METHODOLOGY AND EXPERIMENTS

(Anomaly Detection in Cloud-Based Systems)

DIVYANSH AGRAWAL

SUPERVISOR: ANDRIY MIRANSKYY

JULY 12, 2025

AIM OF THE PROJECT

The project aims to build and compare machine learning models for anomaly detection in structured system logs. The aim is to utilize automated detection of unusual behavior in high-speed log data across various datasets that simulate different computing environments. Key goals are:

- To feature-extract from raw log data the structural and semantic information
- To apply supervised learning for labeled datasets and unsupervised learning for unlabeled datasets
- Testing multiple machine learning algorithms across varied system logs
- Using suitable methods to prepare data for anomaly detection
- Improving model understanding through analysis of important features

RESPONSE AND INDEPENDENT VARIABLES

Response variables are the binary anomaly labels assigned to each log entry:

- BGL Dataset uses supervised binary labels (0 normal, 1 anomaly), with 143 anomalies in 2000 entries.
- HDFS, OpenStack, Thunderbird, and Zookeeper datasets are used in an unsupervised setup, without label guidance during training.

The feature extraction process creates 91-154 features per dataset, categorized as:

1. Basic Text Features:
    - log_length: Character count of log content
    - word_count: Number of words in log entry
    - char_count_no_spaces: Character count excluding spaces
2. Dataset-Specific Pattern Features:
    - Hardware patterns (cache, core, instruction, parity, exception, alignment)
    - System patterns (CIOD, socket, message, directory, chdir)
    - Error patterns (CE sym, failed, error, interrupt, TLB, storage)
    - Numeric patterns (hex addresses, IP addresses, port numbers, core numbers)
3. Event Template Features:
    - event_template_length: Template complexity
    - event_template_word_count: Template detail level
    - has_parameters: Parameter presence
    - parameter_count: Number of parameters

4. Component and Level Features:
    - Component dummy variables (dataset-specific)
    - Level dummy variables (ERROR, FATAL, INFO, SEVERE, WARNING)
    - Event ID dummy variables (common events only)
5. Temporal Features:
    - hour_of_day: Time-based patterns
    - day_of_week: Day-based patterns
    - is_weekend: Weekend vs weekday patterns
    - is_business_hours: Business hours patterns

FACTORS AND LEVELS

Factors tested in this experiment include:

1. Learning Paradigm:
    - Level 1: Supervised Learning (BGL dataset)
    - Level 2: Unsupervised Learning (HDFS, OpenStack, Thunderbird, Zookeeper)
2. Algorithm Types:
    - Level 1: Random Forest (supervised)
    - Level 2: Neural Network (supervised)
    - Level 3: Isolation Forest (unsupervised)
    - Level 4: Autoencoder (unsupervised)
3. Feature Engineering Approaches:
    - Level 1: Generic pattern extraction (initial approach)
    - Level 2: Dataset-specific pattern optimization (final approach)
4. Model Parameters:
    - Random Forest: n_estimators=200, max_depth=15, class_weight='balanced'
    - Neural Network: 4 hidden layers (256,128,64,32), dropout=0.2-0.4, batch_norm
    - Isolation Forest: contamination=0.1, n_estimators=100
    - Autoencoder: encoding_dim=64, epochs=100, batch_size=32

EXPERIMENTAL DESIGN

a) Data Preprocessing and Feature Extraction

The experimental design follows a systematic approach to feature engineering:

Phase 1: Initial Generic Feature Extraction

- Applied common log analysis patterns across all datasets
- Extracted 100+ features including generic hardware, system, and error patterns
- Included temporal, component, and event template features
- Resulted in sparse feature matrices with many zero-value columns

Phase 2: Dataset-Specific Optimization

- Analyzed actual log content for each dataset to identify recurring patterns
- Removed generic patterns that don't appear in specific datasets
- Focused on domain-specific terms (e.g., BGL: CIOD, double-hummer; HDFS: block, replica)
- Eliminated features with all-zero values or minimal variance
- Reduced feature count by 15-25% while improving model performance

Phase 3: Data Quality Enhancement

- Handled missing values by filling with appropriate defaults
- Converted boolean strings to integers for model compatibility
- Applied feature scaling (StandardScaler) for neural networks
- Ensured all features are numeric and properly formatted

b) Train/Test Split Strategy

Supervised Learning (BGL):

- Stratified split: 80% training, 20% testing
- Maintains class distribution across splits
- Random state fixed for reproducibility

Unsupervised Learning:

- No explicit train/test split required
- Models learn normal patterns from entire dataset
- Anomaly detection based on deviation from learned patterns

c) Cross-Validation and Model Selection

Supervised Models:

- Early stopping for neural networks (patience=20)
- Learning rate reduction on plateau
- Class weight balancing for imbalanced data

Unsupervised Models:

- Contamination parameter tuning (0.001-0.1)
- Threshold selection based on reconstruction error percentiles
- Model selection based on anomaly rate consistency

EXPERIMENT PERFORMANCE AND REVISIONS

A series of experiments were conducted to optimize feature extraction and model performance:

Experiment 1: Generic Feature Extraction

- Applied uniform feature extraction across all datasets
- Results: High feature dimensionality, poor model performance
- Issues: Sparse features, irrelevant patterns, computational overhead
- Decision: Move to dataset-specific optimization

Experiment 2: Dataset-Specific Feature Engineering

- Analyzed each dataset's unique characteristics
- BGL: Focused on hardware and supercomputer-specific patterns
- HDFS: Emphasized distributed system and block management patterns
- OpenStack: Cloud infrastructure and service patterns
- Thunderbird: Email system and communication patterns
- Zookeeper: Distributed coordination and consensus patterns
- Results: Improved model performance, reduced feature count
- Decision: Adopt optimized approach for all datasets

Experiment 3: Model Architecture Optimization

- Random Forest: Optimized tree parameters and class weights
- Neural Network: Tested different architectures and regularization
- Isolation Forest: Tuned contamination and ensemble size
- Autoencoder: Varied encoding dimensions and training parameters
- Results: Identified optimal configurations for each model type

Experiment 4: Unsupervised Model Robustness

- Tested models on datasets with no known anomalies
- Implemented graceful handling of edge cases
- Added AUC calculation safeguards for single-class scenarios
- Results: Robust models that handle various data distributions

MEASURING MODEL PERFORMANCE

Performance metrics are selected based on the learning paradigm:

Supervised Learning Metrics (BGL):

- AUC (Area Under ROC Curve): Primary metric for imbalanced classification
- Precision: Accuracy of positive predictions
- Recall: Sensitivity to actual anomalies
- F1-Score: Harmonic mean of precision and recall
- Feature Importance: Model interpretability

Unsupervised Learning Metrics:

- Anomaly Detection Rate: Percentage of samples flagged as anomalous
- Reconstruction Error: Mean squared error for autoencoders
- Isolation Score: Average path length for isolation forest
- Model Stability: Consistency across different runs

Cross-Dataset Comparison:

- Feature Efficiency: Performance per feature count
- Computational Cost: Training and inference time
- Model Interpretability: Feature importance and decision explanations
- Robustness: Performance across different data distributions

METHODOLOGICAL INSIGHTS

- Dataset-specific feature engineering significantly improves performance
- Supervised learning outperforms unsupervised when labels are available
- Model selection should consider both performance and interpretability
- Robust error handling is essential for production deployment
- Feature optimization reduces computational cost without sacrificing performance