Parallelizing the Random Cut Forest classifier

SANTHOSHI KAREDDY  
M.S IN COMPUTER SCIENCEUNT ID: 11600006  
santhoshikareddyviii@my.unt.edu

SATHVIKA REDDY MULA  
M.S IN COMPUTER SCIENEC  
UNT ID:11630415  
 sathvikareddymula@my.unt.eduSAHITHI PRIYA BHAVANAM  
M.S IN COMPUTER SCIENCE  
UNT ID: 11605815  
sahithipriyabhavanam@my.unt.edu

GOWTHAM REDDY PAPPULA  
M.S IN COMPUTER SCIENCE  
LUNT ID:11583362  
gowthamreddypappula@my.unt.edu

SANHITHA VARMA KONDURU  
M.S IN COMPUTER SCIENCEUNT ID:11630415  
sanhithavarmakonduru@my.unt.edu

*Abstract*— Random cut forest is an unsupervised learning algorithm, RCF is the best algorithm to find anomalies in large time-series or streaming data sets. This algorithm will detect anomaly data points in well patterned data. In this study, we want to implement parallel processing techniques on Random Cut Forest (RCF) to improve efficiency of the algorithm by accelerating training and prediction processes, when dealing with large datasets in real-time systems. We implement strategies like Data partitioning, parallel model training, parallel prediction and batch processing and we examine the results, challenges related to parallelizing the Random cut Forest algorithm. And comparing the performance with traditional Isolation Forest algorithm.

Keywords— Random Cut Forest, Parallel Processing, anomaly detection.

# Introduction

The fundamental concept of RCF is to find anomalies by analyzing well patterned data and how they deviate from the other data points. During the training stage of the model, RCF constructs forest of trees by recursively cutting feature dimension. At each node split, random subset of feature dimension and random threshold is chosen to divide data points. This process will continue until all data points are isolated. Once the forest of trees is constructed using training data points, RCF uses collective response of all trees to identify the anomalies in new data points. Based on depth of the data point in tree, an anomaly score is assigned, data point with highest anomaly score is identified as an anomaly data point.

RCF (Random cut forest) is a powerful algorithm to find anomaly in high-dimensional data streams. RCF works by developing collection of binary trees known as forest, where each tree is built using random selection of feature dimensions and random partitioning of data points. This random selection enables RCF to efficiently detect anomalies in various types of data, including time-series, multivariate, and high-dimensional data.

# Problem Statement

In real time world to find anomalies on large dataset we need efficient algorithms which uses less resources and time to perform the tasks. Traditional algorithms like Isolation Forest algorithm takes a greater number of resources and time to execute the tasks. Through this project we are trying to develop robust and efficient algorithm this is achieved by using the Random Cut Forest Algorithm with open MPI.

# Literature

## Random Cut Forest

Random cut forest is used for anomaly detection for large datasets where traditional methods like isolation forest might be inefficient.

RCF divides the feature space after first choosing features at random from the dataset. This random partitioning procedure is used to create many binary trees, ensuring that each tree is unique from the others. When all the trees are combined, a forest of data points is produced. Every data point pass through a tree and depending on how divergent it is from the average of the points in that partition, the tree provides an anomaly score. How far into the tree the data point may go before hitting a leaf determines the score.   
The total anomaly score of the data point is then calculated by averaging the anomaly values from each tree.

Key Features of RCF:

* Scalability: It can handle large datasets.
* Flexibility: It doesn’t require lot of data preprocessing
* Robustness: It can adapt to different data distribution

Random Cut Forest is a powerful algorithm or anomaly detection, specifically in scenarios where data is streaming, and traditional methods may not be optional due to scale or complexity.

## Isolation Forest

Isolation Forest is an anomaly detection algorithm that works by isolating outliers in a dataset. The main idea behind it is to identify anomalies by isolating them in fewer steps compared to most of the data points which are non-anomalous.

The algorithm begins by randomly picking a feature and a split value within the feature’s range, creating a partition. This random partitioning is recursively repeated, each time randomly selecting a feature and split value to create partitions until each data point is isolated. The number of partitions needed to isolate a data point indicates its anomaly level. Anomalies are expected to be isolated quicker than normal data points. This process is repeated to build multiple isolation trees independently. To calculate the anomaly score for each data point, the average path length to isolation across all trees is determined. Data points with shorter average path lengths are more likely to be anomalies. Finally, a decision threshold is used to classify data points as anomalies based on their anomaly scores. Data points with anomaly scores above the threshold are classified as anomalies.

By utilizing the ideas of random division and isolation in binary trees, Isolation Forest is an effective outlier identification approach that can effectively isolate abnormalities. Because of this, it may be used to a variety of tasks, including anomaly detection, fraud detection, and network security monitoring.

# Implementation

## Random Cut Forest

### System Setup:

OS: Ubuntu (Linux).

Libraries Used: Mpi4py, Pandas, NumPy.

Programming language: Python.

Number of Process: 1, 4, 8.

Dataset: nyc\_taxi.csv

### MPI Implementation:

We scatter the entire data set across different processes, and then trains multiple trees in the forest on each process using assigned dataset. Each process builds its tree independently.

By dividing the Random Cut Forest algorithm's work over several processes, MPI makes it possible to train huge datasets in parallel and compute anomaly scores quickly. Working with part of the data, each process contributes to the outcome. This distributed computing is made possible by the mpi4py package, which controls data distribution and process communication.

### Random Cut Forest Implementation:

RCF was implemented from scratch without using libraries python language using NumPy, Pandas, MPI and random libraries for processing, computing, and presenting the dataset.

Methods developed:

* Building Tree:

This method divides the incoming data X among many MPI processes. It uses a portion of the data to create several trees (num\_trees) for each process. The forest of trees is applied to the complete dataset X to generate anomaly scores. The anomaly scores are returned by the technique.

* Computing Anomaly Score:

Uses the forest of trees to calculate anomaly scores for the whole dataset X. Assigns scores to each sample based on the threshold values from each tree.

* Fitting dataset:

This method divides the incoming data X among many MPI processes. It uses a portion of the data to create several trees (num\_trees) for each process. The forest of trees is applied to the complete dataset X to generate anomaly scores. The anomaly scores are returned by the technique.

## Isolation Forest

### System Setup:

OS: Ubuntu (Linux).

Libraries Used: Mpi4py, Pandas, NumPy.

Programming language: Python.

Number of Process: 1, 4, 8.

Dataset: nyc\_taxi.csv

### MPI Implementation:

All other processes receive the loaded data through broadcast from the root process. The data is now stored in each process. To divide the load, each process separates the data according to its rank. This ensures that a subset of the dataset is worked on by each process.For each portion of the dataset, each process fits an isolation tree on its own.For the data it contains, each process calculates path lengths and associated anomaly scores.Each process computes anomaly scores, which are collected at the root process.

### Isolation Forest Implementation:

Isolation Forest was implemented from scratch without using libraries python language using NumPy, Pandas, MPI and random libraries for processing, computing, and presenting the dataset.

Methods developed:

* Building Tree:

Builds the isolation tree recursively. It terminates when the number of occurrences of X at a node is less than or equal to 1, or when the current\_height reaches height\_limit. The split\_feature and split\_value are chosen at random to divide the data into left and right child nodes. Depending on the split condition, keeps constructing the tree recursively for the left and right child nodes.Find Path Length.

* Computing Anomaly Score

Calculates an instance x path length (depth) down the isolation tree. use the split feature and split value at each node as a basis for its traversal of the tree using the helper recursive function \_path\_length\_helper. The number of edges (or levels) that x must traverse from the root to get to an external node (leaf) is represented by the path length.

* Fitting dataset

Fits the Isolation Tree to the dataset.

### Random Cut Forest Software

from mpi4py import MPI

import numpy as np

import pandas as pd

import random

import time

df = pd.read\_csv("/home/sk2349/project/nyc\_taxi.csv")

df['timestamp'] = pd.to\_datetime(df['timestamp'])

#is anomaly? : True => 1, False => 0

df['anomaly'] = 0

df.index = df['timestamp']

df.drop(['timestamp'], axis=1, inplace=True)

class RandomCutForestMPI:

def \_\_init\_\_(self, num\_trees=100, num\_samples\_per\_tree=256, num\_dimensions=None, threshold=None):

self.num\_trees = num\_trees

self.num\_samples\_per\_tree = num\_samples\_per\_tree

self.num\_dimensions = num\_dimensions

self.threshold = threshold

self.comm = MPI.COMM\_WORLD

self.rank = self.comm.Get\_rank()

self.size = self.comm.Get\_size()

self.trees = []

#print(self.rank)

#print(self.size)

def fit\_predict(self, X):

if self.num\_dimensions is None:

self.num\_dimensions = X.shape[1] if len(X.shape) > 1 else 1

# Scatter the data to different processes

local\_X = comm.scatter(np.array\_split(X, self.size), root=0)

# Build the forest

for \_ in range(self.num\_trees):

tree = self.\_build\_tree(local\_X)

self.trees.append(tree)

# Compute anomaly scores

anomaly\_scores = self.\_compute\_anomaly\_scores(X)

return anomaly\_scores

def \_build\_tree(self, X):

num\_samples = X.shape[0]

# Randomly select samples for the tree

sample\_indices = np.random.choice(num\_samples, size=self.num\_samples\_per\_tree, replace=False)

tree\_X = X[sample\_indices]

# Randomly select features for the tree

feature\_indices = np.random.choice(self.num\_dimensions, size=int(np.sqrt(self.num\_dimensions)), replace=False)

tree\_X = tree\_X[:, feature\_indices]

# Create a tree structure (e.g., using a simple threshold-based approach)

tree = {'feature\_indices': feature\_indices}

if self.threshold is None:

# Compute threshold as the mean of a random feature

random\_feature = np.random.choice(tree\_X.shape[1])

tree['threshold'] = np.mean(tree\_X[:, random\_feature])

else:

tree['threshold'] = self.threshold

return tree

def \_compute\_anomaly\_scores(self, X):

anomaly\_scores = np.zeros(X.shape[0])

for tree in self.trees:

feature\_indices = tree['feature\_indices']

thresholds = tree['threshold']

# Apply the tree to each sample

tree\_scores = np.mean(X[:, feature\_indices] < thresholds, axis=1)

# Update the anomaly scores

anomaly\_scores += tree\_scores

return anomaly\_scores / self.num\_trees

if \_\_name\_\_ == "\_\_main\_\_":

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

print(f"Rank {rank} of {size} processes")

# Assuming `df` is the DataFrame containing the dataset with the feature(s) for anomaly detection

# Extract the feature for anomaly detection (assuming 'value' column)

features = np.array(df['value']).reshape(-1, 1)

# Process 0's work

start\_time = time.time()

# Instantiate and fit RandomCutForestMPI

rcf\_mpi = RandomCutForestMPI(num\_trees=100, num\_samples\_per\_tree=256, num\_dimensions=features.shape[1], threshold=None)

anomaly\_scores = rcf\_mpi.fit\_predict(features)

# Your MPI code here for process 0

end\_time = time.time()

if rank == 0:

elapsed\_time = end\_time - start\_time

print(f"Elapsed Time: {elapsed\_time} seconds")

# Update DataFrame with anomaly scores

df['anomaly'] = anomaly\_scores

# Display DataFrame with anomaly scores

print(df)

print(f"Process {rank} completed")

## IsolationTree

from mpi4py import MPI

import numpy as np

import pandas as pd

class Node:

def \_\_init\_\_(self, left, right, split\_feature, split\_value):

self.left = left

self.right = right

self.split\_feature = split\_feature

self.split\_value = split\_value

class IsolationTree:

def \_\_init\_\_(self, height\_limit):

self.height\_limit = height\_limit

def fit(self, X):

self.root = self.\_make\_tree(X, 0)

def \_make\_tree(self, X, current\_height):

if current\_height >= self.height\_limit or len(X) <= 1:

return Node(None, None, None, None)

else:

split\_feature = np.random.choice(X.shape[1])

split\_value = np.random.uniform(np.min(X[:, split\_feature]), np.max(X[:, split\_feature]))

left\_mask = X[:, split\_feature] < split\_value

right\_mask = ~left\_mask

return Node(self.\_make\_tree(X[left\_mask], current\_height + 1),

self.\_make\_tree(X[right\_mask], current\_height + 1),

split\_feature,

split\_value)

def path\_length(self, x):

return self.\_path\_length\_helper(x, self.root, 0)

def \_path\_length\_helper(self, x, node, current\_path\_length):

if node.left is None and node.right is None:

return current\_path\_length

elif x[node.split\_feature] < node.split\_value:

return self.\_path\_length\_helper(x, node.left, current\_path\_length + 1)

else:

return self.\_path\_length\_helper(x, node.right, current\_path\_length + 1)

def main():

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

# Load data

if rank == 0:

df = pd.read\_csv("/home/sk2349/project/nyc\_taxi.csv")

df['timestamp'] = pd.to\_datetime(df['timestamp'])

df['anomaly'] = 0

df.set\_index('timestamp', inplace=True)

data = df[['value']].values # Extract 'value' feature for anomaly detection

else:

data = None

# Broadcast data

data = comm.bcast(data, root=0)

data = np.array\_split(data, size)[rank]

# Fit isolation tree

it = IsolationTree(height\_limit=200)

it.fit(data)

# Compute path lengths

path\_lengths = np.array([it.path\_length(x) for x in data])

# Compute anomaly scores

anomaly\_scores = path\_lengths / it.height\_limit

# Gather anomaly scores

anomaly\_scores = comm.gather(anomaly\_scores, root=0)

# Print anomaly scores

if rank == 0:

anomaly\_scores = np.concatenate(anomaly\_scores)

ascore=pd.DataFrame(anomaly\_scores)

#print(len(anomaly\_scores))

df['anomaly'] = anomaly\_scores

df.to\_csv('/home/sk2349/project/it.csv')

#print(len(ascore))

if \_\_name\_\_ == '\_\_main\_\_':

main()

# Results

1. Execution time for different processes (in Seconds)

| Classifiers | No.of Processes | | |
| --- | --- | --- | --- |
| N=1 | N=4 | N=8 |
| RCF | 0.024 | 0.016 | 0.106 |
| Isolation Forest | 149.744 | 136.731 | 376.6173 |

From the above table we can observe execution time taken by each algorithm with different processes.

1. CPU Utilization

| Classifiers | No.of Processes | | |
| --- | --- | --- | --- |
| N=1 | N=4 | N=8 |
| RCF | 1.1500000000000008% | 5.322000000000008% | 12.45400000000001% |
| Isolation Forest | 0.5519999999999998% | 11.706999999999983% | 58.501% |

From the above table we can observe CPU Utilization by each algorithm with different processes.

1. Memory usage

| Classifiers | No.of Processes | | |
| --- | --- | --- | --- |
| N=1 | N=4 | N=8 |
| RCF | 861.27 MB | 1140.15 MB | 1588.28 MB |
| Isolation Forest | 1367.88 MB | 2976.55 MB | 3750.60 MB |

From the above table we can observe Memory Usage by each algorithm with different processes.

1. Over Heads

| Classifiers | No.of Processes | |
| --- | --- | --- |
| N=4 | N=8 |
| RCF | 1.5 | 0.226 |
| Isolation Forest | 1.095 | 0.398 |

From the above table we can observe Overheads of each algorithm with different processes.

# DISCUSSION

The RCF is the key algorithm, to find anomaly in real-time with large datasets, it consumes less CPU Utilization and Memory Usage. It can find anomalies in very less time and with minimum resources compared to traditional algorithms like Isolation Forest

# Conclusion

The project successfully demonstrated the efficiency and accelerated performance in RCF using MPI parallel processing. Implementing RCF on MPI resulted in a robust anomaly detector capable of handling large amount of datasets in real-time computations.

# References

1. <https://docs.aws.amazon.com/sagemaker/latest/dg/randomcutforest.html>
2. <https://medium.com/analytics-vidhya/random-cut-forest-321aae4d8a59>
3. https://opensearch.org/blog/random-cut-forests/
4. C. C. Aggarwal. Outlier Analysis. Springer New York, 2013.ndon, 1892, pp.68–73.
5. L. Breiman. Random forests. Machine Learning, 45(1):532, 2001.K. Elissa, “Title of paper if known,” unpublished.
6. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Chapman and Hall & CRC, 1984.
7. S. Guha, N. Mishra, G. Roy, and O. Schrijver. Robust random cut based anomaly detection in dynamic data streams. Proc. of ICML (JMLR Proceedings Series), 2016.

H. Ishwaran. The effect of splitting on random forest. Machine learning, 99:75–118, 2014.