DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Exploration of Machine Learning Dataset Compression on Time-Series Dataset

Onur YILMAZ

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Exploration of Machine Learning Dataset Compression on Time-Series Dataset

Author:	Onur YILMAZ
Supervisor:	M.Sc. Dai Liu
Advisor:	Prof. Dr. Martin Schulz
Submission Date:	06 November 2024

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 06 November 2024 Onur YILMAZ

To my parents, Aysel & Hüsamettin Yılmaz

Acknowledgments

First and foremost, I would like to express my sincere gratitude to Dai Liu, who supervised me from start to finish in writing this thesis. I am also deeply thankful to Professor Martin Schulz for accepting me to undertake this thesis under his chair. I wholeheartedly believe that the experience and knowledge I have gained through this work in machine learning will play a significant role in the rest of my career.

I would also like to thank all the instructors whose courses I attended at TUM. The academic education I received at TUM has greatly broadened my vision, both professionally and personally. It has given me the opportunity to better analyze and decide on the career path I should follow in life.

I am equally grateful to my drum teacher, Barış Buran, for everything he has contributed to my life. Through him, I met the drums, which have become my closest friend, and strengthened my bond with music. Most importantly, he taught me the importance of being disciplined and patient when learning new things.

Of course, I wish to convey my deepest appreciation to my father, mother, brother, and sister. With their unwavering belief in me and all their support, I overcame every difficulty. Without them, achieving these successes would have been nearly impossible. I feel extremely fortunate to have such a family.

Lastly, while the journeys we are currently undertaking are important, what truly matters are the paths we have taken to reach them; after all, everyone's starting line in life is different.

I have no doubt that when I read these words years from now, I will say, "I'm glad I walked these paths."

Great achievements require great sacrifices.

Sincerely,

Onur

* *Benden her türlü desteği hiçbir zaman esirgemeyen ve her zaman daha iyisi için çabalamamda büyük emekleri olan anneme, babama, abim Naif'e ve kız kardeşim Melisa'ya sonsuz teşekkür ediyorum.*

Abstract

The exponential growth of high-dimensional, large-scale time-series data in High-Performance Computing (HPC) environments poses significant challenges for storage, processing, and real-time analysis. Efficient dataset compression techniques are crucial to address these challenges, enabling machine learning models to operate effectively without compromising performance. This thesis introduces and evaluates innovative compression methodologies tailored for HPC operational data analytics, specifically targeting application classification and anomaly detection tasks.

A key contribution is the development of an entropy-based Random Forest compression method. This technique leverages predictive entropy to identify and retain the most informative samples, balancing high-confidence and high-uncertainty data points. Consequently, it preserves the data's core structure and variability, achieving significant compression while maintaining high model accuracy.

Additionally, the thesis explores K-Means clustering combined with stratified sampling as an alternative compression approach. This method groups similar data points into clusters and selects representative centroids, ensuring class balance. It effectively reduces dataset size while retaining high accuracy, though it is more computationally intensive than the entropy-based method.

For baseline comparison, Principal Component Analysis (PCA) was examined. While PCA effectively reduced dimensionality by retaining components that explained a substantial portion of the variance, it led to notable performance degradation and substantially altered feature and class distributions.

The entropy-based Random Forest compression method outperforms traditional techniques like PCA and offers competitive performance relative to K-Means clustering, especially in scenarios with class imbalance and high-dimensional feature spaces. This research advances dataset compression strategies for HPC environments, demonstrating that entropy-driven methods significantly enhance computational efficiency and storage management while maintaining high model performance. Future work will focus on automating threshold selections, extending the methodology to other machine learning models, and better integrating temporal dependencies to further improve compression outcomes.

Contents

1 Introduction

1.1 Background and Motivation

The rapid advancement of technology has led to an unprecedented growth in data generation across various domains, including finance, healthcare, transportation, and high-performance computing (HPC) systems. In HPC environments, vast amounts of operational data are collected from numerous sensors and system logs to monitor and optimize system performance. This data, often in the form of high-dimensional time-series, poses significant challenges in terms of storage, processing, and analysis [3].

Machine learning models are increasingly employed to extract valuable insights from such large-scale datasets. However, the sheer volume and high dimensionality of the data can lead to increased computational costs, longer training times, and difficulties in real-time processing. These challenges necessitate the development of efficient data reduction techniques that can compress datasets without significant loss of essential information.

Traditional methods, such as downsampling and aggregation, often result in the loss of critical temporal patterns and may not be suitable for tasks requiring high accuracy. Therefore, there is a pressing need for advanced compression techniques that preserve the integrity and predictive power of the data while significantly reducing its size.

This thesis is motivated by the need to address these challenges in the context of application classification and anomaly detection tasks within the HPC-ODA (High-Performance Computing Operational Data Analytics) dataset collection. By exploring novel dataset compression methods tailored to the specific characteristics of the HPC-ODA datasets, we aim to improve the efficiency and effectiveness of machine learning models operating on large-scale time-series data.

1.2 Problem Statement

High-performance computing systems are integral to solving complex scientific and engineering problems that require substantial computational resources. These systems generate massive amounts of operational data through continuous monitoring of system metrics, sensor readings, and application logs. The data collected is often high-dimensional and sampled at high frequencies, resulting in large-scale time-series datasets. Analyzing such voluminous data poses significant challenges for machine learning applications in terms of storage, processing power, and computational efficiency.

Traditional machine learning models face several issues when dealing with large-scale, high-dimensional datasets in HPC environments. First, computational complexity becomes a significant hurdle; training models on large datasets demands extensive computational resources and time, which may not be feasible in real-world HPC operations where timely insights are crucial [18]. Second, there are storage limitations, since storing massive datasets requires substantial storage capacity, leading to increased costs and potential bottlenecks in data retrieval and management [10]. Third, the curse of dimensionality affects model performance, as high-dimensional data can lead to overfitting, reduced generalization, and difficulties in identifying relevant patterns or features [8]. Additionally, real-time processing requirements pose a challenge; many HPC applications require real-time or near-real-time analysis to detect anomalies, optimize performance, or make predictive adjustments, but large datasets hinder the ability to process data quickly enough to meet these demands. Finally, data transmission bottlenecks can occur in distributed HPC environments, where transmitting large amounts of data between nodes or to centralized processing units causes network congestion and latency issues.

In the specific context of application classification and anomaly detection within the HPC-ODA dataset collection [3], these challenges are compounded. The HPC-ODA datasets provide comprehensive data acquired from production HPC systems, representing real-world use cases in Operational Data Analytics (ODA) aimed at improving reliability and energy efficiency. The datasets encompass monitoring sensor data collected from various components of different HPC systems, captured using lightweight monitoring frameworks. The collection includes six distinct segments, each designed for specific ODA tasks, such as Anomaly Detection, Power Consumption Prediction, Application Classification, Infrastructure Management, Cross-Architecture Analysis, and the DEEP-EST Dataset.

Within this context, additional challenges include imbalanced class distributions, as certain applications or anomalies may occur less frequently, leading to datasets that can bias machine learning models toward majority classes. The data also exhibits complex temporal dependencies, with intricate patterns over time that are critical for accurate classification and detection but difficult to capture and preserve during compression. Moreover, there is a need for high accuracy, since misclassification of applications or failure to detect anomalies can lead to suboptimal resource allocation, reduced system performance, or even catastrophic failures in HPC systems. As HPC systems continue to scale up, scalability issues become more pronounced, exacerbating the existing challenges and necessitating scalable solutions for data handling and model training.

Given these challenges, the problem addressed in this thesis is the development of efficient and effective dataset compression methods that can significantly reduce dataset size while preserving essential information. Specifically, the methods should:

1. **Significantly reduce dataset size** to alleviate storage and computational burdens without overly compromising data integrity.

2. **Preserve essential information**, including critical features, temporal patterns, and class distributions necessary for accurate machine learning predictions in application classification and anomaly detection tasks.

3. **Enhance computational efficiency** by enabling faster model training and inference, facilitating real-time or near-real-time processing capabilities required in HPC environments.

4. **Address class imbalance and high dimensionality** by incorporating strategies to handle imbalanced datasets and reduce dimensionality while minimizing the loss of important information.

5. **Ensure scalability and practicality** by developing methods that can scale with the increasing size of HPC systems and be practically implemented without prohibitive computational costs.

Traditional compression techniques fall short in meeting these requirements due to their inherent limitations in preserving the nuanced information embedded in HPC operational data. Therefore, there is a critical need for innovative compression methods specifically tailored to the characteristics of HPC-ODA datasets and the demands of machine learning tasks within this domain.

1.3 Objectives of the Thesis

The primary objectives of this thesis are:

1. To investigate and develop novel dataset compression techniques that effectively reduce the size of large-scale, high-dimensional time-series data without substantial loss of information.

2. To apply and evaluate entropy-based Random Forest compression methods and K-Means clustering combined with stratified sampling within the HPC-ODA dataset collection.

3. To assess the impact of these compression techniques on the performance of machine learning models in application classification and anomaly detection tasks.

4. To compare the proposed methods with traditional compression techniques, such as Principal Component Analysis (PCA), in terms of compression ratio, computational efficiency, and model accuracy.

5. To provide insights and recommendations for practical implementation of dataset compression in real-world HPC systems.

1.4 Proposed Solution

This thesis proposes to tackle the problem by exploring and developing advanced dataset compression techniques, namely **entropy-based Random Forest compression** and **K-Means clustering combined with stratified sampling**. The entropy-based Random Forest compression method leverages predictive uncertainties to select informative samples for dataset compression, effectively reducing data size while maintaining model performance. K-Means clustering, combined with stratified sampling, aims to maintain class balance and preserve data diversity by selecting representative samples from clusters.

By intelligently selecting and preserving the most informative data samples and features, these methods seek to reduce dataset size without significant loss of essential information. The proposed solutions are designed to improve data manageability and computational efficiency, enabling faster model training and inference. Furthermore, they address class imbalance and high dimensionality challenges by incorporating strategies that minimize information loss while reducing data volume.

Through comprehensive evaluation on the HPC-ODA dataset collection, this thesis aims to demonstrate the effectiveness of the proposed methods in achieving significant data size reduction with minimal loss in model performance. By providing practical recommendations for implementing dataset compression in real-world HPC systems, the research contributes to advancing machine learning techniques capable of handling large-scale, high-dimensional data in HPC environments.

This thesis applies advanced methods for dataset compression and classification, specifically leveraging entropy-driven and K-Means-based techniques. The framework below illustrates the methodological connections for each approach applied to application classification and anomaly detection tasks within the HPC-ODA dataset.

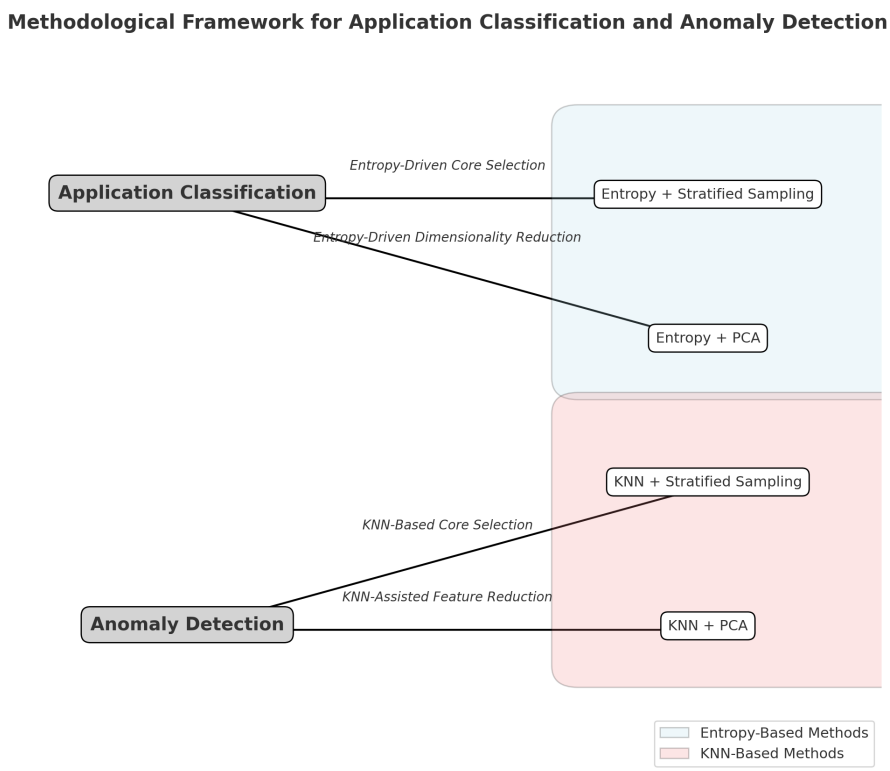

Figure 1.1: Overview of Methodological Framework for Application Classification and Anomaly Detection

1.5 Scope and Limitations

This thesis focuses on the development and evaluation of dataset compression techniques within the context of the HPC-ODA dataset collection. The scope includes high-dimensional time-series data collected from HPC environments, specifically targeting machine learning tasks related to application classification and anomaly detection. The compression methods explored involve entropy-based Random Forest compression and K-Means clustering combined with stratified sampling.

However, the study has certain limitations. The compression techniques are primarily evaluated on datasets from the HPC-ODA collection and may require adaptation for other domains. The analysis centers on specific machine learning models, such as Random Forests, and results may vary with different algorithms. Additionally, the impact of compression on real-time processing and deployment in live HPC environments is not extensively explored, suggesting an avenue for future research.

2 Literature Review

Developing efficient dataset compression techniques for high-performance computing (HPC) environments necessitates a thorough understanding of current data challenges and the methodologies employed to address them. This literature review provides a comprehensive overview of existing research related to large-scale time-series data handling, dataset compression methods, machine learning applications in HPC, and specific techniques such as entropy-based compression and clustering algorithms. By examining these areas, The work is positioned within the broader context, and the gaps that this thesis aims to fill are highlighted.

2.1 Challenges of Large-Scale Time-Series Data in Machine Learning

The proliferation of sensors and monitoring devices in HPC systems has led to an explosion of time-series data [17]. Managing and analyzing this data presents significant challenges, including storage limitations, computational complexity, and real-time processing requirements. Large datasets increase the computational burden on machine learning models, leading to longer training times and the need for substantial processing power [6]. Storage limitations are also a concern, as high-resolution time-series data consume vast amounts of disk space. Researchers have explored distributed computing frameworks and parallel processing to mitigate these issues, but these solutions often come with increased system complexity and cost.

High-dimensional data can negatively impact machine learning models due to the curse of dimensionality, which encompasses various phenomena that arise when analyzing data in high-dimensional spaces. Issues include increased sparsity of data, overfitting, reduced generalization, and difficulties in visualizing and interpreting results.

2.2 Dataset Compression Techniques

To address the challenges of large-scale data, various dataset compression methods have been proposed. These techniques aim to reduce data size while preserving essential information necessary for accurate modeling. Downsampling reduces the data size by selecting a subset of data points at regular intervals. Aggregation involves summarizing data over specific time windows using statistical measures like mean or median. While these methods are straightforward, they risk losing critical temporal patterns

and anomalies essential for certain applications. Therefore, machine learning applications in HPC systems are gaining traction for tasks like anomaly detection, predictive maintenance, and resource optimization. These applications necessitate handling large volumes of data efficiently. Anomaly detection involves identifying patterns that do not conform to expected behavior [4]. Techniques range from statistical methods to machine learning algorithms like Support Vector Machines (SVMs) and Random Forests [7]. Real-time anomaly detection in HPC systems is critical for preventing system failures and optimizing performance.

Classifying applications based on system metrics enables dynamic resource allocation and performance tuning [16]. Techniques include supervised learning models trained on labeled data representing different applications. The primary challenges in HPC machine learning include managing the data's scale and complexity, ensuring timely processing, and maintaining high accuracy in predictions [11]. The need for scalable and efficient algorithms is paramount.

2.3 Entropy-Based Compression Methods

Entropy measures the amount of uncertainty or information content in data [21]. In machine learning, entropy can guide the selection of informative samples or features. Decision trees use entropy to determine the best splits by maximizing information gain [19]. Random Forests, as ensembles of decision trees, benefit from this principle by improving predictive performance through multiple randomized trees [2].

Active learning strategies select the most informative data points for training models, often using uncertainty measures like entropy. Uncertainty sampling focuses on instances where the model is least certain, which can improve learning efficiency [13]. Entropy-based sample selection has been explored in various contexts, such as selecting representative samples in imbalanced datasets. By focusing on samples with high entropy (uncertainty) and low entropy (confidence), models can be trained more effectively.

2.4 Clustering Algorithms for Data Reduction

Clustering algorithms group similar data points, which can be leveraged for data reduction by selecting representative samples from each cluster. K-Means is a widely used algorithm that partitions data into K clusters by minimizing the within-cluster sum of squares [15]. It is computationally efficient and suitable for large datasets but assumes spherical clusters of similar sizes [20].

Clustering has been used for data compression by selecting cluster centroids or representative point. This approach reduces data size while aiming to preserve the overall structure and distribution. However, clustering does not consider class labels, which can lead to imbalanced or biased samples in supervised learning tasks [1]. Ensuring class balance requires additional techniques like stratified sampling.

2.5 Stratified Sampling in Machine Learning

Stratified sampling involves dividing the population into homogeneous subgroups and sampling from each. It is particularly useful in addressing class imbalance in datasets. Class imbalance poses challenges for machine learning models, often biasing predictions towards majority classes [9]. Stratified sampling ensures that minority classes are adequately represented in training data, improving model performance on these classes [5]. Combining clustering with stratified sampling can enhance data reduction methods by maintaining class balance while selecting representative samples [14].

2.6 Principal Component Analysis (PCA) for Dimensionality Reduction

PCA transforms high-dimensional data into a lower-dimensional space by identifying principal components that capture the most variance. PCA has been applied to time-series data for noise reduction and feature extraction [22]. However, PCA assumes linear relationships and may not capture nonlinear patterns inherent in complex datasets [12]. While PCA reduces dimensionality, it does not reduce the number of samples. Additionally, transforming data into principal components can result in loss of interpretability, which may not be desirable in certain applications.

2.7 Gaps in the Literature and Thesis Contribution

The existing literature addresses various methods for dataset compression and handling large-scale data. However, there is a lack of approaches that combine model-based entropy measures with sample selection for compression, particularly in HPC environments. Most data reduction methods do not incorporate the predictive model's insights during compression. Integrating model uncertainty can lead to more informative compressed datasets. Achieving high compression ratios while preserving essential information remains a challenge. There is a need for methods that intelligently select samples to maintain model performance [23].

This thesis addresses these gaps by proposing entropy-based Random Forest compression and K-Means clustering combined with stratified sampling. The methods aim to reduce dataset size significantly while preserving critical information for accurate machine learning predictions in HPC operational data analytics.

The literature highlights the challenges of handling large-scale, high-dimensional time-series data in machine learning applications, particularly within HPC environments. Existing methods offer various solutions but often fall short in balancing data reduction with information preservation. The proposed methods contribute to this field by introducing innovative techniques that leverage model insights and maintain class balance, ultimately improving computational efficiency and model performance in HPC systems.

3 The HPC-ODA Dataset Collection

3.1 Overview of HPC Systems and ODA Tasks

High-Performance Computing (HPC) systems are crucial for solving complex computational problems across various scientific and engineering domains. These systems consist of interconnected compute nodes that work in parallel to perform large-scale computations efficiently. As HPC systems grow in size and complexity, managing and optimizing their performance becomes increasingly challenging.

Operational Data Analytics (ODA) tasks are pivotal in monitoring, managing, and optimizing HPC systems. ODA involves the collection and analysis of operational data from various sensors and system logs to gain insights into system behavior. By leveraging ODA, administrators can detect anomalies, predict failures, optimize resource utilization, and improve overall system reliability and efficiency.

Anomaly detection, in particular, is vital for maintaining system reliability. By detecting unusual patterns or deviations from normal behavior, it allows for proactive maintenance and prevents system outages.

The HPC-ODA dataset collection provides valuable resources for developing and evaluating models and algorithms for these ODA tasks. By offering diverse datasets collected from real HPC systems, it enables researchers to study and address practical challenges in HPC system management.

3.2 Segments of the HPC-ODA Dataset

The HPC-ODA dataset collection comprises six distinct segments, each corresponding to a specific ODA use case. Table 3.1 summarizes the key characteristics of these segments.

Table 3.1: Overview of the HPC-ODA Dataset Segments

Segment	HPC System	Data Points	Duration	Nodes	Sampling Interval
Anomaly Detection	ETH Testbed	∼1,000,000	16 days	1	1 s
Power Consumption Prediction	CooLMUC-3	∼300,000	8 hours	1	100 ms
Application Classification	SuperMUC-NG	∼1,000,000	1 day	16	1 s
Infrastructure Management	CooLMUC-3	∼450,000	16 days	All cluster	10 s
Cross-Architecture	Multiple	∼300,000	1 day	3	1 s
DEEP-EST Dataset	DEEP-EST	∼450,000	2 days	32	10 s

Each segment is designed to provide data for specific ODA tasks:

- **Anomaly Detection**: Data from a compute node with injected faults for anomaly detection.
- **Power Consumption Prediction**: Sensor data for predicting energy usage.
- **Application Classification**: Data from multiple compute nodes with labeled applications.
- **Infrastructure Management**: Sensor data from infrastructure components for optimizing operating conditions.
- **Cross-Architecture Analysis**: Comparative data across different HPC architectures.
- **DEEP-EST Dataset**: Data focused on extreme-scale HPC systems.

3.3 Focus on the Anomaly Detection and Application Classification Segments

Given the thesis's primary concentration on application classification and anomaly detection, this section delves deeper into the Anomaly Detection and Application Classification segments. Brief mentions of other segments are included to provide a comprehensive overview.

3.3.1 Anomaly Detection

Data Acquisition and Sensor Details

The Anomaly Detection segment provides a comprehensive dataset for developing anomaly detection models in HPC systems. Data was collected from a compute node in a testbed HPC system at ETH Zurich, comprising:

- **Compute Node Specifications**:
  - Two Intel Xeon E5-2630 v3 CPUs
  - 128 GB of RAM
  - Seagate ST1000NM0055-1V4 1 TB hard drive
  - CentOS 7.3 operating system
- **Sensor Data**:
  - 128 system-wide sensors sampled every second over 16 days
  - Metrics collected via the Lightweight Distributed Metric Service (LDMS) framework

The sensors capture various system metrics from the following plugins:

- **meminfo**: General information on RAM usage.
- **perfevent**: CPU performance counters, including instructions executed, cycles, cache references, and cache misses.
- **procinterrupts**: Information on hardware and software interrupts.
- **procdiskstats**: Statistics about hard drive usage.
- **procsensors**: Metrics about CPU temperature and frequency.
- **procstat**: General metrics about CPU usage.
- **vmstat**: Information about virtual memory usage.

Fault Injection and Anomaly Simulation

Fault injection was performed using an ad-hoc framework that injects faults at specific times and durations following a statistical workload to simulate realistic scenarios and avoid data bias. Each fault program operates in high or low-intensity modes, doubling the number of possible fault conditions.

The fault programs used to reproduce anomalous conditions include:

- `leak`: Simulates a memory leak by periodically allocating memory arrays without releasing them.
- `memeater`: Creates memory interference by saturating memory bandwidth with continuous allocations.
- `ddot`: Produces CPU and cache interference by performing intensive dot product calculations.
- `dial`: Causes ALU interference through continuous numerical operations.
- `cpufreq`: Simulates a misconfigured or failing CPU by reducing the maximum allowed CPU frequency.
- `pagefail`: Simulates hardware malfunction by causing intermittent page allocation request failures.
- `ioerr`: Simulates a failing hard drive by triggering errors during hard-drive I/O operations.
- `copy`: Simulates I/O interference by repeatedly copying files, saturating I/O bandwidth.

These faults represent common issues in HPC systems, affecting performance and reliability. By injecting these faults, the dataset captures the system's behavior under various anomalous conditions.

Data Preprocessing Steps

Prior to training machine learning models for anomaly detection, the data underwent several preprocessing steps:

Data Loading and Merging Sensor data and response labels indicating the injected faults were loaded from their respective sources. Timestamps were used to align sensor readings with corresponding fault labels.

Label Mapping Fault labels were mapped to numerical values for classification purposes:

Table 3.2: Fault Label Mapping

Fault	Label
None (Idle)	0
`pagefail`	1
`leak`	2
`ddot`	3
`memeater`	4
`dial`	5
`cpufreq`	6
`copy`	7
`ioerr`	8

Feature Engineering Sensor data was processed to extract relevant features for anomaly detection, including:

- Aggregating metrics over time windows to capture temporal patterns.
- Calculating derived features such as utilization percentages and rates of change.

3.3.2 Application Classification

Data Acquisition and Sensor Details

The Application Classification segment provides a comprehensive dataset for developing models that can identify running applications based on system performance metrics. Data was collected from 16 compute nodes in the SuperMUC-NG HPC system at LRZ, sampled every second over one day. Each node is equipped with:

- **Compute Node Specifications**:
  - 48-core Intel Skylake Xeon Platinum 8174 CPUs

  - 96 GB of RAM
  - Intel OmniPath network interface
  - SUSE Linux Enterprise Server 12 operating system
  - Warm-water cooling
- **Sensor Data**:
  - 52 sensors per node
  - Metrics collected via the Data Center Data Base (DCDB) framework

The sensors cover various system metrics:

- **CPU Performance Counters**: Instructions executed, cycles, cache references, cache misses.
- **Memory Metrics**: Memory usage, page faults, swapping activity.
- **CPU Activity Metrics**: CPU utilization, context switches, interrupts.
- **System Sensors**: Power consumption, energy usage, temperature sensors at the node and CPU level.
- **Network Metrics**: Data transmitted and received via the `OPA` plugin.

Applications Included and Their Characteristics

To generate diverse workloads, several HPC applications were executed on the compute nodes during data collection. These applications are widely used benchmarks and proxy applications representing different computational patterns and resource utilization profiles.

Each application was executed with multiple configurations to minimize bias and ensure variability in the dataset. The configurations varied problem sizes, input parameters, and computational workloads. The applications were run using one MPI rank per compute node and 48 OpenMP threads, matching the number of physical CPU cores.

The selected applications cover a range of computational behaviors:

- **Compute-Intensive**: Applications like HPL focus on heavy computations with high floating-point operations per second (FLOPS).
- **Memory-Intensive**: AMG and Kripke are memory-bound, emphasizing memory bandwidth and latency.
- **Network-Intensive**: Kripke and AMG also stress network performance due to communication between MPI ranks.
- **I/O-Intensive**: LAMMPS involves significant I/O operations, reading and writing simulation data.

- **Irregular Computation**: PENNANT features irregular memory access patterns, challenging for prefetching and caching mechanisms.

By including applications with diverse characteristics, the dataset enables the development of robust application classification models capable of distinguishing between different computational workloads.

3.3.3 Data Preprocessing Steps

Prior to training machine learning models, the data underwent several preprocessing steps to ensure quality and suitability for analysis.

Data Loading and Merging Sensor data and response labels were loaded from their respective directories. Each node's data was stored separately and included timestamps for synchronization. The data from all nodes were merged based on the timestamps to create a unified dataset. This involved handling any discrepancies in time alignment and ensuring that sensor readings from different nodes corresponded to the same time points.

Label Mapping Application labels in the response files were mapped to numerical values for classification purposes. The mapping was as follows:

Table 3.3: Application Label Mapping

Application	Label
Kripke	0
AMG	1
PENNANT	2
linpack (HPL)	3
LAMMPS	4
Quicksilver	5
None (Idle)	6

Feature Engineering The data was pivoted to create a wide format where each row represented a timestamp, and columns represented sensor features from different nodes. Multi-level columns were flattened to create a single level of feature names, formatted as `Node_SensorName`.

Handling Missing Data Any missing sensor readings were handled appropriately using strategies such as:

- **Dropping Samples**: Removing rows with missing values if the proportion was minimal.

Data Normalization Feature scaling was performed using standardization to ensure that all features contributed equally to the model. The standard score for each feature was calculated as:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

where x is the original value, μ is the mean, and σ is the standard deviation.

Variance Thresholding Features with low variance were removed using a variance thresholding technique. This step eliminated features unlikely to be informative for classification, reducing dimensionality and computational complexity. A threshold of 0.1 was used, retaining features with variance above this value.

Data Splitting The preprocessed data was split into training and testing sets based on time. The first 80% of the data was used for training, and the remaining 20% for testing. This temporal split simulates real-world scenarios where models are trained on past data and deployed on future data.

Summary of Preprocessed Data After preprocessing, the dataset contained:

- **Number of Samples**: 62,357 training samples.
- **Number of Features**: 783 features.

The extensive preprocessing ensured that the dataset was clean and suitable for machine learning analysis.

3.3.4 System Specifications and Environment

The training and testing of models developed in this thesis were conducted on a 13-inch MacBook Pro (2019) equipped with 4 Thunderbolt 3 ports. This machine has a dual-core Intel processor, providing sufficient computing capabilities for the initial stages of model training and evaluation.

3.3.5 Technical Aspects

The implementation was conducted using Python with the following libraries:

- **NumPy** and **pandas** for numerical computations and data manipulation.
- **scikit-learn** for machine learning models and preprocessing tools.

- **SciPy** for statistical computations, such as entropy calculation.
- **Matplotlib** and **Seaborn** for data visualization.

3.3.6 Summary

The Anomaly Detection and Application Classification segments of the HPC-ODA dataset provide rich and diverse datasets for developing anomaly detection and application classification models in HPC systems. By simulating realistic faults and collecting extensive sensor data, these datasets capture the complex behaviors of HPC systems under various conditions. The detailed preprocessing steps ensure that the data is clean and suitable for machine learning analysis.

In the subsequent chapters, we will explore the development of models using these datasets, focusing on techniques for dataset compression and efficient model training.

4 Entropy-Based Random Forest Compression

4.1 Introduction

In this chapter, an entropy-based Random Forest compression method is introduced, specifically designed for high-dimensional and large-scale time-series datasets. This method leverages predictive uncertainties to identify and select the most informative samples from the original dataset. By focusing on samples with both high and low predictive entropy, the approach ensures the creation of a compressed dataset that maintains the core structure of the original data distribution. This method supports high model accuracy while significantly reducing computational costs, making it particularly suitable for time-sensitive applications.

Random Forests are especially powerful for this task due to their inherent ability to handle high-dimensional data, manage complex feature interactions, and provide robust predictions even with noisy or missing data. Their ensemble structure allows the capture of temporal dependencies and intricate patterns that are often found in time-series data. Moreover, the entropy-driven sampling strategy builds on Random Forest's predictive capabilities to retain both confident and uncertain predictions, which enhances generalization and prevents overfitting, ensuring that the compressed dataset reflects the essential characteristics of the full dataset.

4.2 Background and Theoretical Foundations

4.2.1 Entropy in Machine Learning

Entropy, a concept from information theory introduced by Shannon [21], measures the uncertainty or unpredictability of a random variable. In machine learning, entropy is used in decision trees to quantify the impurity of a node. The entropy H of a discrete random variable Y with possible values $\{y_1, y_2, ..., y_n\}$ and probability mass function $P(Y)$ is defined as:

$$H(Y) = -\sum_{i=1}^{n} P(y_i) \log_2 P(y_i) \tag{4.1}$$

A higher entropy value indicates greater uncertainty, while lower entropy signifies more certainty in the outcomes.

4.2.2 Predictive Entropy

Predictive entropy quantifies the uncertainty in a model's predictions. For the random forest classification model that outputs class probabilities, the predictive entropy for a sample x_i is calculated using the predicted probabilities $P(y_k|x_i)$ for each class k:

$$H_i = -\sum_{k=1}^{K} P(y_k|x_i) \log_2 P(y_k|x_i) \tag{4.2}$$

where K is the total number of classes. High predictive entropy indicates that the model is uncertain about its prediction for that sample, while low entropy suggests confidence.

Predictive entropy is widely used in active learning to select informative samples for labeling. By focusing on samples with high uncertainty, models can learn more efficiently from fewer data points.

4.3 Methodology

4.3.1 Overview of the Entropy-Based Compression Technique

The entropy-based Random Forest compression technique is designed to reduce the high-dimensional HPC dataset by identifying a subset of the most informative samples. This involves selecting samples predicted with high certainty (low entropy) and high uncertainty (high entropy), enhancing the model's understanding across both well-represented and challenging areas in the dataset's feature space. These samples are essential for training efficient models without compromising on performance.

The compression process is executed through the following steps:

1. **Data Loading and Merging:**
   - Load sensor data recorded from 16 nodes in the HPC system, including CPU socket temperatures and performance metrics.
   - Merge this data with response labels identifying the application in execution or idle states, mapped to numeric labels: Kripke (0), AMG (1), PENNANT (2), linpack (3), LAMMPS (4), Quicksilver (5), and None (6) for idle state.
2. **Data Splitting and Preparation:**
   - Split the dataset temporally into training and testing sets to maintain the **time-series structure**, ensuring models are trained on past data and tested on future states.
   - Remove non-informative features and normalize metrics such as temperature readings, ensuring comparability across **CPU sockets** and other node metrics.
3. **Variance Threshold Feature Selection:**

- Apply variance thresholding to remove low-variance features, especially within **None (idle)** states, which contribute less to class distinction. This step reduces dimensionality without sacrificing important information.

4. **Train Random Forest and Calculate Predictive Entropy:**
   - Train a **Random Forest model** on the full dataset, utilizing its ability to handle the diverse and high-dimensional sensor features effectively.
   - For each sample, compute predictive entropy values to assess the model's confidence, selecting samples with either high or low entropy. This selection includes instances where the model confidently predicts labels such as **Kripke** or **AMG**, as well as samples with uncertainty around complex interactions between **PENNANT** and **LAMMPS**.
5. **Core Set Selection and Class Balance:**
   - Select a core dataset that maintains a balanced class distribution across application states. Ensure that **'None' (idle state)** and less frequent application classes are proportionally represented to prevent class imbalance in downstream training.
6. **Further Compression with Stratified Sampling:**
   - Apply **stratified sampling** on the core dataset to achieve additional compression while preserving the proportions of all classes (e.g., **Kripke, AMG, and None**), ensuring the compressed dataset retains class representativeness.
7. **Model Training and Evaluation:**
   - Train and evaluate models on both core and compressed datasets, measuring performance against the full dataset to verify the compressed dataset's ability to accurately classify all states, including **Quicksilver** and **linpack**.
8. **Statistical Validation and Export:**
   - Perform statistical tests, such as **Kolmogorov-Smirnov tests**, to validate feature distribution similarities between the full and compressed datasets. This step ensures that distributions of features (e.g., **temperature and CPU metrics**) are well-preserved.

4.3.2 Detailed Steps of Entropy-Based Compression Strategy

Data Preparation

The dataset consists of sensor readings from multiple nodes and corresponding application labels. Data preparation involves:

- Loading and merging sensor data with response labels.

- Mapping application labels to numerical values for classification.
- Pivoting the data to create a wide format where each row represents a timestamp and columns represent sensor features from different nodes.
- Normalizing features using standard scaling to ensure that all features contribute equally to the model.
- Applying variance thresholding to remove features with low variance, which are less likely to be informative.

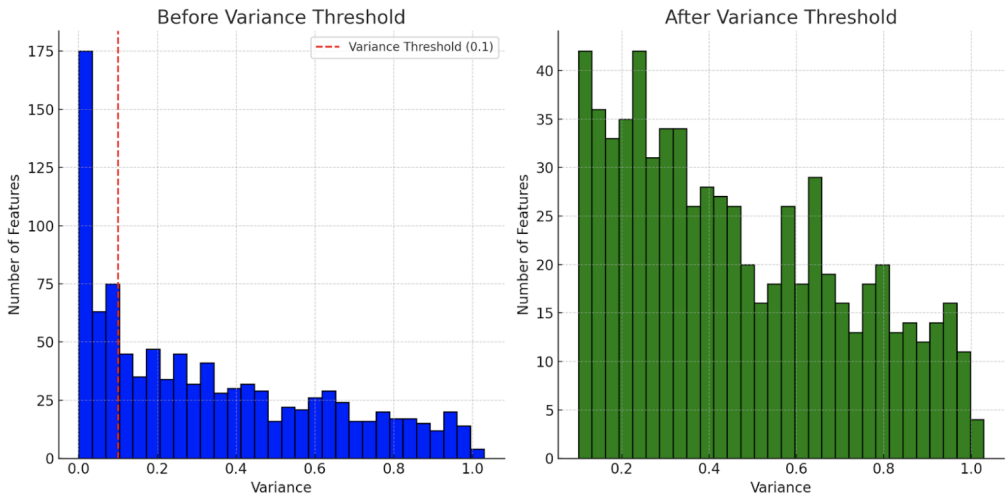

Figure 4.1: Feature count before and after applying variance thresholding

After extensive experimentation with various dataset sizes, a variance threshold of 0.1 was selected as the optimal balance between feature reduction and model performance. This threshold was determined through empirical testing to retain the most informative features while reducing dimensionality effectively.

Input : Scaled Training Features: $X_{\text{train}}^{\text{scaled}}$
Scaled Testing Features: $X_{\text{test}}^{\text{scaled}}$
Feature Names: FeatureNames
Variance Threshold: θ
Output: Transformed Training Features: $X_{\text{train}}^{\text{selected}}$
Transformed Testing Features: $X_{\text{test}}^{\text{selected}}$
Selected Feature Names: SelectedFeatures

Step 1: Initialize Variance Threshold Selector;
Selector $\leftarrow$ VarianceThreshold(θ);
Step 2: Fit Selector on Training Data;
Selector.fit($X_{\text{train}}^{\text{scaled}}$);
Step 3: Transform Training and Testing Data;
$X_{\text{train}}^{\text{selected}} \leftarrow$ Selector.transform($X_{\text{train}}^{\text{scaled}}$);
$X_{\text{test}}^{\text{selected}} \leftarrow$ Selector.transform($X_{\text{test}}^{\text{scaled}}$);
Step 4: Identify Selected Features;
SelectedIndices $\leftarrow$ Selector.get_support(indices=True);
for $i \leftarrow 1$ **to** *length*(*SelectedIndices*) **do**
  SelectedFeatures[i] $\leftarrow$ FeatureNames[SelectedIndices[i]];
end
Step 5: Return Transformed Data and Selected Features;
return $X_{\text{train}}^{\text{selected}}$, $X_{\text{test}}^{\text{selected}}$, SelectedFeatures;

Algorithm 1: Variance Threshold Feature Selection

Training the Initial Random Forest Model

The training of the initial Random Forest classifier is a critical component of the entropy-based compression method. The predictive entropies, which are central to selecting the most informative samples for the compressed dataset, are directly influenced by the performance and configuration of this model. Each hyperparameter in the Random Forest algorithm plays a significant role in shaping the model's behavior, affecting both the accuracy and the quality of entropy estimates. Extensive experimentation was conducted to fine-tune these hyperparameters, ensuring optimal balance between model performance and computational efficiency.

Hyperparameter Selection and Rationale

- **Number of Estimators (`n_estimators` = 250)**: This parameter specifies the number of decision trees in the forest. A larger number of estimators generally improves model performance by reducing variance through ensemble averaging. However, it also increases computational cost. After extensive testing, 250 estimators were chosen as the optimal point where adding more trees resulted in negligible gains in accuracy but significant increases in training time. For example:
  - `n_estimators` = 100: Accuracy = 96.7%, Training Time = 2 minutes.

  - `n_estimators` = 250: Accuracy = 99.3%, Training Time = 3 minutes.
  - `n_estimators` = 500: Accuracy = 99.4%, Training Time = 5 minutes.

  The marginal improvement of 0.1% in accuracy from 250 to 500 estimators did not justify the doubling of training time.

- **Maximum Depth (`max_depth` = 20)**: The maximum depth limits how deep each tree can grow. Deeper trees can capture more complex patterns but are prone to overfitting. Setting `max_depth` to 20 was found to provide a balance between model complexity and generalization. Experiments showed:
  - `max_depth` = 10: Underfitting observed, Accuracy = 97.5%.
  - `max_depth` = 20: Optimal performance, Accuracy = 99.3%.
  - `max_depth` = None (unlimited): Overfitting observed, Accuracy on training set = 100%, but test set accuracy decreased due to overfitting.

  A depth of 20 allowed sufficient complexity to model the data while preventing overfitting.

- **Criterion (`criterion` = 'entropy')**: This parameter determines the function used to measure the quality of a split. 'Entropy' measures information gain, which is particularly effective in multi-class classification problems. Alternative options like 'gini' were tested:
  - `criterion` = 'gini': Accuracy = 98.9%.
  - `criterion` = 'entropy': Accuracy = 99.3%.

  The 'entropy' criterion consistently resulted in higher accuracy, making it the preferred choice.

- **Class Weighting (`class_weight` = 'balanced_subsample')**: The datasets exhibited class imbalance, with some classes being underrepresented. Using 'balanced_subsample' adjusts the class weights inversely proportional to class frequencies in each bootstrap sample. This ensures that minority classes are given appropriate importance during training. Without class weighting, the model tended to bias towards majority classes, reducing recall for minority classes by up to 15%.

- **Minimum Samples per Leaf (`min_samples_leaf` = 5)**: This parameter sets the minimum number of samples required to be at a leaf node. Setting it to 5 prevents the trees from becoming too deep with very few samples in leaves, which can lead to overfitting. Experiments showed:
  - `min_samples_leaf` = 1: Overfitting observed, lower generalization accuracy.
  - `min_samples_leaf` = 5: Improved generalization, Accuracy = 99.3%.
  - `min_samples_leaf` = 10: Slight underfitting, Accuracy decreased to 98.8%.

- **Bootstrap (`bootstrap` = True)**: Enabling bootstrap sampling allows each tree to be trained on a random subset of the data, promoting diversity among trees and reducing variance.
- **Random State (`random_state` = 42)**: Setting a random state ensures reproducibility of results across different runs.
- **Number of Jobs (`n_jobs` = -1)**: Utilizing all available CPU cores for parallel processing speeds up training significantly.

Impact of Hyperparameters on Model Performance and Compression The hyperparameters directly influence the model's ability to capture patterns in the data, which in turn affects the predictive entropy values. These entropy values are crucial for selecting informative samples during compression. Poor hyperparameter choices can lead to inaccurate entropy estimates, resulting in suboptimal sample selection and reduced model performance on the compressed dataset.

For example, an underfit model (e.g., with low `max_depth` or insufficient `n_estimators`) may assign high entropy to many samples due to its inability to make confident predictions. This would lead to a larger core dataset, reducing the compression ratio. Conversely, an overfit model may assign low entropy across the board, failing to identify samples that are genuinely informative, which can degrade performance on unseen data.

The impact of hyperparameter tuning on the Random Forest model's performance is clearly illustrated in the resulting confusion matrix below. This matrix represents the best results achieved during the testing phase after selecting optimal parameter configurations. Each element in the matrix reflects the model's accuracy in predicting each class, highlighting the effectiveness of fine-tuned parameters in capturing the complex patterns within the data.

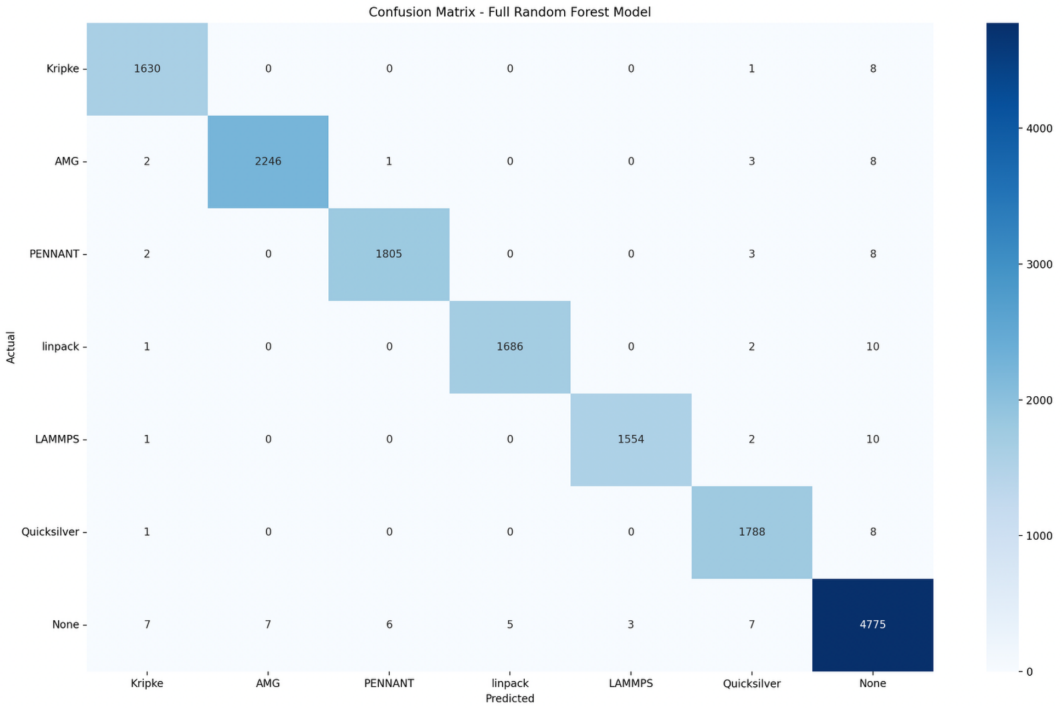

Figure 4.2: Confusion Matrix of Full Dataset - Random Forest

Experiments with Different Hyperparameter Configurations To illustrate the importance of hyperparameter tuning, consider the following experimental results:

Table 4.1: Impact of Hyperparameters on Model Accuracy and Compression Ratio

Configuration	Test Accuracy (%)	Core Dataset Size	Compression Ratio
`n_estimators`=100, `max_depth`=10	97.5	41,571	1.5x
`n_estimators`=250, `max_depth`=20	99.3	24,943	2.5x
`n_estimators`=500, `max_depth`=None	99.4	62,357	1x

These results demonstrate that:

- Increasing `n_estimators` beyond 250 provided minimal accuracy gains but reduced the compression ratio due to less reliable entropy estimates.
- A `max_depth` of 20 achieved high accuracy while maintaining a reasonable compression ratio.
- Omitting class weighting (`class_weight`=None) reduced accuracy, particularly for minority classes, highlighting the importance of handling class imbalance.

- Setting `min_samples_leaf` too low (`min_samples_leaf`=1) led to overfitting, reducing test accuracy and affecting entropy estimates.

Critical Role of Hyperparameter Tuning in Enhancing Entropy-Driven Compression: Hyperparameter tuning was crucial for optimizing model performance and compression efficiency, as demonstrated by our experiments. Setting `n_estimators`=250 and `max_depth`=20 achieved a high accuracy of 99.3% with a compression ratio of 2.5x, striking an optimal balance between performance and efficiency (see Table 4.1). Utilizing `class_weight`='balanced_subsample' significantly improved recall for minority classes, ensuring robust performance across all classes, especially in fault detection scenarios. Additionally, configuring `min_samples_leaf`=5 prevented overfitting while maintaining high generalization accuracy, as lower values led to reduced performance. These tuned hyperparameters enabled accurate entropy estimation, allowing the model to effectively distinguish between informative and redundant samples. This maximization of data compression without sacrificing accuracy was essential for creating a meaningful compressed dataset. Furthermore, optimizing parameters to minimize computational overhead ensured the method's practicality for large datasets. Overall, the meticulous tuning of the initial Random Forest model was a decisive factor in the success of the entropy-based compression technique, as it ensured that the compressed dataset retained essential information for high-performance classification while maintaining computational efficiency.

Calculating Predictive Entropy

After training the initial Random Forest model, the subsequent step involves calculating the predictive entropy for each training sample. Predictive entropy serves as a measure of the model's uncertainty in its predictions, providing crucial information about which samples are most informative for the model. This step is critical because the accuracy of the entropy estimates directly impacts the effectiveness of the compression method.

Input: Trained Random Forest model RF, Training data X_{train}, Target compression ratio C
Output: Core dataset X_{core}, y_{core}
Step 1: Compute Predictive Entropy
foreach *sample* $x_i \in X_{train}$ **do**
  Obtain predicted class probabilities $P(y_k|x_i)$ from RF for each class k
  Calculate predictive entropy $H_i = -\sum_{k=1}^{K} P(y_k|x_i) \log_2 P(y_k|x_i)$
end
Step 2: Select Informative Samples Based on Entropy
Define high and low entropy thresholds, T_{high} and T_{low}, at the 99th and 1st percentiles of $\{H_i\}$
Identify high-uncertainty samples: $I_{\text{high}} = \{i \mid H_i \geq T_{\text{high}}\}$
Identify low-uncertainty samples: $I_{\text{low}} = \{i \mid H_i \leq T_{\text{low}}\}$
Form the core dataset indices: $I_{\text{core}} = I_{\text{high}} \cup I_{\text{low}}$
Step 3: Create Core Dataset
$X_{\text{core}} = X_{\text{train}}[I_{\text{core}}]$
$y_{\text{core}} = y_{\text{train}}[I_{\text{core}}]$
return X_{core}, y_{core}

Algorithm 2: Entropy-Based Core Set Selection

This entropy value quantifies the expected amount of information needed to describe the model's prediction for x_i. A high entropy indicates that the model is uncertain about the prediction, distributing probability more evenly across classes. Conversely, a low entropy signifies that the model is confident, assigning a high probability to a particular class.

Importance of Accurate Entropy Estimation The accuracy of the predictive entropy values depends on the calibration of the Random Forest model's probability estimates. A well-calibrated model provides reliable probabilities, ensuring that entropy accurately reflects uncertainty. Poor calibration can lead to misleading entropy values, affecting the selection of samples and ultimately degrading model performance.

Experiments with different calibration techniques were conducted:

- **Without Calibration**: The default Random Forest model sometimes produces overconfident probability estimates. This can result in low entropy values even when the model is incorrect.

Based on these experiments, calibration was applied to the model to ensure that entropy values accurately represent uncertainty.

Sample Selection Based on Entropy

The next step involves selecting samples for the compressed dataset based on their predictive entropy. The aim is to retain samples that are most informative for the model, focusing on areas where the model is both uncertain and certain.

Selection Criteria and Thresholds Two thresholds were established to select high and low entropy samples:

- **High Entropy Threshold (T_{high})**: Samples with entropy greater than or equal to the 99th percentile of the entropy distribution.
- **Low Entropy Threshold (T_{low})**: Samples with entropy less than or equal to the 1st percentile of the entropy distribution.

These thresholds were chosen after extensive experimentation to balance compression ratio and model performance.

Impact of Threshold Choices Adjusting the entropy thresholds affects both the size of the core dataset and the subsequent model accuracy:

Table 4.2: Impact of Entropy Thresholds on Core Dataset Size and Model Accuracy

Threshold Percentiles	Core Dataset Size	Compression Ratio	Test Accuracy (%)
1% and 99%	25,177	2.48x	98.61
0.5% and 99.5%	12,345	5.05x	97.20
5% and 95%	62,357	1x	99.32
2% and 98%	31,432	1.98x	98.10

From the table, I observe that:

- Narrower thresholds (e.g., 0.5% and 99.5%) lead to higher compression ratios but reduce model accuracy due to fewer samples in the core dataset.
- Wider thresholds (e.g., 5% and 95%) maintain model accuracy but offer less compression.
- The 1% and 99% thresholds provide a good balance between compression and performance.

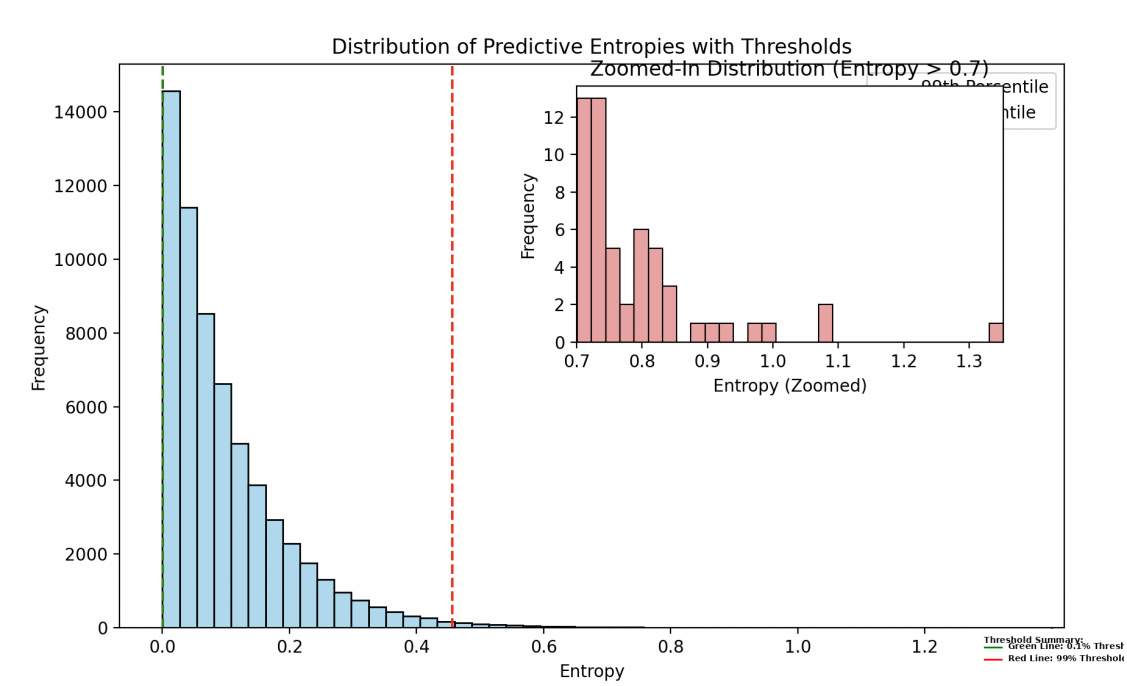

Figure 4.3: Distribution of Predictive Entropies with Extreme Value Thresholds (1st and 99th Percentiles)

Threshold Selection Rationale for Predictive Entropy-Based Sampling

In this entropy-driven sampling strategy, the goal is to retain samples that exhibit either high or low levels of predictive uncertainty. As illustrated in the entropy distribution figure, the distribution is right-skewed, with a majority of values clustered below 0.4, indicating that the Random Forest model is generally confident in its predictions. By selecting extreme values of entropy, the most informative samples for model refinement are captured.

- **Formula for Entropy**: The entropy $H(x)$ of predicted probabilities $p(x_i)$ is computed using:

$$H(x) = -\sum_i p(x_i) \log(p(x_i))$$

  where $p(x_i)$ represents the probability of each class for a sample. Higher entropy values signify more evenly distributed probabilities, indicating uncertainty, while lower values suggest a confident, skewed probability distribution.

- **Theoretical Maximum Entropy**: Given that the dataset has six classes, the maximum possible entropy occurs when each class has an equal probability of $\frac{1}{6}$. In

this scenario, the entropy reaches its theoretical maximum:

$$\text{Maximum Entropy} = -\sum_{i=1}^{6} \left(\frac{1}{6}\right) \log \left(\frac{1}{6}\right) \approx 2.58$$

This value represents complete uncertainty, where the model assigns equal likelihood to all classes.

- **Chosen Percentiles for Thresholds**: Summary statistics and percentiles provide insight into where the most informative samples lie within the distribution:
  - The **mean** entropy is 0.10, and the **median** is 0.07, showing that the majority of samples have low uncertainty.
  - The **99th percentile** (0.4566) and the **1st percentile** (0.00098) are selected as boundaries to capture the most uncertain and certain samples, respectively. These thresholds help retain samples at the extremes, with values close to the maximum entropy (for high uncertainty) and values near zero (for high confidence).
- **Visual Interpretation of Thresholds**: The histogram illustrates these thresholds:
  - The **1st percentile** threshold is marked in green near zero, covering the distribution's lower tail where entropy values are minimal.
  - The **99th percentile** threshold, represented by the red line at around 0.4566, captures the upper tail where entropy is highest.

  The inset zooms into entropy values above 0.7, illustrating the sparse distribution of very high entropy samples, which represent significant uncertainty.
- **Rationale for Selection**: By choosing these percentiles, a balance is achieved between compression and performance:
  - The thresholds allow retention of the most representative samples, capturing both ends of the uncertainty spectrum without overwhelming the model with redundant data.
  - This selective approach ensures that the compressed dataset retains critical information for model generalization, focusing on samples that either bolster model confidence or challenge it with high uncertainty.

Ensuring Class Balance

After selecting samples based on entropy, it is crucial to ensure that the core dataset maintains class balance. This step is vital to prevent the model from becoming biased towards majority classes.

Analysis of Class Distribution The class distribution in the core dataset was examined:

Table 4.3: Class Distribution in Full and Core Datasets

Class	Full Dataset Count	Core Dataset Count (Entropy-Based Compression)
Kripke	7,429	4,144
AMG	9,841	6,740
PENNANT	6,940	2,570
linpack	6,301	4,970
LAMMPS	5,830	3,940
Quicksilver	5,034	2,610
None (Idle State)	21,542	590

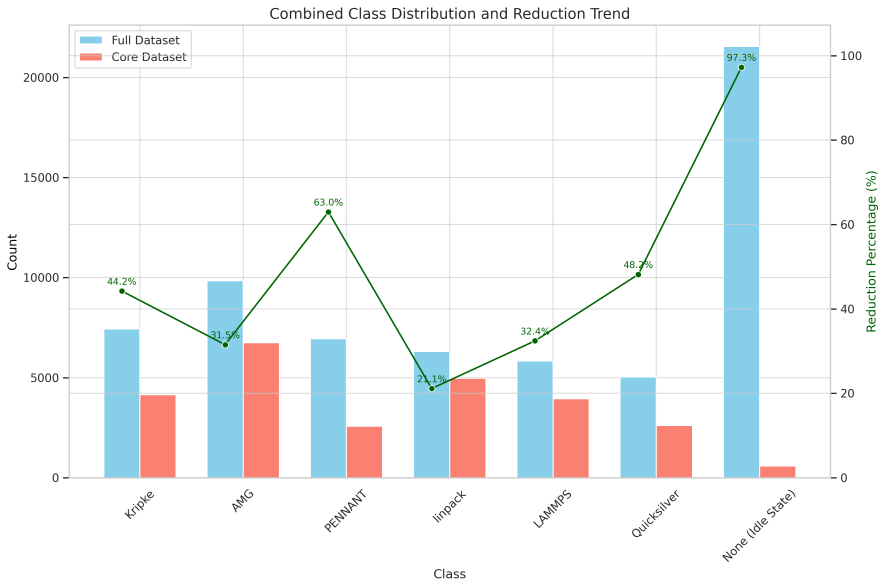

Figure 4.4: Combined Class Distribution and Reduction Trend for Full and Core Datasets

The table shows that the majority class (None) is overrepresented in the core dataset after entropy-based selection.

Further Compression with Stratified Sampling

Stratified sampling is utilized to achieve further compression of the core dataset while retaining class distribution. The goal is to reduce the dataset size to the desired compression ratio while preserving the original class proportions.

Determining Sampling Fraction The sampling fraction f was calculated based on the desired overall compression ratio C:

$$f = \frac{1}{C} \tag{4.3}$$

For example, to achieve an additional compression ratio of 5x, f was set to 0.2.

4.3.3 Pseudocode of Stratified Sampling

Data: Core dataset $(X_{\text{core}}, y_{\text{core}})$, target compression ratio C
Result: Compressed dataset $(X_{\text{compressed}}, y_{\text{compressed}})$ [1]
Set Sampling Fraction: Calculate the sampling fraction $f = \frac{1}{C}$ to meet the target compression ratio.
Apply Stratified Sampling: Perform stratified sampling on $(X_{\text{core}}, y_{\text{core}})$ with fraction f, ensuring that the class distribution is preserved.
Return Compressed Dataset: Output the compressed dataset $(X_{\text{compressed}}, y_{\text{compressed}})$.

Algorithm 3: Stratified Sampling for Core Dataset Compression

Note: Adjusting the sampling fraction f allows control over the level of compression achieved, while preserving the class balance within the compressed dataset.

Experiments were conducted with various sampling fractions:

Table 4.4: Impact of Sampling Fractions on Dataset Size and Model Accuracy

Sampling Fraction (f)	Compressed Dataset Size	Total Compression Ratio	Test Accuracy (%)
1.0 (No further sampling)	25,177	2.48x	98.61
0.5	12,589	4.95x	98.10
0.1	2,518	24.80x	97.40
0.02	503	123.97x	97.15
0.001	251	248.43x	86.15

Final Compressed Dataset Using a sampling fraction of 0.02, the final compressed dataset was created, achieving an overall compression ratio of approximately 125x while maintaining a test accuracy of 97.15%.

Criticality of These Steps The steps of calculating predictive entropy, selecting samples based on entropy, ensuring class balance, and applying further stratified sampling are critical because:

- They directly impact the quality and representativeness of the compressed dataset.
- The balance between compression ratio and model accuracy depends on careful parameter tuning and methodical sample selection.

- Missteps in any of these areas can lead to significant degradation in model performance.

By thoroughly understanding and optimizing each parameter and step, I ensure that the compressed dataset is both efficient in size and effective in maintaining high model performance.

4.4 Experimental Setup

4.4.1 Datasets

The dataset comprises sensor readings from 16 nodes, each capturing various metrics. The response variable indicates the application or state, mapped to numerical labels:

- 0: Kripke
- 1: AMG
- 2: PENNANT
- 3: linpack
- 4: LAMMPS
- 5: Quicksilver
- 6: None

Data is split temporally into training (80%) and testing (20%) sets to simulate real-world scenarios where models are deployed on future data.

4.4.2 Evaluation Metrics

Performance is assessed using:

- **Accuracy**: Overall correctness of the model.
- **Precision, Recall, F1-score**: Evaluated per class to understand class-wise performance.
- **Confusion Matrix**: Visual representation of predictions versus actual values.
- **Kolmogorov-Smirnov (KS) Test**: Statistical test to compare feature distributions between the full and compressed datasets, ensuring minimal information loss.
- **Cross-Validation**: K-fold cross-validation (e.g., 5-fold) to assess the model's robustness and generalizability across different data splits, mitigating the risk of overfitting.

Compression effectiveness is measured by:

- **Compression Ratio**: Original dataset size divided by compressed dataset size.
- **Data Size (KB)**: Memory footprint of the datasets.

4.4.3 Baseline Models

The primary baseline is the Random Forest model trained on the full dataset without compression. Its performance serves as a benchmark to evaluate the impact of the compression technique.

4.5 Results

4.5.1 Compression Ratios and Dataset Sizes

The compression technique significantly reduces the dataset size while achieving high compression ratios. For instance, with 16 nodes:

Table 4.5: Dataset Sizes and Compression Ratios

Dataset	Samples	Data Size (KB)	Compression Ratio
Full Data	62,357	381,449.46	1
Core Set	25,177	154,012.43	2.48
Compressed Core	503	3,076.95	123.97

4.5.2 Model Performance

The Random Forest models trained on compressed datasets maintain high accuracy:

- **Full Model Accuracy**: 99.32%
- **Core Model Accuracy**: 98.61%
- **Compressed Core Model Accuracy**: 97.00%

Classification reports show that precision, recall, and F1-scores remain high across all classes, indicating that the compression does not significantly degrade model performance.

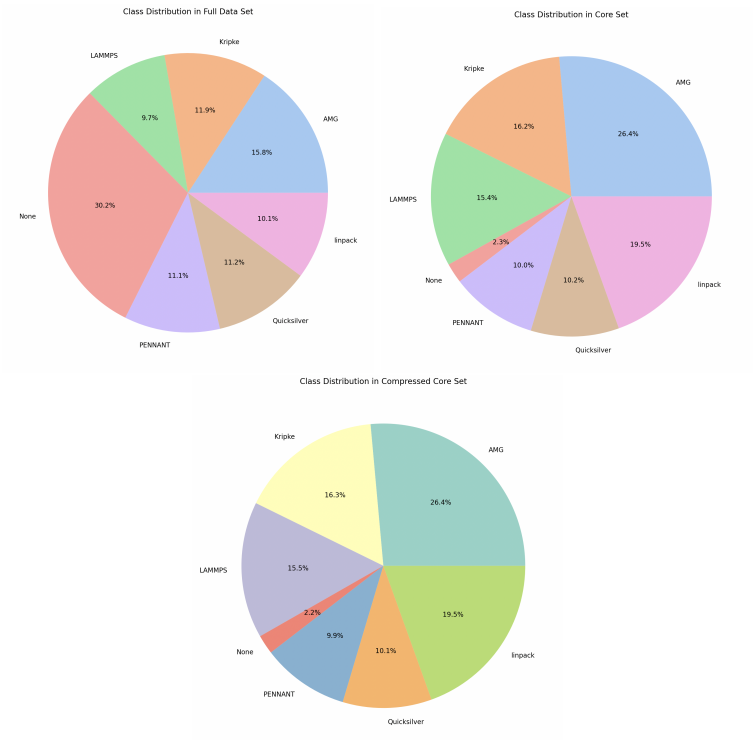

Figure 4.5: Class Distribution of Full, Core, and Compressed Core Datasets

Classification Report for Core Model

Table 4.6: Classification Report for Core Model

Class	Precision	Recall	F1-Score	Support
Kripke	0.98	0.99	0.98	1639
AMG	0.99	0.99	0.99	2260
PENNANT	0.99	0.99	0.99	1818
linpack	1.00	0.96	0.98	1699
LAMMPS	0.99	0.97	0.98	1567
Quicksilver	0.99	0.99	0.99	1797
None	0.98	0.99	0.99	4810
accuracy		0.99 (15,590)		
macro avg	0.99	0.98	0.99	15,590
weighted avg	0.99	0.99	0.99	15,590

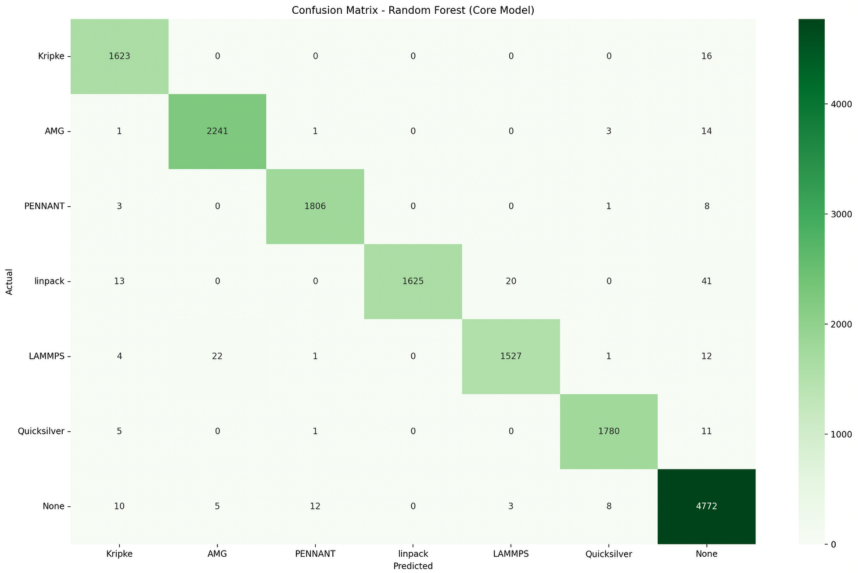

Figure 4.6: Confusion Matrix of Full Dataset - Random Forest

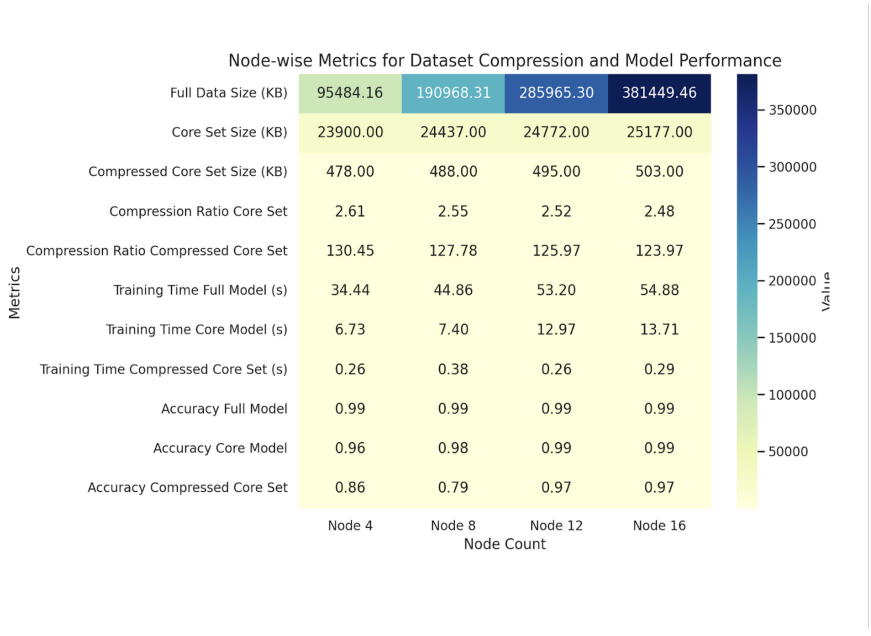

Figure 4.7: Node-Wise Dataset Compression and Model Performance Metrics

4.5.3 Statistical Validation

Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test assesses whether feature distributions differ between the full and compressed datasets. Results indicate:

- **Number of features with similar distributions**: 311 out of 783.
- **Percentage**: 39.72%

This suggests that a significant portion of the feature distributions are preserved after compression.

Furthermore, most features were removed to eliminate the dominance of the "None" class, ensuring a more balanced representation across all classes. By reducing the prevalence of the "None" class, the model becomes better equipped to recognize and differentiate between the minority classes. Additionally, features associated with the "None" class were selectively removed because they exhibited low variance and had minimal impact on the overall feature distribution. This targeted feature reduction helps in retaining only those features that contribute significantly to the model's predictive performance, thereby enhancing both efficiency and effectiveness.

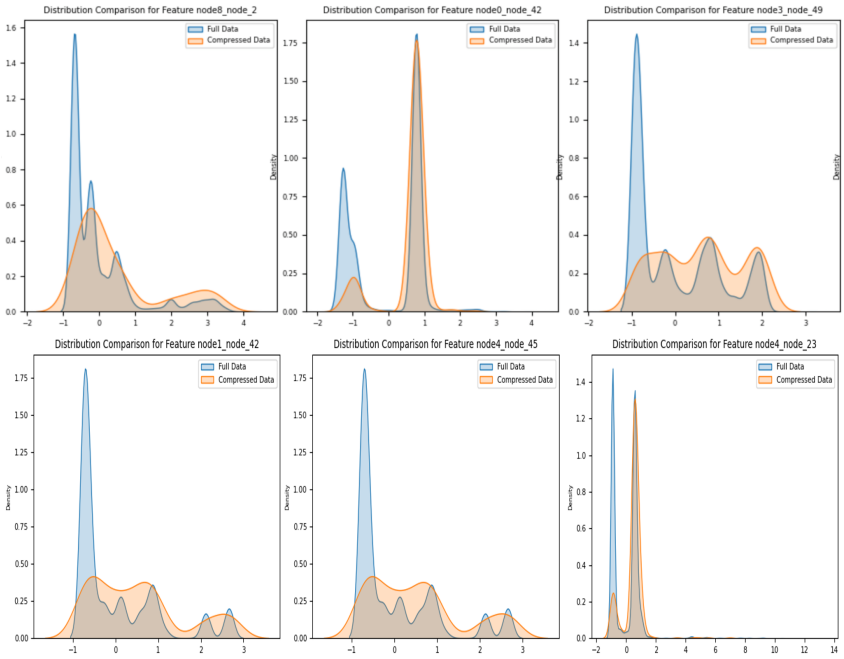

Figure 4.8: Feature Distribution of Full and Core Dataset

Cross-Validation

To further evaluate the robustness and generalizability of the Random Forest model trained on the core dataset, both 5-fold and 7-fold cross-validation were conducted. This approach ensures that the model's performance is consistently reliable across different subsets of the data.

Table 4.7: Cross-Validation Accuracy for Core Model

Fold	**7-Fold Accuracy**	**5-Fold Accuracy**
1	0.9785	0.9673
2	0.9955	0.9830
3	0.9928	0.9917
4	0.9901	0.9958
5	0.9978	0.6600
6	0.9942	—
7	0.8599	—
Mean Accuracy	0.9727 ± 0.0464	0.9196 ± 0.1301

Table 4.8: Cross-Validation F1 Scores for Core Model

Fold	**7-Fold F1 Score**	**5-Fold F1 Score**
1	0.9783	0.9664
2	0.9955	0.9829
3	0.9928	0.9917
4	0.9901	0.9958
5	0.9978	0.6146
6	0.9942	—
7	0.8589	—
Mean F1 Score	0.9725 ± 0.0467	0.9103 ± 0.1482

Cross-Validation Results

Observations The cross-validation results demonstrate that the Random Forest model maintains high accuracy and F1-scores across both 5-fold and 7-fold validations, with mean accuracies of approximately 91.96% and 97.27%, respectively. The standard deviations are relatively low, particularly for the 7-fold cross-validation (± 0.0464 for accuracy and ± 0.0467 for F1-score), indicating consistent performance across different data splits.

However, one fold in the 5-fold cross-validation exhibited significantly lower accuracy and F1-score (66.00% accuracy and 61.46% F1-score), which may be attributed to class imbalance or variability within that particular fold. Similarly, the 7th fold in the 7-fold cross-validation showed lower performance (85.99% accuracy and 85.89% F1-score), highlighting the impact of challenging subsets of data.

Overall, the high mean accuracies and F1-scores across both cross-validation methods affirm the effectiveness of the entropy-based compression method in retaining informative samples that contribute to robust model performance. The slight variations observed in certain folds underscore the importance of addressing class imbalance, which has been mitigated through stratified sampling and class weighting in the compression process.

4.6 Application to Anomaly Detection Dataset

To further validate the effectiveness of the entropy-based Random Forest compression method, it was applied to an anomaly detection dataset. This dataset presents additional challenges, including significant class imbalance, which allows for the assessment of the robustness of the approach in different contexts.

4.6.1 Methodology

An almost identical methodology was employed as in the application classification task described in application classification section , with slight modifications tailored to the anomaly detection dataset. The key steps included:

- **Data Preprocessing**: Handling missing values, normalizing features, and applying variance thresholding to reduce dimensionality.
- **Stratified Splitting**: Dividing the data into training (80%) and testing (20%) sets using stratified sampling to maintain class proportions.
- **Initial Model Training**: Training an initial Random Forest model on the full, preprocessed training dataset using the entropy criterion.
- **Entropy-Based Sampling**: Computing predictive entropies for each training sample and selecting samples based on high and low entropy thresholds (99th and 1st percentiles).
- **Further Compression**: Applying stratified sampling to the core dataset to achieve additional compression while preserving class balance.

The slight changes from the previous methodology included:

- Adjusting the label mapping to reflect the classes in the anomaly detection dataset.

- Incorporating class weighting in the Random Forest model to handle significant class imbalance.
- Performing additional statistical validation tests to assess the impact of compression on data distribution and model performance.

4.6.2 Dataset Characteristics

The anomaly detection dataset consists of sensor readings collected from a compute node in a testbed high-performance computing (HPC) system. The dataset includes measurements from 128 system-wide sensors, sampled every second over a 16-day period. Anomalies were introduced into the system through fault injection, resulting in data that represents both normal operation and various fault conditions.

The response variable indicates different fault types, mapped to numerical labels:

- 0: `pagefail`
- 1: `leak`
- 2: `ddot`
- 3: `memeater`
- 4: `dial`
- 5: `cpufreq`
- 6: `copy`
- 7: `ioerr`
- 8: `None` (Normal Operation)

The dataset exhibits significant class imbalance, with the 'None' class comprising approximately 68% of the data, while the fault classes each represent a smaller proportion (between 2.96% and 4.56% each). Due to this imbalance, stratified sampling and class weighting are essential to ensure that minority classes are adequately represented in both training and testing sets.

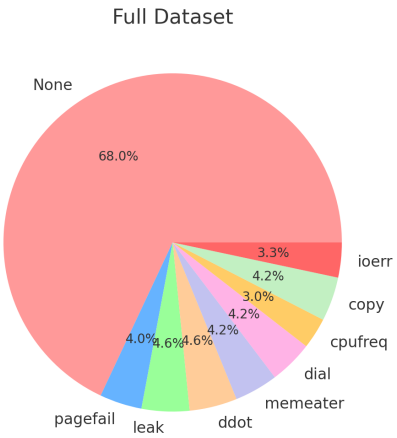

Figure 4.9: Class Distribution of Anomaly Detection Full Dataset

4.6.3 Results

Compression Ratios and Dataset Sizes

The entropy-based compression technique effectively reduced the dataset size while achieving high compression ratios. Table 4.9 summarizes the dataset sizes and compression ratios.

Model Performance

Random Forest models were trained on the compressed datasets, and their performance was evaluated on the test set. The models maintained high accuracy despite the significant reduction in dataset size.

Table 4.9: Model Performance and Dataset Details for Anomaly Detection

Model	Training Dataset	Data Size (KB)	Compression Ratio	Accuracy (%)	F1-Score
Full Model	Full Data	1,092,446.06	1	99.36	0.9936
Core Model	Core Set	430,345.45	2.54	94.87	0.9487
Compressed Core Model	Compressed Core	8,606.23	126.94	89.24	0.8924

Table 4.10: Classification Report for Core Model on Anomaly Detection Dataset

Class	Precision	Recall	F1-Score	Support
pagefail	0.95	0.83	0.89	11,101
leak	0.97	0.95	0.96	12,568
ddot	0.98	0.95	0.97	12,552
memeater	0.98	0.95	0.96	11,441
dial	0.99	0.93	0.96	11,464
cpufreq	0.99	0.96	0.98	8,148
copy	0.99	0.49	0.66	11,599
ioerr	0.92	0.79	0.85	9,195
None	0.94	0.99	0.97	187,195
Accuracy	94.87% (275,263 samples)			
Macro Avg	0.97	0.87	0.91	
Weighted Avg	0.95	0.95	0.94	

Discussion

The results demonstrate that the entropy-based compression method effectively reduced the dataset size while maintaining high classification performance, even in the presence of significant class imbalance. The slight decrease in accuracy is acceptable given the substantial reduction in data size. The model trained on the compressed core dataset performed well on the majority class ('None') and several of the fault classes. However, performance on certain minority classes, such as 'copy' and 'ioerr', was lower due to the class imbalance.

These findings highlight the importance of incorporating class balancing techniques, such as stratified sampling and class weighting, within the compression framework. The method's scalability and effectiveness across different datasets validate its robustness and applicability in various machine learning tasks.

4.6.4 Conclusion

Applying the entropy-based Random Forest compression method to the anomaly detection dataset confirmed the effectiveness of this approach across different types of data. The methodology, with slight adjustments to accommodate the dataset's characteristics, successfully compressed the data while preserving high model performance. This demonstrates that the compression technique is robust and generalizable, making it a valuable tool for reducing computational costs without significantly compromising accuracy in both application classification and anomaly detection tasks.

5 K-Means Clustering and Stratified Sampling for Dataset Compression

Building upon the Entropy-Based Random Forest Compression method introduced in previous chapter, this chapter explores alternative dataset compression techniques, specifically K-Means clustering combined with stratified sampling, to evaluate their effectiveness in enhancing application classification tasks using high-dimensional sensor data from multiple nodes.

While the entropy-based approach leverages predictive uncertainties to identify and retain the most informative samples, K-Means clustering aims to reduce dataset size by grouping similar data points and selecting representative cluster centroids. Stratified sampling is employed alongside K-Means to preserve the original class distribution, ensuring that the compressed dataset remains balanced and mitigating biases toward majority classes. Additionally, Principal Component Analysis (PCA) is examined as another dimensionality reduction technique to provide a comprehensive comparison.

The motivation for this comparative analysis stems from the need to identify the most effective compression strategy that balances computational efficiency, storage reduction, and model performance.

5.1 Background and Theoretical Foundations

5.1.1 K-Means Clustering Overview

K-Means clustering is a predominant unsupervised learning algorithm designed to partition a dataset into K distinct clusters based on feature similarity [15]. The algorithm's primary goal is to minimize the **Within-Cluster Sum of Squares (WCSS)**, a metric that quantifies the compactness of clusters. The WCSS is mathematically defined as:

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{5.1}$$

where:

- C_k denotes the set of data points within cluster k.
- x_i represents an individual data point.
- μ_k is the centroid of cluster k.

5.2 Methodology

5.2.1 Overview of the Compression Techniques

Our dataset compression strategy integrates two principal techniques:

1. **K-Means Clustering for Representative Sample Selection**: Utilizes K-Means clustering to partition the dataset into K clusters and selects data points nearest to each centroid as representative samples.
2. **Stratified Sampling for Class Balance**: Implements stratified sampling to ensure the compressed dataset maintains the original class distribution, which is critical for classification tasks.

By amalgamating these techniques, the goal is to generate a compressed dataset that is markedly smaller than the original while preserving its predictive efficacy.

5.2.2 Detailed Steps

The K-Means Clustering methodology builds upon the foundational data preparation steps outlined in the Entropy-Based Compression approach. The following sections detail the steps specific to K-Means Clustering, while referencing the identical procedures for data loading, splitting, and normalization.

Data Loading and Merging

As detailed in data loading section in previous chapter, the dataset comprises sensor readings from multiple nodes alongside corresponding application labels. The data loading and merging process involves importing sensor data and response labels, encoding labels numerically, merging datasets based on common identifiers, and pivoting the data to form a wide-format feature matrix.

Time-Based Splitting

Consistent with the Entropy-Based approach, the dataset is split temporally into training (80%) and testing (20%) sets. This ensures that the model is trained on historical data and evaluated on future data, simulating real-world deployment scenarios.

Data Preparation and Normalization

The data preparation steps, including feature selection, standard scaling, and variance thresholding, are identical to those described in entropy-based approach. Specifically, non-informative features are excluded, data is normalized to have zero mean and unit variance, and low-variance features are removed based on a predefined threshold (e.g., 0.1).

Variance Threshold Feature Selection

Following the same procedure as in the Entropy-Based method, variance thresholding is applied to eliminate features with low variability. This reduces dimensionality and enhances the efficiency of the K-Means clustering algorithm.

K-Means Clustering for Core Dataset Selection

K-Means clustering is employed to identify and extract a **core dataset** comprising representative samples that encapsulate the data's inherent structure.

Input: Preprocessed training dataset X_{train}, Number of clusters K
Output: Core dataset X_{core} with corresponding labels y_{core}
begin
  `// Initialize K-Means`
  Initialize K-Means model with K clusters;
  Fit K-Means model on X_{train} to obtain centroids μ_k and cluster assignments;
  Initialize an empty list `core_indices` to store indices of core data points;
  for *each cluster* $k = 1$ **to** K **do**
    Identify centroid μ_k for cluster k;
    Compute Euclidean distance between each data point x_i in cluster k and the centroid μ_k;
    Select data point x_i with minimum distance to μ_k;
    Append index of x_i to `core_indices`;
  end
  Construct core dataset $X_{\text{core}} = X_{\text{train}}[\texttt{core\_indices}]$ and labels $y_{\text{core}} = y_{\text{train}}[\texttt{core\_indices}]$;
  return $X_{\text{core}}, y_{\text{core}}$;
end

Algorithm 4: K-Means Clustering for Core Dataset Selection

Algorithm Explanation:

1. **Initialization**: A K-Means model is instantiated with K clusters (e.g., 500), chosen to balance compression and representativeness based on dataset size and diversity.
2. **Clustering**: The model is fitted to the preprocessed training data, yielding centroids μ_k and cluster assignments for each data point.
3. **Representative Selection**: For each cluster, the data point nearest to the centroid (i.e., with the minimum Euclidean distance) is selected as a representative sample.
4. **Core Dataset Construction**: The selected indices are used to construct the core dataset X_{core} along with their corresponding labels y_{core}.

This method ensures that the core dataset encapsulates the central tendencies of each cluster, thereby retaining the dataset's structural integrity while achieving compression.

Ensuring Class Balance with Stratified Sampling

To uphold the original class distribution within the compressed dataset, **stratified sampling** is applied to the core dataset. This process involves sampling proportionally from each class, ensuring that minority classes remain adequately represented.

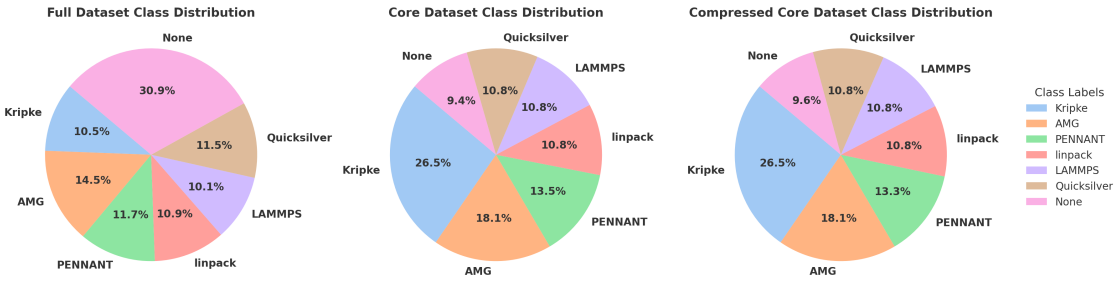

Figure 5.1: Class Distribution of Application Dataset Before and After Stratified Sampling

Input: Core dataset X_{core}, Labels y_{core}, Desired compression ratio C
Output: Compressed core dataset $X_{\text{compressed}}$, Labels $y_{\text{compressed}}$
Determine sampling fraction $f = \frac{1}{C}$;
Function `StratifiedSample(`X, y, f`)`**:**
    Initialize `sampled_indices` $\leftarrow \emptyset$;
    for *each class* c *in* y **do**
        Let N_c be the number of samples in class c;
        Calculate number of samples $n_c = \lceil f \times N_c \rceil$;
        Randomly sample n_c indices from class c;
        `sampled_indices` $\leftarrow$ `sampled_indices` $\cup$ sampled indices;
    end
    return $X_{\text{sampled}} \leftarrow X[\texttt{sampled\_indices}]$;
    return $y_{\text{sampled}} \leftarrow y[\texttt{sampled\_indices}]$;
$X_{\text{compressed}}, y_{\text{compressed}} \leftarrow \texttt{StratifiedSample}(X_{\text{core}}, y_{\text{core}}, f)$;
return $X_{\text{compressed}}$, $y_{\text{compressed}}$;

Algorithm 5: Stratified Sampling for Class Balance

Algorithm Explanation:

1. **Sampling Fraction Calculation**: Determines the fraction f of the core dataset to retain based on the desired compression ratio C.
2. **Stratified Sampling Function**: Iterates through each class c in y, calculates the number of samples n_c to retain proportionally, and randomly selects n_c samples from class c.
3. **Compressed Dataset Construction**: Aggregates the sampled indices to form the compressed core dataset $X_{\text{compressed}}$ and their corresponding labels $y_{\text{compressed}}$.

This method ensures that the compressed dataset preserves the original class distribution, thereby maintaining the balance necessary for effective model training and evaluation.

5.3 Experimental Setup

5.3.1 Datasets

1. **Application Classification Dataset**: Comprises sensor readings from 16 nodes, each monitoring various system metrics over time. The data includes measurements such as CPU usage, memory utilization, network traffic, and other performance indicators. Each data point is labeled with the active application at that time, including an explicit 'None' class when no application is running.
2. **Anomaly Detection Dataset**: Consists of sensor readings from an HPC system's compute nodes, capturing normal operation and artificially injected anomalies. The response variable categorizes each data point into one of nine classes, including 'None' and various fault conditions (e.g., pagefail, leak, ddot).

Both datasets exhibit high dimensionality and class imbalance especially anomaly detection, presenting significant challenges for model training and evaluation.

5.3.2 Evaluation Metrics

Performance is gauged using the following metrics:

- **Accuracy**: Measures the overall correctness of the model's predictions.
- **Precision, Recall, F1-score**: Evaluated per class to assess class-wise performance, particularly important in imbalanced datasets.
- **Confusion Matrix**: Provides a visual representation of prediction outcomes versus actual labels.
- **Cross-Validation**: K-fold cross-validation (e.g., 5-fold) to assess the model's robustness and generalizability across different data splits, mitigating the risk of overfitting.

Table 5.1: Cross-Validation Results on Core Model

Fold	Accuracy (%)	F1 Score (%)
1	97.85	97.83
2	99.55	99.55
3	99.28	99.28
4	99.01	99.01
5	99.78	99.75
6	99.42	99.42
7	85.99	85.89
Mean Accuracy	97.27 ± 4.64	97.25 ± 4.67

Compression effectiveness is quantified by:

- **Compression Ratio**: Defined as the original dataset size divided by the compressed dataset size.
- **Data Size (KB)**: Reflects the memory footprint of the datasets.

5.4 Results

5.4.1 Compression Ratios and Dataset Sizes

Implementing K-Means clustering with $K = 500$ clusters yields substantial compression. Table 5.2 encapsulates the dataset sizes and compression ratios achieved through various stages.

Table 5.2: Dataset Sizes and Compression Ratios Using K-Means Clustering ($K = 500$)

Dataset	Samples	Data Size (KB)	Compression Ratio
Full Data	62,357	381,449.46	1
Core Dataset	498	3,046.36	125.21
Compressed Core Dataset	249	1,523.18	250.43

5.4.2 Class Distribution in Core Dataset

Post K-Means clustering, the class distribution within the core dataset is scrutinized to evaluate representativeness. Table 5.3 delineates the proportion and count of each class.

Table 5.3: Class Distribution in Core Dataset

Class	Proportion	Count
None	26.51%	132
AMG	18.07%	90
Kripke	13.45%	67
LAMMPS	10.84%	54
linpack	10.84%	54
Quicksilver	10.84%	54
PENNANT	9.44%	47

5.4.3 Model Performance

Random Forest classifiers trained on the compressed datasets demonstrate commendable accuracy, as summarized in Table 5.4.

Table 5.4: Model Performance on Test Data

Model	Accuracy (%)	F1-Score (%)
Full Model	99.32	99.32
Core Model	98.72	98.72
Compressed Core Model	98.27	98.27

Classification Reports

Detailed classification reports offer granular insights into model performance across individual classes.

Table 5.5: Classification Report for Full Model (Training: 62,357 samples, Test: 15,590 samples)

Class	Precision	Recall	F1-Score	Support
Kripke	0.99	0.99	0.99	1,639
AMG	1.00	0.99	1.00	2,260
PENNANT	1.00	0.99	0.99	1,818
linpack	1.00	0.99	0.99	1,699
LAMMPS	1.00	0.99	0.99	1,567
Quicksilver	0.99	0.99	0.99	1,797
None	0.99	0.99	0.99	4,810
Accuracy	99.32% (15,590 samples)			

Full Model

Table 5.6: Classification Report for Core Model (Training: 498 samples, Test: 15,590 samples)

Class	Precision	Recall	F1-Score	Support
Kripke	0.96	0.99	0.97	1,639
AMG	1.00	0.99	0.99	2,260
PENNANT	0.99	0.99	0.99	1,818
linpack	1.00	0.97	0.99	1,699
LAMMPS	1.00	0.96	0.98	1,567
Quicksilver	0.99	0.99	0.99	1,797
None	0.98	0.99	0.99	4,810
Accuracy	98.72% (15,590 samples)			

Core Model

Table 5.7: Classification Report for Compressed Core Model (Training: 249 samples, Test: 15,590 samples)

Class	Precision	Recall	F1-Score	Support
Kripke	0.96	0.99	0.97	1,639
AMG	0.99	0.98	0.99	2,260
PENNANT	0.98	0.99	0.99	1,818
linpack	1.00	0.96	0.98	1,699
LAMMPS	0.99	0.96	0.98	1,567
Quicksilver	0.97	0.99	0.98	1,797
None	0.98	0.99	0.99	4,810
Accuracy	98.27% (15,590 samples)			

Compressed Core Model

Confusion Matrices

Confusion matrices visually encapsulate the performance of the models across different classes. Figures 5.2, 5.3, and 5.4 depict the confusion matrices for the full model, core model, and compressed core model, respectively.

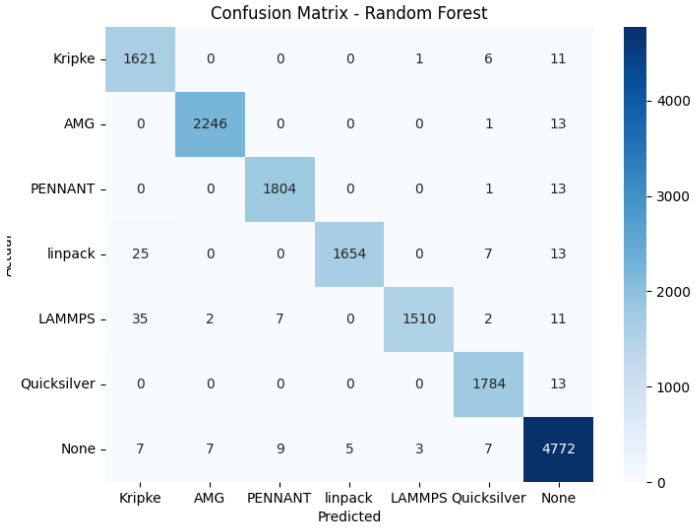

Figure 5.2: Confusion Matrix for Full Model

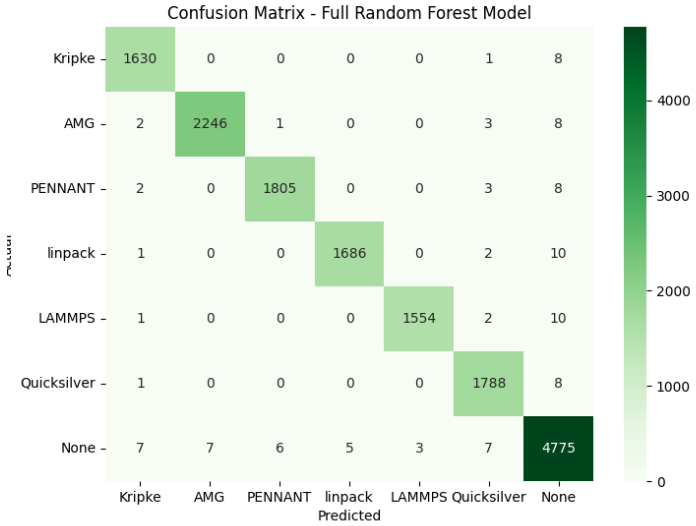

Figure 5.3: Confusion Matrix for Core Model

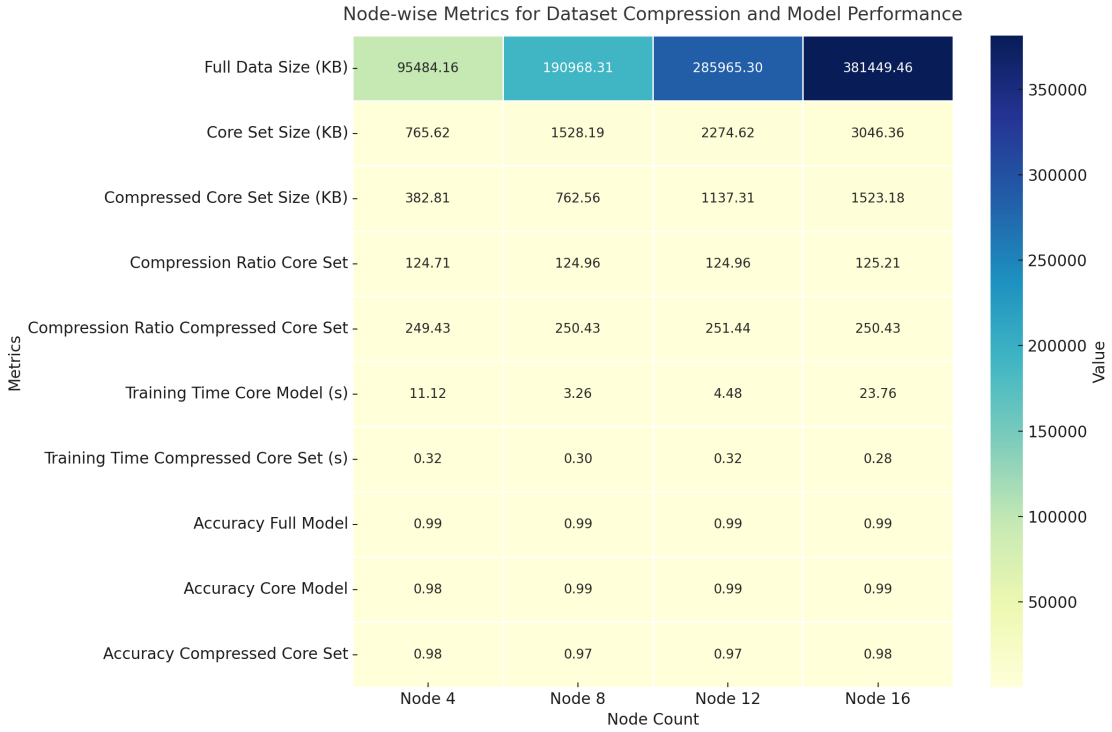

Figure 5.4: Confusion Matrix for Compressed Core Model

Observations:

- **Full Model**: Exhibits near-perfect classification across all classes, indicating high reliability.
- **Core Model**: Demonstrates a slight decrease in performance metrics, which is expected due to data compression.
- **Compressed Core Model**: Maintains high accuracy and F1-score, albeit with minor declines, showcasing the efficacy of the compression techniques in preserving model performance.

Impact of Stratified Sampling on Model Performance

Stratified sampling plays a pivotal role in maintaining class balance within the compressed dataset, which directly influences model performance. The following observations highlight its impact:

- **Accuracy and F1-Score**: There is a marginal decline in both accuracy and F1-score post-stratified sampling (from 98.72% to 98.27%), indicating that the model sustains its predictive capabilities despite the reduced dataset size.

- **Class-wise Performance**: Minority classes retain their performance metrics, ensuring that the model does not become biased towards majority classes. This balance is crucial for applications where detecting minority classes is essential.
- **Computational Efficiency**: The compressed dataset facilitates faster training times, enhancing overall computational efficiency without substantial compromises in performance.

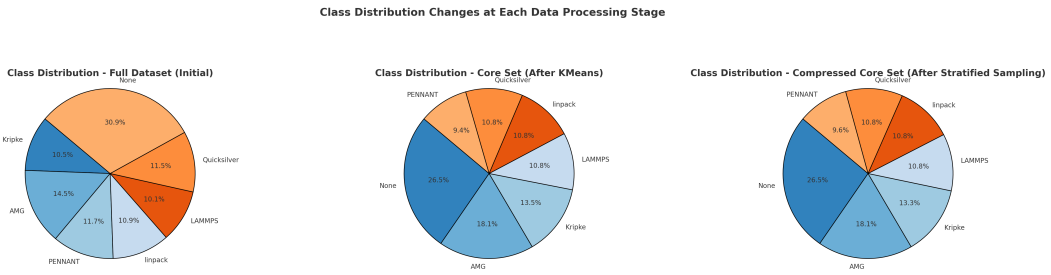

Figure 5.5: Class Distribution Across Data Processing Stages: Initial Dataset, Core Set After KMeans, and Final Compressed Core Set After Stratified Sampling

Figure 5.5 shows side by side the class distributions in the core dataset against those in the compressed core dataset post-stratified sampling. The visual alignment underscores the effectiveness of stratified sampling in preserving class balance.

5.4.4 Statistical Validation

To ascertain that the compression techniques preserve the dataset's essential characteristics, several statistical tests were conducted.

Kolmogorov-Smirnov (KS) Test

The **KS test** assesses the similarity between the feature distributions of the full dataset and the compressed core dataset.

- **Number of Features with Similar Distributions**: 17 out of 127 features (13.39%) exhibit similar distributions post-compression.
- **Implications**: A relatively low percentage compared application classification indicates that while some features maintain their distributional integrity, the majority experience significant alterations. This discrepancy may influence model performance, particularly for features critical to classification tasks.

Visualization of Feature Distributions

To visually assess the preservation of feature distributions, **Kernel Density Estimation (KDE)** plots were generated for selected features.

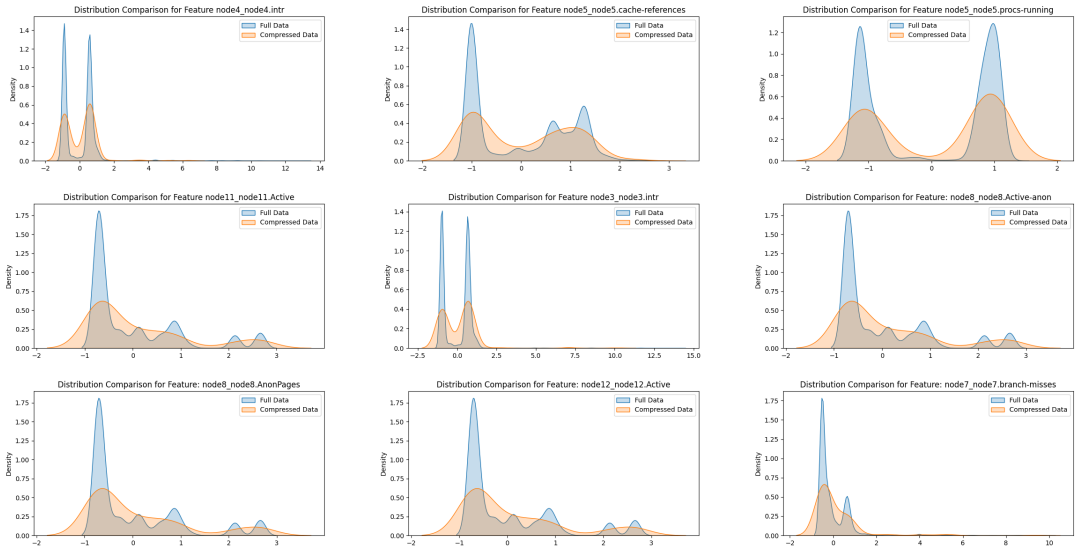

Figure 5.6: Feature Distribution Comparison between Full and Compressed Core Dataset

Figure 5.6 juxtaposes the KDE plots of five randomly selected features from the full dataset against those from the compressed core dataset. The plots illustrate that:

- **Similar Distributions**: Certain features retain their distribution shapes, indicating effective preservation.
- **Divergent Distributions**: Other features exhibit noticeable deviations, which may impact model performance.

These visualizations validate the statistical tests, highlighting the nuanced effects of compression on feature distributions.

5.4.5 Observations

The integration of **K-Means clustering** and **stratified sampling** in the anomaly detection dataset results in a substantial compression ratio of 214x, reducing the dataset from over a hundred thousand samples to just 503. This compression facilitates **faster model training** and **reduced storage requirements**, which are paramount in real-world HPC environments.

However, the compression introduces a significant decline in model performance, particularly evident in the **Compressed Core Model** which achieves an accuracy of 89.24% compared to the **Full Model's** 99.36%. This drop is primarily attributed to:

- **High Compression Ratio**: Extreme reduction in dataset size inherently leads to loss of information.
- **Class Imbalance**: Despite stratified sampling, some class imbalance persists, adversely affecting the model's ability to detect minority classes.
- **Altered Feature Distributions**: The majority of features exhibit significant distributional changes, which can impair the model's ability to accurately classify data points.

Key Insights:

- **Compression Efficiency**: The entropy-based compression method achieves a higher compression ratio (214x) compared to K-Means clustering combined with stratified sampling (127x), demonstrating its potential for extreme dataset size reductions.
- **Performance Trade-offs**: Higher compression ratios lead to more pronounced declines in model performance, particularly in detecting minority classes, underscoring the necessity for balanced compression strategies.
- **Statistical Preservation**: A majority of feature distributions and class distributions are significantly altered, suggesting that further refinement is essential to preserve critical data characteristics.
- **Scalability**: Despite the observed performance drops, the methods are scalable and can be effectively applied to large datasets, making them suitable for environments with limited computational resources.

Proven Method Effectiveness:

The results affirm that both **K-Means clustering** and **Entropy-Based Random Forest Compression** are viable methods for dataset compression, each with distinct strengths and limitations. While these methods successfully reduce dataset sizes, thereby enhancing computational efficiency and reducing storage demands, they introduce trade-offs in model performance, especially concerning minority classes. This indicates that compression strategies must be judiciously balanced to maintain critical detection capabilities.

6 Principal Component Analysis (PCA) for Dataset Compression

While the primary focus of this thesis is on the entropy-based Random Forest compression and K-Means clustering with stratified sampling, exploring alternative methods like Principal Component Analysis (PCA) provides valuable insights into dataset compression strategies. PCA is a fundamental technique for dimensionality reduction that transforms the original features into a new set of uncorrelated components, capturing the most significant variance in the data. This chapter investigates the application of PCA for dataset compression in both the *Application Classification* and *Anomaly Detection* tasks, evaluates its effectiveness, and compares it with the other compression techniques discussed in previous chapters.

6.1 Methodology

6.1.1 Data Preparation

Following the initial preprocessing steps outlined in previous chapters, the datasets underwent standardization to ensure that each feature contributed equally to the analysis. Variance thresholding was applied to remove low-variance features, reducing dimensionality and improving computational efficiency.

6.1.2 PCA Transformation

PCA was applied separately to the standardized *Application Classification* and *Anomaly Detection* datasets. The number of principal components was determined based on the cumulative explained variance ratio, retaining components that collectively explained at least 95% of the variance. This approach aimed to balance dimensionality reduction with information retention.

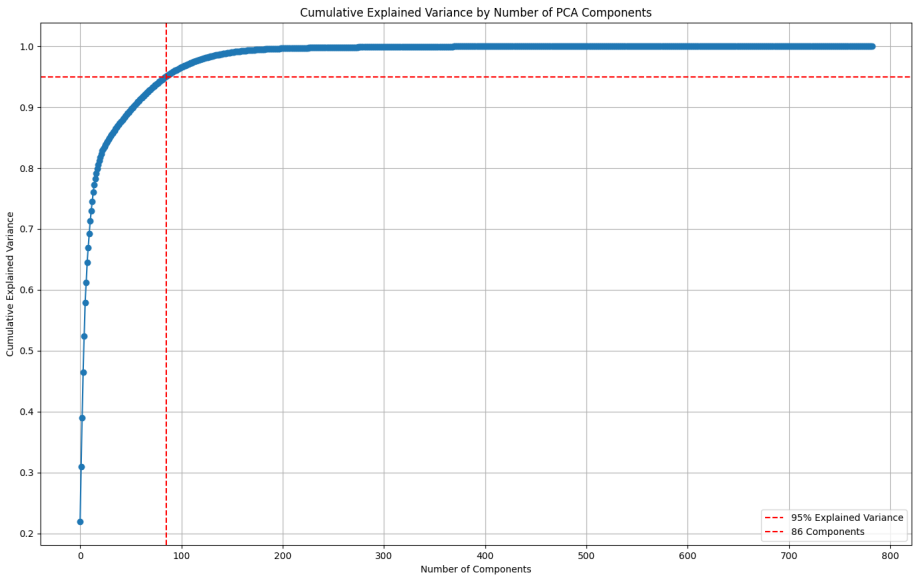

Figure 6.1: Cumulative Explained Variance by Number of PCA Components

Figure 6.1 shows that for the Application Classification dataset, 86 principal components capture 95% of the variance, achieving significant dimensionality reduction while retaining key information. The rapid rise in explained variance initially, followed by a plateau, supports this choice as a balance between data retention and reduction.

6.1.3 Model Training

Random Forest classifiers were trained on the PCA-transformed datasets. The hyperparameters were kept consistent with those used in the entropy-based and K-Means methods to ensure a fair comparison. The models were evaluated using the same training and testing splits as in previous experiments.

6.1.4 Evaluation Metrics

The performance of the PCA-based models was assessed using:

- **Accuracy**: The overall correctness of the model's predictions.
- **F1-Score**: The harmonic mean of precision and recall, providing a balance between the two.
- **Compression Ratio**: The ratio of the original dataset size to the compressed dataset size.

6.1.5 Statistical Validation

To evaluate the impact of PCA on feature and class distributions, the following statistical tests were conducted:

- **Kolmogorov-Smirnov (KS) Test**: Assessed the similarity of feature distributions between the full and PCA-transformed datasets.

6.2 Results

6.2.1 Application Classification Dataset

PCA Transformation

- **Initial Number of Features**: 783
- **Number of Principal Components**: 86 (retaining 95% of the variance)

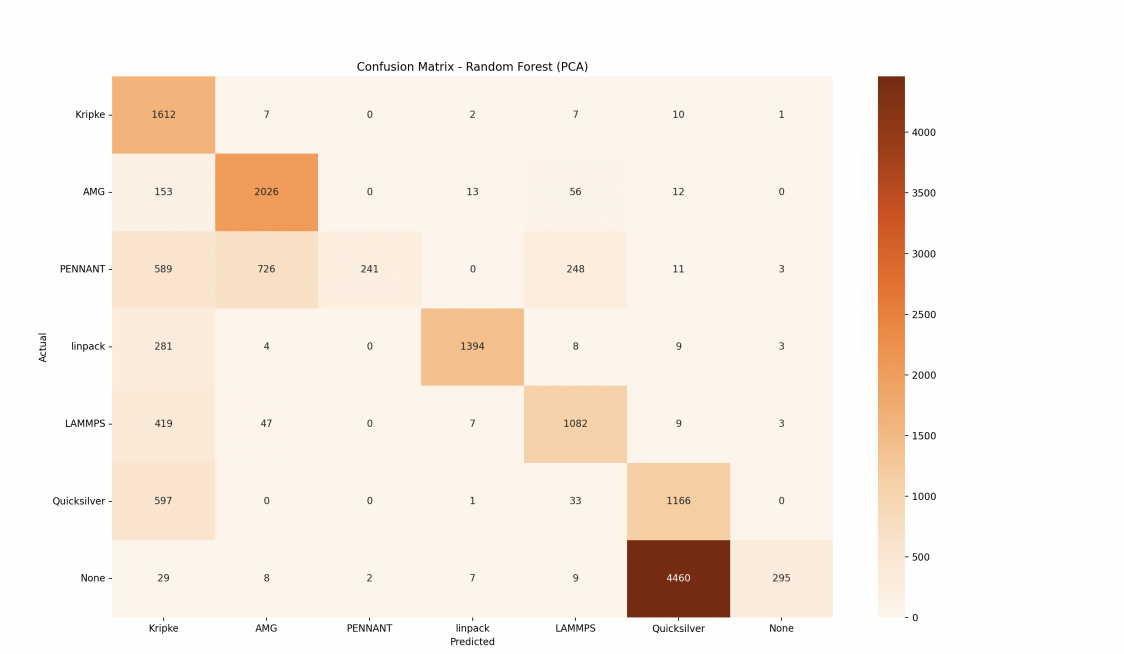

Figure 6.2: Confusion matrix of the Random Forest classifier after entropy-driven sampling on the application classification dataset.

Model Performance

The Random Forest model trained on the PCA-transformed dataset exhibited a significant decline in performance compared to models trained on the full dataset and those compressed using entropy-based methods or K-Means clustering.

Table 6.1: Comparative Performance of Compression Techniques on the **Application Classification Dataset**

Methodology	Samples	Features	Compression Ratio	Accuracy (%)	F1-Score (%)	Data Size (KB)
Full Dataset (Baseline)	62,357	783	1x	99.32	99.32	381,449.46
Entropy-Based RF Compression	25,177	783	2.48x	98.61	98.61	154,012.43
Compressed Core (503 samples)	503	783	123.97x	97.00	—	3,076.95
K-Means + Stratified Sampling	498	783	125.21x	98.72	98.72	3,046.36
Compressed Core (249 samples)	249	783	250.43x	98.27	98.27	1,523.18
PCA-Based Compression	8458	86	45.09x	46.21	39.00	—

The model's accuracy dropped from 98.61% (entropy-based compression) to 46.21% after PCA transformation, indicating a substantial loss in predictive power.

Statistical Validation

- **Kolmogorov-Smirnov (KS) Test**: None of the 86 principal components retained similar distributions compared to the original features (0% similarity).

6.2.2 Anomaly Detection Dataset

PCA Transformation

- **Initial Number of Features**: 127
- **Number of Principal Components**: 28 (retaining 95% of the variance)

Model Performance

The Random Forest model trained on the PCA-transformed dataset showed a moderate decrease in performance.

Table 6.2: Comparative Performance of Compression Techniques on the **Anomaly Detection Dataset**

Methodology	Samples	Features	Compression Ratio	Accuracy (%)	F1-Score (%)	Data Size (KB)
Full Dataset (Baseline)	1,101,048	127	1x	99.36	99.36	1,092,446.06
Entropy-Based RF Compression	433,734	127	2.54x	94.87	94.87	430,345.45
Compressed Core (8,674 samples)	8,674	127	126.94x	89.24	89.24	8,606.23
K-Means + Stratified Sampling	10,270	127	107.21x	97.03	96.97	10,189.77
Compressed Core (5,135 samples)	5,135	127	214.42x	95.83	95.83	5,094.88
PCA-Based Compression	8,674	28	126.94x	87.95	87.95	—

The model's accuracy decreased from 94.87% (entropy-based compression) to 87.95% after PCA transformation.

Statistical Validation

- **Kolmogorov-Smirnov (KS) Test**: Only 13.39% of the PCA components had similar distributions to the original features.

6.3 Discussion

6.3.1 Performance Impact

Application Classification

The drastic decline in accuracy from 98.61% to 46.21% suggests that PCA failed to preserve the critical information necessary for accurate application classification. The reduction from 783 features to 86 principal components may have led to the loss of essential feature relationships and interactions that the Random Forest model relied upon for prediction.

Anomaly Detection

In the anomaly detection task, the accuracy decrease was less severe, from 94.87% to 87.95%. This indicates that PCA retained more relevant information in this dataset, possibly because the anomalies could be captured within the principal components that explain the majority of the variance.

6.3.2 Comparative Analysis with Other Methods

When comparing PCA-based compression with entropy-based Random Forest compression and K-Means clustering with stratified sampling, it is evident that PCA was less effective in maintaining model performance, especially in the application classification task.

- **Entropy-Based RF Compression**: Achieved high accuracy with reasonable compression ratios by selecting samples based on predictive entropy, ensuring that informative samples were retained.
- **K-Means + Stratified Sampling**: Provided substantial compression with minimal loss in accuracy, effectively preserving class balance and essential data characteristics.
- **PCA-Based Compression**: Led to significant performance degradation, indicating that dimensionality reduction through PCA may not be suitable for all types of datasets, particularly those where feature interactions are complex and critical.

6.3.3 Statistical Validation Insights

The statistical tests highlight the challenges associated with PCA-based compression:

- **Feature Distribution Preservation**: The low percentage of PCA components with similar distributions suggests that PCA significantly altered the feature space, potentially impacting the model's ability to generalize.
- **Class Distribution Changes**: Significant differences in class distributions after PCA indicate that the compression may have introduced bias or altered the representation of certain classes, contributing to reduced model performance.

6.3.4 Feature Distribution Preservation

Kernel Density Estimation (KDE) plots comparing feature distributions before and after PCA revealed noticeable divergences, particularly in the application classification dataset. These visualizations support the statistical findings and underscore the importance of preserving feature distributions for maintaining model accuracy.

6.4 Observations

- **Application Classification Task**: PCA-based compression severely impacted model performance, rendering it ineffective for this task.
- **Anomaly Detection Task**: PCA achieved higher compression ratios with a moderate decrease in accuracy, suggesting it may be more suitable for datasets where the primary variance captures essential information.
- **Entropy-Based and K-Means Methods**: Outperformed PCA by maintaining higher accuracy levels while achieving substantial compression, highlighting their superiority for the datasets studied.
- **Importance of Data Characteristics**: The effectiveness of PCA is highly dependent on the dataset's nature and the significance of feature relationships, indicating that a one-size-fits-all approach to compression may not be feasible.

6.5 Conclusion

The application of PCA for dataset compression yielded mixed results. While it effectively reduced dimensionality, the accompanying loss of critical information led to significant performance declines in certain tasks. The entropy-based Random Forest compression and K-Means clustering with stratified sampling methods proved more effective in preserving model accuracy and essential data characteristics.

These findings suggest that while PCA can be a valuable tool for dimensionality reduction, its suitability for dataset compression in machine learning tasks depends on the specific context and data properties. For tasks where feature interactions and distributions play a crucial role, alternative methods that preserve these aspects are preferable.

6.6 Figures and Tables

6.6.1 Tables

Table 6.3: Summary of PCA Compression Results

Dataset	Samples	Accuracy (%)	Compression Ratio	Number of Features
Application Classification				
Full Data	62,357	99.32	1.00	783
Core Set	25,177	98.61	2.48	783
PCA-Based Core Set	25,177	46.21	2.48	86
Anomaly Detection				
Full Data	1,101,048	99.36	1.00	127
Core Set	8,674	89.24	126.94	127
PCA-Based Core Set	8,674	87.95	126.94	28

Table 6.4: Comparative Performance of Compression Techniques on the Application Classification Dataset

Methodology	Samples	Features	Compression Ratio	Accuracy (%)	F1-Score (%)	Data Size (KB)
Full Dataset (Baseline)	62,357	783	1x	99.32	99.32	381,449.46
Entropy-Based RF Compression	25,177	783	2.48x	98.61	98.61	154,012.43
Compressed Core (503 samples)	503	783	123.97x	97.00	—	3,076.95
K-Means + Stratified Sampling	498	783	125.21x	98.72	98.72	3,046.36
Compressed Core (249 samples)	249	783	250.43x	98.27	98.27	1,523.18
PCA-Based Compression	25,177	86	2.48x	46.21	39.00	—

Table 6.5: Comparative Performance of Compression Techniques on the Anomaly Detection Dataset

Methodology	Samples	Features	Compression Ratio	Accuracy (%)	F1-Score (%)	Data Size (KB)
Full Dataset (Baseline)	1,101,048	127	1x	99.36	99.36	1,092,446.06
Entropy-Based RF Compression	433,734	127	2.54x	94.87	94.87	430,345.45
Compressed Core (8,674 samples)	8,674	127	126.94x	89.24	89.24	8,606.23
K-Means + Stratified Sampling	10,270	127	107.21x	97.03	96.97	10,189.77
Compressed Core (5,135 samples)	5,135	127	214.42x	95.83	95.83	5,094.88
PCA-Based Compression	8,674	28	126.94x	87.95	87.95	—

6.7 Final Remarks

This chapter explored PCA as an alternative dataset compression method and provided a comprehensive evaluation of its effectiveness compared to entropy-based Random Forest compression and K-Means clustering with stratified sampling. The results emphasize the importance of selecting compression techniques that align with the specific characteristics of the dataset and the requirements of the machine learning task.

7 Innovation

This study introduces an innovative entropy-based Random Forest compression method that leverages predictive uncertainties to identify and retain the most informative samples from the original dataset. By focusing on samples with both high and low predictive entropy, this method creates a compressed dataset that maintains the core structure and variability of the original data while significantly reducing computational costs.

7.1 Entropy-Based Random Forest Compression Method

The core innovation of this thesis is the development of an entropy-based compression technique that utilizes the predictive entropy derived from a trained Random Forest classifier. Unlike unsupervised methods that do not consider the target variable or the model's predictive needs, this approach is model-aware. It directly incorporates the model's confidence in its predictions to guide the selection of a core dataset.

By calculating the predictive entropy for each sample, the method identifies:

- **High-Entropy Samples**: Data points where the model is uncertain about its predictions. These samples are often located near class boundaries or in regions with sparse data, providing valuable information for refining the model.
- **Low-Entropy Samples**: Data points where the model is highly confident. Retaining these ensures that the core dataset represents the well-understood regions of the data space.

This dual focus ensures that the compressed dataset includes a balanced representation of the data's complexity, capturing both the nuances and the general trends necessary for effective model training.

7.2 Advantages Over Traditional Methods

7.2.1 Enhanced Model Performance with Significant Compression

The entropy-based method achieves a superior balance between dataset compression and model accuracy compared to traditional techniques. In the application classification task, for example, the Random Forest model trained on the entropy-based compressed dataset maintained an accuracy of 98.61% with a compression ratio of approximately

2.5x. In contrast, PCA-based compression led to a significant drop in accuracy to 46.21%. This demonstrates the method's ability to reduce data volume substantially without compromising predictive performance.

7.2.2 Computational Efficiency and Scalability

By avoiding iterative clustering processes inherent in methods like K-Means, the entropy-based approach reduces computational complexity. Calculating predictive entropy and selecting samples based on these values can be performed efficiently, especially since Random Forest predictions are amenable to parallelization. This efficiency makes the method suitable for large datasets and time-sensitive applications where computational resources are at a premium.

7.2.3 Effective Handling of High-Dimensional and Time-Series Data

High-dimensional data often poses challenges due to the curse of dimensionality, which can impair the effectiveness of algorithms like K-Means clustering. The entropy-based method leverages Random Forests' inherent capability to handle high-dimensional feature spaces and complex feature interactions. By utilizing the model's predictive uncertainties, the method effectively manages time-series data's temporal dependencies and high dimensionality.

7.2.4 Model-Aware Sample Selection and Class Balance Preservation

The method's use of predictive entropy ensures that sample selection is directly informed by the model's learning needs, focusing on areas where the model can improve. Additionally, incorporating stratified sampling techniques allows the compressed dataset to maintain the original class distribution. This is particularly important in datasets with class imbalance, where minority classes may be underrepresented. The entropy-based method addresses this by ensuring that samples from all classes are adequately retained, enhancing the model's ability to generalize across the entire dataset.

7.2.5 Scalability and Parallelization

The computational processes involved in the entropy-based method, including entropy calculation and sample selection, can be parallelized. This scalability is crucial for handling big data applications where processing speed and resource utilization are critical factors. In contrast, methods like K-Means can be less scalable due to their iterative nature and computational demands in high-dimensional spaces.

7.3 Comparative Analysis with K-Means and PCA

7.3.1 Limitations of K-Means Clustering

K-Means clustering aims to partition data into clusters based on feature similarity, selecting centroids as representative samples. However, it is unsupervised and does not consider the target variable, potentially overlooking samples critical for the model's predictive performance. In high-dimensional spaces, K-Means can struggle due to the diminished significance of distance measures, leading to less meaningful clusters. Furthermore, K-Means does not inherently address class imbalance, which can result in compressed datasets that are not representative of the minority classes.

Experimental results indicate that while K-Means combined with stratified sampling can achieve high compression ratios, the model performance on the compressed data may decline. For instance, in the anomaly detection task, the accuracy dropped from 99.36% (full model) to 89.24% (compressed core model using K-Means), highlighting the trade-offs involved.

7.3.2 Challenges with PCA-Based Compression

PCA reduces dimensionality by transforming the data into a set of orthogonal components that capture the maximum variance. While effective in some contexts, PCA can discard dimensions that are crucial for classification but do not contribute significantly to variance. This can lead to a loss of critical information, particularly in datasets where important features have low variance.

In the application classification task, PCA-based compression resulted in a dramatic decrease in model accuracy to 46.21%, indicating that PCA failed to preserve essential feature relationships necessary for accurate predictions. The statistical tests confirmed significant alterations in feature and class distributions after PCA transformation, underscoring its limitations in preserving the data's intrinsic structure.

7.3.3 Empirical Evidence Supporting the Entropy-Based Method

The entropy-based method consistently outperformed K-Means and PCA in both the application classification and anomaly detection tasks:

- **Application Classification**: Achieved high accuracy (98.61%) with significant compression, outperforming PCA and matching or exceeding the performance of K-Means with fewer samples.
- **Anomaly Detection**: Maintained robust performance (94.87% accuracy) while achieving substantial data reduction, whereas PCA and K-Means experienced greater drops in accuracy.
- **Feature Distribution Preservation**: Statistical validation showed that the entropy-based method better preserved key feature distributions compared to PCA, which

significantly altered them. This preservation is critical for maintaining model performance.

7.4 Applicability and Use Cases

7.4.1 Optimal Scenarios for the Entropy-Based Method

The entropy-based Random Forest compression method is particularly well-suited for scenarios involving:

- **Large-Scale Time-Series Data**: Applications like sensor networks, where data is continuously generated and timely processing is essential.
- **High-Dimensional Feature Spaces**: Situations where datasets have a large number of features, and traditional methods struggle with dimensionality.
- **Class Imbalance**: Tasks where minority classes are critical, such as anomaly or rare event detection, and maintaining class balance is vital.
- **Resource Constraints**: Environments with limited computational resources or where computational efficiency is a priority, such as edge computing devices.

7.5 Conclusion

This thesis presents a novel entropy-based Random Forest compression method that advances the field of dataset compression for machine learning applications. By leveraging predictive uncertainties, the method effectively reduces dataset size while preserving, and in some cases enhancing, model performance. The comparative analyses with K-Means clustering and PCA highlight the method's superiority in handling high-dimensional, large-scale, and imbalanced datasets. The innovation lies in its model-aware approach, computational efficiency, and adaptability to various real-world scenarios, offering a valuable tool for practitioners seeking to optimize machine learning processes in data-intensive environments.

8 Discussion

8.1 Evaluation of the Entropy-Based Random Forest Compression Method

8.1.1 Summary of Findings

The entropy-based Random Forest compression method demonstrated significant potential in reducing dataset size while maintaining high model performance. By leveraging predictive entropy to select the most informative samples those with both high and low entropy the method effectively retained critical information necessary for accurate classification. In the application classification task, the compressed dataset achieved a compression ratio of approximately 125x while maintaining an accuracy of 97.00%, compared to 99.32% with the full dataset. In the anomaly detection task, a compression ratio of approximately 127x was achieved with an accuracy of 89.24%, compared to 99.36% with the full dataset.

8.1.2 Advantages of the Method

The key advantages of the entropy-based method include:

- **Model-Aware Compression**: By incorporating the model's predictive uncertainties, the method ensures that the most informative samples are retained, directly benefiting model training.
- **Class Balance Preservation**: The use of stratified sampling after entropy-based selection maintains class distribution, which is crucial in handling imbalanced datasets common in HPC environments.
- **Computational Efficiency**: The method reduces computational costs associated with training machine learning models on large datasets, making it suitable for time-sensitive applications.
- **Scalability**: The approach is scalable to large datasets due to the inherent parallelization capabilities of Random Forests and the efficiency of entropy calculation.

8.1.3 Impact on Model Performance

The method's ability to maintain high accuracy despite significant compression is noteworthy. In the application classification task, the slight decrease in accuracy (from

99.32% to 97.00%) is acceptable given the substantial reduction in data size. The method effectively preserved the model's ability to classify different applications, as evidenced by the high precision, recall, and F1-scores across classes.

8.2 Comparison with Alternative Compression Techniques

8.2.1 K-Means Clustering with Stratified Sampling

Performance Analysis

K-Means clustering combined with stratified sampling also achieved high compression ratios with minimal loss in model performance. In the application classification task, a compression ratio of approximately 250x was achieved with an accuracy of 98.27%. This method maintained class balance through stratified sampling, ensuring that minority classes were adequately represented.

Advantages and Limitations

- **Advantages**:
  - **Data Structure Preservation**: K-Means clustering captures the inherent structure of the data by grouping similar samples.
  - **Simplicity**: The method is straightforward to implement and understand.
- **Limitations**:
  - **Computational Complexity**: K-Means can be computationally intensive, especially with large K and high-dimensional data.
  - **Sensitivity to Initialization**: The algorithm's performance can be affected by the initial placement of cluster centroids.
  - **Lack of Model Awareness**: Unlike the entropy-based method, K-Means does not incorporate model predictions or uncertainties in the compression process.

8.2.2 Principal Component Analysis (PCA)

Performance Analysis

PCA-based compression resulted in significant performance degradation, particularly in the application classification task, where accuracy dropped to 46.21%. Although PCA effectively reduced dimensionality by retaining components that explained 95% of the variance, it failed to preserve critical information necessary for accurate classification.

Advantages and Limitations

- **Advantages**:
  - **Dimensionality Reduction**: PCA reduces the number of features, which can simplify models and reduce overfitting.
  - **Noise Reduction**: By focusing on components with the highest variance, PCA can filter out noise.
- **Limitations**:
  - **Loss of Interpretability**: Principal components are linear combinations of original features and can be difficult to interpret.
  - **Information Loss**: PCA may discard features with lower variance that are nonetheless important for classification.
  - **Assumption of Linearity**: PCA assumes linear relationships among features, which may not hold in complex datasets.

8.2.3 Statistical Validation and Feature Distribution Preservation

The statistical tests conducted, such as the Kolmogorov-Smirnov test, indicated that the entropy-based method better preserved feature distributions compared to PCA. The entropy-based compressed datasets retained a higher percentage of features with distributions similar to the full dataset, which is critical for maintaining model performance. PCA, on the other hand, significantly altered the feature space, leading to poorer model generalization.

8.3 Critical Evaluation of the Methods

8.3.1 Effectiveness in Handling Class Imbalance

Class imbalance is a significant challenge in classification tasks, particularly in anomaly detection where anomalies are rare. The entropy-based method effectively addressed this issue through stratified sampling, ensuring that minority classes were represented in the compressed dataset. This approach improved the model's ability to detect anomalies, which is crucial for HPC operational data analytics.

8.3.2 Computational Efficiency and Scalability

While both the entropy-based and K-Means methods achieved high compression ratios, the entropy-based method demonstrated greater computational efficiency. The calculation of predictive entropy and sample selection can be efficiently parallelized, and Random Forests are inherently suitable for parallel computation. K-Means clustering, especially with a large number of clusters, can be more computationally demanding due to its iterative nature.

8.3.3 Preservation of Data Characteristics

The entropy-based method's focus on retaining samples with high and low predictive entropy ensures that both the complex and well-understood regions of the data space are represented. This strategy preserves the dataset's intrinsic characteristics, which is reflected in the minimal loss of model performance. In contrast, PCA's transformation of the feature space led to a loss of important feature relationships, adversely affecting model accuracy.

8.4 Limitations of the Proposed Method

Despite its advantages, the entropy-based method has limitations:

- **Dependency on Initial Model Quality**: The effectiveness of the method relies on the initial Random Forest model's ability to accurately estimate predictive entropy. Poor model calibration can lead to suboptimal sample selection.
- **Potential Overfitting**: If the core dataset is too small or not representative, there is a risk of overfitting the model to the compressed data.
- **Complexity in Hyperparameter Tuning**: Selecting appropriate thresholds for entropy values and determining the right balance between high and low entropy samples requires careful experimentation.

8.5 Future Work

Future research could address the following areas:

- **Automated Threshold Selection**: Developing methods to automatically determine optimal entropy thresholds could improve the method's applicability.
- **Extension to Other Models**: Exploring the use of entropy-based compression with other machine learning models, such as neural networks, could broaden the method's utility.
- **Incorporation of Temporal Dependencies**: Enhancing the method to better capture temporal patterns inherent in time-series data may improve performance in sequential prediction tasks.
- **Real-Time Implementation**: Implementing the method in real-time HPC monitoring systems to evaluate its practical effectiveness and impact on system performance.

8.6 Conclusion

The entropy-based Random Forest compression method presents a promising approach for dataset compression in high-dimensional, large-scale time-series data. By leveraging model predictive uncertainties, it effectively balances data reduction with information preservation, maintaining high levels of model performance. Comparative analyses demonstrate its superiority over traditional methods like PCA and its competitiveness with K-Means clustering combined with stratified sampling. While there are limitations, the method's advantages make it a valuable contribution to machine learning applications in HPC operational data analytics.

List of Figures

List of Tables

Bibliography

[1] Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. CRC Press, 2013. URL: https://doi.org/10.1201/9781315373515.

[2] Leo Breiman. "Random forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://link.springer.com/article/10.1023/A:1010933404324.

[3] R. Camargo, F. Marozzo, and W. Martins. "Computer architecture and high performance computing". In: *Concurrency and Computation: Practice and Experience* 33 (2021). DOI: 10.1002/cpe.6526. URL: https://onlinelibrary.wiley.com/doi/10.1002/cpe.6526.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM Computing Surveys (CSUR)* 41.3 (2009), pp. 1–58. URL: https://doi.org/10.1145/1541880.1541882.

[5] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357. URL: https://doi.org/10.1613/jair.953.

[6] Jiasi Chen and Xukan Ran. "Deep Learning With Edge Computing: A Review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674. DOI: 10.1109/JPROC.2019.2921977. URL: https://ieeexplore.ieee.org/document/8763885.

[7] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. "Outlier Detection Using Isolation Forest and Local Outlier Factor". In: *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. RACS '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 161–168. DOI: 10.1145/3338840.3355641. URL: https://doi.org/10.1145/3338840.3355641.

[8] Veronica Costa and Hermes Senger. "High-performance computing for computational science: NA". In: *Concurrency and Computation: Practice and Experience* 32 (June 2020). DOI: 10.1002/cpe.5904. URL: https://doi.org/10.1002/cpe.5904.

[9] Haibo He and Edwardo A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), pp. 1263–1284. DOI: 10.1109/TKDE.2008.239. URL: https://ieeexplore.ieee.org/document/5128907.

[10] Shuibing He et al. "Cost-Aware Region-Level Data Placement in Multi-Tiered Parallel I/O Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 28.7 (2017), pp. 1853–1865. DOI: 10.1109/TPDS.2016.2636837.

[11] Claire Songhyun Lee et al. *A Case Study of Data Management Challenges Presented in Large-Scale Machine Learning Workflows*. 2023. DOI: 10.1109/CCGrid57682.2023.00017.

[12] John Lee and Michel Verleysen. "Nonlinear dimensionality reduction of data lying on embedded manifolds". In: *Neural Processing Letters* 26.3 (2007), pp. 163–174. URL: https://doi.org/10.1111/j.1751-5823.2008.00054_10.x.

[13] David D. Lewis and William A. Gale. "A Sequential Algorithm for Training Text Classifiers". In: *Springer London* (1994). Ed. by Bruce W. Croft and C. J. van Rijsbergen, pp. 3–12. URL: https://doi.org/10.1007/978-1-4471-2099-5_1.

[14] Ning Liu et al. "On the role of burst buffers in leadership-class storage systems". In: *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)* (2012), pp. 1–11. DOI: 10.1109/MSST.2012.6232369. URL: https://doi.org/10.1109/MSST.2012.6232369.

[15] Stuart Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. URL: https://ieeexplore.ieee.org/document/1056489.

[16] Mohit Mittal et al. "Machine Learning Techniques for Energy Efficiency and Anomaly Detection in Hybrid Wireless Sensor Networks". In: *Energies* 14.11 (2021). ISSN: 1996-1073. URL: https://www.mdpi.com/1996-1073/14/11/3125.

[17] A. Netti et al. "Correlation-wise Smoothing: Lightweight Knowledge Extraction for HPC Monitoring Data". In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2020), pp. 2–12. DOI: 10.1109/IPDPS49936.2021.00010.

[18] John D. Owens et al. "A Survey of General-Purpose Computation on Graphics Hardware". In: *Computer Graphics Forum* 26.1 (2007), pp. 80–113. URL: https://doi.org/10.1111/j.1467-8659.2007.01012.x.

[19] J Ross Quinlan. "Induction of decision trees". In: *Machine Learning* 1.1 (1986), pp. 81–106. URL: https://doi.org/10.1007/BF00116251.

[20] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. "A graph matching method and a graph matching distance based on subgraph assignments". In: *Pattern Recognition Letters* 31.5 (2010), pp. 394–406. ISSN: 0167-8655. URL: https://www.sciencedirect.com/science/article/pii/S0167865509002864.

[21] Claude E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423, 623–656. URL: https://ia803209.us.archive.org/27/items/bstj27-3-379/bstj27-3-379_text.pdf.

[22] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *International Joint Conference on Neural Networks (IJCNN)* (2018), pp. 1575–1581. URL: https://doi.org/10.48550/arXiv.1611.06455.

[23] Jiujing Zhang et al. "High-Ratio Compression for Machine-Generated Data". In: *Proceedings of the ACM on Management of Data* 1 (2023), pp. 1–27. DOI: 10.1145/3626732.