Log Dataset Collection - Data Dictionary

Dataset Overview

This collection contains system logs from multiple sources used for developing and testing model drift detection algorithms. The datasets represent various system types and exhibit different patterns of behavior, errors, and potential drift indicators.

Dataset Statistics

Volume Statistics

- HDFS: 2,000 entries
- Apache: 52,004 entries
- HealthApp: 253,395 entries
- Additional Systems:
  - BGL (Blue Gene/L Supercomputer)
  - HPC (High-Performance Computing)
  - Linux System Logs
  - Mac System Logs

Common Fields

Timestamp

- **Description**: Log entry generation time
- **Data Type**: DateTime
- **Format Variations**:
  - HDFS: `YYMMDD HHMMSS`
  - Apache: `[Day Mon DD HH:MM:SS YYYY]`
  - HealthApp: `YYYYMMDD-HH:MM:SS:mmm`
  - BGL: `YYYY.MM.DD-HH.MM.SS.mmm`
  - HPC: `YYYY-MM-DD HH:MM:SS`
  - Linux/Mac: `MMM DD HH:MM:SS`

- **Standardization**:
  - Conversion to ISO 8601 format
  - UTC timezone alignment
  - Millisecond precision where available

Component

- **Description**: Source system component
- **Data Type**: String
- **Examples by System**:
  - HDFS: `DataNode` , `NameNode` , `FSNamesystem`
  - Apache: `mod_ssl` , `mod_auth` , `core`
  - HealthApp: `ActivityMonitor` , `SensorManager`
  - BGL: `KERNEL` , `APP` , `NETWORK`
  - HPC: `scheduler` , `compute_node` , `storage`
  - Linux/Mac: `kernel` , `daemon` , `system`

Severity

- **Description**: Message importance level
- **Data Type**: Categorical
- **Standardized Levels**:
  - ERROR: Critical issues
  - WARNING: Potential problems
  - INFO: Normal operations
  - DEBUG: Detailed information
- **System-Specific Mappings**:
  - HDFS: Direct mapping
  - Apache: `error` → ERROR, `warn` → WARNING
  - HealthApp: Derived from context
  - BGL: `FATAL` → ERROR, `SEVERE` → WARNING
  - HPC: System-specific severity mapping
  - Linux/Mac: Standard syslog levels

Message

- **Description**: Log content
- **Data Type**: String

- **Characteristics**:
    - Structured patterns
    - Variable content
    - System-specific formatting
    - Error codes and descriptions

Feature Engineering

1. Temporal Features

```python
def extract_temporal_features(log_data, window_size='1H'):
    """
    Extract time-based features from log data

    Parameters:
    - log_data: DataFrame with log entries
    - window_size: Time window for aggregation

    Returns:
    - Dictionary of temporal features
    """
    features = {
        'event_count': count_events_in_window(log_data, window_size),
        'error_rate': calculate_error_rate(log_data, window_size),
        'component_activity': analyze_component_activity(log_data, window_size),
        'pattern_frequency': analyze_pattern_frequency(log_data, window_size)
    }
    return features
```

2. Component Features

```python
def extract_component_features(log_data):
    """
    Extract component-related features

    Parameters:
    - log_data: DataFrame with log entries

    Returns:
    - Dictionary of component features
    """
    features = {
        'active_components': identify_active_components(log_data),
        'interaction_patterns': analyze_component_interactions(log_data),
        'error_distribution': analyze_component_errors(log_data),
        'state_transitions': analyze_state_changes(log_data)
    }
    return features
```

3. Message Features

```python
def extract_message_features(log_data):
    """
    Extract features from log messages

    Parameters:
    - log_data: DataFrame with log entries

    Returns:
    - Dictionary of message features
    """
    features = {
        'message_length': calculate_message_lengths(log_data),
        'pattern_complexity': analyze_message_patterns(log_data),
        'error_patterns': extract_error_patterns(log_data),
        'variable_content': identify_variable_content(log_data)
    }
    return features
```

Drift Detection Features

1. Distribution Features

- Message pattern distributions
- Component activity distributions
- Error rate distributions
- Time interval distributions

2. Pattern Features

- Sequential patterns
- Periodic patterns
- Anomalous patterns
- State transition patterns

3. Performance Features

- Response time patterns
- Resource usage patterns
- Error frequency patterns
- Component load patterns

Quality Metrics

1. Data Quality

- Completeness (missing values)
- Consistency (format adherence)
- Accuracy (value validation)
- Timeliness (temporal order)

2. Feature Quality

- Feature completeness
- Feature stability
- Feature correlation
- Feature importance

3. Pattern Quality

- Pattern stability
- Pattern significance
- Pattern evolution
- Pattern anomalies

Usage Guidelines

1. Data Loading

```python
def load_log_data(log_type, input_path):
    """
    Load and validate log data

    Parameters:
    - log_type: Type of log data
    - input_path: Path to log file

    Returns:
    - DataFrame with validated log data
    """
    # Load data
    data = read_log_file(input_path)

    # Validate format
    validate_log_format(data, log_type)

    # Extract fields
    parsed_data = parse_log_fields(data, log_type)

    # Standardize fields
    standardized_data = standardize_fields(parsed_data)

    return standardized_data
```

2. Feature Extraction

```python
def extract_features(log_data, config):
    """
    Extract all relevant features

    Parameters:
    - log_data: DataFrame with log entries
    - config: Feature extraction configuration

    Returns:
    - Dictionary of extracted features
    """
    features = {
        'temporal': extract_temporal_features(log_data, config['window_size']),
        'component': extract_component_features(log_data),
        'message': extract_message_features(log_data),
        'performance': extract_performance_features(log_data)
    }
    return features
```

3. Drift Analysis

```python
def analyze_drift(current_features, baseline_features):
    """
    Analyze drift between current and baseline features

    Parameters:
    - current_features: Current feature set
    - baseline_features: Baseline feature set

    Returns:
    - Dictionary of drift analysis results
    """
    results = {
        'distribution_drift': analyze_distribution_drift(current_features, baseline_fea
        'pattern_drift': analyze_pattern_drift(current_features, baseline_features),
        'performance_drift': analyze_performance_drift(current_features, baseline_featu
    }
    return results
```

References

1. System Log Analysis Documentation
2. Feature Engineering Documentation
3. Drift Detection Documentation
4. Data Quality Guidelines