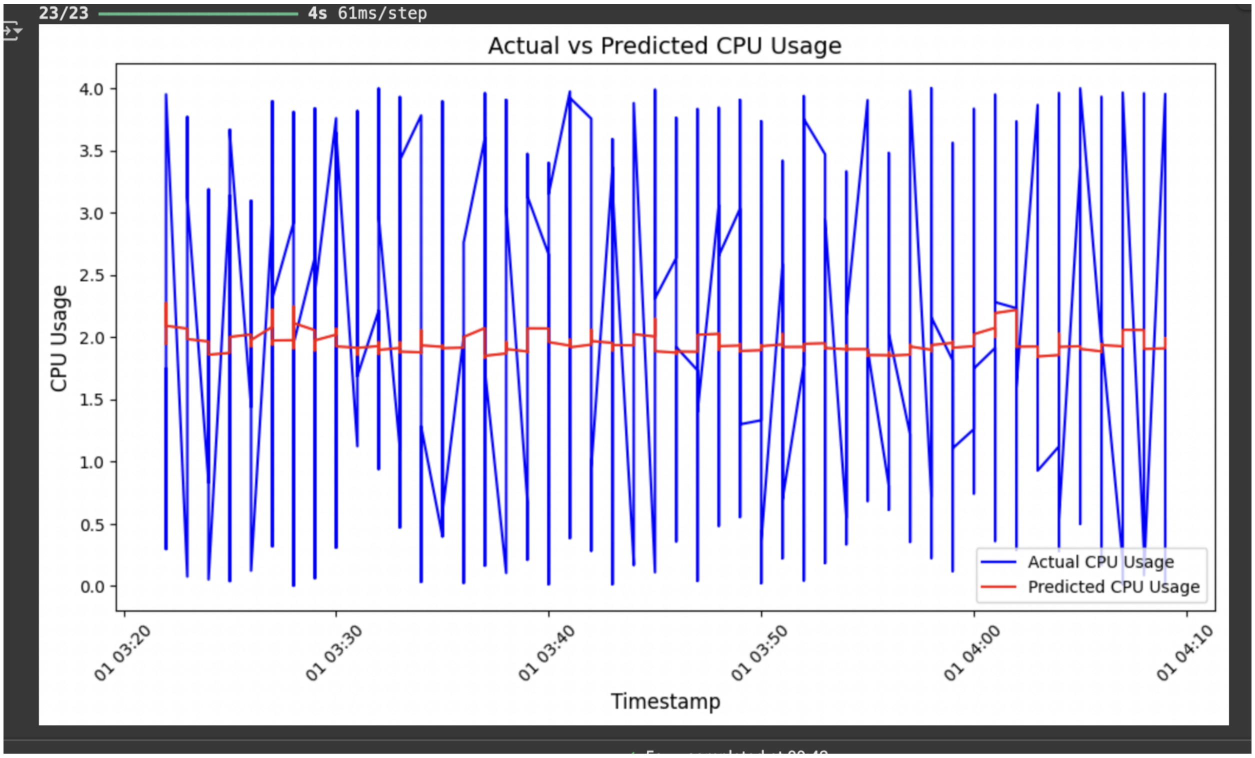
- **Better System Monitoring**: Real-time detection of anomalies alongside forecasting provides a proactive approach to resource management.

This hybrid technique successfully integrates anomaly detection and forecasting, making it suitable for real-time system monitoring and optimization.

Model Evaluation

- Plotted training and validation loss curves.
- Predicted future values and compared them with actual values.
- Visualized the actual vs predicted CPU usage over time.

The actual vs predicted CPU usage over time is shown below.



Below is the output result from testing the model with sample data points.

```
Enter CPU Usage (0 to 1): 0.7
Enter Memory Usage (0 to 1): 0.5
Enter Network Usage (0 to 1): 0.6
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 874ms/step

Predicted Future Resource Usage:
CPU Usage: 0.4233
Memory Usage: 0.2533
Network Usage: 0.5386
```

Deployment & Model Saving

- Saved the trained MinMaxScaler and LSTM model for future inference.
- Provided a mechanism to take real-time CPU, memory, and network usage inputs for prediction.

Conclusion

This project successfully demonstrates the effectiveness of a hybrid approach by combining One-Class SVM and Isolation Forest for anomaly detection with LSTM for time-series forecasting of Kubernetes resource usage. The integration of anomaly-aware forecasting enhances system monitoring by identifying unusual patterns and improving predictive accuracy. Through the practical implementation of machine learning and deep learning techniques, this approach enables proactive resource management. Future improvements include hyperparameter tuning, incorporating additional features, and integrating real-time anomaly detection to further optimize system performance.

Model 2: Pod status prediction using Random Forest classifier

File Source: GW1.ipynb

1. Dataset Used

The dataset used for **Kubernetes Resource Allocation Dataset**, which contains information about pod resource usage, deployment strategies, scaling policies, and pod statuses. The dataset includes the following features:

- **Numerical Features**: CPU request, CPU limit, memory request, memory limit, CPU usage, memory usage, restart count, uptime seconds, network bandwidth usage.
- **Categorical Features**: Deployment strategy, scaling policy, and pod status (target variable).
- **Target Variable:Pod status**, which is encoded into numerical values for model training.

2. Model Used

The **Random Forest Classifier** from sklearn.ensemble was selected for this task due to its robustness in handling both numerical and categorical features, as well as its capability to deal with complex decision boundaries.

3. Data Preprocessing

Several preprocessing steps were performed:

- **Dropping Irrelevant Columns**: pod_name, namespace, and node_name were removed as they do not contribute to the prediction.
- **Handling Missing Values**:
  - Numerical features were filled with their median values.
  - Categorical features (deployment_strategy, scaling_policy) were filled with their mode.
- **Encoding Categorical Variables**:
  - pod_status was encoded using LabelEncoder().
  - deployment_strategy and scaling_policy were mapped to numerical values .
- **Feature and Target Variable Selection**:
  - X: All numerical and encoded categorical variables.
  - y: The encoded pod_status.
- **Train-Test Split**: The dataset was split into training (80%) and testing (20%) sets using train_test_split().

5. Model Training

A **RandomForestClassifier** with 100 estimators was trained on the dataset. The classifier was fitted using the training data (X_train, y_train) to learn the relationships between the features and the pod status.

6. Model Evaluation

The trained model was evaluated on the test set (X_test, y_test). The following metrics were used:

- **Accuracy**: accuracy_score(y_test, y_pred)
- **Classification Report**: Includes precision, recall, and F1-score for each pod status category.

The model achieved a satisfactory accuracy score, indicating its ability to generalize well to unseen data.

```
Accuracy: 0.203
              precision    recall  f1-score   support

           0       0.21      0.24      0.22       599
           1       0.17      0.19      0.18       565
           2       0.21      0.17      0.19       630
           3       0.21      0.23      0.22       604
           4       0.22      0.19      0.20       602

    accuracy                           0.20      3000
   macro avg       0.20      0.20      0.20      3000
weighted avg       0.20      0.20      0.20      3000
```

7. Model Prediction for Unseen Data and Justification

A new pod data instance was introduced with specific resource allocations and deployment settings. The model predicted its pod status based on learned patterns. The predicted status was mapped back using the label encoding dictionary.

```
# Example usage:
new_pod = {
    "cpu_request": 1.5, "cpu_limit": 3.6, "memory_request": 3200,
    "memory_limit": 5000, "cpu_usage": 3.3, "memory_usage": 2100,
    "restart_count": 0, "uptime_seconds": 76536,
    "deployment_strategy": "RollingUpdate", "scaling_policy": "Manual",
    "network_bandwidth_usage": 450
}
```

```
Predicted Pod Status: Pending
```

Why the Pod is in "Pending" Status?

1. **Resource Issues:**
   - The pod's CPU usage (3.3) is higher than its request (1.5), which may cause delays.
   - Memory usage (2100) is close to the limit (5000), indicating possible contention.
   - High network bandwidth usage (450) could also affect scheduling.

2. **No Restarts & Long Uptime:**
   - The pod hasn’t restarted, meaning no failures occurred.
   - It has been running for 21 hours, possibly waiting for resources.

3. **Deployment & Scaling:**
   - "RollingUpdate" strategy may delay scheduling until older pods are replaced.
   - "Manual" scaling requires admin intervention for more resources.

4. **Model's Decision:**
   - The Random Forest model predicts based on past patterns.
   - Key factors like CPU, memory, and scaling policy likely influenced the "Pending" status.