In [3]:
```
import pandas as pd
import time
import os
import psutil
import gc
import sys
import numpy as np
import math
from tqdm import tqdm
import line_profiler
```

In [5]:
```
df1=pd.read_csv('recut.csv',header=None)
testing = df1.iloc[:,2:]
```

In [6]:
```
df2 = pd.read_csv('newRecut.csv',header=None,sep='\t')
training =  df2.iloc[:,3:]
```

In [7]:
```
testing
```

Out[7]:

	2	3	4	5	6	7	8	9	10	11	...	481	482	483	484	
0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
1	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
2	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
3	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
4	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
146091	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
146092	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
146093	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
146094	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6
146095	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	65535.0	...	65535.0	65535.0	65535.0	65535.0	6

146096 rows × 489 columns

In [8]:
```
training
```

Out[8]:

	3	4	5	6	7	8	9	10	11	12	...	482	483	484	
0	0.052075	0.052360	0.055625	0.058591	0.060967	0.063452	0.068622	0.071691	0.074371	0.080955	...	0.432697	0.436374	0.433194	0.
1	0.051643	0.053531	0.055357	0.059845	0.061159	0.065157	0.069040	0.072510	0.075409	0.081626	...	0.443428	0.445660	0.440417	0.
2	0.052075	0.052360	0.055625	0.058591	0.060967	0.063452	0.068622	0.071691	0.074371	0.080955	...	0.432697	0.436374	0.433194	0.
3	0.051643	0.053531	0.055357	0.059845	0.061159	0.065157	0.069040	0.072510	0.075409	0.081626	...	0.443428	0.445660	0.440417	0.

4 rows × 489 columns

In [9]:
```
def getStatistics(startTime):
    # Statistics of the Algorithm after execution
    # print("Dataset name:", "SampleRunTrainFile.txt")
    print("Total Execution time of proposedAlgo", time.time() - startTime)
    process = psutil.Process(os.getpid())
    memory = process.memory_full_info().uss
    memory_in_KB = memory / 1024
    print("Memory of proposedAlgo in KB:", memory_in_KB)  # in bytes
```

In [10]:
```
def rasterFuzzyTSC(training, testing,topElements):
    startTime = time.time()

    # To drop the first column which is a class label of each sample
    # training.drop(0, axis='columns', inplace=True)

    # separating classes into different dictionary variables and creating mean, min and max curves
    maxData = pd.DataFrame(training.max(axis=0)).T
    minData = pd.DataFrame(training.min(axis=0)).T
    meanData = pd.DataFrame(training.mean(axis=0)).T

    del training
    gc.collect()

    # Classification Phase

    counter = {}
    num_rows, num_columns = testing.shape

    # print(num_rows, num_columns)
    newColumn = list()
    # print(num_columns)
    correct = 0
    for i in tqdm(range(num_rows)):
        counter = 0.0
        for j in range(1, num_columns):
            if testing.iloc[i, j] >= meanData.iloc[0,j]:
                if testing.iloc[i, j] <= maxData.iloc[0,j]:
                    counter = counter + 0.5 * (testing.iloc[i, j] - meanData.iloc[0,j]) / (maxData.iloc[0,j] - meanData.iloc[0
                else:
                    counter = counter + 1
```

```
            else:
                if testing.iloc[i, j] >= minData.iloc[0,j]:
                    counter = counter + 0.5 * (meanData.iloc[0,j] - testing.iloc[i, j]) / (meanData.iloc[0,j] - minData.iloc[0
                else:
                    counter = counter + 1

        counter = counter / num_columns

        newColumn.append(counter)

    testing['RD'] = newColumn

    # N = int(input("Enter the how many number of top elements to be retrieved between: 1 to " + str(num_rows) + ":"))
    N = topElements
    sortedDF = testing.sort_values('RD').head(N)
    getStatistics(startTime)
    return testing, sortedDF
```

In [17]:
```
import tarunParallelFuzzyOneClassClassifier as alga
```

In [19]:
```
trainingDF=training.iloc[:,:]
```

In [21]:
```
testingDF=testing.iloc[:,:-1]
```

In [25]:
```
_,topDFSingle =  alga.rasterFuzzyTSC(trainingDF,testingDF,10, 'single')
```

```
Total Execution time of proposedAlgo 35.36864757537842
```

In [26]:
```
_,topDFParallel =  alga.rasterFuzzyTSC(trainingDF,testingDF,10, 'parallel')
```

```
Total Execution time of proposedAlgo 2.735771894454956
```

In [27]:
```
_,topDFcuda =  alga.rasterFuzzyTSC(trainingDF,testingDF,10, 'cuda')
```

```
Total Execution time of proposedAlgo 1.3574156761169434
```

In [28]:
```
topDFSingle
```

Out[28]:

	2	3	4	5	6	7	8	9	10	11	...	481	482	48
137520	0.050999	0.052464	0.055512	0.058662	0.060770	0.064085	0.067578	0.072179	0.074534	0.079875	...	0.441750	0.443071	0.4368
140681	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
141078	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
137911	0.051450	0.052915	0.055312	0.058845	0.061210	0.064394	0.068205	0.072050	0.074690	0.080993	...	0.434885	0.436782	0.4319
137121	0.050776	0.052658	0.055525	0.058978	0.060430	0.063482	0.068262	0.072171	0.074006	0.081057	...	0.438391	0.439715	0.4339
106941	0.052166	0.052759	0.055443	0.059056	0.060892	0.064411	0.068399	0.071596	0.075050	0.081266	...	0.436593	0.437834	0.4319
107338	0.052444	0.053321	0.055288	0.058923	0.060762	0.064796	0.068301	0.072262	0.074928	0.080618	...	0.436704	0.438588	0.4333
139108	0.050541	0.053066	0.056100	0.059940	0.061225	0.064496	0.068740	0.072144	0.074929	0.081277	...	0.437010	0.438189	0.4321
98229	0.051337	0.053317	0.055940	0.059308	0.060839	0.064683	0.067961	0.072245	0.075379	0.080848	...	0.439286	0.441581	0.4331
102157	0.051710	0.052726	0.055379	0.058907	0.060457	0.064931	0.068451	0.071895	0.075247	0.081486	...	0.436058	0.436741	0.4306

10 rows × 489 columns

In [29]:
```
topDFParallel
```

Out[29]:

	2	3	4	5	6	7	8	9	10	11	...	481	482	48
137520	0.050999	0.052464	0.055512	0.058662	0.060770	0.064085	0.067578	0.072179	0.074534	0.079875	...	0.441750	0.443071	0.4368
140681	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
141078	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
137911	0.051450	0.052915	0.055312	0.058845	0.061210	0.064394	0.068205	0.072050	0.074690	0.080993	...	0.434885	0.436782	0.4319
137121	0.050776	0.052658	0.055525	0.058978	0.060430	0.063482	0.068262	0.072171	0.074006	0.081057	...	0.438391	0.439715	0.4339
106941	0.052166	0.052759	0.055443	0.059056	0.060892	0.064411	0.068399	0.071596	0.075050	0.081266	...	0.436593	0.437834	0.4319
107338	0.052444	0.053321	0.055288	0.058923	0.060762	0.064796	0.068301	0.072262	0.074928	0.080618	...	0.436704	0.438588	0.4333
139108	0.050541	0.053066	0.056100	0.059940	0.061225	0.064496	0.068740	0.072144	0.074929	0.081277	...	0.437010	0.438189	0.4321
98229	0.051337	0.053317	0.055940	0.059308	0.060839	0.064683	0.067961	0.072245	0.075379	0.080848	...	0.439286	0.441581	0.4331
102157	0.051710	0.052726	0.055379	0.058907	0.060457	0.064931	0.068451	0.071895	0.075247	0.081486	...	0.436058	0.436741	0.4306

10 rows × 489 columns

In [30]:
```
topDFcuda
```

Out[30]:

	2	3	4	5	6	7	8	9	10	11	...	481	482	48
137520	0.050999	0.052464	0.055512	0.058662	0.060770	0.064085	0.067578	0.072179	0.074534	0.079875	...	0.441750	0.443071	0.4368
140681	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
141078	0.051406	0.052683	0.055191	0.058703	0.059723	0.064666	0.067597	0.071206	0.074184	0.080440	...	0.438394	0.441505	0.4370
137911	0.051450	0.052915	0.055312	0.058845	0.061210	0.064394	0.068205	0.072050	0.074690	0.080993	...	0.434885	0.436782	0.4319
137121	0.050776	0.052658	0.055525	0.058978	0.060430	0.063482	0.068262	0.072171	0.074006	0.081057	...	0.438391	0.439715	0.4339
106941	0.052166	0.052759	0.055443	0.059056	0.060892	0.064411	0.068399	0.071596	0.075050	0.081266	...	0.436593	0.437834	0.4319
107338	0.052444	0.053321	0.055288	0.058923	0.060762	0.064796	0.068301	0.072262	0.074928	0.080618	...	0.436704	0.438588	0.4333
139108	0.050541	0.053066	0.056100	0.059940	0.061225	0.064496	0.068740	0.072144	0.074929	0.081277	...	0.437010	0.438189	0.4321
98229	0.051337	0.053317	0.055940	0.059308	0.060839	0.064683	0.067961	0.072245	0.075379	0.080848	...	0.439286	0.441581	0.4331
102157	0.051710	0.052726	0.055379	0.058907	0.060457	0.064931	0.068451	0.071895	0.075247	0.081486	...	0.436058	0.436741	0.4306

10 rows × 489 columns

In [ ]: