Anomaly Detection and Failure Prediction in Kubernetes Clusters

Operations in Kubernetes clusters usually experience issues like pod crashes, resource bottlenecks, network failures, and service disruptions. To anticipate these challenges ahead of time, we introduce an integrated approach consisting of anomaly detection and failure prediction by machine learning models. This document summarizes our procedure, spanning data preprocessing, feature engineering, training the model, and evaluation.

Data Preprocessing and Category Encoding

The initial step in our method is the conversion of categorical labels into numeric values through category encoding. The problem is specifically in the issue_type column, which represents the nature of failure. Numerical labels are given to it as follows:
0 → Node Failure
1 → Resource Exhaustion
2 → Network Issue
3 → Service Disruption
This makes the model capable of interpreting and analyzing various types of failures properly.

Feature Selection

We employ a range of independent variables that describe the state of the system, gathering important metrics from the Kubernetes platform. The chosen features are:
1) cpu_usage
2) memory_usage
3) disk_usage
4) network_in (MB/sec)
5) network_out (MB/sec)
6) node_status
7) pod_count
8) packet_loss (%)

9) api_request_count
10) api_error_rate (%)
These features give a wide perspective on the performance of the cluster, and the model uses them to forecast and classify prospective failures.

Time-Series Analysis and Lagged Features

To reflect historical trends, we create lagged features with the shift() function. We create three lagged versions of key measures such as CPU and memory usage. This assists the model in identifying patterns over time, enhancing forecasting accuracy on time-series data.

Anomaly Detection Using Isolation Forest

We use Isolation Forest to identify anomalies, which reflect abnormal behavior in the cluster. Isolation Forest is good at finding rare and abnormal instances by isolating data with randomly created trees. The contamination parameter is 0.05, so 5% of the data is marked as anomalies.
Anomalies are marked as -1 and encoded to binary for ease (1 for anomaly, 0 for normal).
The is_anomaly column marks rows that are identified as anomalies, and this is a key input to subsequent failure analysis.

Model Training with Random Forest

For prediction of failures, we employ a Random Forest Classifier, an ensemble learning method which aggregates multiple decision trees to achieve stable and reliable predictions. The model is set up with the following hyperparameters:
1) 300 trees for adequate model stability
2) Depth of 20 to capture complex relationships
3) Random seed for reproducibility
The model is trained on the labeled past Kubernetes metrics and the respective failure types. The model can then make the prediction of the failure type based on new, unseen data after training.

Evaluation Metrics

We use the following evaluation metrics to judge the performance of the model:

1) Accuracy: It measures the ratio of correctly classified instances.
2) F1 Score: It gives a balance between precision and recall and is particularly useful with imbalanced datasets.
3) Precision: This measures the ratio of correctly predicted positive instances.
4) Recall: Quantifies the capacity of the model to identify real failures.

Real-Time Monitoring and Deployment

The trained model is deployed in a real-time monitoring setup with the help of FastAPI. The API takes real-time Kubernetes metrics, performs anomaly detection, and returns predictions on the probability of failures. This provides early warnings and active resolution of impending problems.

Conclusion

We introduced a new method leveraging anomaly detection and failure prediction that contributes to improving the robustness of Kubernetes clusters. Our approach incorporates time-series analysis, Isolation Forest, and Random Forest models to help identify and alleviate failures in a scalable and accurate manner. A methodology like this is perfect for use in production as it promotes cluster stability and minimize downtime.

Results

```
✅ Node Failure Prediction Accuracy: 0.8162
✅ F1 Score: 0.8097
✅ Precision: 0.8100
✅ Recall: 0.8162

📄 Classification Report:
                   precision    recall  f1-score   support

       node_failure       0.77      0.77      0.77        95
resource_exhaustion       0.90      0.95      0.92       110
      network_issue       0.83      0.95      0.89       104
 service_disruption       0.73      0.58      0.64        99

           accuracy                           0.82       408
          macro avg       0.81      0.81      0.81       408
       weighted avg       0.81      0.82      0.81       408
```

Potential Improvements

1) **Inculcating SHAP, Optuna/** GridSearchCV, LIME, Prometheus, Grafana
2) Instead of global oversampling, use cluster-based SMOTE or Adaptive Synthetic Sampling (ADASYN) to prevent synthetic noise in minority classes.