A Performance Comparison between Isolation Forest and K-Means-Based Isolation Forest on Anomaly Detection Tasks

Răzvan-Nicolae Căpățînă Mihnea-Vicențiu Bucă
University of Bucharest, Romania

Abstract

This paper conducts a detailed comparative presentation analyzing the foundational Isolation Forest (**IF**) method and its novel extension, the k-Means-Based Isolation Forest (**kMIF**), specifically focusing on their performance in anomaly detection.

The **kMIF** approach fundamentally augments the isolation process by incorporating k-Means clustering during the construction of search trees. This modification replaces the binary splits of the classic **IF** with multi-branch node divisions determined by the optimal number of clusters, aiming to better fit the underlying data structure.

To empirically evaluate the efficacy of this innovative structure, we execute a comprehensive series of experiments on both controlled **synthetic datasets** and a real-world **Embedded System Network Security Dataset (ESNSDP)**. Our analysis moves beyond standard isolation metrics, including accuracy, precision, recall, and F1-scores, to conduct a deeper technical and functional comparison.

Crucially, we incorporate an Agreement Analysis to quantify the concordance and disagreement between the two models' anomaly rankings. Furthermore, we provide an extensive assessment of computational overhead, benchmark execution speed, and analyze performance under parallelization constraints, which are vital considerations for deployment in resource-limited embedded systems.

The objective of this research is to clearly delineate the functional trade-offs and structural advantages of each model; isolation quality versus computational efficiency; to definitively determine a suitable anomaly detection method for various data scenarios, particularly those characterized by high noise, extreme class imbalance, and the unique demands of network security environments.

1 Introduction

The task of detecting anomalies, also widely known as outlier detection, is one of the most critical and challenging problems in data science across numerous fields, from finance and manufacturing to security and healthcare. Fundamentally, this is considered an **ill-posed problem** in a mathematical sense. The inherent difficulty stems from the fact that an observation can only be deemed anomalous depending on the **context, the application, or the domain** in which the data resides. Unlike classification, where class boundaries are clearly defined, an anomaly lacks a uniform, objective definition. This absence of a standardized ground truth means there is no single, unique solution to the problem, necessitating a continuous evolution of domain specific and general-purpose techniques.

The search for effective anomaly detection methods has led to the documentation of several distinct approaches throughout the years:

- **Statistical Methods**: These techniques, such as **Z-scores** and Gaussian-based models, rely on fitting a statistical distribution to the data and identifying points that fall into low-probability regions as outliers.
- **Distance- and Density-Based Methods**: Approaches like **k-Nearest Neighbors** (k-NN) and **Local Outlier Factor** (LOF) [1] quantify the degree of isolation by measuring the distance or density of a point relative to its immediate neighbors.

- **Reconstruction-Based Methods (Deep Learning)**: Modern techniques, including **Autoencoders (AEs)** [2] and **Variational Autoencoders (VAEs)** [3], learn a compressed normal representation of the data. Anomalies are then detected by their high reconstruction error, as the model fails to efficiently compress data it has never truly learned.
- **Isolation-Based Methods**: These models, such as Isolation Forest (IF) or Extended Isolation Forest (EIF) [4], are built on the premise that anomalies, being "few and different", can be isolated much faster and closer to the root of a decision tree structure than normal points.

The classical Isolation Forest (IF), introduced in 2008 [5], quickly established itself as a highly successful isolation-based technique. Its success is attributed to its **efficiency** in terms of running time and memory footprint, as well as its ability to provide **easily interpretable results** with robust performance across various datasets.

However, the classical IF suffers from an **inflexible spatial partition**. By design, it recursively divides the hyperspace using a hyperplane that is strictly perpendicular to a randomly selected dimension, resulting in a series of rigid, binary splits. This limitation restricts the model's expressiveness, leading to identifiable structural artifacts in complex datasets.

To address this fundamental shortcoming, the k-Means-Based Isolation Forest (kMIF) was introduced in 2020 [6]. kMIF attempts to enhance the model's structural flexibility by modifying the space partitioning logic. Instead of a binary split, the model uses **k-Means clustering** at each node to divide the space into an **arbitrary number of subspaces** of varying sizes. This modification allows the trees to learn a more data-aware partition, substantially increasing the expressive capacity and providing a more intuitive isolation score based on cluster membership.

2 Algorithmic Implementation

Both models discussed, the Isolation Forest (IF) and the k-Means-Based Isolation Forest (kMIF), are ensemble methods. They are respectively composed of many Isolation Trees and k-Means-Based Isolation Trees, with the core difference lying in how each tree node partitions its data. [1]

2.1 Implementation of the Classic Isolation Tree

The construction of a classical Isolation Tree follows a recursive partitioning strategy designed to isolate samples quickly.

2.1.1 Tree Construction (Binary Splitting)

An Isolation Tree begins as a single root node receiving a subset of the training data. The splitting process at any given node T is governed by two uniform random selections:

1. **Feature Selection:** A dimension (feature) q is chosen randomly (uniformly) from the set of all dimensions.
2. **Split Value Selection:** A split value v is chosen randomly (uniformly) from the range $[\min_q, \max_q]$, where $\min_q$ and $\max_q$ are the minimum and maximum values of feature q within the data subset currently held by node T.

The data are then separated into two disjoint sets: $D_L = \{x | x_q < v\}$ and $D_R = \{x | x_q \geq v\}$. The node then instantiates exactly two children, each receiving one of the resulting data subsets. This process is recursively applied to the child nodes.

2.1.2 Termination Conditions

The recursive partitioning stops, and the current node T becomes a leaf node if any of the following conditions are met:

- **Maximum Depth**: The node has reached a preset maximum depth, l.
- **Sample Size:** The node contains either zero or exactly one sample.
- **Identical Samples:** All samples within the node are identical across all dimensions, making any further split impossible (as it would result in an empty set and an unchanged initial set).

[1] The source code implementation, including all technical and tuning parameters, is available for consultation in the dedicated **Project's GitHub Repository**: `https://github.com/Razvan48/Project-Anomaly-Detection-AD`.

2.1.3 Anomaly Score Calculation

The Isolation Forest (or a single Isolation Tree) determines a sample's anomaly score based on the path length $h(x)$, which is the number of splits required to isolate the sample x.

The anomaly score $s(x, n)$ for a single sample x in an ensemble is calculated as:

$$s(x, n) = 2^{-\frac{\mathbb{E}[h(x)]}{c(n)}}$$

Where:

- n is the number of samples used during the training of each tree.
- $\mathbb{E}[h(x)]$ is the mean path length of sample x, averaged over all trees in the ensemble.
- $c(n)$ is a normalizing factor, representing the **average path length of an unsuccessful search** in a binary search tree built using n samples. It is given by:

$$c(n) = 2[\ln(n-1) + 0.5772156649] - \frac{2(n-1)}{n}$$

- where 0.5772156649 is the Euler–Mascheroni constant, and $\ln(i)$ is used to estimate the harmonic number $H(i)$.
- $h(x)$ is the path length of sample x from the root to the terminating node. If the node terminates due to the maximum depth constraint, the final path length for all samples in that node is adjusted by adding the correction factor $c(m)$, where m is the number of samples remaining in that leaf node ($m > 1$). This compensation accounts for how much further the node *would* have expanded if the depth limit had not been enforced.

Below is an illustration (1) depicting how the **Two Moons Dataset 2D** is partitioned by a single classic Isolation Tree.

Visually, we can observe the characteristic rigidity of the IF approach: the space is divided into a mosaic of rectangular regions, where every split is a line (a hyperplane in higher dimensions) parallel to either the X or Y axis. Crucially, the partitioning is performed **uniformly and randomly**, regardless of the local data density in each area. This random and rigid split is a direct consequence of having exactly two children for each node and choosing the split value from a uniform distribution over the current data range.

As a result, large, low-density regions (like the one spanning $X \in [-4, 4]$ and $Y \in [-4, -4]$) are given the same split consideration as the high-density regions containing the crescent shapes. While this approach quickly isolates true outliers (points far from the crescents), it is **inefficient** in separating the two dense moons from each other, demonstrating the model's inflexibility when dealing with complex or non-convex clusters.

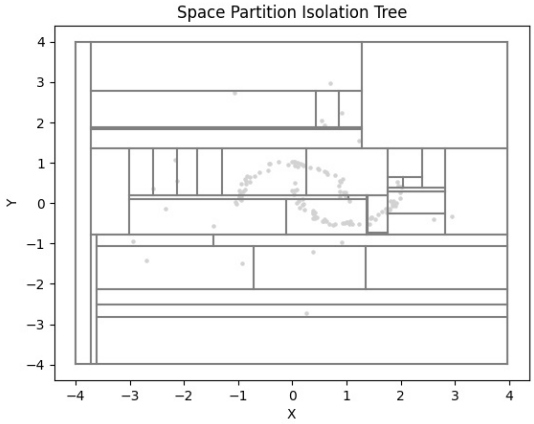

Figure 1: Space Partition Classic on Two Moons 2D Dataset

2.2 Isolation Forest Ensemble

The Isolation Forest ensemble is created by building t Isolation Trees, each typically using a randomly sub-sampled set of the training data.

The original paper recommends:

- **Sub-sampling size n:** $n = 256$ is highly effective.
- **Maximum Tree Depth l:** $l = \lceil \log_2 n \rceil$. For $n = 256$, this is $l = 8$.
- **Ensemble Size t:** $t = 100$.

During the scoring phase, a final threshold is required to label samples as Anomalous/Normal. While a contamination rate from the training set can provide this threshold, the models will default to a score of **0.5** if the rate is unknown.

2.3 Implementation of the k-Means-Based Isolation Tree

The k-Means-Based Isolation Forest (kMIF) replaces the classic IF's rigid binary split with a data-driven, multi-branch split. This architectural change allows the tree structure to better adapt to the intrinsic local geometry of the data.

2.3.1 Tree Construction via Multi-Branch Clustering

For a single k-Means-Based Isolation Tree, the core logic of recursive space partitioning is preserved, but the splitting criterion is modified:

1. **Feature Selection:** A dimension (feature) q is chosen randomly and uniformly.
2. **k-Means Clustering:** Instead of selecting a random split value, k-Means clustering is applied to all samples' values along the chosen dimension q.
3. **Optimal k Determination:** The optimal number of clusters k is determined dynamically at each node using the **Elbow Rule**[2] on the projected data's WCSS. In our implementation, we test a restricted range of cluster counts, specifically $k \in \{2, 3, 5, 7\}$, to maintain performance. The optimal k is selected as the point maximizing the distance to the baseline connecting the minimum and maximum tested k values. Furthermore, to stabilize the calculation of the cluster center radius, the final cluster centers are **recalculated** as the mean of their assigned data points.
4. **Partitioning:** The dataset is split into k subsets based on each sample's membership in one of the generated clusters.

The kMIF approach calculates an anomaly score $\alpha(x)$ by quantifying how "normal" a sample x is at every split it encounters. A "normality score" $s_j(x)$ is computed at each node j.

The single-split normality score $s_j(x)$ is calculated using the formula:

$$s = 1 - \frac{d(x, c_q)}{d(c_l, c_q)}$$

[2]Elbow method

Where:

- $d(x, c_q)$ is the distance from the sample x to the center of the cluster c_q to which it belongs along the feature q.
- $d(c_l, c_q)$ is the distance from the cluster limit c_l to the cluster center c_q (representing the cluster's radius). These radii are saved during the training phase.

The final ensemble anomaly score $\alpha(x)$ is computed by averaging the cumulative normality scores across all t trees:

$$\alpha(x) = 1 - \frac{1}{t} \sum_{i=1}^{t} \sum_{j=1}^{M_i} s_j(x)$$

Where M_i is the actual depth of the path traced by x in tree i.

Score Range Considerations

This method introduces two important considerations regarding the score range:

1. **Normality Score Range**: The distance ratio $\frac{d(x,c_q)}{d(c_l,c_q)}$ can exceed 1.0 for out-of-sample data, and the score s can theoretically become negative. To maintain the probabilistic interpretation of a "normality" score, our implementation **clamps** the result to the $[\mathbf{0}, \mathbf{1}]$ range using: $s_j(x) = \max(0, 1 - d(x, c_q)/d(c_l, c_q))$.
2. **Anomaly Score Range:** The final anomaly score $\alpha(x)$ is **not normalized to a fixed range**, unlike classic IF. Since each $s_j(x) \in [0, 1]$, the inner sum $\sum_{j=1}^{M_i} s_j(x)$ has a range of $[0, M_i]$. Therefore, $\alpha(x)$ is dependent on the tree depth M_i, ranging from $1 - \mathbb{E}[M]$ to 1.

2.4 k-Means-Based Isolation Forest Ensemble

The kMIF ensemble is constructed by building t such multi-branch trees.

- The original paper recommends a maximum depth $l = 9$, a sub-sampling size n of 128 or 256, and an ensemble size $t = 100$ [6].

- As with the classic IF, the training contamination rate is needed to set a meaningful anomaly threshold for labeling.

Below is an illustration (2) depicting how the Two Moons Dataset is partitioned by a single k-Means-Based Isolation Tree. We can clearly observe the **data-aware partitioning**: dense areas (like the clusters themselves) result in nodes with a greater number of children (higher k), while sparse areas are less frequently split or are split into fewer children, dynamically optimizing the partitioning effort based on the local data density.

In stark contrast to the classic Isolation Tree (IF), the k-Means-Based Isolation Tree (kMIF) demonstrates a **data-aware and non-uniform partition**. The multi-branch splits, determined by the optimal cluster count k from the Elbow Rule, result in regions that are highly reflective of the underlying data density:

- **Dense Regions (The Moons):** Where the data is highly concentrated, the tree performs numerous fine-grained splits, often resulting in narrow, stacked boxes. This signifies that a high k value was chosen for that node's dimension, allowing the tree to quickly learn the boundaries of the dense clusters.
- **Sparse Regions (Outer Space):** In areas with few samples, the tree either stops partitioning quickly or splits the region into very few large children (a low k value).

This mechanism allows the kMIF to devote its partitioning resources only where the data is dense and complex, directly addressing the classic IF's inability to adapt its splitting strategy to the local data structure. This structural flexibility is hypothesized to lead to a more accurate distinction between clustered normal data and isolated anomalies.

In summary, both Isolation based approaches leverage the same core insight that anomalies isolate faster; however, the kMIF achieves isolation through data driven clustering along a selected feature, resulting in a more expressive and adaptable tree structure.

Elbow Rule for Clustering

The primary challenge in implementing the k-Means-Based Isolation Tree is the need to dynamically select the optimal number of children (k) for each node. This is solved by applying the **Elbow Rule (or Elbow Method)**.

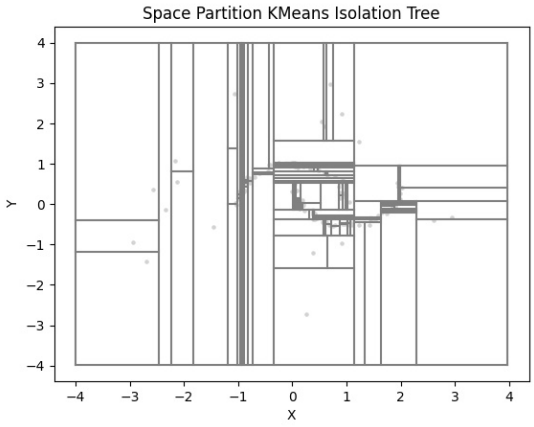

Figure 2: Space Partition of a k-Means-Based Isolation Tree on the Two Moons 2D Dataset. Note the non-uniform, data-driven partitioning.

The technique involves running the k-Means algorithm multiple times with an increasing number of clusters (k) and tracking the **Inertia**, defined as the **Within-Cluster Sum of Squares (WCSS)**.

- When the number of clusters k is very small, the Inertia value is high.
- When k is equal to the number of samples, the Inertia converges to its minimum of zero (as every point is its own cluster center).

The goal of the Elbow Rule is to identify the point on the resulting plot where the Inertia begins to converge **very slowly**, the "bend" or "elbow." This point signifies the optimal k, as any further increase in clusters provides a negligible marginal reduction in WCSS, indicating that the core structure has been found.

An illustration of the Inertia curve is provided in Figure 3. In the example shown, the optimal point is identified at $k = 11$ (marked by the red star). This point is often found algorithmically by identifying the k that yields the greatest distance to the line connecting the first and last Inertia points (the gray *Baseline* line in the figure).

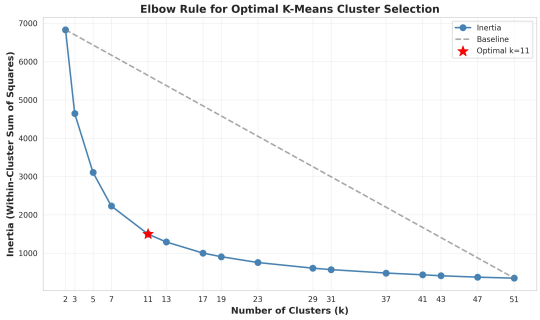

Figure 3: The optimal $k = 11$ is selected at the point where the rate of decrease in Inertia sharply diminishes.

3 Algorithmic Details

The core logic for both algorithms is implemented in Python, leveraging NumPy for vectorization and `joblib` for parallel execution. Below, we outline the recursive tree building process for both models.

Tree Constructions

Algorithm 1: Isolation Tree

Input: D, l_{max}, depth
Output: Node
if *depth* $\geq l_{max}$ ***or*** $|D| \leq 1$ ***or*** *AllSamplesIdentical(D)* **then**
  return LeafNode(D)
$q \leftarrow$ ChooseRandomFeature(D);
$v \leftarrow$ ChooseRandomSplitValue(D_q);
$D_L \leftarrow \{x \in D \mid x_q < v\}$;
$D_R \leftarrow \{x \in D \mid x_q \geq v\}$;
Node.Left $\leftarrow$ BuildTree(D_L, l_{max}, *depth* $+ 1$);
Node.Right $\leftarrow$ BuildTree(D_R, l_{max}, *depth* $+ 1$);
return Node

Features of Isolation Forest (IF)

Node Split Cost	$O(1)$
Overall Training Bottleneck	$O(t \cdot n \cdot \log n)$ The complexity of traversing n nodes in the tree, leading to the highly efficient $O(n \log n)$ complexity per tree.

Algorithm 2: k-Means Isolation Tree

Input: D, l_{max}, *depth*
Output: Node
if *depth* $\geq l_{max}$ ***or*** $|D| \leq 1$ ***or*** *Identical(D)* **then**
  return LeafNode(D)
$q \leftarrow$ ChooseRandomFeature(D);
$k_{opt} \leftarrow$ FindOptimalK(D_q along q);
$C, M, R \leftarrow$ KMSplit(D_q, k_{opt});
foreach $C_i \in C$ **do**
  Node.Child$_i \leftarrow$ BuildTree(C_i, l_{max}, *depth* $+ 1$);
Node.Centers $\leftarrow M$;
Node.Radii $\leftarrow R$;
return Node

Features of k-Means Isolation Forest (kMIF)

Node Split Cost	$O(P \cdot k \cdot n' \cdot d')$
Overall Training Bottleneck	**Significantly Higher** The repeated execution of the Elbow Rule (running k-Means P times, where $P = 4$ in our implementation) at every node introduces considerable **overhead**.

While the overall complexity of the classic IF remains exceptionally fast at $O(t \cdot n \cdot \log n)$ for t trees and a sub-sample size of n, the k-Means operation within the kMIF fundamentally increases the total number of computations. This makes the **training phase** the primary theoretical computational bottleneck for kMIF compared to its classic counterpart.

3.1 Parallelization as a Solution

To mitigate the wall-clock time required for training and scoring, especially for the computationally heavier kMIF, our implementation explicitly leverages the intrinsic parallelism of ensemble methods: **each tree in the forest can be built and scored independently.**

Our implementation uses the `joblib` library to efficiently exploit multi-core architectures, enabling parallelism in the following ways:

1. **Ensemble Training:** The `fit` method for both `IsolationForest` and `KMeansIsolation Forest` distributes the construction of all t trees to separate CPU cores. This ensures that the wall-clock training time is effectively reduced by a factor proportional to the number of jobs (n_{jobs}), making the kMIF's training overhead competitive. **Reproducibility** is guaranteed

by generating unique seeds sequentially in the main process and passing them to the parallel workers (`Irandom_state` logic in `Iforest.py`).

2. **Ensemble Scoring:** The `scores` method similarly parallelizes the process of calculating the final anomaly score. By distributing the individual path length (IF) or normality score (kMIF) calculations across available cores, the scoring time is significantly reduced, which is vital when processing large test datasets or when real-time performance is required.

Forest Ensemble Constructions

Algorithm 3: Parallel Ensemble Training

Input: Training data X, Ensemble size t, Subsample size s, Max depth l, Cores p
Output: Forest F, threshold τ

FitForest X, t, s, l, p
Seeds $\leftarrow$ GenerateSeeds(t)
TreeType $\leftarrow$ Choose(IF / kMIF)
`// parallel on p cores`
for $i = 1$ **to** t **do**
  $F[i] \leftarrow$ BuildTreeWorker(X, s, l, Seed_i)
$F \leftarrow \{\text{F}_1, \ldots, \text{F}_t\}$
Scores $\leftarrow$ ScoreForest(F, X, p)
$\tau \leftarrow$ Quantile(Scores, $1 - \text{contam.}$)
return F, τ

Forest Ensemble Scoring

Algorithm 4: Parallel Forest Scoring

Input: Trained forest F
Test data X_{test}
Cores p
Output: Anomaly scores A

ScoreForest F, X_{test}, p
`// parallel on p cores`
for $i = 1$ **to** $|F|$ **do**
  $S[i] \leftarrow$ ScoreTreeWorker$(F[i], X_{\text{test}})$
if *TreeType* $=$ *IF* **then**
  $A \leftarrow 2^{-\frac{\text{Mean}(S)}{\text{ExpectedPathLength}}}$
else if *TreeType* $=$ *kMIF* **then**
  $A \leftarrow 1 - \text{Mean}(S)$
return A

3.2 Efficiency Analysis

To empirically validate the necessity and effectiveness of the parallelization strategy, we bench-marked the training time of both the Isolation Forest (IF) and k-Means-Based Isolation Forest (kMIF) implementations. The test was performed on a synthetic dataset.

The results, presented in Figure 4, clearly illustrate the significant computational trade-off between the two models and the power of parallel processing.

3.2.1 Analysis of Training Time (Figure 4, Left)

The bar chart on the left highlights the sequential wall-clock time for model training:

1. **kMIF Sequential Overhead:** The sequential training time for **kMIF** (green bars) is substantial, consistently exceeding **43 seconds** across all tested dataset sizes (500-5,000 samples). This directly confirms the theoretical complexity analysis, where the repeated sequential execution of the Elbow Rule at every node constitutes the dominant bottleneck.

2. **kMIF Parallel Efficiency:** By leveraging parallel processing (red bars), the training time for kMIF is dramatically reduced to approximately **4.1-4.6 seconds**. This improvement renders kMIF viable for practical deployment.

3. **IF Performance:** The training time for the **Isolation Forest** (blue/orange bars) remains minimal in both sequential and parallel settings (sub-second). The IF's $O(1)$ node-split cost results in inherently low training time, thereby limiting the achievable gains from parallelization.

3.2.2 Analysis of Parallelization Speedup (Figure 4, Right)

The line plot on the right, which shows the speedup (Sequential Time / Parallel Time), quantifies this efficiency.

1. **kMIF Speedup:** kMIF achieves an impressive speedup of approximately **9.3–10.5**×. This

near-linear scaling indicates that the computational workload is effectively distributed across the available CPU cores, validating our $n_{\text{jobs}} = -1$ parallel implementation strategy. The consistent speedup across dataset sizes (up to 5,000 samples) further demonstrates the robustness of the approach.

2. **IF Speedup:** The speedup for the Isolation Forest (blue line) remains modest, ranging between **1.0–1.3**× (with a maximum of 1.28×). This behavior is expected; because the sequential runtime is already minimal (below 0.4 s); the overhead of inter-process communication can outweigh the benefits of parallel execution. Consequently, the speedup occasionally drops below unity (e.g., 0.69× at 500 samples), although it generally stays close to the "No speedup" baseline.

This experiment confirms that the choice to implement parallel training was essential for the kMIF, transforming a method with high theoretical overhead into a practically efficient alternative to the classic IF.

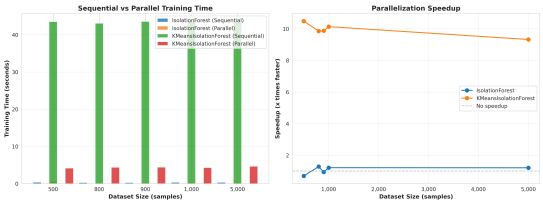

Figure 4: Comparison of sequential vs. parallel training time and speedup for IF and kMIF on synthetic high-dimensional datasets.

3.3 PyOD, sklearn and H2O

To contextualize the efficiency of our custom implemented Isolation Forest (IF) and $k-$Means Isolation Forest (kMIF), we bench-marked their training times against three industry standard libraries: the `PyOD` anomaly detection library, the `sklearn` implementation of Isolation Forest, and the `H2O` implementation of Extended Isolation Forest. The results confirm the trade-off between the computational overhead of kMIF and the highly optimized speed of existing methods.

3.3.1 Other Algorithms Benchmarks

Figure 5 presents a benchmark comparing our implementations with a selection of algorithms from the `PyOD` library and `H2O` on a dataset of 50,000 samples.

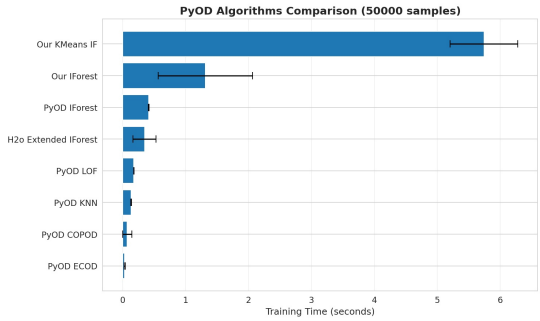

Figure 5: Training time comparison of our custom algorithms against H2O EIF and various PyOD models on 50,000 samples.

The comparison highlights two key findings:

1. **kMIF Performance Ceiling:** Consistent with our prior analysis, `Our KMeans IF` registers the longest training time at **5.74** seconds. This result, even with parallelization, firmly establishes the computational cost of the k-Means-based node splitting.

2. **PyOD's Efficiency Frontier:** Algorithms like **PyOD ECOD** and **PyOD COPOD** demonstrate significantly faster training times than any IForest variant, indicating that for maximum speed with this dataset size, non-tree-based methods should be considered.

3. **IF Comparison:** Our Isolation Forest implementation (**1.31** s) is approximately three times slower than the optimized `PyOD IForest` (**0.41** s). This gap suggests that while our custom implementation is functional, it carries a larger constant-factor overhead compared to the production-ready code within PyOD.

4. **EIF Comparison:** H2O Extended Isolation Forest (**0.34** s) proved to be faster than PyOD IForest (**0.41** s) and both our IF and kMIF implementations.

3.3.2 Scaling Comparison with Scikit-learn

To assess scaling performance, Figure 6 provides a direct comparison of our two models against the highly optimized `sklearn IsolationForest` implementation across various dataset sizes, from 1,000 to 20,000 samples.

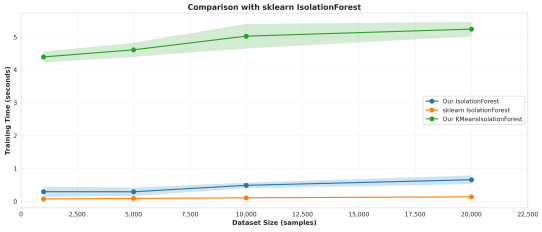

Figure 6: Training time comparison with `sklearn IsolationForest` across varying dataset sizes.

The results reinforce the performance hierarchy:

1. **kMIF Overhead Across Scale:** The training time for `Our KMeansIsolationForest` is stable. While high, the slow rate of increase validates that the parallelization strategy effectively manages the overhead such that the wall-clock time scales gently with n.

2. **sklearn as the Speed Benchmark:** The `sklearn IsolationForest` remains the fastest, completing training in **0.07** s at 1,000 samples and only increasing to **0.14** s at 20,000 samples.

3. **Custom IF Scaling:** `Our IsolationForest` training time scales from 0.60 s to 0.53 s. While this is significantly faster than kMIF, it is still substantially slower than the 'sklearn' baseline, demonstrating the performance cost inherent in a pure-Python custom implementation.

3.3.3 Isolation Forest: Four-way Timing Comparison

Finally, Figure 7 isolates the performance of the Isolation Forest across four distinct implementations: our custom version (`Our IForest`), `PyOD IForest`, `scikit-learn IForest`, and `h2o` Extended IForest, scaling the dataset up to 50,000 samples.

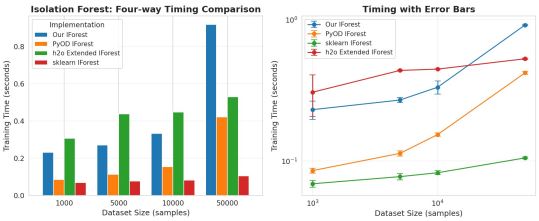

Figure 7: Direct comparison of four distinct Isolation Forest implementations across dataset scales.

The bar chart and log-scale line plot clearly illustrate the scaling differences.

1. **High-Performance Baseline:** The `sklearn IForest` implementation maintains the lowest training time and the gentlest scaling curve across all dataset sizes. It completed the 50,000 sample test in just **0.11** seconds, demonstrating a substantial performance advantage.

2. **Mid-Range Scaling:** The `PyOD IForest` scales efficiently, consistently remaining the second fastest implementation. At 50,000 samples, it records a training time of **0.42** seconds.

3. **Slower Implementations:** Both `h2o Extended IForest` and `Our IForest` demonstrate higher training times, with `h2o` being the slowest at 1,000 samples (**0.31** s). However, the `h2o Extended IForest` exhibits better scaling than `Our IForest`. At the largest scale (50,000 samples), `h2o Extended IForest` records **0.53** seconds, while `Our IForest` is the slowest at **0.92** seconds.

The efficiency trade-off of the novel kMIF method is substantial, but its training time has been reduced to a practical range due to our effective parallelization strategy.

3.3.4 Prediction Time Analysis

The prediction (scoring) phase follows a similar performance ordering to training; however, it exhibits different scaling behavior. The highly optimized `sklearn IsolationForest` remains the fastest overall, exhibiting near-negligible scoring times across all dataset sizes. In contrast, our `KMeansIsolationForest` incurs the largest total

batch latency due to the additional cost of traversing k-Means-based trees.

Importantly, the normalized prediction time (per sample) decreases substantially as batch size increases, demonstrating the effectiveness of our parallel scoring strategy. This indicates that the computational overhead introduced by the more complex tree structure is efficiently amortized over large test batches. As a result, kMIF becomes increasingly competitive in high-throughput settings, which is a key consideration for practical deployment.

4 Experiments and Results

All experiments were run on a dedicated machine with a pre established seed to ensure consistent timing measurements. The system specifications include an **AMD Ryzen 9 7845HX CPU** (12 cores, 24 threads) and **64 GB DDR5 RAM**, running on **Ubuntu 24.04 LTS**.

4.1 Synthetic Datasets Tests

Several experiments were conducted to evaluate the comparative performance of the Isolation Forest (IF) and its extension, the k-Means-Based Isolation Forest (kMIF), across a spectrum of synthesized datasets. These datasets were strategically chosen to represent varying geometric challenges in anomaly detection: the **9-Rectangle Dataset** (rectilinear, axis-aligned clusters), the **Circles Dataset** (concentric, non-convex clusters), the **Moons Dataset** (interlocking, non-convex manifolds), and the **S-Curve Dataset** (complex, high-dimensional manifold). For all tests, the models were trained as ensembles of $t = 100$ trees, with a sub-sampling size of $n = 256$ and a maximum tree depth of $l = 9$. A sweep of contamination rates was performed to find the optimal threshold for each model, with the results from the determined optimal threshold being used for the metric comparison. The empirical results demonstrate a profound functional trade-off between the models: IF consistently exhibits a superior ability to discriminate between normal and anomalous points across complex data structures, while kMIF introduces a computational overhead that does not translate into a commensurate gain in isolation quality on these canonical benchmarks.

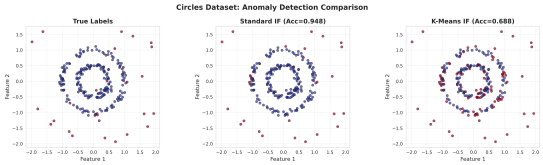

Figure 8: Visual Comparison of Anomaly Detection on the Circles Dataset. The classic IF (center) successfully isolates the non-convex rings, while kMIF (right) misclassifies dense core points as anomalies.

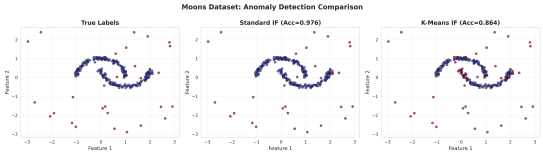

Figure 9: Visual Comparison of Anomaly Detection on the Moons Dataset. The classic IF (center) maintains high accuracy, whereas kMIF (right) exhibits significant misclassification within the two moon-shaped clusters.

4.1.1 Performance Evaluation on Synthetic Data

The comparative evaluation, summarized by key metrics including Balanced Accuracy, F1-Score, and area under the curve for the Receiver Operating Characteristic (ROC-AUC) and Precision-Recall (PR-AUC), reveals an unexpected performance hierarchy (Figure 10 and 11).

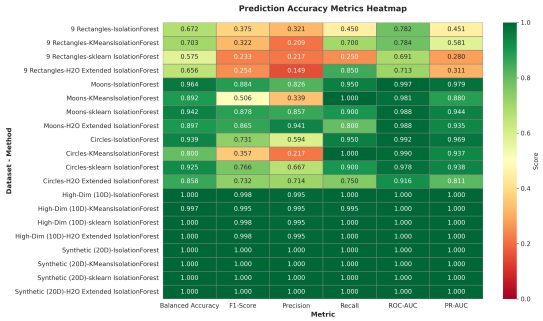

Figure 10: Prediction Accuracy Metrics Heatmap for IF and kMIF across various synthetic datasets. Darker green indicates higher performance.

The principal findings from the synthetic data experiments are as follows:

- **Non-Convex Challenge (Moons and Circles):** Contrary to the theoretical advantage

expected from kMIF's data-aware partitioning, the classic **Isolation Forest significantly outperforms kMIF** on the Moons and Circles datasets. For instance, on the Moons dataset, IF achieves an F1-Score of **0.883**, whereas kMIF achieves only **0.506**. The visual comparison (Figure 8 and 9) clarifies this failure: kMIF misclassifies a substantial portion of the high-density normal points within the cluster core as anomalies, leading to a high false positive rate and, consequently, low Precision and F1-Scores (Figure 11). This suggests that kMIF's k-Means splits, even when locally optimal along a single random feature, fail to capture the complex, non-linear manifold boundaries as effectively as the purely random and flexible cuts of the classic IF.

- **Rectilinear and High-Dimensional Data:** On the 9-Rectangles dataset, which consists of naturally rectilinear clusters, the performance gap narrows, with kMIF slightly leading in Balanced Accuracy (**0.702** vs. **0.672**) but trailing in F1-Score (**0.321** vs. **0.375**). However, on higher-dimensional data (10D and 20D synthetic sets), both models perform nearly perfectly, demonstrating the robustness of isolation-based methods in high-dimensional space, where sparsity aids anomaly detection. (see Appendix 22)

- **Area Under the Curve (AUC) Metrics:** Both models exhibit high ROC-AUC (typically ≥ 0.98 for Moons/Circles), indicating that they are excellent at ranking anomalies correctly. However, a notable trade-off appears in PR-AUC (Precision-Recall AUC). For the challenging non-convex datasets, the PR-AUC of kMIF drops severely (e.g., **0.880** for Moons vs. IF's **0.979**), confirming that kMIF's issue is specifically with the Precision of its labeling, due to the misidentification of normal core points as outliers.

The **Contamination Sweep** analysis (Figure 12) further reinforces that IF is generally more robust across varying levels of contamination, maintaining a higher F1-Score on the Moons and Circles datasets than kMIF, especially as the contamination rate increases. This stability is a significant practical advantage.

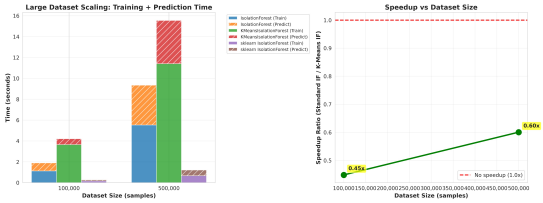

Figure 11: Bar Chart Comparison of key metrics across the most challenging synthetic datasets. IF shows a marked advantage in F1-Score and Precision for non-rectilinear data.

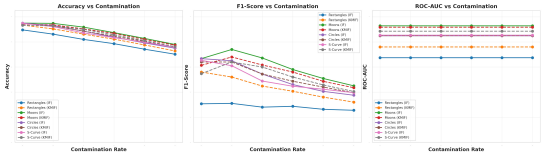

Figure 12: Performance comparison of IF and kMIF across varying contamination rates on synthetic datasets. IF (solid lines) generally shows greater stability and higher performance on complex data.

4.1.2 Disagreement and Ranking Analysis

The underlying functional differences between the models were explored through an **Agreement Analysis** on the 9-Rectangle Dataset (Figure 13). This analysis quantifies how consistently the two models rank the same samples as being more or less anomalous.

- **Score Correlation:** The Spearman rank correlation coefficient between the IF and kMIF anomaly scores was found to be only moderate (**0.531**), and the Kendall's Tau correlation was even lower (**0.376**). This moderate agreement confirms that the fundamental mechanism of isolation (random binary split vs. data-driven multi-branch k-Means split) leads to significantly different anomaly score distributions and rankings (Figure 13, top right and bottom right).

- **Spatial Distribution of Disagreement:** The spatial distribution of the absolute rank differences (Figure 13, bottom left) shows that the largest disagreements occur primarily in the regions *between* the dense clusters (the 9 rectangles) and in the vicinity of the cluster boundaries. This phenomenon is expected: in these transitional zones, the rigidly defined hyper-rectangles of IF are conceptually different

from the smoother, cluster-aware boundaries of kMIF, leading to distinct scoring decisions.

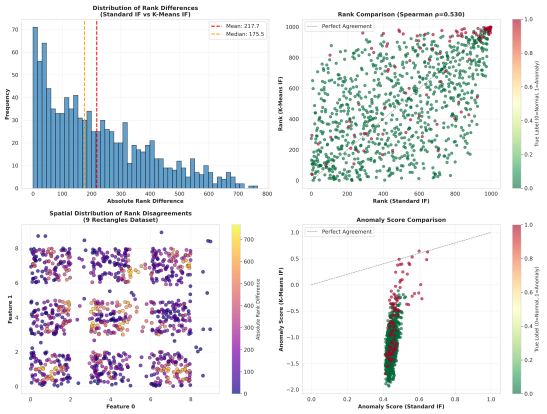

Figure 13: Algorithm Agreement Analysis on the 9-Rectangle Dataset. The moderate correlation and concentrated spatial rank differences highlight the core structural dissimilarity between IF and kMIF.

The rank analysis definitively shows that the two algorithms do not merely implement the same concept differently; they fundamentally model the data space in divergent ways, leading to distinct predictions, particularly for samples near the periphery of the normal class.

4.1.3 Overall Comparative Ranking and Trade-offs

Aggregating the performance across all tested synthetic datasets (Figure 14), a clear performance profile emerges for each model (see Appendix 23):

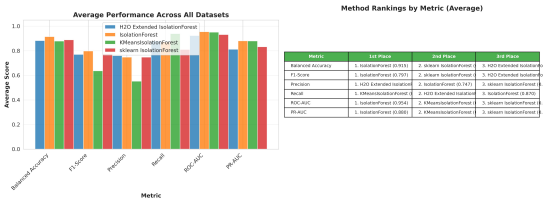

Figure 14: Average Performance and Ranking Summary across all synthetic datasets. IF shows superior average performance in most discrimination metrics.

- **Classic Isolation Forest (IF)** achieves the highest average Balanced Accuracy, F1-Score, and PR-AUC. It is a robust discriminator, excelling in correctly classifying both anomalies and normal data points. It is structurally the fastest model in training and prediction, as established in Section 2, due to its $O(1)$ node-split operation.
- **k-Means-Based Isolation Forest (kMIF)** achieves the highest average Recall, indicating its propensity to aggressively flag samples as anomalous. However, this is at the expense of Precision (low F1-Score) due to the misclassification of normal points in dense clusters. Structurally, kMIF is the most computationally expensive due to the repeated k-Means clustering at every node.

The empirical evidence suggests that the inherent simplicity and randomness of the classic Isolation Forest's splitting mechanism provide a more effective and robust anomaly score aggregation for non-rectilinear data than the supposedly more adaptive, but computationally costly, k-Means-based approach. The $O(1)$ split cost of the classic IF translates directly into a practical and performance advantage that the kMIF's complex split process fails to overcome.

4.2 Embedded System Network Security Dataset (ESNSDP) Results

To evaluate the models' practical utility, a comprehensive comparison was conducted on the real-world **Embedded System Network Security Dataset (ESNSDP)**. This dataset is characterized by an extreme class imbalance, with Normal traffic constituting **90.0**% (900 samples) and Anomaly traffic only **10.0**% (100 samples), as illustrated in the **Label Distribution** plots. This imbalance is a critical challenge, as it necessitates models that prioritize precision and recall for the minority class rather than simple accuracy.

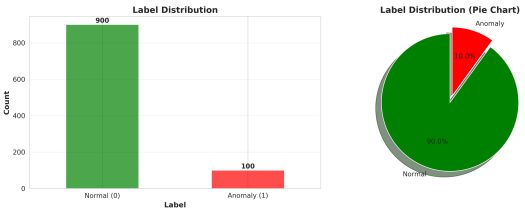

Figure 15: Label Distribution in the ESNSDP dataset showing extreme class imbalance: 90% Normal (900 samples) vs 10% Anomaly (100 samples).

4.2.1 Overall Discrimination Performance

The experimental metrics confirm that distinguishing Anomaly from Normal traffic in this dataset is inherently difficult for both isolation-based methods (see table and Figure 19).

- **Low Overall Scores:** Both models register low scores across the board. The Balanced Accuracy is **0.494** for IF and **0.467** for kMIF, and the ROC-AUC is only slightly better than random chance (**0.524** for IF and **0.440** for kMIF). This poor performance is likely due to the high degree of feature overlap between the two classes. As shown in the **packet_size Distribution** and other feature distribution plots, the histograms for Normal (green) and Anomaly (red) data are highly non-separable, meaning the anomalous behavior is not easily isolated along single feature dimensions.
- **Anomaly Score Overlap:** The **Anomaly Score Distribution** plots further validate this difficulty. For the Isolation Forest, the mean score for Normal samples (0.389) and Anomaly samples (0.390) is virtually indistinguishable, making effective separation impossible. The kMIF scores show a slightly larger separation (-3.204 vs -3.257), but the distributions remain highly mixed.

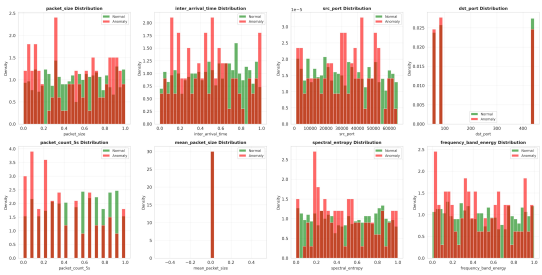

Figure 16: Feature distributions for Normal (green) and Anomaly (red) traffic in the ESNSDP dataset. Note the high overlap between classes, making isolation-based detection challenging.

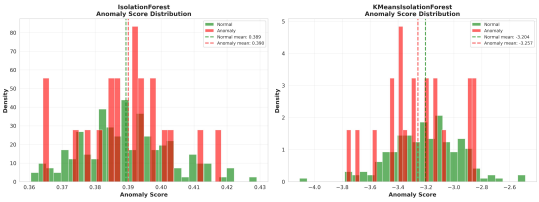

Figure 17: Anomaly Score Distributions for IF and kMIF on ESNSDP. The overlapping distributions (Normal mean: 0.389 vs Anomaly mean: 0.390 for IF) demonstrate the difficulty of threshold-based classification.

4.2.2 Precision vs. Recall Trade-off

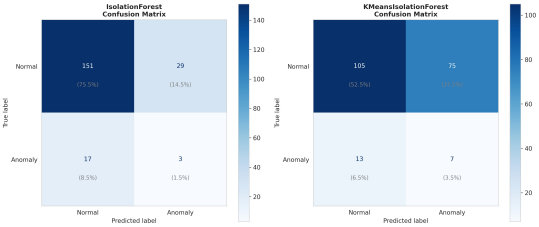

Figure 18: Confusion Matrix Comparison on the ESNSDP. The Isolation Forest (IF) prioritizes True Negatives (151), reflecting a Precision-focused strategy, while the k-Means Isolation Forest (kMIF) is Recall-aggressive, achieving a higher True Positive count (7) but incurring a large False Positive penalty (75).

A critical difference emerges in the models' approach to classification, as seen in the **Confusion Matrix** plots and the metrics:

- **Classic IF: Precision-Focused:** The Isolation Forest is the more conservative classifier, exhibiting higher Precision (**0.094** vs. **0.085**) and better True Negative performance (151 correct Normal predictions). It correctly identifies only **3** out of 200 samples as anomalies (Recall: **15.0**%). This low recall suggests IF's rigid splits fail to isolate anomalies in the complex, high-dimensional space of network features.
- **kMIF: Recall-Aggressive:** The k-Means Isolation Forest demonstrates a much more aggressive classification strategy, achieving significantly higher Recall (**0.350** vs. **0.150**), meaning it correctly identified **7** anomalies. However, this comes at a heavy cost in terms of False Positives. The kMIF misclassifies **75** Normal

samples as Anomalous, compared to only **29** for IF. This high false positive rate renders kMIF impractical for security applications where minimizing false alarms is paramount.

- **Ranking Quality:** Despite the aggressive recall, the kMIF's overall ranking quality is poorer. The ROC-AUC and PR-AUC for kMIF (**0.440** and **0.108**) are lower than those for IF (**0.524** and **0.116**), as visually confirmed by the **ROC Curves Comparison** and **Precision-Recall Curves Comparison** plots. This confirms that the classic IF's score distribution is a more reliable measure for ranking anomalies, even if its ultimate classification decision is more conservative.

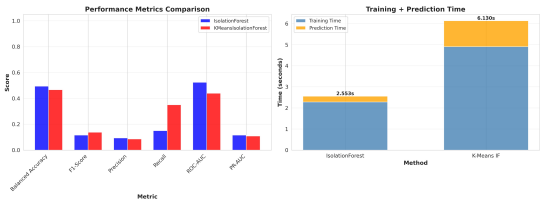

Figure 19: Performance Metrics Comparison for IF and kMIF on the ESNSDP dataset. Note the low overall scores for both models.

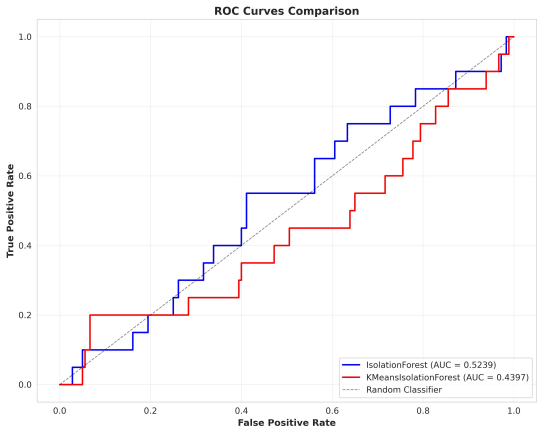

Figure 20: Receiver Operating Characteristic (ROC) Curves Comparison on the ESNSDP. The classic IF (AUC: 0.524) demonstrates a ranking ability slightly above random chance, while kMIF's performance (AUC: 0.440) is worse than random, confirming the high feature overlap between classes.

4.2.3 Trade-off on Real-World Data

The computational cost difference is stark in the context of this real-world dataset:

- **Training Time:** The kMIF required **4.91s** to train, which is more than twice the time required by the Isolation Forest (**2.28s**).
- **Prediction Time:** The scoring overhead is even more pronounced, with kMIF requiring **1.22s**, which is approximately **4.5**× slower than the IF's **0.27s** prediction time.

The empirical results on the ESNSDP demonstrate that the increased complexity of the kMIF, intended to improve structural fit, does not translate into a performance advantage for non-separable, real-world data and instead introduces a substantial computational and false-positive penalty.

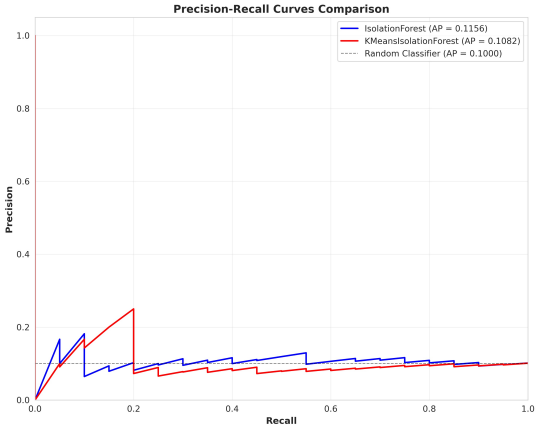

Figure 21: Precision-Recall (PR) Curves Comparison on the highly imbalanced ESNSDP. IF (PR-AUC: 0.116) shows a marginally superior average precision in identifying the minority Anomaly class compared to kMIF (PR-AUC: 0.108).

5 Conclusion

This paper presents a rigorous comparative analysis of the classic Isolation Forest (IF) and the k-Means-Based Isolation Forest (kMIF), focusing on the critical trade-offs between isolation quality and computational efficiency across various data structures, including synthetic benchmarks and a real-world Embedded System Network Security Dataset (ESNSDP). The core architectural difference lies

in the space partitioning logic: IF uses random binary splits, while kMIF employs data-driven, multi-branch divisions based on k-Means clustering.

The study yielded several definitive conclusions:

1. **Computational Overhead is Significant but Mitigated:** The theoretical complexity analysis and empirical benchmarking confirmed that kMIF's node split operation, constrained by the Elbow Rule-driven k-Means clustering, introduces a significant computational overhead compared to the $O(1)$ split of IF. Our parallelization strategy was essential, transforming kMIF from a sequentially time-prohibitive method into a practical alternative with competitive wall-clock training times, achieving a speedup of approximately $9\times$. Nevertheless, on the ESNSDP, the kMIF still demonstrated a $\mathbf{2}\times$ to $\mathbf{4.5}\times$ increase in training and prediction latency, confirming that IF remains the model of choice for resource-constrained environments.

2. **Classic IF Exhibits Superior Robustness and Precision:** Empirically, the classic Isolation Forest demonstrated superior performance in terms of Balanced Accuracy, F1-Score, and PR-AUC on the challenging non-convex synthetic datasets (Moons and Circles). The structural modification in kMIF, intended to capture complex data geometry, led to a high rate of false positives by misclassifying dense, normal points within the core of the clusters as anomalies. The random, rigid partitioning of IF proves to be a more effective and robust isolation heuristic for general-purpose outlier detection than the local 1D k-Means splitting of kMIF.

3. **Divergent Ranking Systems:** The Agreement Analysis on synthetic data confirmed that the two models produce significantly different anomaly score rankings, particularly in the boundary regions between clusters (Spearman correlation ≈ 0.531). This disparity underscores the profound functional difference introduced by kMIF's architectural change, where the IF score is derived from path length and the kMIF score from cluster distance.

4. **Performance on Real-World Security Data:** On the highly imbalanced and non-separable ESNSDP, both models performed poorly, but their trade-offs were distinct. The kMIF was an aggressive classifier, achieving higher Recall ($\mathbf{35.0}\%$) but suffering from a critically high False Positive rate, which is often unacceptable in network security. The classic IF, while more conservative, maintained a superior ranking quality (higher ROC-AUC) and provided a more Precision-focused solution, which is generally preferred when alerts must be trustworthy.

5. **Best Model for Task:** For scenarios where computational resource limits are not a primary concern and a slightly more aggressive recall is required, kMIF offers a viable, albeit more complex, alternative. However, the **classic Isolation Forest** remains the optimal choice for general-purpose anomaly detection: it offers a far greater speed advantage (both in training and prediction), greater robustness across different data complexities, and superior overall F1-Score and ranking performance, particularly on non-rectilinear manifolds and complex, high-dimensional data.

In summary, the novelty introduced by the k-Means-Based Isolation Forest, while structurally interesting and theoretically appealing, does not empirically translate into a reliable performance gain for many common data structures, and it comes at a substantial increase in computational cost. Future work could focus on optimizing the dimensionality of the k-Means clustering at each node, refining the anomaly score aggregation function to better reflect the cluster membership of a sample, or exploring hybrid models that leverage the k-Means split only when a high local density is detected.

References

[1] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104. ACM, 2000.

[2] Christopher M. Bishop. Novelty detection and neural network validation. *IEE Proceedings - Vision, Image and Signal Processing*, 141(4):217–222, 1994.

[3] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. In *Special Lecture on IE*, pages 1–18, 2015.

[4] Sahand Hariri, Matthias Kind, and Robert J. Brunner. Extended isolation forest. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 398–407. IEEE, 2019.

[5] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[6] Paweł Karczmarek, Adam Kiersztyn, Witold Pedrycz, and Ebru Al. K-means-based isolation forest. *Knowledge-based systems*, 195:105659, 2020.

Appendix: Supplementary Figures

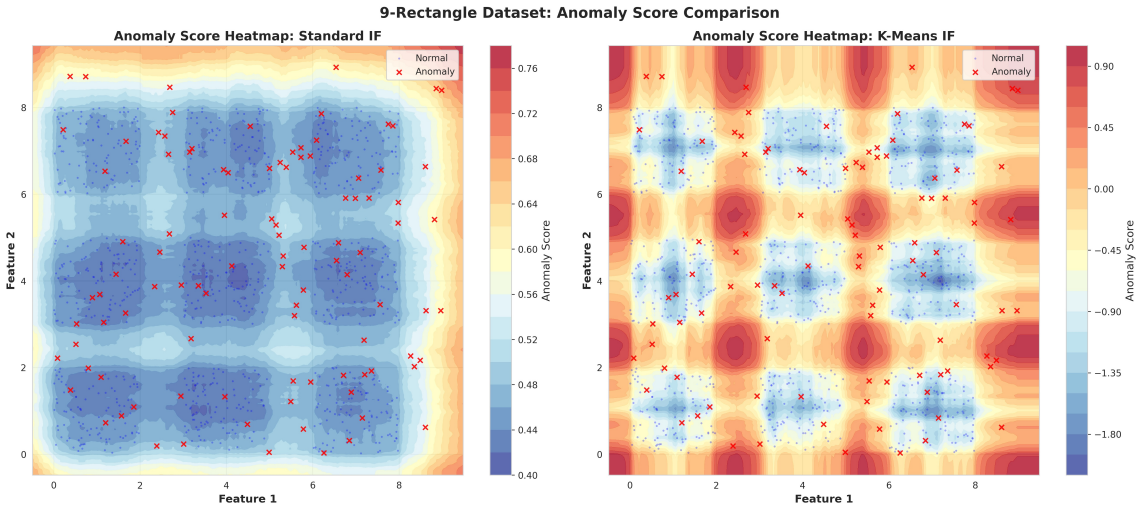

Figure 22: Comparison of anomaly score heatmaps generated by the Standard Isolation Forest (left) and the KMeans Isolation Forest (right) on the 9-Rectangle dataset. The KMeansIF method shows a clearer distinction and lower anomaly scores within the nine distinct, convex normal regions (blue), compared to the Standard IF, which exhibits more uniform scores across the feature space.

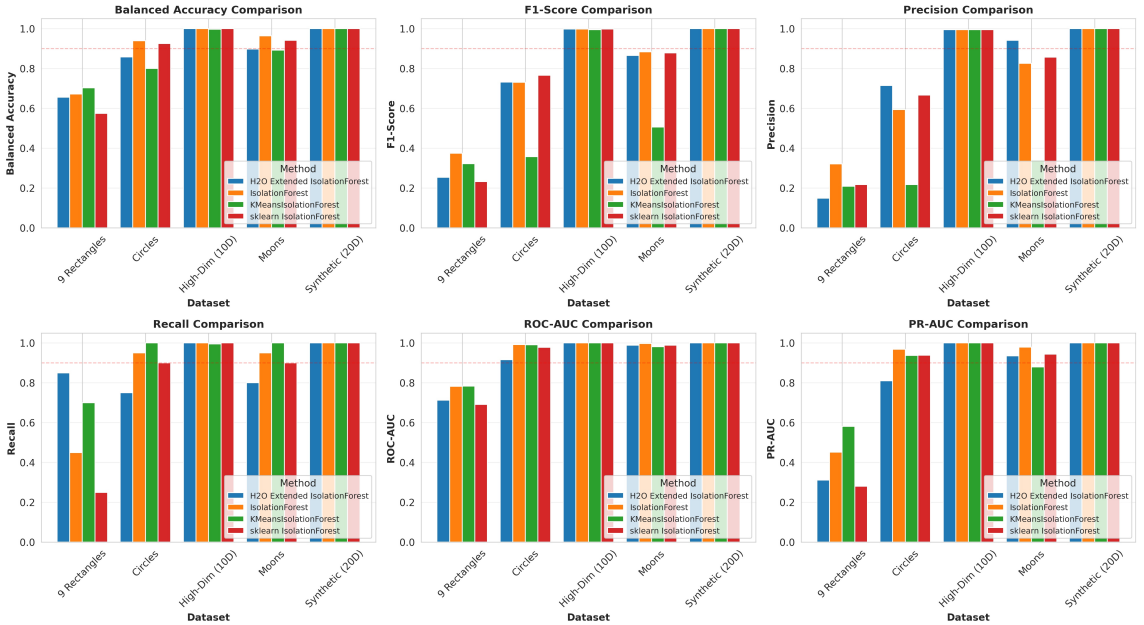

Figure 23: Performance comparison of four different Isolation Forest variants: H2O Extended Isolation Forest, standard Isolation Forest, KMeans Isolation Forest (KMeansIF), and Scikit-learn's Isolation Forest, across six different datasets. The evaluation includes six metrics: Balanced Accuracy, F1-Score, Precision, Recall, ROC-AUC, and PR-AUC. The relative performance varies across datasets, with all methods achieving near-perfect scores on the High-Dim, Moons, and Synthetic datasets, but showing notable differences on the non-convex 9 Rectangles and Circles datasets.