CAN MACHINE LEARNING MODEL'S DECISION BOUNDARY BE A FRACTAL? *

Dhruv Gupta, Aditya Nagarsekar, Vraj Shah
Department of Computer Science and Information Systems
BITS Pilani K K Birla Goa Campus
403726, Goa, India
dhruvgupta110205@gmail.com, adityanagarsekar2108@gmail.com, vrajshah2785@gmail.com

Nithin Nagaraj
Complex Systems Programme
National Institute of Advanced Studies, Indian Institute of Science Campus
Bengaluru, 560012, Karnataka, India
nithin@nias.res.in

Harikrishnan N B
Department of Computer Science and Information Systems
BITS Pilani K K Birla Goa Campus
403726, Goa, India
harikrishnannb@goa.bits-pilani.ac.in

ABSTRACT

Modern datasets often contain a large number of features connected through complex dependency structures. To effectively analyze such data, dimensionality reduction plays a crucial role, with many methods relying on the notion of intrinsic dimension (id) as a measure of dataset complexity. Estimating this quantity, however, is challenging: the value of the id can vary significantly with the scale at which interpoint distances are evaluated. At very small scales, noise tends to inflate the estimated id, whereas at larger scales the estimates stabilize, yielding lower and more scale-invariant values. This paper proposes a novel, highly parallelizable method for id estimation, based on a measure of local connectivity we term as "Connectivity Factor" (CF). We empirically show that our method is robust to noise, and produces comparable estimates to other state-of-the-art methods. We use our method to detect fractality of decision boundaries with high accuracy, and justify the use of CF for object degradation analysis and fraud detection.

***K*eywords** Fractal · Online Learning · Machine Learning · Decision Tree

1 Introduction

1.1 A Brief Overview

Topology, a branch of mathematics focused on properties of space that remain invariant under continuous deformations such as stretching or bending (but excluding tearing or gluing), offers a powerful framework increasingly utilized in data analysis. Within Computer Science, topology finds several significant applications, including:

- **Understanding Intrinsic Dimensionality:** Complex high-dimensional data frequently reside on or near a lower-dimensional manifold embedded within the ambient space (Manifold Hypothesis). Topological methods

*Citation: **Authors. Title. Pages.... DOI:000000/11111.**

assist in estimating this *intrinsic dimension*, thereby uncovering the true degrees of freedom in the data. This insight is vital for effective dimensionality reduction, data visualization, and understanding the generative mechanisms underlying the dataset, ultimately alleviating the curse of dimensionality.

- **Robustness to Scale and Deformation (via Persistence):** The genuine structure of data may manifest differently across scales, while real-world data often undergo nonlinear distortions. Persistent homology, a cornerstone tool in Topological Data Analysis (TDA), tracks the emergence and disappearance of topological features over a continuum of scales (filtrations). Features that persist across extensive scale ranges are regarded as robust, whereas short-lived features are commonly attributed to noise or artifacts. This multiscale approach obviates the need for selecting an arbitrary single scale and ensures resilience to continuous transformations such as stretching or bending.
- **Robustness to Noise:** Topological features identified through persistence exhibit stability under moderate noise perturbations. Although noise may introduce ephemeral, small-scale topological artifacts, the persistent homology framework discriminates these from stable, significant features reflective of the underlying signal. Consequently, TDA proves valuable for analyzing noisy, real-world datasets.
- **Characterizing Fractal Structures:** Fractals are geometric constructs featuring self-similarity across scales and often possess non-integer dimensions. Classical topology addresses integer dimensions; however, topological concepts related to dimension-such as Hausdorff or Box-Counting dimensions, which can be estimated with topological techniques-are essential for identifying and quantifying the complexity of fractal-like structures found in natural phenomena and datasets (e.g., coastlines, turbulence, biological networks). Topology thereby provides a formal framework to articulate dimension in these intricate cases.

1.2 Related Works and Differences

1.2.1 Literature Review

A work by Isha Sehgal and K.S. Venkatesh [1] largely resembles what we are trying to accomplish here in the aspects of the algorithm being extendable to n dimensions while utilizing a parallel processing approach to ensure complete utilization of system resources for processing.

- The concept of dividing the grids into multiple parts after which we assign each part to a separate processor and then combine them at the end is the same. However the difference lies the number of parts we divide the data into. The method used by [1] allows us to divide the image/data into 2 or 4 parts wheareas with the help of our GridGenerator class discussed in section 7.2.1 we can generate a divided grid for n dimensions with us having to specify how many divisions on each axis are required. For example if I take divisions as 50 then I will have 2500 divisions total with the load of data being reduced for highly dense grids with a base spacing of say 0.0005 to the image size of 1. This helps us largely in a Boundary Extraction Algorithm in an n dimensional space that we have discussed in section 7.1.
- The algorithm extends it's concepts to an n dimensional space. The method of achieving this effect is however quite different. [1] leverages the shape of the array to get the dimensions of the array on the other hand we are using a in essence a recurrence pattern to generate neighbours of a point in n dimensions which can be extended by giving a range to make a whole n dimensional grid through just the point as a 1D array with each feature representing a value on an axis in those dimensions. Through this we essentially create an n dimensional cube which is centered around the given point. We can now move this cube easily by shifting the origin technique and we compare our values as such. This process is discussed in sections 7.5, 7.1 and 3.3.
- The algorithm of connected components works to establish a connection between two groups of pixels which can be achieved through the concept of average connectivity factor increase between two components when they are considered a single component.

A survey by Francesco Camastra [2] has a section titled **'The near neighbour algorithm'** which talks about how we can estimate the Intrinsic Dimension of an object. The author references Trunk [3] who's method was described as follows.

"An initial value of an integer parameter k is chosen and the k nearest neighbours (KNN) to each pattern in the given data set are identified. The subspace spanning the vectors from the ith pattern to its k nearest neighbors is constructed for all patterns. The angle between the (k+1)th near neighbor of pattern i and the subspace constructed for pattern i is then computed for all i. If the average of these angles is below a threshold, ID is k. Otherwise, k is incremented by 1 and the process is repeated."

Camastra [2] further goes on to point out the fact that the weakness of Trunk's [3] method was that there was no clear way to fix a suitable value for thresholds. This problem is addressed in Research Gap section. 1.3.

1.3 Research Gap

Intrinsic dimension estimation in computerized spaces encounters challenges due to the discrete nature of data representations and the intrinsic scale- and perspective-dependence of topological features. Unlike continuous mathematical objects, point clouds comprise finite, often sparse, samples that inherently limit infinite resolution and connectivity assumptions. This limitation complicates the identification of neighbors and dimension, especially in high-dimensional settings.

Thus, selecting an appropriate scale and neighborhood definition is non-trivial and critical for meaningful results. Existing approaches that rely on continuous metric spaces and naive distance thresholds often fail to generalize efficiently in high dimensions or handle scale ambiguity robustly.

State-of-the-art intrinsic dimension estimation methods often rely on global pairwise distance computations or recursive topological constructions, which limits their ability to be efficiently parallelized due to intensive data dependencies and synchronization requirements.

1.4 Our Approach: eDCF (Empirically-weighted Distributed Connectivity Factor)

This paper introduces **eDCF**, an empirically-weighted Distributed Connectivity Factor framework that addresses these challenges through several key innovations.

First, we employ a **grid-based neighbor framework**, where directions and neighbors are defined discretely via vector and set constructs instead of continuous distance metrics. This enables computationally efficient algorithms for neighbor generation and grid alignment. It also enables us to discretize space into independent subregions, enabling **fully parallel computation** of neighbor relations on separate grid cells without cross-dependencies. This facilitates scalable execution on large, high-dimensional datasets using multiprocessing and thread-based parallelism.

Instead of traditional scale selection, we introduce an alternative information-theoretic measure called *Information Percentage*, which guides scale selection based on data coverage and information retention. This avoids manual or heuristic scale choices.

We also define a novel *Connectivity Factor (CF)*, a measure of local connectivity, based on the above gridded neighbor framework and the Information Percentage. We derive theoretical bounds for intrinsic dimension of manifolds using this CF. Then, using a weighted average of this CF, called *Distributed Connectivity Factor (DCF)*, we provide a theoretical method to estimate intrinsic dimension of arbitrary manifolds, under the assumption that the number of data points is sufficiently large.
To extend it to practical scenarios, we use an empirical method to weigh the CF in DCF, finally introducing *eDCF*. This method robustly captures multi-scale connectivity while accommodating noise and sparsity typical in real-world data. It overcomes obstacles posed by scale sensitivity, neighborhood ambiguity, and computational complexity in high-dimensional intrinsic dimension estimation, and is validated through extensive synthetic manifold benchmarks against state-of-the-art methods like TwoNN and MLE.

2 Preliminaries and Problem Definition

2.1 Notational Definition

Basic Notations:

- $\mathbb{N}$: Set of all natutal numbers.
- $\mathbb{Z}$: Set of all integer numbers.
- $\mathbb{Z}^n$: $\{(x_1, x_2, ..., x_n) : x_i \in \mathbb{Z}, i \in \mathbb{N}, i \in [1, n]\}$
- $\mathbb{R}$: Set of all real numbers.
- $\mathbb{R}^n$: $\{(x_1, x_2, ..., x_n) : x_i \in \mathbb{R}, i \in \mathbb{N}, i \in [1, n]\}$
- $\sum$: Summation Operator
- $\cap$: Intersection over sets.
- $|X|$: Cardinality of set X.

$\mathcal{CF}$ - Connectivity Factor :

Connectivity Factor is a measure which ranges between [0, 1] and $\mathcal{CF} \in \mathbb{R}$

- $\mathcal{CF}^n_m$: Connectivity Factor of an object in m dimensional space in an n dimensional space.
- $\mathcal{CF}^m_m$: Connectivity Factor of an object in m dimensional space in that space itself.

  The following is based on Minimal Sets and Limits which is discussed in section 3.3.3
- ${}^{L}CF^n_m$: Lower Bound of Connectivity Factor of an object in m dimensional space in an n dimensional space.
- ${}^{M}CF^n_m$: Most Probable Connectivity Factor of an object in m dimensional space in an n dimensional space.
- ${}^{U}CF^n_m$: Upper Bound of Connectivity Factor of an object in m dimensional space in an n dimensional space.

$\text{Neighbours}(X) -$ Neighbour Set of a Point X (the set of immediate neighbors of a point in a given dimension according to a gridded space)

$\mathcal{DCF}$ - Distributed Connectivity Factor
$\rceil\mathcal{DCF}$ - Empirically - weighted Distributed Connecitivity Factor

2.2 Problem Definition

Providing theoretical and empirical bounds on Topological Dimension of lower dimensional objects embedded in higher dimensions.

3 Proposed Method

3.1 Formal Description of Neighbors

In the context of an n-dimensional grid $\mathbb{Z}^n$, neighbors of a point are defined as the set of points that surround the given point within a single step along each axis of the grid. This definition leverages the standard basis vectors of the grid to systematically characterize neighbor relationships.

3.1.1 Mathematical Definition:

A point $\mathbf{p} \in \mathbb{Z}^n$ is considered a neighbor of another point $\mathbf{q} \in \mathbb{Z}^n$ if it can be expressed as:

$$\mathbf{p} = \mathbf{q} + c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n,$$

where:

- $\mathbf{v}_i$ are the standard basis vectors of $\mathbb{Z}^n$, each representing movement along a single axis.
- $c_i \in \{-1, 0, 1\}$ allows for displacement by one step in the positive direction, negative direction, or no movement along each axis.

3.1.2 Exclusion of the Point Itself:

To ensure that a point is not considered its own neighbor, the case $c_1 = c_2 = \cdots = c_n = 0$ is explicitly excluded from the set of neighbors.

3.1.3 Number of Neighbors:

The total number of neighbors of a point in an n-dimensional grid is given by:

$$R_n = 3^n - 1,$$

which represents all 3^n possible combinations of c_i minus the case where all $c_i = 0$.

3.1.4 Extension to Real Number Space:

This idea naturally extends to a Real Number Space when considering a specific uniform spacing $s \in \mathbb{R}$ along each axis of the grid. In this setting, points $p, q \in \mathbb{R}^n$ are neighbors if:

$$\mathbf{p} = \mathbf{q} + c_1 s\mathbf{v}_1 + c_2 s\mathbf{v}_2 + \cdots + c_n s\mathbf{v}_n,$$

with $c_i \in \{-1, 0, 1\}$, discretizing the continuous space into a uniform grid system.

3.2 Dynamic Spacing: Information Percentage

Dynamic Spacing Adjustment is a coarse-to-fine search procedure devised to efficiently identify grid spacings that yield a target Information Percentage (IP) bucket. The Information Percentage is defined as

$$IP = \frac{\text{Number of output (grid-assigned) data points}}{\text{Number of data points in the original set}} \times 100,$$

and quantifies the proportion of retained grid points after discretization.

At each candidate grid spacing, points from the original dataset are assigned to discrete grid buckets. Inevitably, some points map to overlapping bins, causing a reduction in IP. The minimum attainable IP occurs when all points fall into a single grid cell. Because buckets are discrete, IP can only take certain quantized values for a finite dataset. Selecting a finer grid spacing (i.e., smaller δx) leads to more possible buckets and thus finer attainable IP increments.

This strategy allows for adaptive refinement: the algorithm first explores coarse spacings quickly, and then incrementally narrows the search in regions where the IP approaches the specified target.

3.3 Connectivity Factor

3.3.1 Formal Definition:

We formally define connectivity factor as follows:

If we have a structure in the form of a set of points S, and S_i represents the i^{th} element of this set, then:

$$\mathcal{CF} = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{|\text{Neighbours}(S_i) \cap S|}{|\text{Neighbours}(S_i)|}$$

We know that in an n-dimensional space,

$$|\text{Neighbours}(S_i)| = 3^n - 1$$

Thus,

$$\mathcal{CF} = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{|\text{Neighbours}(S_i) \cap S|}{3^n - 1}$$

3.3.2 Space Conversion Formula:

In a n-dimensional space, for topological dimension m:

$$\mathcal{CF}^n_m = \frac{3^m - 1}{3^n - 1} \mathcal{CF}^m_m.$$

Here, $0 \leq m \leq n$, and $n > 0$. When $n = 0$, the denominator $3^n - 1 = 0$, and we define this base case explicitly as

$$\mathcal{CF}^0_0 = 1.0,$$

(note that since $n = 0$ and $m \leq n$, we have $m = 0$).

$\mathcal{CF}^m_m$ is the structure S as embedded in an m-dimensional space where at least one point in the set S has a contributing connectivity factor of 1.0.

3.3.3 Some Important CF Factors:

Minimal Set in m Dimensions: To determine the minimal set whose points possess an intrinsic topological dimension m under the connectivity factor formalism, we require the set to contain the fewest points such that exactly one point achieves a connectivity factor of 1.0. Intuitively, this means that all immediate neighbors of exactly one central point, as defined by the grid system for dimension m, must also be present in the set.

Examples of minimal sets:

$m = 0$:

X

$m = 1$:

X	X	X

$m = 2$:

X	X	X
X	X	X
X	X	X

Lower Bound for CF:

More formally, for a minimal set S with $|S| = 3^m$, the lower bound on the connectivity factor is given by:

$$^{L}\mathcal{CF}_m^m = \frac{X_m}{(3^m - 1)(3^m)}$$

where

$$X_m = \sum_{i=1}^{3^m} X_i', \qquad X_i' = |\text{Neighbours}(S_i) \cap S|.$$

The value X_m counts all neighbor-interactions for this fully packed minimal neighborhood.

To compute the term X_m, we consider a grid of size 3^n, where n denotes the ambient dimension, and every point belongs to a minimal set with uniform spacing (assumed to be 1 for simplicity). Each such point can be naturally represented as a vector:

$$V^T = [c_1, c_2, c_3, \ldots, c_n]$$

where the c_i specify the offset along each axis.

Let us now formalize the interaction counts associated with different types of points through a recursive series:

- a_0 captures the interactions for the central element, i.e., the point with all $c_i = 0$ for $i \in [1, n]$. This central point possesses the maximal number of possible neighbors, so $a_0 = 3^n - 1$.
- a_1 counts the interactions for those points lying on the axes, i.e., having $c_1 \in \{-1, 1\}$ and $c_i = 0$ for $i \geq 2$, without loss of generality. Since the "pivot" axis can be chosen in n ways and each can take two signs, there are $2^1 \cdot {}^nC_1$ such points.
- Extending this logic, a_m counts points where m of the n coordinates are nonzero (each ± 1), and the remainder are zero. Thus, there are $2^m \cdot {}^nC_m$ such points for each $m = 0, 1, \ldots, n$.

Given these definitions, the total number of neighbors for the central point is $a_0 = 3^n - 1$. To understand the structure recursively, consider what occurs when making a single step away from the center along one axis: precisely one third of the grid points become inaccessible as neighbors, which yields the relationship

$$a_1 = a_0 - \frac{1}{3}(a_0 + 1).$$

This recursive structure extends further; at each level, a similar exclusion applies, producing the general recurrence

$$a_m = a_{m-1} + \frac{1}{3}(a_{m-1} + 1).$$

By solving this recurrence, we obtain the closed-form:

$$a_m = 2^m 3^{n-m} - 1. \tag{1}$$

Accumulating the contributions for all point types yields the aggregate interaction sum, and thus,

$$^{L}\mathcal{CF}_{m}^{m} = \frac{\sum_{i=0}^{m}[2^{i}3^{m-i} - 1] \cdot 2^{i} \cdot {}^{m}C_{i}}{(3^{m}-1)(3^{m})}.$$

We can simplify this further using the binomial theorem, giving:

$$^{L}\mathcal{CF}_{m}^{m} = \frac{(4+3)^{m} - (2+1)^{m}}{(3^{m}-1)(3^{m})} = \frac{7^{m} - 3^{m}}{(3^{m}-1)(3^{m})}.$$

Therefore, for minimal sets in m dimensions, the lower bound for the connectivity factor is:

$$^{L}\mathcal{CF}_{m}^{m} = \begin{cases} \dfrac{7^{m} - 3^{m}}{(3^{m}-1)(3^{m})}, & \text{if } m > 0 \\ 1, & \text{if } m = 0 \end{cases}, \quad \text{where } m \in \mathbb{W}.$$

Most Probable Value for CF:

Consider the scenario where, in an m-dimensional space, our set contains all possible points corresponding to a given grid spacing ds, effectively filling the space entirely. In this setting, as the set size $|S|$ tends to infinity, every point in the m-dimensional structure has all of its possible neighbors within the set - that is, each point satisfies

$$|\text{Neighbours}(S_i) \cap S| = 3^{m} - 1.$$

This yields a connectivity factor of

$$\mathcal{CF}_{m}^{m} = \frac{1 \times |S|}{|S|} = 1,$$

and thus, in the infinite limit,

$$\lim_{|S| \to \infty} \mathcal{CF}_{m}^{m} = 1.$$

To relate the intrinsic m-dimensional structure to its representation when embedded in a higher-dimensional space of dimension n, we employ the space conversion formula for the most probable configuration:

$$^{M}\mathcal{CF}_{m}^{n} = \frac{3^{m} - 1}{3^{n} - 1}.$$

A justification for why this represents the most probable value for $\mathcal{CF}$ is provided in the appendix.

Upper Bound for CF:

The goal of this derivation is to establish a precise mathematical formula for $^{U}\mathcal{CF}_{m}^{n}$, which represents the theoretical **maximum achievable Connectivity Factor** for an idealized, non-fractal structure possessing topological dimension $\boldsymbol{m}$ embedded within a higher-dimensional space of ambient dimension $\boldsymbol{n}$.

Step 1: Conceptual Foundation — A Perfectly Filled Grid We begin by considering an infinitely large, fully populated n-dimensional integer lattice where every coordinate point exists. This perfect grid acts as the conceptual "universe" in which idealized topological objects are constructed.

Step 2: Point Classification by Geometric Type (a_t) Within this lattice, each point is classified into a type $\boldsymbol{a_t}$, depending on its relative geometric position to a local origin, typically $(0, 0, \ldots, 0)$. The type corresponds to the number of nonzero coordinates (i.e., coordinates with value ± 1):

- **Type $\boldsymbol{a_0}$**: The origin point itself, serving as the center of the local n-cube.
- **Type $\boldsymbol{a_1}$**: Points where exactly one coordinate is nonzero, e.g., $(1, 0, \ldots, 0)$.
- **Type $\boldsymbol{a_2}$**: Points with exactly two nonzero coordinates, e.g., $(1, 1, 0, \ldots, 0)$.
- **Type $\boldsymbol{a_t}$**: In general, points with exactly t nonzero coordinates. The total number of such points within a single n-cube is

$$2^{t} \cdot {}^{n}C_{t}.$$

The number of points of type m is denoted $\alpha_m^n = 2^m \cdot {}^nC_m$.

System Space: In the context of recursive neighborhood analysis on a discretized n-dimensional grid, the **system space** is a conceptual x-dimensional combinatorial structure formed by the collection of subsystems s_m^{n-x} associated with points of geometric type a_x. Each subsystem s_m^{n-x} represents a lower-dimensional grid of dimension $n-x$ centered at points with m nonzero coordinates, where $m \leq x$.

Intuitively, the system space organizes the local neighborhood of a point of type a_x as an x-dimensional lattice of smaller subsystems of dimension $n-x$, enabling a recursive decomposition of the original n-dimensional neighborhood counting problem into tractable lower-dimensional analyses. This hierarchical structure allows the counting of neighbor interactions and connectivity contributions by aggregating over the subsystems within the system space.

Formally, for a fixed x, the system space is the x-dimensional hypercube whose nodes correspond to subsystems s_m^{n-x} with types $m = 0, \ldots, x$, arranged such that the overall neighborhood patterns can be expressed as sums over these constituent subsystems.

We refer to the system centered on type a_0 points as the 0-centric system s_0^n. Increasing the "type" by 1 generates the 1-centric system s_1^n, and in general, an m-centric system s_m^n, where the dimension of the system space corresponds to m.

Step 3: Core Insight — Recursive Dimensional Reduction The key insight is that analyzing neighbors from the perspective of a type a_x point reduces the problem to one in a lower-dimensional space of dimension $(n-x)$. This recursive framing simplifies complex n-dimensional neighbor counting into tractable lower-dimensional analyses.

- **Logic**: Analyzing neighbors from a type a_x point transforms the counting problem into that of several $(n-x)$-dimensional systems.
- This perspective incrementally increases the "system space" dimension x while reducing the system dimension accordingly.
- For example, at $x = 1$, the system space comprises the multiset $\{s_0^{n-1}, s_1^{n-1}, s_0^{n-1}\}$.
- At $x = 2$, the system space expands to a multiset of nine elements:
$$\begin{Bmatrix} s_0^{n-2} & s_1^{n-2} & s_0^{n-2} \\ s_1^{n-2} & s_2^{n-2} & s_1^{n-2} \\ s_0^{n-2} & s_1^{n-2} & s_0^{n-2} \end{Bmatrix}.$$
- The system space given x and n, acts as an x dimensional cube itself with the systems being of types 0, 1, 2, ... x; mapped to s_x^{n-x}, s_{x-1}^{n-x}, ... , s_m^{n-x}, ... s_0^{n-x} (s_x^{n-x} as the center, and so forth).
- The number of s_m^{n-x} type systems in this space is given by
$$\alpha^x_{s_m^{n-x}} = 2^m \cdot {}^xC_m.$$
- For an m-centric system with ambient dimension reduced from n to $n-x$, we utilize a direct mapping previously established:
$$\alpha_{t+m}^{n-x} = 2^t \cdot {}^{n-x}C_t,$$
where $0 \leq t < n-x$. Note that, since this is an m-centric system, the subscript of α involves the sum $t+m$. Thus, to obtain the count for a specific point type $p = t+m$, we rearrange to $t = p-m$ and apply this in the context of the reference system s_0.
- Consequently, for a fixed ambient dimension n and a chosen perspective x, we compute the number of points of type m present as:
$${}^x\alpha_m^n = \sum_{i=1}^{x} 2^x \cdot {}^xC_i \left[2^{m-(x-i)} \cdot {}^{n-x}C_{m-(x-i)}\right].$$
This expression characterizes the distribution of type-m points in the original n-dimensional space from the viewpoint of the perspective x.
- In our analysis, self-interactions (i.e., of the center point with itself) are excluded. Therefore, when calculating valid interactions for points of type x, we subtract a factor of 1 to correct for the center-center count. This adjustment is formally represented via the combinatorial term ${}^0C_{x-m}$, which equals 1 if $x = m$ and 0 otherwise.
- Incorporating this correction leads us to the final formula for the number of interactions:
$${}^x\alpha_m^n = \left[\sum_{i=1}^{x} 2^x \cdot {}^xC_i \left(2^{m-(x-i)} \cdot {}^{n-x}C_{m-(x-i)}\right)\right] - {}^0C_{x-m}.$$

Step 4: Elimination of Point Types For a topology of dimension m, all point types less than $n-m$ are excluded:

$$\text{Remove types } 0 \to (n-m-1).$$

Step 5: Contribution Formula (χ) The contribution of a single point of type t to the connectivity factor is:

$$^{m}\chi_t^n = \frac{\sum_{i=n-m}^{n} {}^{t}\alpha_i^n}{3^n - 1}, \quad \text{where } n-m \le t \le n.$$

Step 6: Frequency Formula (f_t) To compute the average connectivity, the frequency of remaining point types f_t within the idealized structure is needed.
Calculation of equivalents (e_t):

$$e_t = \frac{\text{Number of type } t \text{ points under fixed perspective } (x=0)}{\text{Number of hypercubes the point } t \text{ contributes to } (x=t)},$$

which simplifies to

$$e_t = \begin{cases} \frac{^{0}\alpha_t^n}{^{t}\alpha_0^n} & 0 < t \le n \\ 1 & t = 0 \\ \text{Invalid} & \text{otherwise} \end{cases} = {}^{n}C_t.$$

The frequency of points of type t is then

$$\boxed{f_t = \frac{^{n}C_t}{\displaystyle\sum_{i=n-m}^{n} {}^{n}C_i}}.$$

Step 7: Final Upper Bound Formula The overall upper bound on the connectivity factor is the weighted average of contributions, weighted by frequencies:

$$\boxed{^{U}\mathcal{CF}_m^n = \sum_{t=n-m}^{n} f_t \cdot {}^{m}\chi_t^n}.$$

We now propose two methods for theoretical estimation of intrinsic dimension, namely the LMU-CF (LMU Flag based CF) and the DCF (Distributed CF).

3.4 LMU Flag Overlaps and Influence

With the introduction of the Lower (L), Most Probable (M), and Upper (U) flags on the Connectivity Factor (CF) scale, one might initially consider using the L and U flags as strict boundaries for inferring topological dimension. However, these bounds exhibit significant overlaps that complicate such direct interpretations.

Illustrative Examples:

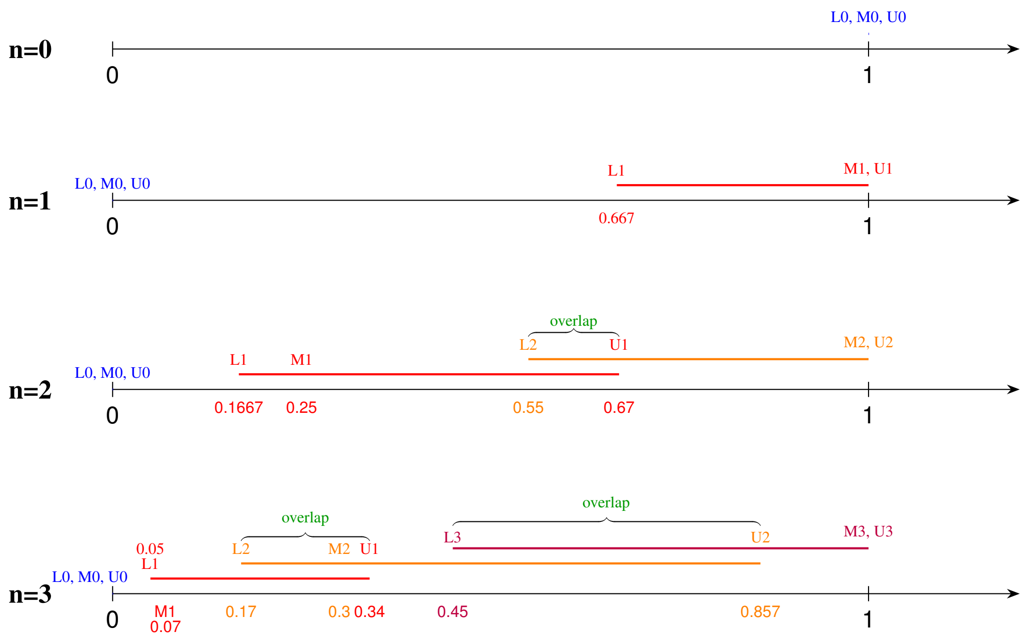


Figure 1: Ranges and potential overlaps of the Lower (L), Most Probable (M), and Upper (U) Connectivity Factor (CF) values for various topological dimensions within an n-dimensional ambient space (where Xk denotes the flag for embedded dimension k).

It is therefore not advisable to interpret these ranges independently as strict bounds, given the significant degree of overlap. The Lower and Upper flags represent extreme cases of the topological dimension and hence should be rare in typical datasets.

When ambiguity arises, the Most Probable (M) flag provides the most reliable estimate for the topology of a structure, with its degree of alignment depending on the curvature and complexity of the underlying manifold.

Consequently, a probabilistic influence function that integrates the Lower, Most Probable, and Upper flags is required. This function compares the influence of all possible topologies at a given CF value and assigns the topology with the highest inferred probability.

LMU Influence An influence function is thus formulated to estimate the topological dimension probabilistically based on the CF value of a point set. Key characteristics of this function include:

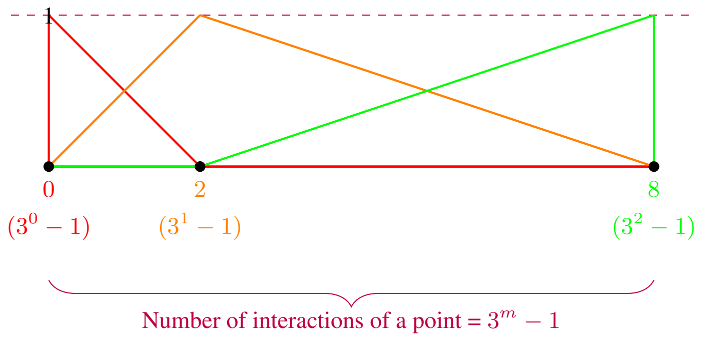
- It accepts as input a scalar CF value.
- The output is a vector of influences over all candidate topologies, typically of dimension $n + 1$, reflecting the probability or degree of membership of the structure in each possible topological dimension.
- Common implementations of such influence functions include triangular cap functions, Gaussian distributions, or learned weights via machine learning techniques.
- Aggregating the influence vector over all points allows computation of a final topological dimension estimate for the dataset.

3.5 Distributed Connectivity Factor (DCF)

The Distributed Connectivity Factor (DCF) method acknowledges that individual points may not be purely of a single topological dimension but instead contribute fractionally to multiple dimensions. This framework models each point as having a fractional membership across different topologies.

Fractional contributions are computed using a cap function, which bounds the influence of a point toward each topology based on its local connectivity. For simplicity and interpretability, a linear cap function is often employed.

Example: $n = 2$



$\rightarrow$ Based on this function, we determine each point's fractional contribution to each topology.

For instance, in a two-dimensional ambient space ($n = 2$), a point with $x = 3$ interactions results in fractional memberships:

$$1 \rightarrow 0.833 \quad \text{(83.3\% contribution to 1-D topology)},$$
$$2 \rightarrow 0.167 \quad \text{(16.7\% contribution to 2-D topology)}.$$

Thus, the fractionally assigned memberships reflect the mixed topological nature of the point.

Aggregating this fractional influence $f(x, t)$ over all points and normalizing enables a probabilistic estimation of the dataset's topological structure.

3.5.1 Fraction Vector Function F

Define the vector-valued fractional membership function for any point as:

$$F(x, n) = \begin{bmatrix} f_n(x, 0) \\ f_n(x, 1) \\ \vdots \\ f_n(x, n) \end{bmatrix},$$

where n is the ambient dimension and x is the absolute neighbor count. Each component $f_n(x, t)$ is a cap function corresponding to topology t.

The aggregate influence vector for the set S is thus:

$$\vec{I}(S, n) = \frac{1}{|S|} \sum_{x \in S} F(x, n).$$

This vector encodes the relative prominence of each topological dimension within the dataset.

3.6 Empirically-weighted Distributed Connectivity Factor (eDCF)

The eDCF method extends the Distributed Connectivity Factor (DCF) framework by incorporating an empirically generated reference model for neighbor counts, improving robustness and accuracy under real-world noise and finite sampling conditions.

3.6.1 Relation to DCF

Definitions and notation for intrinsic dimension, neighbor counts, and point-wise connectivity contributions remain unchanged from DCF. The eDCF replaces the fixed theoretical bounds with empirically derived neighbor count references obtained from an offline calibration procedure. This calibration uses synthetic datasets sampled from n-dimensional hyperspheres with comparable size and noise as the input data to accurately model expected neighbor counts r_t for each intrinsic dimension t.

3.6.2 Empirical Reference Model Generation

For each $t \in [1, D_{\max}]$, a synthetic t-dimensional hypersphere is sampled with the same point count and noise level as the dataset. The average neighbor count r_t is computed by pairwise comparisons within a grid of spacing ϵ, and then taking their mean:

$$\text{For } i = 1 \dots N, \quad \text{for } j = i+1 \dots N :$$
$$\text{Increment counts if } \|x_i - x_j\|_\infty \leq \epsilon \text{ and } i \neq j.$$

We then take the mean to get r_t. These r_t values form an empirical lookup that replaces theoretical bounds for connectivity-based dimension estimation. A map is made from {number of points, noise, dimension} to {average neighbor count}, which is then stored to act as a look-up table for future runs. By caching this way, and creating buckets for values (such as number of points), we can reduce the size of this map while maintaining accuracy.
To pass the value of noise to the reference model generator, we require the amount of noise in the dataset, which we obtain using fast graph-based denoising (FGBD) by <reference> (the time taken for noise estimation is negligible).

3.6.3 Weighted Membership Assignment

Using the empirical counts r_t, eDCF employs triangular cap functions $f_t(x)$ as:

$$f_t(x) = \begin{cases} 0, & x \notin [r_{t-1}, r_{t+1}] \\ \dfrac{x - r_{t-1}}{r_t - r_{t-1}}, & x \in [r_{t-1}, r_t] \\ \dfrac{r_{t+1} - x}{r_{t+1} - r_t}, & x \in (r_t, r_{t+1}] \\ 1, & x = r_t \text{ when } r_{t-1} = r_t \text{ or } r_t = r_{t+1} \end{cases},$$

where $r_{-1} = -\infty$, $r_{D_{\max}+1} = +\infty$.

Each point's absolute neighbor count c_i is converted into a fractional membership vector $F(c_i) = [f_0(c_i), \dots, f_{D_{\max}}(c_i)]^\top$.

3.6.4 Intrinsic Dimension Estimation

The dataset's influence vector $I(S, d)$ is defined as the average of the fractional membership vectors $F(c_i)$ over all points $x_i \in S$, where c_i is the observed neighbor count of the point. Formally,

$$I(S, d) = \frac{1}{|S|} \sum_{i=1}^{|S|} F(c_i) = \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_d \end{bmatrix},$$

where

- S is the set of sampled points,
- $|S|$ is the number of points,
- $d \leq n$ is the maximum intrinsic dimension considered,
- c_i is the absolute neighbor count of the i-th point,
- $F(c_i) = [f_0(c_i), f_1(c_i), \dots, f_d(c_i)]^\top$ is the fractional membership vector for point i,
- I_t is the empirical average membership to topology dimension t over the entire dataset.

The estimated intrinsic dimension $\hat{d}$ is then computed as the weighted average of all possible dimensions, weighted by their respective average memberships:

$$\hat{d} = \frac{\sum_{t=0}^{d} t \cdot I_t}{\sum_{t=0}^{d} I_t}.$$

This weighted average captures the expected intrinsic dimension by aggregating contributions from all candidate dimensions according to their influence on the data.

3.6.5 Computational Complexity and Optimizations

The original theoretical bounds and neighbor count calculations exhibit runtime complexity on the order of $O(N^2 d)$, where N is the number of points and d the ambient dimension.

Introducing the empirical reference model adds an amortized overhead for computing neighbor counts across multiple intrinsic dimensions $D_{\max}$. However, this cost is amortized across multiple runs by caching reference values keyed on intrinsic dimension, point count, and noise level.

Further computational improvements include:

- Highly parallelized pairwise neighbor calculations, yielding effective runtime

$$T_{\text{total}}^{\text{pair}}(N, d, D_{\max}, k) \;=\; \Theta\!\left(\frac{N^2\, D_{\max}^2}{k} \;+\; \mathbf{1}_{[d\le 3]}\,\frac{N\, 3^d}{k} \;+\; \mathbf{1}_{[d>3]}\,\frac{N^2\, d}{k}\right)$$

  with k processors.

- Potential replacement of brute-force neighborhood counts with KD-tree methods, lowering complexity to approximately:

$$T_{\text{total}}^{\text{kd}}(N, d, D_{\max}, k) \;=\; \Theta\!\left(\frac{N \log N\, D_{\max}^2}{k} \;+\; \mathbf{1}_{[d\le 3]}\,\frac{N\, 3^d}{k} \;+\; \mathbf{1}_{[d>3]}\,\frac{N \log N\, d}{k}\right)$$

- Bucketing and approximate caching strategies to generalize the empirical lookup, eventually reducing amortized cost to:

$$T_{\text{total, cached}}^{\text{kd}}(N, d, k) \;=\; \Theta\!\big(\mathbf{1}_{[d\le 3]}\,\tfrac{N\, 3^d}{k} \;+\; \mathbf{1}_{[d>3]}\,\tfrac{N \log N\, d}{k}\big)$$

- Note: For high values of d, the method reduces back to brute-force search due to KD-Tree limitations, and has a time complexity of:

$$T_{\text{total, cached}}^{\text{pair}}(N, d, k) \;=\; \Theta\!\big(\mathbf{1}_{[d\le 3]}\,\tfrac{N\, 3^d}{k} \;+\; \mathbf{1}_{[d>3]}\,\tfrac{N^2\, d}{k}\big)$$

- Since most work happens in higher ambient dimensions, usually the brute-force degraded form applied, as does in state-of-the-art methods such as TwoNN and MLE.

3.6.6 Adaptive Target Scaling Heuristic

To better capture neighborhood statistics in higher ambient dimensions, we adopt an adaptive target percentage heuristic:

$$\text{Target Information Percentage} = \min\big(95.0, \text{base\_target} + 3\sqrt{n}\big)\,,$$

where n is the ambient dimension and base_target is a tunable hyperparameter (in practice, we have observed that 50 % works best for a wide variety of cases).

This empirically improves neighborhood coverage, particularly in higher-dimensional settings. We do note that this is just a heuristic and is not necessary, nor is it optimal and can be further improved.

4 Results

4.1 eDCF results on Benchmark Manifolds

We compare TwoNN and MLE against our method (eDCF) on benchmark manifolds provided in the scikit-dimension (skdim) library, which generates a commonly used benchmark set of synthetic manifolds with known intrinsic dimension described by Hein et al. and Campadelli et al. The TwoNN and MLE implementations used are also from the skdim library. We compare the three methods using Mean Absolute Error (MAE), Mean Signed Error and Accuracy.

For eDCF, we use a value of 50 % on the IP scale. We use manifolds with 1 % noise, 10 % noise and 30 % noise.

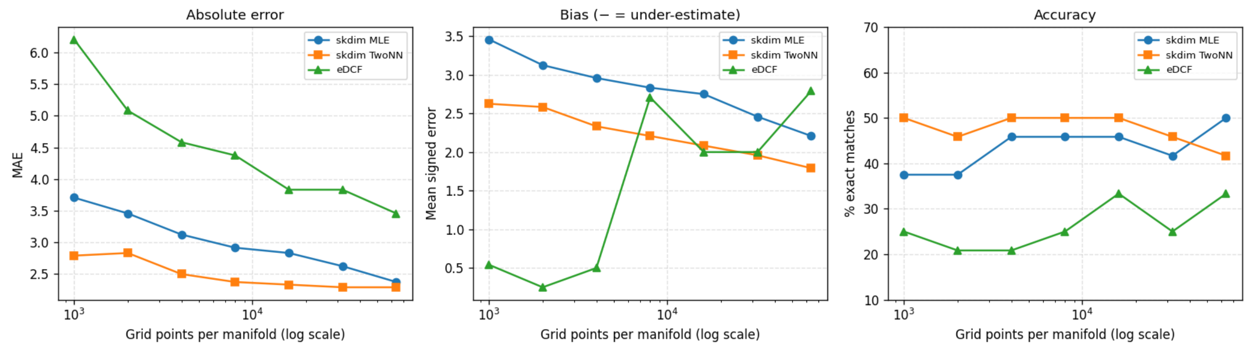


Figure 2: The figure describes MAE, Mean Signed Error and Accuracy against number of points per manifold (log scale) for TwoNN, MLE and eDCF on 24 Benchmark Manifolds with 1 % noise.

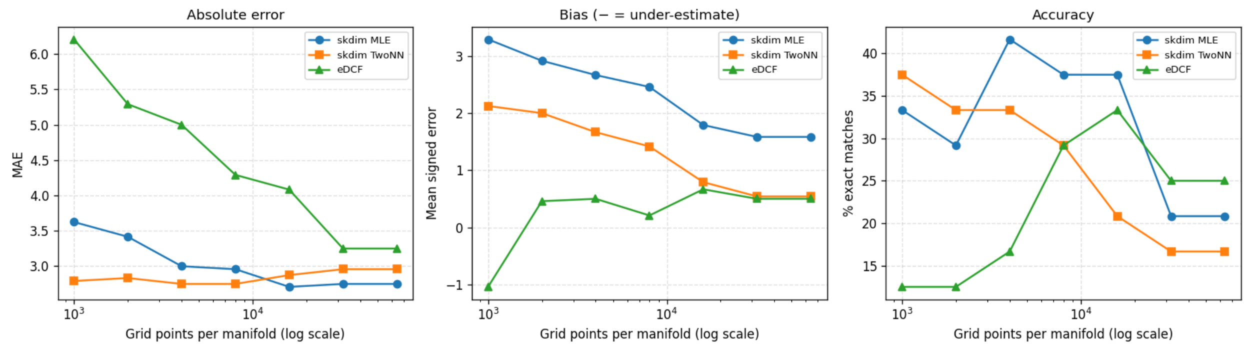


Figure 3: Comparison of TwoNN, MLE and eDCF on Benchmark Manifolds with 10 % noise.

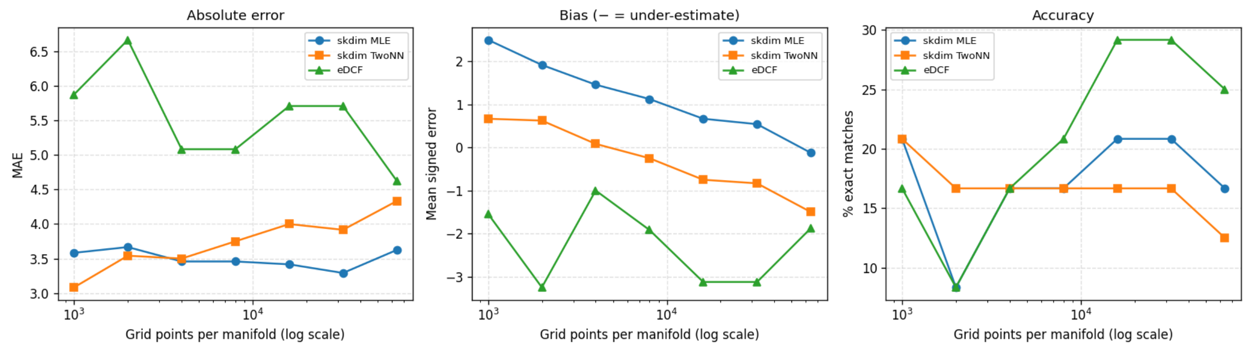


Figure 4: Comparison of TwoNN, MLE and eDCF on Benchmark Manifolds with 30 % noise.

Across all noise levels, eDCF's performance tightens with increasing data: its error decreases and the percentage of exact dimension estimates rises as sample size grows. At 1% noise, eDCF's mean absolute error (MAE) drops markedly from **6.208** (at 1k samples) to **3.458** (at 64k), while exact matches increase from **16.7%** to **25.0%**—peaking at **29.2%** in the 16k–32k regime. Its bias (signed error) slightly increases from **+0.542** at 1k to **+2.792** by 64k.

At the intermediate noise setting (10% noise), eDCF contracts MAE from **6.208** to **3.250** between 1k and 64k samples, and its exact-match rate rises steadily from **12.5%** (1–2k) to **25.0%** (32k–64k), with a maximum of **33.3%** at 16k. The signed error remains small and moves toward zero ($\mathbf{-1.042}$ at 1k to $\mathbf{+0.500}$ at 64k), indicating well-balanced estimation in large samples.

For 30% noise, eDCF maintains improvement in MAE with sample size (**5.875** at 1k to **4.625** at 64k) and its exact-match rate closely follows the trends seen at lower noise (**16.7%** to **25.0%**, peaking at **29.2%**), though the signed error remains moderately negative.

We note that while eDCF's MAE is somewhat higher than that of classic baselines such as skdim MLE and TwoNN (e.g., at 64k: **2.375/2.292** at 1% noise; **2.750/2.958** at 10%; **3.625/4.333** at 30%), eDCF frequently outperforms both in exact dimension recovery for large N, especially in moderate to high noise. For example, at 64k points and 10–30% noise, eDCF achieves **25.0%** exact matches, versus MLE's **20.8%** and TwoNN's **16.7–12.5%**. For eDCF, as sample size increases, its precision in identifying the correct dimension improves - even when its average error remains marginally above the baseline methods.

4.2 DCF results on Synthetic Data

The following results are for the DCF framework, with high number of points in the dataset and boundary, thus fitting the criteron for DCF usage. We run KNN on the Concentric Circles dataset (CCD), Decision Tree on Overlapping Concentric Circles dataset (OCCD), KNN on Barnsley Fern (BF) dataset, and Decision Tree on Sierpinski Carpet (SC) dataset. All dataset generation details and further experiments are provided in the appendix.

4.2.1 KNN:

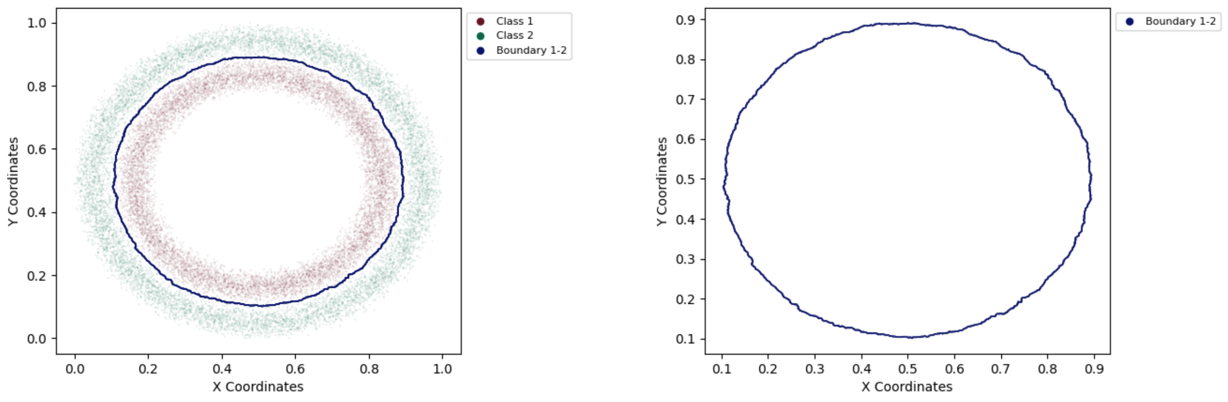


Figure 5: Plots of KNN on CCD Dataset corresponding to Table 5

Fractal Dimension (Boundary)	1.0306
Connectivity Factor (Boundary)	0.3968
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0000
Weight (Topology 1, Boundary)	0.8042
Weight (Topology 2, Boundary)	0.1957
Fractal Dimension (Object 1)	1.7473
Fractal Dimension (Object 2)	1.7120

Table 1: KNN boundary characteristics results

4.2.2 Decision Tree:

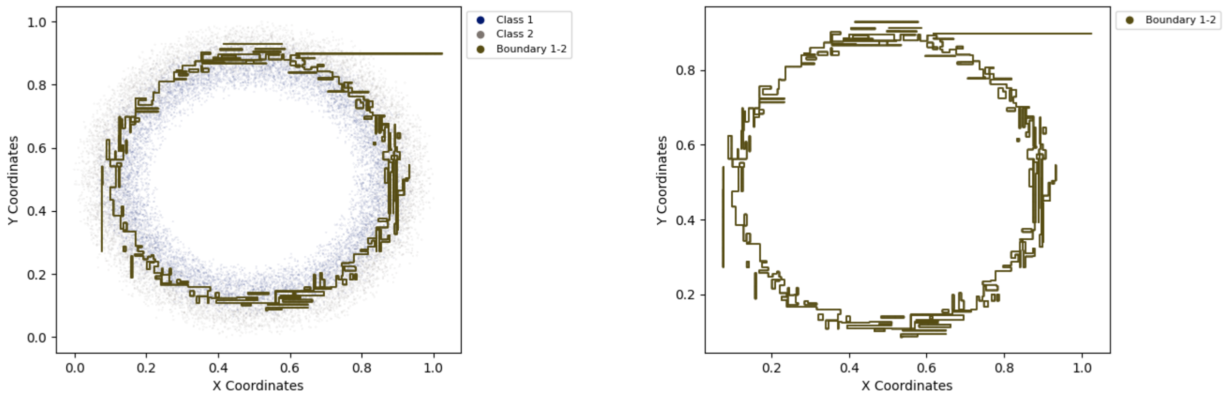


Figure 6: Plots of Decision Tree on OCCD Dataset corresponding to Table 8

Fractal Dimension (Boundary)	1.4441
Connectivity Factor (Boundary)	0.2713
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	4.7567
Weight (Topology 1, Boundary)	0.9715
Weight (Topology 2, Boundary)	0.0284
Fractal Dimension (Object 1)	1.7625
Fractal Dimension (Object 2)	1.7662

Table 2: Boundary characteristics results

4.2.3 KNN:

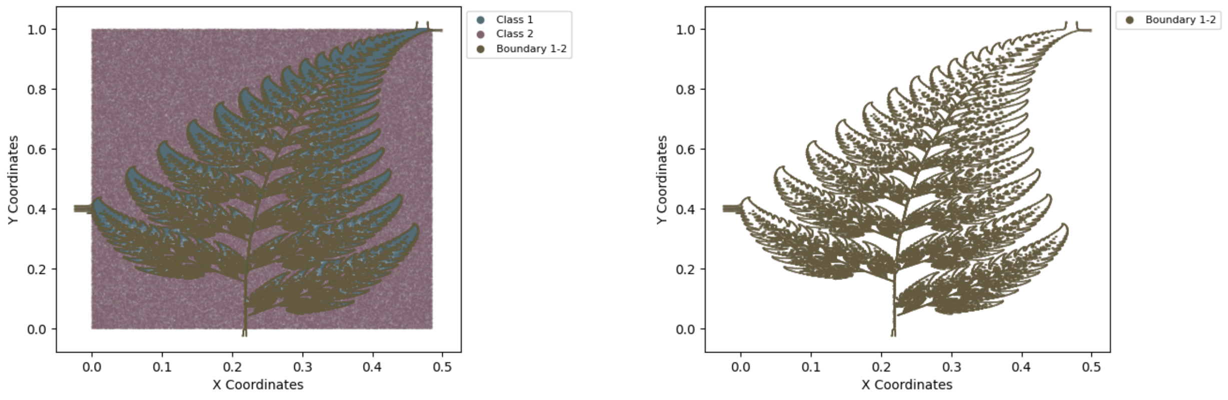


Figure 7: Plots of KNN on BF Dataset corresponding to Table 15

Fractal Dimension (Boundary)	1.7930
Connectivity Factor (Boundary)	0.4325
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	5.1683
Weight (Topology 1, Boundary)	0.7565
Weight (Topology 2, Boundary)	0.2434
Fractal Dimension (Object 1)	1.8340

Table 3: KNN boundary characteristics results

4.2.4 Decision Tree:

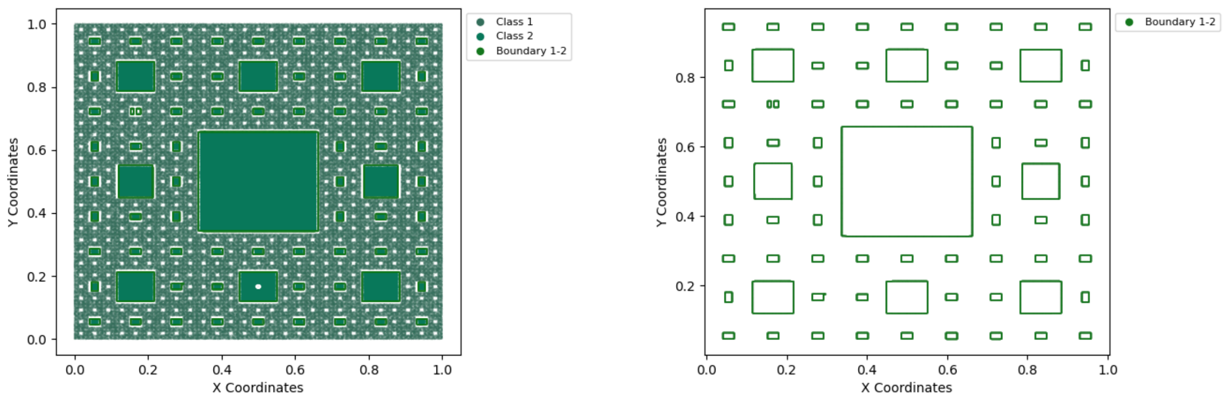


Figure 8: Plots of Decision Tree on SCD Dataset corresponding to Table 17

Fractal Dimension (Boundary)	1.0188	Weight (Topology 0, Boundary)	0.0000
Connectivity Factor (Boundary)	0.2543	Weight (Topology 1, Boundary)	0.9941
Topological Dimension (CF Based)	1	Weight (Topology 2, Boundary)	0.0058
Topological Dimension (Weight Based)	1	Fractal Dimension (Object 1)	1.8967

Table 4: Boundary characteristics results

Across datasets like Concentric Circle Data, Overlapping Circles, Barnsley Fern, and Sierpinski Carpet, the estimated boundary topological dimension was consistently correctly estimated as 1. Fractal dimensions of boundaries ranged from about 1.0 to 1.79, with connectivity factors between roughly 0.25 and 0.43. Weighted topology contributions showed a dominant one-dimensional membership with smaller fractions attributed to zero- and two-dimensional topologies. For fractal data, fractal dimensions aligned with known fractal characteristics, mostly between 1.6 and 1.9. Boundary plots confirmed effective identification of data boundaries consistent with the underlying dataset geometry. KNN CCD boundary was correctly identified as non-fractal, Decision Tree OCCD was identified as having some fractal nature due to inherent complexity and self similarity across sections of the boundary, KNN BF boundary was correctly identified as having fractal nature, and Decision Tree SC boundary was correctly identified and being non-fractal (while SC is a fractal, the decision boundary was too simple and not self similar to a great enough depth to truly be considered a fractal).

5 Discussion

The results demonstrate that the Empirically-weighted Distributed Connectivity Factor (eDCF) method provides a robust, scalable, and noise-resilient approach for intrinsic dimension estimation on diverse synthetic benchmark manifolds. Its design, centered on a grid-based neighbor framework and a novel Connectivity Factor formulation, enables efficient parallelization and scalability, distinguishing it from traditional distance-metric-dependent methods that face challenges in high-dimensional spaces.

A key factor underlying eDCF's robustness to noise is its reliance on local connectivity patterns rather than purely distance-based metrics. By discretizing space into uniform grid cells and defining neighbors through a fixed neighborhood structure, eDCF mitigates sensitivity to small perturbations in point locations caused by noise. Additionally, the empirical weighting mechanism calibrates neighbor contributions using reference models from synthetic noisy manifolds, allowing accurate fractional membership assignment under sampling variability and outliers. This contrasts with classical methods, such as TwoNN and MLE, whose dependence on nearest-neighbor distances can be disproportionately impacted by noise.

Quantitatively, eDCF exhibits consistent improvement with increasing sample size across all noise levels: mean absolute error (MAE) decreases and the proportion of exact dimension matches rises. At low noise (1%), the method achieves substantial reduction in error alongside increasing exact-match rates, indicating reliable convergence to true intrinsic dimensionality as data density grows. The signed error shows a mild positive bias at small sample sizes but stabilizes with larger samples.

At moderate noise (10%), the signed error approaches zero for large samples, signifying balanced and unbiased dimension estimation. The highest exact-match rate at intermediate sample sizes (around 16k) suggests an optimal scale where the local neighborhood structure is best captured, with diminishing returns beyond this point likely due to saturation or finite sampling effects.

Under high noise conditions (30%), eDCF maintains error reduction and consistent exact-match trends, though gains are more modest. The observed negative bias in signed error indicates a slight underestimation tendency, plausibly resulting from noise-induced disruption of local connectivity.

Comparisons with TwoNN and MLE reveal that despite occasionally higher MAE, eDCF consistently attains superior exact dimension recovery in large-sample, moderate-to-high noise regimes. This highlights eDCF's focus on precise identification of discrete dimensionality rather than minimizing average error alone, making it particularly suited for applications demanding reliable, topology-aware dimension inference.

Further, eDCF's dynamic neighborhood scaling via Information Percentage allows adaptive tuning to data scale without manual parameters, enhancing flexibility in heterogeneous, noisy environments.

For fractal analysis using DCF, as is seen in <Section 4.2>, it is accurate and reliable for low dimensional fractals.

5.1 Limitations

While the empirical results are encouraging, our study has several limitations.

1. **Discretization choices:** The grid discretization and the target Information Percentage (IP) act as structural hyperparameters. We did not present a full ablation over spacing schedules/IP targets, so sensitivity to those choices remains partially characterized.
2. **Empirical calibration:** eDCF relies on an empirical reference table (neighbor-count anchors across point counts/noise). Its fidelity depends on how well the synthetic calibration families match the data under study; domain shift or misestimated noise can bias assignments.
3. **Computation at high ambient dimension:** Although the workflow parallelizes well, neighbor counting can revert to $O(N^2 d)$ behavior in high d, with nontrivial memory pressure. We did not benchmark GPU/approximate variants, nor wall-clock vs. baselines.
4. **Scope of datasets/baselines:** Benchmark manifolds are synthetic; real-world evaluations are limited to illustrative boundary analyses (CCD/OCCD/BF/SC). Baselines focus on TwoNN and MLE; broader comparisons (e.g., DANCo, MiND/ESS, kNN-graph estimators) were not included.
5. **Bias behavior under noise:** While exact-hit rates generally increase with sample size across noise levels, signed-error trends can drift with noise and dataset scale. We did not analyze causes (e.g., density nonuniformity, curvature, class imbalance) in depth.

5.2 Future Work

We outline several directions that address the above limitations and extend applicability.

1. **LMU-CF:** Developing an appropriate influence function for the LMU Flag method; running benchmarks for it.
2. **Ablations and multi-resolution schemes:** Systematic sweeps over grid spacing/IP targets, plus multi-resolution ensembling (e.g., voting or stacking across IP scales) to reduce discretization bias.
3. **Adaptive, data-driven calibration:** Learn the membership caps and anchor counts directly from data via density-aware or noise-aware models.
4. **Scalability:** GPU implementations and approximate neighbor counting (e.g., LSH/IVF-PQ); streaming/online eDCF for evolving datasets; compressed caching for the calibration table.
5. **Broader benchmarks and baselines:** Real-world high-dimensional corpora (vision, audio, graphs) and additional ID estimators beyond MLE/TwoNN; stress tests on anisotropy, heavy-tailed noise, and strong density gradients.
6. **Calibration diagnostics:** Per-dataset reliability diagrams for signed error, bias–variance decompositions, and sample-size curves that contextualize when exact-hits overtake baselines.
7. **Boundary and application studies:** Extend boundary fractality analysis across modern classifiers (e.g., CNNs/transformers) and tasks; evaluate ties to generalization, drift/degradation monitoring, and fraud/outlier detection pipelines.

6 Conclusion

Across synthetic benchmark manifolds with 1–30% noise, eDCF exhibits a consistent empirical pattern: as sample size grows, mean absolute error decreases and the fraction of *exact* intrinsic-dimension hits rises, often rivaling or surpassing baselines at medium–large N, while typically posting slightly higher MAE than MLE/TwoNN overall. On a suite of constructed datasets (CCD/OCCD/BF/SC), the framework also supports boundary-centric analyses. Taken together, these findings position eDCF as a practical, scalable default when reliability and exact recovery at realistic data volumes are prioritized. The identified limitations—most notably discretization sensitivity, empirical calibration dependence, and high-d compute—suggest clear next steps: multi-resolution and adaptive calibration strategies, uncertainty quantification, real-world validation, and engineering for large-scale deployment.

Acknowledgments

This was was supported in part by......

References

[1] Isha Sehgal and K.S. Venkatesh. Connected component labeling for binary images. *International Journal of Advanced Research*, 2019.

[2] Francesco Camastra. Data dimensionality estimation methods: a survey. *Pattern Recognition*, 36:2945–2954, 2003.

[3] Gerard V. Trunk. Statistical estimation of the intrinsic dimensionality of a noisy signal collection. *IEEE Transactions on Computers*, C-25:165–171, 1976.

7 Appendix

7.1 Boundary Extraction

7.1.1 Informal Discussion

The BoundaryExtractor class that we build has the following functionalities:

- a method to enable us to get the boundary by using NCubeNeighbours 7.5 and multiprocessing using the labelled and divided grid data from GridGenerator 7.2.1. This method is able to get all types of boundaries which is to say if there is contact between any two structures from a list of structures then it will be able to get that.
- a method to call the mini grid method defined above for the extraction of complete boundary and all boundaries that can be detected.
- a method to fetch these boundaries from the file they are stored in.

7.1.2 Important Points to Note:

- This method of Boundary Extraction requires the points to be in a Grid System.
- For any two separate entity points which are next to each other, the boundary is assumed to lie somewhere on the line between the two points.
- The point which is chosen on the line is determined by a "weight" which is made such that when the weight is 0.0 then the structure lower in the lower-higher format is taken as the Boundary point. The maximum weight can be 1.0.
- We have taken weight as 0.0 to ensure the Boundary belongs to a Grid System which is required for further Analysis.
- Since we are using NCubeNeighbours 7.5 we are using the Formal Definition of Neighbours. Therefore, when we say two different entity points are next to each other, we mean they are Neighbours of each other.
- This Boundary Extraction is sensitive to the Grid Spacing value which is used to Generate the Grid in the first place.
- Proper Boundary extraction can be ensured if each point in the grid is labelled (at least the points where the boundary lies).

7.1.3 Benefits using this Boundary Extraction Method:

We are using a Grid Generator 7.2.1 which gives us a collection of points which are divided into multiple mini grids. If a mini grid Does not contain a lower and higher point set pair then we skip the boundary extraction of that particular pair for that grid which effectively reduces the amount of Neighbour Comparisons we have to make.

The other major reason we are using this algorithm is that this algorithm ensures that our boundary extracted is extracted without any errors introduced via dimensionality.

This algorithm is dividable to a large degree which makes it such that a larger core system can take advantage of the dividing nature of the algorithm computing the parts of the Boundary simultaneously.

A scan method of change of an entity in a direction would work but it would require parsing through points many times to get all points of the boundary and getting a grid boundary only while our method ensures that we can get a boundary with parsing each mini grid only once and it does not only give us grid points but allows us to check for a boundary in all grid directions which include the diagonals which is not possible in the scan method. This becomes increasingly apparant with increasing dimensions which have diagonals in increasing number.

7.2 Pre-processing for Boundary Extraction:

7.2.1 Grid Generator

Informal Discussion The GridGenerator class takes the trained algorithm hyper-parameters and the classification target data and generates a grid around the data points classifying each grid point and giving it the identity of the set it was classified to. We divide the full grid into multiple parts to enable multiprocessing. Arbitrary amount of divisions are possible. This is done throughout the entire n dimensional space that is required. Hence we have multiple classifications happening simultaneously. This Grid Division approach is also to facilitate Boundary Extraction thereafter.

We shall now see how the GridGenerator class provides:

- super loop which is a method to compute the partitioned boundaries for the n dimensional grid.
- mini grid compute allows us to classify the points via the provided trained algorithm.

Super Loop Algorithm The `super_loop` algorithm systematically partitions a high-dimensional grid into smaller, evenly divided mini-grids through recursive iteration across dimensions. This hierarchical division optimizes computational efficiency by enabling parallel processing and reducing memory usage. The algorithm facilitates scalable handling of complex, multi-dimensional data by breaking it into independently processable sections.

Divided Grid Generation Algorithm The `Divided Grid Generation` algorithm builds on the `super_loop` framework to manage mini-grid creation and processing. It calculates optimal subdivisions and spacing based on grid dimensions and desired detail while leveraging parallel processing to compute grid points efficiently across multiple CPU cores. The algorithm eliminates duplicate grid bounds to ensure unique mini-grids and maximizes resource utilization. The output is a labeled, comprehensive grid structure ready for analysis and visualization.

Integration and Workflow These algorithms form an integrated framework for efficient grid computation. The `super_loop` handles grid partitioning, while the `Divided Grid Generation` optimizes parallel processing and management. Together, they enhance scalability, computational performance, and flexibility across various grid sizes and dimensions.

7.3 Forced Grid Object

7.3.1 Informal Discussion

Force Grid Conversion is used to prepare the object point cloud for any evenly spaced (grid) operation such as finding out connectivity factor. This would work in n dimensions and uses two methods to find the force grid points.

These are the following functionalities provided by the class:

- the first and much less efficient algorithm uses the structure of GridGenerator 7.2.1. We generate the entire grid which makes hatching the object at the coarseness of the number of mini grid divisions possible. We thus get a hatch of the object which gives us an approximate solid filled view of the object which is not possible with the second algorithm. We use a modification which is that we only take those points into considerations which are in the boundary of the mini grid which reduces our number of comparisons for cases where there are no points in that grid and for cases where there are very few points there.

- the second algorithm is both faster and more memory efficient but does not produce a hatch of the object. In this algorithm, instead of generating a grid. We use an integer modulus based method to straight up grid the data without any comparison and store it in a set to ensure uniqueness. Since the data does not depend on any interactions among itself we can safely divide the data up into parts and use multiprocessing while taking the union of all sets which are the result of the computation.

7.3.2 Dynamic Spacing Adjustment

Dynamic Spacing Adjustment is a coarse to fine searching algorithm designed to effectively search for an Information Percentage bucket as specified.

$IP(\text{Information Percent}) = \frac{\text{Number of output data points}}{\text{Number of data points in original set}} \cdot 100$

At each resolution of Grid Spacing we will lose a certain amount of points since points may fall into overlapping buckets and the smallest IP is if all the points land into a single grid bucket and buckets cannot be fractional and thus the IP will only ever take certain values and the $\delta x =$ smallest possible division would lead to finer values as the number of points of the grid increases.

- We know that at a higher spacing we have maximum information loss and the thus only by decreasing the spacing can the IP be increased until we hit 100% after which any further decrement is unnecessary and redundant since the data will simply degrade to 0 dimensional as is the nature of point cloud data.
- Since we know the spacing and IP are monotonically related so we can perform a coarse search with a higher division factor such as 10 which would search first for spacing $1 \rightarrow 0.1 \rightarrow 0.01 \rightarrow \ldots$ in a normalized dataset till we find a spacing which either makes it so that we fall into the IP range that we desire or skip past the range.
- In case we fall into our IP range then we simply take that spacing and the grid data our analysis.
- If we exceed however, then we must perform a binary search on the current spacing that overshot the IP range and current spacing $\cdot$ 10 which undershot it.
- It is very well possible that the data is such that we cannot find the specific spacing which lands us into the desired IP range because of a variety of factors including if the IP range is too small which can be the most common reason.

7.4 Topological Dimension

7.4.1 Perspective Based Approach

Ideal mathematical objects like grids, 2D sheets, etc. are defined such that there is no effect of perspective on topological dimension since they are infinite in nature in terms of points. However we can neither generate no operate on infinite points. It is also impossible to come up with a general equation which identifies all irregular objects and in practice we usually want to find the topological dimension of the structure a point cloud data represents. This causes the topological dimension to be perspective based. Since we simply have points we cannot determine where a structure has a gap and where it does not without talking about the scale at which we are considering the structure. An example of this phenomenon is suppose the line of stars that we see here from earth. A line has a topological dimension of 1 but we know in reality that the line is not truly a line but stars which are very far apart. This can also be thought of as we have to consider the tolerance of continuation which means to say till what spacing do I consider the new point a part of a structure or a point in a new structure.

7.4.2 Role of Connectivity Factor

Connectivity Factor makes use of the fact that we need such a perspective based approach. $\mathcal{CF}$ of an object is calculated on the basis of the average connectivity of every point in the structure to it's neighbours and those neighbours are spacing based.

When we say a perspective based approach it means when we increase the spacing requirement for neighbour classification then we are essentially zooming out to consider points which might be very far apart to be considered part of the same structure such as the line of stars example. Similarly, when we decrease the spacing requirement for neighbour classification then we are essentially zooming in.

$\mathcal{CF}$ is an average value of the entire object and thus low to medium amount of noise or the present of outliers does not affect the data at all for high amount of correct points. That is to say if the noise is not enough to make, say a circle, a solid ring which is having topology 2 we would correctly be able to identify a circle as a 1D topological object.

7.5 N-Cube Neighbours

7.5.1 Informal Discussion

This section is about the construction of an NCubeNeighbours class which provides the following functionalities:

- get an N dimensional cubic space around a given point. If we want just a single point spacing from each axis as allowed then we get the neighbours of the central point provided. We have given the formal description of neighbours in section 3.1.
- find the neighbours in a given set through a tracking algorithm which essentially restricts the neighbours in terms of one axis and then proceeds to the next axis through boolean masking.
- move an NCubeNeighbours object thus the center point to another point thus instead preventing memory wastage. This is done by shifting of origin from center to another point.

7.5.2 Why use Sets when we have Distance for Neighbour Classification?(In Grid System)

In this section, we analyze why the use of distance metrics for defining neighbors fails in high-dimensional spaces, particularly beyond three dimensions. This limitation stems from the inability of a distance-based approach to clearly distinguish between neighbors and non-neighbors as dimensionality increases.

Distance-Based Neighbor Classification To examine this issue, we relax the constraints on the coefficients c_i, allowing $c_i \in \mathbb{Z}$ rather than $\{-1, 0, 1\}$. This generalization highlights the failure of distance-based metrics in distinguishing neighbors effectively.

1. **Maximum Distance of a Neighbor**: Consider a point where $c_1 = c_2 = c_3 = \cdots = c_n = 1$. The distance of this point from the origin **0** is given by:
$$d_{\text{max}} = \sqrt{n},$$
which represents the maximum distance of a neighbor.
2. **Minimum Distance of a Non-Neighbor**: Now consider a point where $c_2 = c_3 = \cdots = c_n = 0$ and $c_1 = 2$. The distance of this point from the origin is:
$$d_{\text{min}} = 2,$$
which represents the minimum distance of a non-neighbor.

Neighbor Classification Criteria For a distance metric to effectively classify neighbors, the maximum distance of a neighbor, $d_{\text{max}} = \sqrt{n}$, must be strictly less than the minimum distance of a non-neighbor, $d_{\text{min}} = 2$. This requirement translates into the inequality:

$$\frac{\sqrt{n}}{2} < 1.$$

Dimensional Analysis We analyze this inequality for various dimensions:

- **2D** ($n = 2$):
$$\sqrt{n} = \sqrt{2} \approx 1.414,$$
which satisfies the inequality $\frac{\sqrt{2}}{2} < 1$. Hence, the distance metric works correctly.
- **3D** ($n = 3$):
$$\sqrt{n} = \sqrt{3} \approx 1.732,$$
which also satisfies $\frac{\sqrt{3}}{2} < 1$, allowing correct neighbor classification.
- **4D** ($n = 4$):
$$\sqrt{n} = \sqrt{4} = 2,$$
which violates the inequality $\frac{\sqrt{4}}{2} < 1$. Consequently, extra points at a distance of 2 are incorrectly classified as neighbors.

- **Higher Dimensions** ($n > 4$): For $n > 4$, $\sqrt{n} > 2$, further invalidating the inequality. The number of misclassified points grows as n increases.

The analysis demonstrates that distance-based neighbor classification fails for dimensions greater than three. Beyond this threshold, the maximum distance of a neighbor ($\sqrt{n}$) exceeds the minimum distance of a non-neighbor (2), leading to ambiguity and misclassification. This issue highlights the limitations of using distance metrics in high-dimensional neighbor definitions and underscores the need for alternative approaches to accurately determine neighbors in such spaces.

7.5.3 Informal Discussion

Connectivity makes use of NCubeNeighbours 7.5 to get the neighbour set of a point and needs the system of points to be evenly spaced for computation which is achieved by 7.3. Also we can observe multiprocessing use in this case as well since points do not affect each other making parallel processing plausible.

The Connectivity class allows us the following functionalities:

- calculating the Connectivity Factor of the entire boundary system.
- calculating the Connectivity Factor for a general set of points.
- finding the adding potential of a point to a set which is to say that if we already know the connectivity factor of a set then adding that one point would change the connectivity by how much.

Infinite Limit of a Particular Topology m in a n dimensional space

Now let us imagine a scenario where we take the highest possible connections that the points can have in a particular topology of m dimension in a space which is n dimensional without any error points. Error points here refer to points which according to their position in the lattice we try to build have too few or too many connections.

Let us start with the simplest possible non-trivial dimension which is 1 dimensional space i.e. $n = 1$. Now for this space let us break down what is happening with the possible dimensions being $\{0, 1\}$.

For $n = 1$

- 0: Ideal scenario for a maximal 0 dimensional object is when the data is arranged in the line just so that for that spacing s we always get that each point has 0 neighbours. This is a trivial case and thus turns out that the connectivity factor is 0 when we take the limit of the average connectivity factor since every point is contributing 0 connectivity i.e. $\lim_{|S|\to\infty} \mathcal{CF}_0^1 = 0.0$.
- 1: Ideal scenario for a maximal 1 dimensional object is when the data is arranged in the line so that for that spacing s we always get that each point has the maximum neighbours possible which make it such that the point can be interpreted as 1D without ambiguity of it being a higher dimension. As it turns out that for 1D in 1D space there is no further analysis required as we can simply say that these data points must have 2 neighbours each since this is the highest dimension possible in this case and the most amount of neighbours a point can have which are actually present in the set itself are all of it's neighbours which are $3^1 - 1 = 2$ in this case. Thus contribution of a single point will be $\frac{2}{3^1-1} = 1$ which implies $\lim_{|S|\to\infty} \mathcal{CF}_1^1 = 1.0$.

Now let us analyse 2 dimensional space so that we may try to come to a generalization:

For $n = 2$

- 0: Ideal scenario for a maximal 0 dimensional object is when the data is arranged in the 2D grid just so that for that spacing s we always get that each point has 0 neighbours. This is a trivial case and thus turns out that the connectivity factor is 0 when we take the limit of the average connectivity factor since every point is contributing 0 connectivity i.e. $\lim_{|S|\to\infty} \mathcal{CF}_0^2 = 0.0$.
- 1: Ideal scenario for a maximal 1 dimensional object is when the data is arranged in the 2D grid so that for that spacing s we always get that each point has the maximum neighbours possible which make it such that the point can be interpreted as 1D without ambiguity of it being a higher dimension. Since now we are dealing with a 2D space we don't have the option of simply saying that an all neighbour contribution is okay obviously. Thus we now need a bit of rigor in how we define the points that we consider in the maximal set that we are trying to build.

We have to now revisit the Minimal Sets and Calculation for Lower Bounds of Spaces. Recall that we had specific types of points in there defined as a_0, a_1, a_2 for a 2D space atleast. Now consider what does an a_0 type point represent. It represents a point which is the central element since it has to have 0 amount of pivots and thus lies at the centre of the set in the minimal set that we are considering. Now let's analyse what the point a_1 represents. It represents the fact that we have a single pivot which can also mean that we are free to go in any 1 Direction that we want to since this pivot can be any one of the n axes that we have in the n dimensional space. We also have to consider what effects does the point a_2 simultaneously and how that is affected if we take a maximally filled space which is still 1D.

This turns out to be for Type $1(a_1) = 6$ and for Type $2(a_2) = 4$. We also note that these points occur in the ratio of 2:1. This causes our upper bound limit to come as $\lim_{|S|\to\infty} \mathcal{CF}_1^2 = \frac{2}{3} * \frac{6}{8} + \frac{1}{3} * \frac{4}{8} = 0.667$.

Now for 2D the upper limit is trivially $\lim_{|S|\to\infty} \mathcal{CF}_2^2 = 1.0$.

This way of calculating absolute upper bound tends to yield a result which is ambiguous since it can cause misinterpretations of data since it is an absolute edge case where we are extremely unsure of whether an object is even 1D in the first place; however, it tends to provide an anchor point for a probability function where the probability of the structure being m dimensional is practically 0.

A Deeper Look at Step 3: Expanded Explanation and Example This principle is the intellectual cornerstone of the entire derivation. The core idea is that by strategically changing our frame of reference, we can break down a complex n-dimensional analysis into a series of simpler, lower-dimensional analyses.

The Detailed Logic: Why Does the Dimension Reduce?
When we choose a point of type a_x as our perspective, we have effectively "fixed" or "accounted for" x of the n dimensions. The coordinates of our chosen point are no longer variables in the problem of finding its neighbors; they are constants that define our new reference frame. The task of finding the neighbors is now reduced to figuring out the possible coordinate combinations for the **remaining** $(n - x)$ **dimensions**. Because we only need to consider these remaining dimensions, the problem's complexity is now equivalent to that of a standard problem in an $(n - x)$-dimensional space. We have peeled away x layers of complexity by choosing a specific point of type a_x as our anchor.

Thorough Example 1: Reducing a 3D Problem to 2D
Let's walk through this process with a concrete example in a **3-dimensional space** ($n = 3$). A point in this space has $3^3 - 1 = 26$ neighbors. Our goal is to see how analyzing from the perspective of a **type $\boldsymbol{a_1}$ point** reduces the problem to a 2D system.

1. **Choose a Perspective Point:** A type a_1 point in 3D space has one non-zero coordinate. Let's choose the point $\mathbf{P} = (1, 0, 0)$. This is a "face-center" on the positive x-axis.
2. **Analyze the Neighbors of P:** The neighbors of P are all points $\mathbf{N} = (x, y, z)$ where each coordinate differs from P's by at most 1.
   - $x \in \{1-1, 1+0, 1+1\} \implies x \in \{0, 1, 2\}$
   - $y \in \{0-1, 0+0, 0+1\} \implies y \in \{-1, 0, 1\}$
   - $z \in \{0-1, 0+0, 0+1\} \implies z \in \{-1, 0, 1\}$

   The point $\mathbf{N} = (1, 0, 0)$ itself is excluded.
3. **The Crucial Insight - Identifying the Reduced System:** The y and z coordinates can independently take any value from $\{-1, 0, 1\}$. The problem of finding the valid combinations for (y, z) is **identical** to the problem of finding the neighbors of the origin $(0, 0)$ in a **2-dimensional (y,z) plane**. A 2D point has $3^2 - 1 = 8$ neighbors.
4. **Mapping the 2D Solution Back to 3D:** The 26 neighbors of our point P are found by combining the fixed x-dimension logic with the 2D (y,z) neighbor logic. This gives us three sets of neighbors for P, as visualized in Figure 9.

Connecting the Logic to the Formula:
The dimensional reduction method demonstrated in the examples leads directly to the components of the general formula. The formula is a combinatorial generalization of this "slicing" process.

- **Choosing a perspective point** like $\mathbf{P} = (1, 0, 0)$ is generalized to choosing a point of type $\boldsymbol{a_x}$.
- The process of **slicing the problem** (e.g., into planes at $x = 0, 1, 2$) is generalized by the summation $\sum_{\boldsymbol{i=0}}^{\boldsymbol{x}}$. This sum iterates over all possible ways the axes of the perspective point and a potential neighbor can overlap.

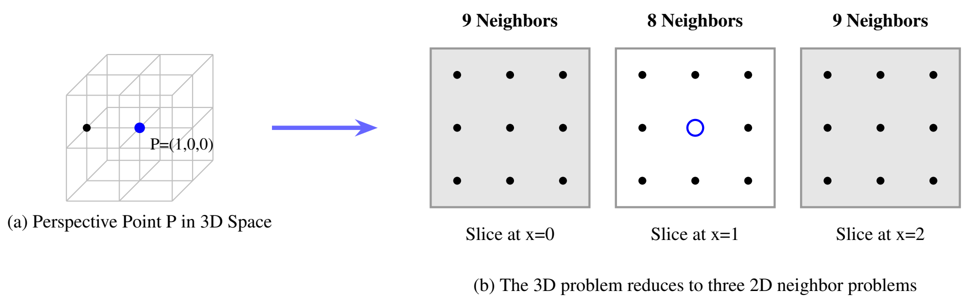

P=(1,0,0)
(a) Perspective Point P in 3D Space
9 Neighbors
8 Neighbors
9 Neighbors
Slice at x=0
Slice at x=1
Slice at x=2
(b) The 3D problem reduces to three 2D neighbor problems

Figure 9: Visualizing the 3D to 2D dimensional reduction. The neighborhood of point P is decomposed into three 2D slices.

- The term $\mathbf{2^i \cdot {}^xC_i}$ handles the combinatorics of the **fixed dimensions**. It counts the ways to choose which of the x fixed axes overlap with the neighbor's non-zero coordinates.
- The term $\mathbf{2^{m-(x-i)} \cdot {}^{n-x}C_{m-(x-i)}}$ handles the combinatorics within the **reduced dimension**. This is the mathematical equivalent of counting the neighbors within each slice.

Step 4: The Mathematical Interaction Formula ($^x\alpha^n_m$) This formula calculates the total number of connections that a single point of perspective-type *x* has with all points of another type *m*.

$$^x\alpha^n_m = \sum_{i=0}^{x} \left[2^i \cdot {}^xC_i \left(2^{m-(x-i)} \cdot {}^{n-x}C_{m-(x-i)}\right)\right] - {}^0C_{x-m} \tag{2}$$

NOTE: Consider $^nC_r = 0$ whenever n < 0 or r < 0 or n < r.

A Complete End-to-End Example: 1D Topology in 2D Space Let's calculate the upper bound for a 1D structure in a 2D space: $^U\mathcal{CF}^2_1$.

- Target Topology: $m = 1$
- Ambient Space: $n = 2$

1. Elimination:
The rule is to eliminate types from a_0 up to a_{n-m-1}. Here, $n - m - 1 = 2 - 1 - 1 = 0$. So, we must eliminate type $\boldsymbol{a_0}$. The **remaining point types** are $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$.

2. Interactions (from Step 4):
We will now use the interaction formula to calculate the total number of connections each remaining point type has with the other remaining point types.

Interactions from an a_1 perspective ($x = 1$): The total connections for an a_1 point is the sum of its connections to other a_1 points and to a_2 points: $^1\alpha^2_1 + {}^1\alpha^2_2$.

- **Connections to a_1 points ($m = 1$):**
$$\begin{aligned} ^1\alpha^2_1 &= \left[2^0 \cdot {}^1C_0 \cdot (2^{1-1} \cdot {}^1C_{1-1})\right] + \left[2^1 \cdot {}^1C_1 \cdot (2^{1-0} \cdot {}^1C_{1-0})\right] - {}^0C_0 \\ &= [1 \cdot 1 \cdot (1 \cdot 1)] + [2 \cdot 1 \cdot (2 \cdot 1)] - 1 = 1 + 4 - 1 = 4 \end{aligned}$$
- **Connections to a_2 points ($m = 2$):**
$$\begin{aligned} ^1\alpha^2_2 &= \left[2^0 \cdot {}^1C_0 \cdot (2^{2-1} \cdot {}^1C_{2-1})\right] + \left[2^1 \cdot {}^1C_1 \cdot (2^{2-0} \cdot {}^1C_{2-0})\right] - {}^0C_{-1} \\ &= [1 \cdot 1 \cdot (2 \cdot 1)] + [2 \cdot 1 \cdot (4 \cdot 0)] - 0 = 2 + 0 - 0 = 2 \end{aligned}$$

Total connections for an a_1 point = $4 + 2 = \mathbf{6}$.

Interactions from an a_2 perspective ($x = 2$): The total connections for an a_2 point is the sum of its connections to a_1 points and to other a_2 points: $^2\alpha^2_1 + {}^2\alpha^2_2$.

- **Connections to a_1 points ($m = 1$):**
$$\begin{aligned} ^2\alpha^2_1 &= [\dots]_{\text{i=0}} + \left[2^1 \cdot {}^2C_1 \cdot (2^{1-1} \cdot {}^0C_{1-1})\right]_{\text{i=1}} + [\dots]_{\text{i=2}} - {}^0C_1 \\ &= 0 + [2 \cdot 2 \cdot (1 \cdot 1)] + 0 - 0 = 4 \end{aligned}$$

- **Connections to a_2 points ($m = 2$):**

$$\begin{aligned} {}^2\alpha_2^2 &= \left[2^0 \cdot {}^2C_0 \cdot (2^{2-2} \cdot {}^0C_{2-2})\right] + [\dots]_{\text{i=1}} + [\dots]_{\text{i=2}} - {}^0C_0 \\ &= [1 \cdot 1 \cdot (1 \cdot 1)] + 0 + 0 - 1 = 0 \end{aligned}$$

Total connections for an a_2 point = $4 + 0 = \mathbf{4}$.

3. Contributions (from Step 6):
Now we calculate the CF contribution of each remaining point type. The total number of neighbors in 2D space is $3^2 - 1 = 8$.

- Contribution of an a_1 point: ${}^1\chi_1^2 = \frac{6}{8}$
- Contribution of an a_2 point: ${}^1\chi_2^2 = \frac{4}{8}$

4. Frequencies (from Step 7):
We calculate the frequency of the remaining points (a_1, a_2) in the idealized structure.

- Total non-eliminated point types for frequency calculation: ${}^2C_1 + {}^2C_2 = 2 + 1 = 3$.
- Frequency of a_1 points: $f_1 = \frac{{}^2C_1}{3} = 2/3$
- Frequency of a_2 points: $f_2 = \frac{{}^2C_2}{3} = 1/3$

5. Final Calculation (from Step 8):
Finally, we calculate the weighted average.

$$ {}^U\mathcal{CF}_1^2 = (f_1 \cdot {}^1\chi_1^2) + (f_2 \cdot {}^1\chi_2^2) = \left(\frac{2}{3} \cdot \frac{6}{8}\right) + \left(\frac{1}{3} \cdot \frac{4}{8}\right) = \frac{12}{24} + \frac{4}{24} = \frac{16}{24} = \frac{2}{3} \approx 0.667 \quad (3)$$

Justification of the Upper Bound for CF The upper bound ${}^U\mathcal{CF}_m^n$ represents the theoretical maximum Connectivity Factor for an idealized, non-fractal object of intrinsic dimension m embedded in an n-dimensional space. This bound assumes an infinite, fully occupied integer lattice where every point and all its neighbors exist, creating a perfect, maximally connected structure.

Point types a_t classify neighbors by their geometric position relative to the center, enabling a recursive decomposition into lower-dimensional subsystems. Eliminating point types incompatible with dimension m ensures the bound is restricted to the intended topology.

The bound is computed as a weighted average of connectivity contributions from valid point types, reflecting their true frequencies in the ideal lattice. Real-world samples, being finite, noisy, or fractal, exhibit fewer connections, making the upper bound unattainable in practice.

Thus, ${}^U\mathcal{CF}_m^n$ serves as a precise, mathematically grounded limit against which observed connectivity factors can be compared within the eDCF framework.

7.6 Dataset Generation Details

7.7 Concentric Circle Data (CCD)

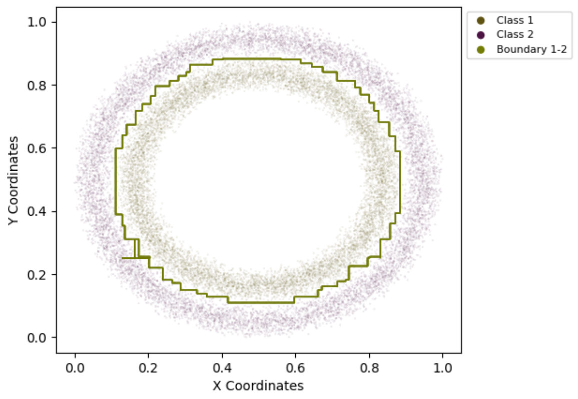
Dataset Creation Details: The Circle Dataset (CCD) generates data points arranged in concentric circular patterns by sampling angles uniformly around each circle and converting them to Cartesian coordinates. Each point's location is determined by its radius and center, and independent noise—controlled by a noise rate parameter—is added to both coordinates.

Table 5: CCD Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	**Value / Class / Formula**	**Description**
θ_{start}	0 radians	Start angle
θ_{end}	2π radians	End angle
θ (samples)	360 samples	Points per circle
radius	3.0 (Circle 1), 4.0 (Circle 2)	Circle radii
x_{center}	0.0	X-center
y_{center}	0.0	Y-center
noise_rate	0.5	Noise magnitude
Training	Circle 1 (1): 8,000 (40%)	Training data
	Circle 2 (2): 8,000 (40%)	
Testing	Circle 1 (1): 2,000 (10%)	Test data
	Circle 2 (2): 2,000 (10%)	
Point Generation	$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} n_x \\ n_y \end{pmatrix}$	Coordinate formula
Noise (x)	$n_x = \text{rand}() \cdot \text{noise\_rate} - \text{rand}() \cdot \text{noise\_rate}$	X-noise
Noise (y)	$n_y = \text{rand}() \cdot \text{noise\_rate} - \text{rand}() \cdot \text{noise\_rate}$	Y-noise

Note: rand() uses uniform distribution in $[0, 1)$

7.7.1 Decision Tree:



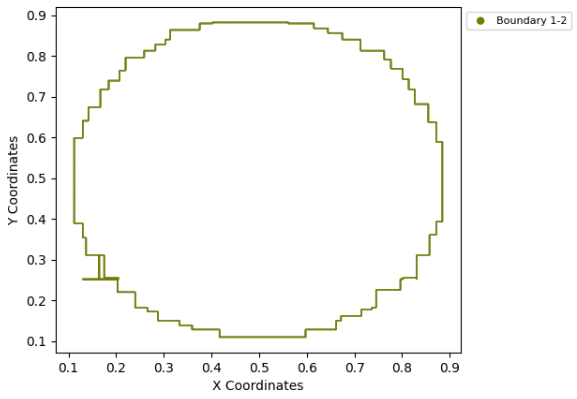


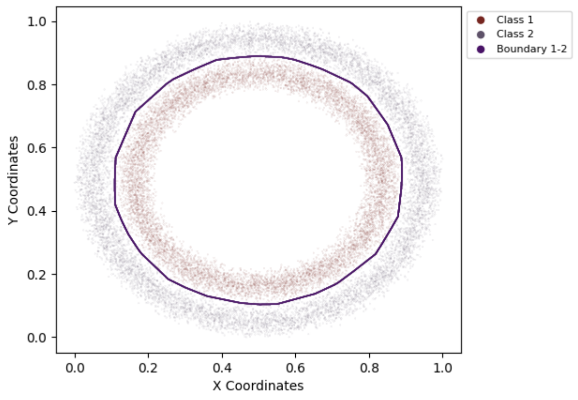
Figure 10: Plots of Decision Tree on CCD Dataset corresponding to Table 5

Fractal Dimension (Boundary)	1.0064
Connectivity Factor (Boundary)	0.2540
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0001
Weight (Topology 1, Boundary)	0.9944
Weight (Topology 2, Boundary)	0.0053
Fractal Dimension (Object 1)	1.7473
Fractal Dimension (Object 2)	1.7120

Table 6: Boundary characteristics results

7.7.2 MLP:



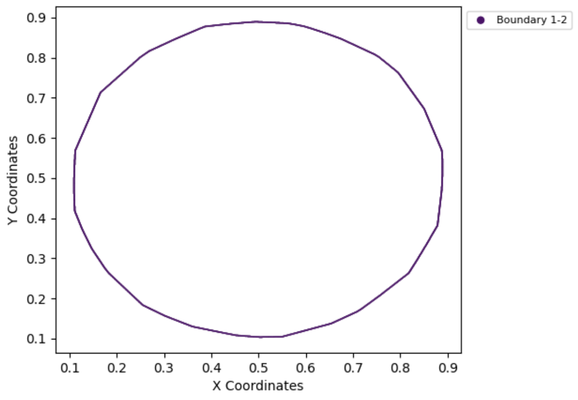


Figure 11: Plots of MLP on CCD Dataset corresponding to Table 5

Fractal Dimension (Boundary)	1.0332
Connectivity Factor (Boundary)	0.3949
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0000
Weight (Topology 1, Boundary)	0.8068
Weight (Topology 2, Boundary)	0.1932
Fractal Dimension (Object 1)	1.7473
Fractal Dimension (Object 2)	1.7120

Table 7: MLP boundary characteristics results

7.8 Overlapping Concentric Circle Data (OCCD)

Dataset Creation Details: The Overlapping Concentric Circle Dataset (OCCD) generates data points arranged in concentric circular patterns with intentional overlap and added noise. Points are sampled by selecting angles uniformly around each circle and converting them to Cartesian coordinates. Each point's position is determined by the specific radius and center of its circle. Independent noise—controlled by a noise rate parameter—is added to both the x and y coordinates, increasing the variability and overlap between circles.

Table 8: OCCD Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	**Value / Class / Formula**	**Description**
θ_{start}	0 radians	Start angle
θ_{end}	2π radians	End angle
θ (samples)	360 samples	Points per circle
radius	3.0 (Circle 1), 3.5 (Circle 2)	Circle radii
x_{center}	0.0	X-center
y_{center}	0.0	Y-center
noise_rate	0.7	Noise magnitude
Training	Circle 1 (1): 8,000 (40%)	Training data
	Circle 2 (2): 8,000 (40%)	
Testing	Circle 1 (1): 2,000 (10%)	Test data
	Circle 2 (2): 2,000 (10%)	
Point Generation	$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} n_x \\ n_y \end{pmatrix}$	Coordinate formula
Noise (x)	$n_x = \text{rand}() \cdot \text{noise\_rate} - \text{rand}() \cdot \text{noise\_rate}$	X-noise
Noise (y)	$n_y = \text{rand}() \cdot \text{noise\_rate} - \text{rand}() \cdot \text{noise\_rate}$	Y-noise

Note: rand() uses uniform distribution in $[0, 1)$

7.8.1 KNN:

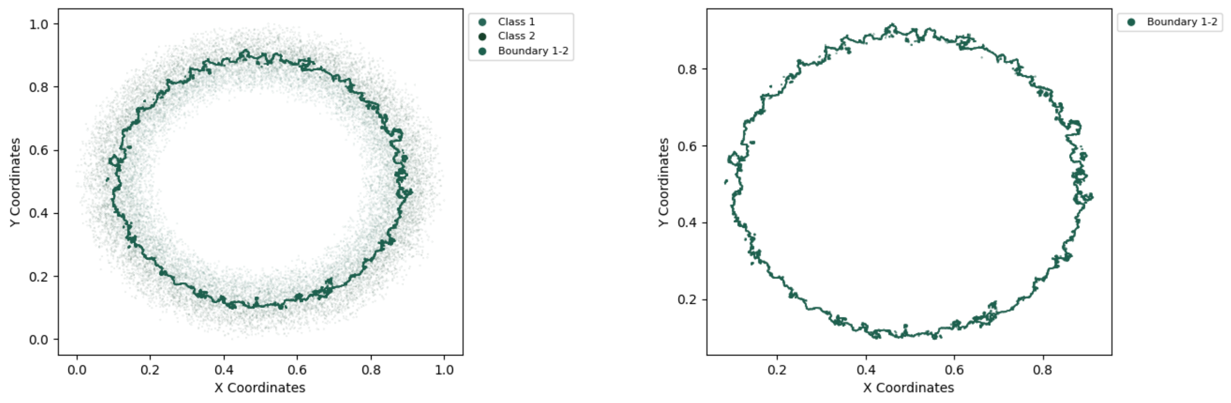


Figure 12: Plots of KNN on CCD Dataset corresponding to Table 8

Fractal Dimension (Boundary)	1.3106
Connectivity Factor (Boundary)	0.4254
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0106
Weight (Topology 1, Boundary)	0.7518
Weight (Topology 2, Boundary)	0.2374
Fractal Dimension (Object 1)	1.7625
Fractal Dimension (Object 2)	1.7662

Table 9: KNN boundary characteristics results

7.8.2 MLP:

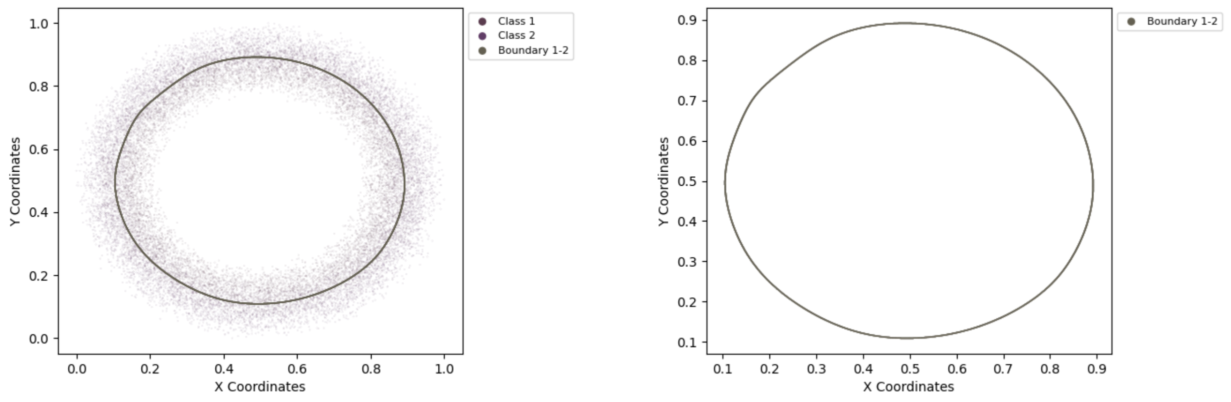


Figure 13: Plots of MLP on OCCD Dataset corresponding to Table 8

Fractal Dimension (Boundary)	1.0258
Connectivity Factor (Boundary)	0.3985
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0000
Weight (Topology 1, Boundary)	0.8018
Weight (Topology 2, Boundary)	0.1981
Fractal Dimension (Object 1)	1.7625
Fractal Dimension (Object 2)	1.7662

Table 10: MLP boundary characteristics results

7.9 Sinusoidal Curve Data (SD):

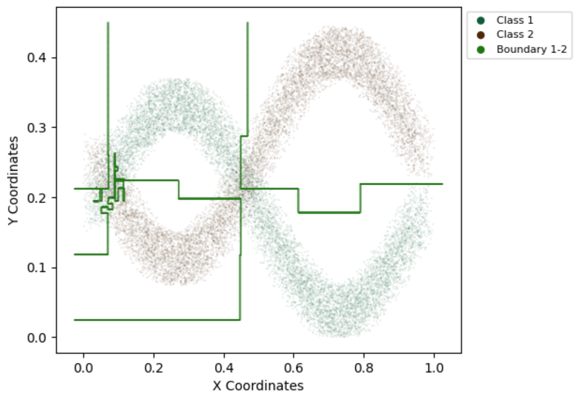
Dataset Creation Details: The Sinusoidal Curve Dataset (SD) with Mean and Phase Difference generates data points arranged along sinusoidal curves with identical amplitude but distinct phase differences and vertical offsets. Each curve has a unique class identity, a specified phase difference, a y-axis offset, and added noise. The sinusoidal curve is represented by the equation: $y = \text{amplitude} \cdot \sin(x + \text{phase\_difference}) + y_{\text{center}} + \text{noise}_y$

Table 11: SD Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	Value / Class / Formula	Description
x_{min}	0.0	Start of x-sampling interval
x_{max}	2π	End of x-sampling interval
x distribution	evenly spaced	Horizontal distribution along sinusoidal wave
amplitude	1.0 (both curves)	Peak vertical distance from center line
phase_difference	0.0 radians (Curve 1), π radians (Curve 2)	Horizontal shift of sinusoidal wave
y_{center}	0.0 (Curve 1), 0.5 (Curve 2)	Vertical offset along y-axis
noise_rate	0.5	Level of noise around each point's mean position
Training	Curve 1: 80%, Curve 2: 80%	Training samples per curve
Testing	Curve 1: 20%, Curve 2: 20%	Test samples per curve
Point Generation	$y = \text{amplitude} \cdot \sin(x + \text{phase\_difference}) + y_{\text{center}} + \text{noise}_y$	Sinusoidal curve equation
Noise (y)	$\text{noise}_y = \text{rand}() \cdot \text{noise\_rate} - \frac{\text{noise\_rate}}{2}$	Random noise added to y-coordinate

Note: rand() uses a uniform distribution in $[0, 1)$.

7.9.1 Decision Tree:



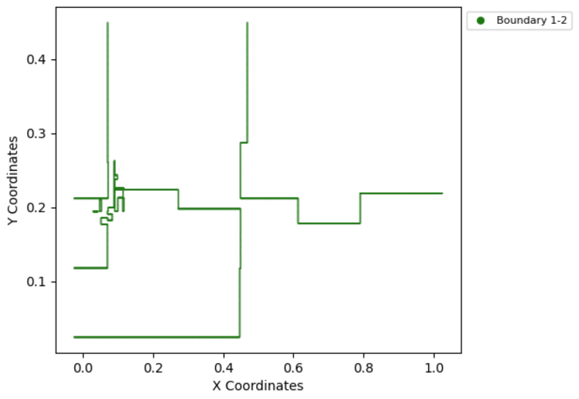


Figure 14: Plots of Decision Tree on SD Dataset corresponding to Table 11

Fractal Dimension (Boundary)	1.0319
Connectivity Factor (Boundary)	0.2523
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0005
Weight (Topology 1, Boundary)	0.9962
Weight (Topology 2, Boundary)	0.0032
Fractal Dimension (Object 1)	1.7631
Fractal Dimension (Object 2)	1.7348

Table 12: Boundary characteristics results

7.9.2 KNN:

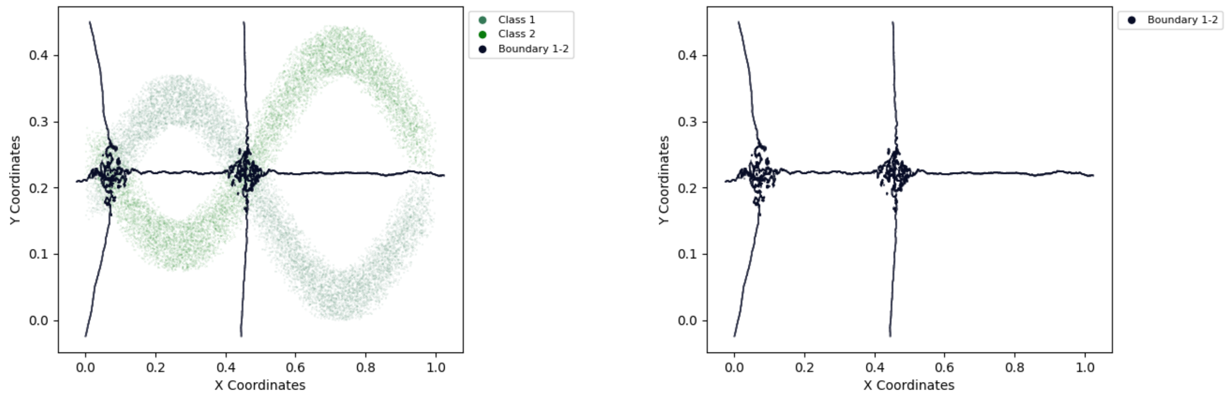


Figure 15: Plots of KNN on SD Dataset corresponding to Table 11

Fractal Dimension (Boundary)	1.2515
Connectivity Factor (Boundary)	0.4076
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0087
Weight (Topology 1, Boundary)	0.7780
Weight (Topology 2, Boundary)	0.2131
Fractal Dimension (Object 1)	1.7631
Fractal Dimension (Object 2)	1.7348

Table 13: KNN boundary characteristics results

7.9.3 MLP:

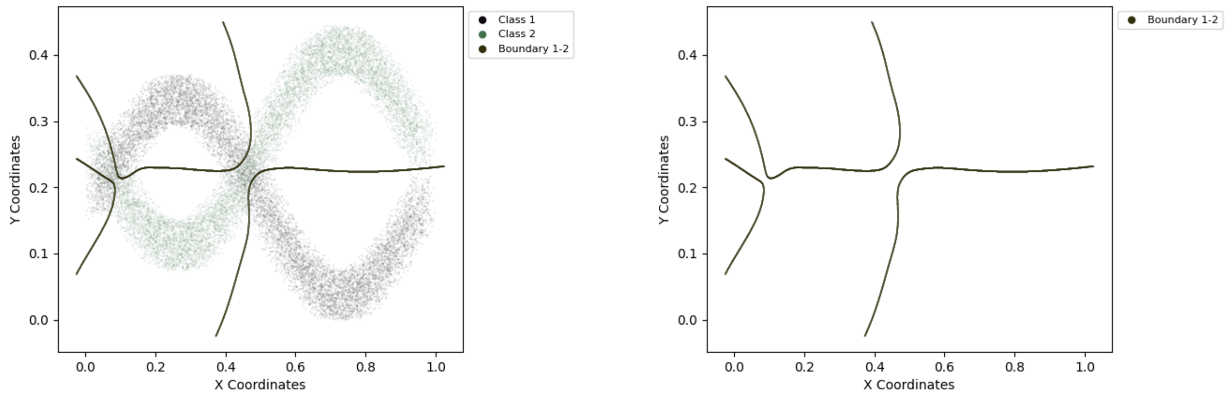


Figure 16: Plots of MLP on SD Dataset corresponding to Table 11

Fractal Dimension (Boundary)	1.0066
Connectivity Factor (Boundary)	0.3678
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0003
Weight (Topology 1, Boundary)	0.8423
Weight (Topology 2, Boundary)	0.1572
Fractal Dimension (Object 1)	1.7631
Fractal Dimension (Object 2)	1.7348

Table 14: MLP boundary characteristics results

7.10 Barnsley Fern Data (BF)

Dataset Creation Details: The Barnsley Fern Dataset (BF) generates data points that collectively form the fractal structure known as the Barnsley fern. Each point is produced iteratively using the Chaos Game method, which applies one of four affine transformations at each step. The transformation is chosen probabilistically, based on predefined probabilities, and the process starts from the initial point (0.0 , 0.0) (0.0,0.0). The dataset includes both fern points and randomly sampled non-fern points within the fern's bounding box, resulting in two classes.

Table 15: BF Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	**Value / Class / Formula**	**Description**
Initial Point	$(0.0, 0.0)$	Starting coordinates for the iterative process
Number of Fern Points	5,000,000	Total points generated for the fern structure (Class 1)
Number of Non-Fern Points	5,000,000	Random points within the fern's bounding box (Class 0)
Affine Transformations	Four, with coefficients: 1: $a = 0.00$, $b = 0.00$, $c = 0.00$, $d = 0.16$, $e = 0.00$, $f = 0.00$ 2: $a = 0.85$, $b = 0.04$, $c = -0.04$, $d = 0.85$, $e = 0.00$, $f = 1.6$ 3: $a = 0.20$, $b = -0.26$, $c = 0.23$, $d = 0.22$, $e = 0.00$, $f = 1.6$ 4: $a = -0.15$, $b = 0.28$, $c = 0.26$, $d = 0.24$, $e = 0.00$, $f = 0.44$	shape and branching of the fern
Transformation Probabilities	$p_1 = 0.01$, $p_2 = 0.85$, $p_3 = 0.07$, $p_4 = 0.07$	Probability of selecting each transformation
Training	Fern (1): 4,000,000 samples (40%) Non-Fern (0): 4,000,000 samples (40%)	Number and fraction of training samples for each class
Testing	Fern (1): 1,000,000 samples (10%) Non-Fern (0): 1,000,000 samples (10%)	Number and fraction of test samples for each class
Point Generation	$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$	Affine transformation applied at each iteration

Note: At each iteration, a random number in $[0, 1)$ determines which transformation is applied, according to the specified probabilities. The dataset comprises two classes: Class 1 (fern points) and Class 0 (random non-fern points).

7.10.1 Decision Tree:

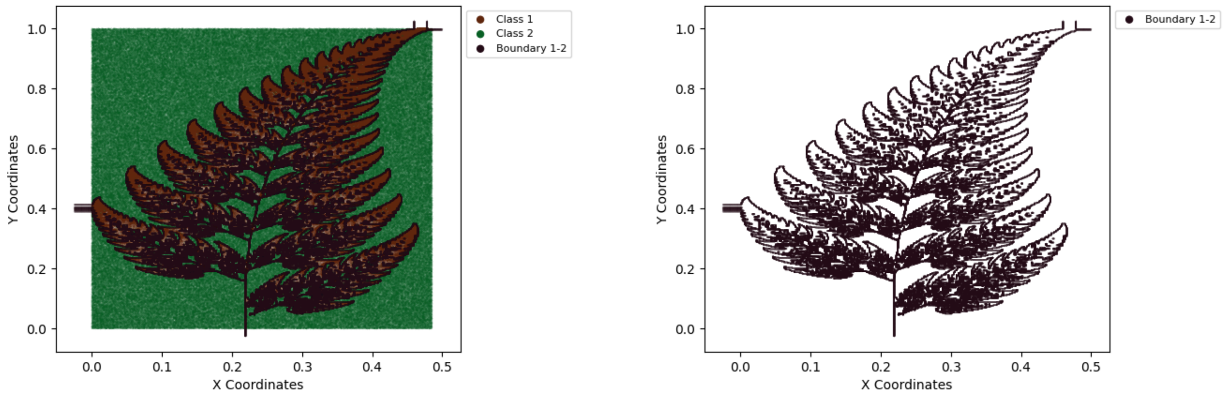


Figure 17: Plots of Decision Tree on BF Dataset corresponding to Table 15

Fractal Dimension (Boundary)	1.7415	Weight (Topology 0, Boundary)	0.1450
Connectivity Factor (Boundary)	00.3289	Weight (Topology 1, Boundary)	0.5764
Topological Dimension (CF Based)	1	Weight (Topology 2, Boundary)	0.2785
Topological Dimension (Weight Based)	1	Fractal Dimension (Object 1)	1.8340

Table 16: Boundary characteristics results

7.11 Sierpinski Carpet Data (SC)

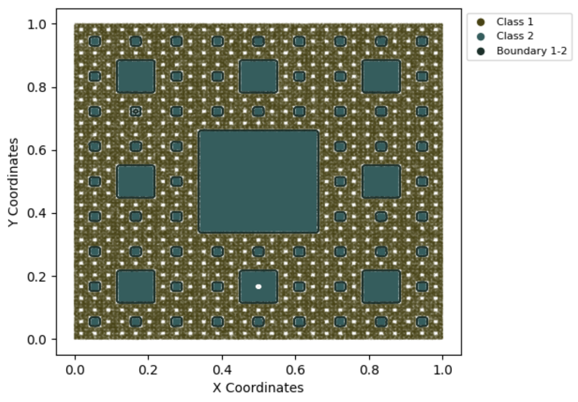
Dataset Creation Details: The Sierpinski Carpet Dataset (SC) is generated using the Chaos Game method, which employs iterative affine transformations. Each new point is computed from the previous one using the following equation:

Table 17: SCD Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	Value / Class / Formula	Description
Initial Point	$(0.5, 0.5)$	Starting coordinates for the iterative process
Number of Carpet Points	1,000,000	Total points generated for the carpet structure (Class 1)
Number of Non-Carpet Points	1,000,000	Random points within the carpet's bounding box (Class 0)
Affine Transformations	Eight, with coefficients:	Recursive structure of the carpet
	1: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.0$, $f = 0.0$	
	2: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.333$, $f = 0.0$	
	3: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.666$, $f = 0.0$	
	4: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.0$, $f = 0.333$	
	5: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.666$, $f = 0.333$	
	6: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.0$, $f = 0.666$	
	7: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.333$, $f = 0.666$	
	8: $a = 0.333$, $b = 0.0$, $c = 0.0$, $d = 0.333$, $e = 0.666$, $f = 0.666$	
Transformation Probabilities	p_1–$p_8 = 0.125$ each	Probability of selecting each transformation
Training	Carpet (1): 800,000 samples (40%) Non-Carpet (0): 800,000 samples (40%)	Number and fraction of training samples for each class
Testing	Carpet (1): 200,000 samples (10%) Non-Carpet (0): 200,000 samples (10%)	Number and fraction of test samples for each class
Point Generation	$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$	Affine transformation applied at each iteration

Note: At each iteration, a random number in $[0, 1)$ determines which transformation is applied, according to the specified probabilities. The dataset comprises two classes: Class 1 (carpet points) and Class 0 (random non-carpet points).

7.11.1 KNN:



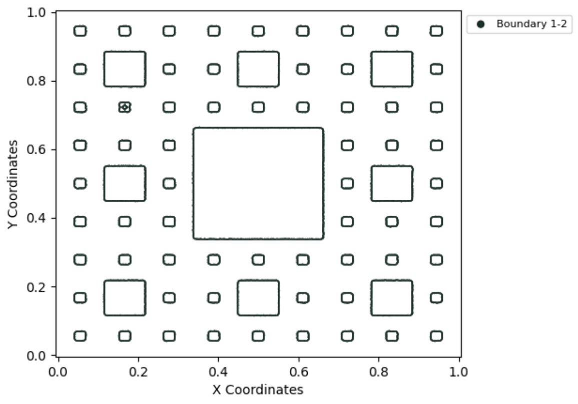


Figure 18: Plots of KNN on SCD Dataset corresponding to Table 17

Fractal Dimension (Boundary)	1.0768
Connectivity Factor (Boundary)	0.3962
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1
Weight (Topology 0, Boundary)	0.0000
Weight (Topology 1, Boundary)	0.8049
Weight (Topology 2, Boundary)	0.1950
Fractal Dimension (Object 1)	1.8967

Table 18: KNN boundary characteristics results

7.12 Sierpinski Triangle (ST)

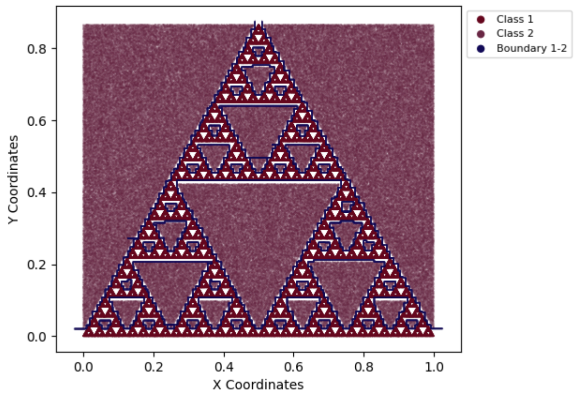
The Sierpinski Triangle dataset is generated using the Chaos Game method, which employs iterative affine transformations.

Table 19: STD Dataset Parameters, Train-Test Distribution, and Formulas

Parameter / Subset / Formula	**Value / Class / Formula**	**Description**
Initial Point	$(0.0, 0.0)$	Starting coordinates for the iterative process
Number of Triangle Points	1,000,000	Total points generated for the triangle structure (Class 1)
Number of Non-Triangle Points	1,000,000	Random points within the triangle's bounding box (Class 0)
Affine Transformations	Three, with coefficients: 1: $a = 0.5, b = 0.0, c = 0.0, d = 0.5, e = 0.0, f = 0.0$ 2: $a = 0.5, b = 0.0, c = 0.0, d = 0.5, e = 0.5, f = 0.0$ 3: $a = 0.5, b = 0.0, c = 0.0, d = 0.5, e = 0.25, f = 0.433$	Defines the recursive structure of the triangle
Transformation Probabilities	$p_1 = p_2 = p_3 = 0.333$	Probability of selecting each transformation
Training	Triangle (1): 800,000 samples (40%) Non-Triangle (0): 800,000 samples (40%)	Number and fraction of training samples for each class
Testing	Triangle (1): 200,000 samples (10%) Non-Triangle (0): 200,000 samples (10%)	Number and fraction of test samples for each class
Point Generation	$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$	Affine transformation applied at each iteration

Note: At each iteration, a random number in $[0, 1)$ determines which transformation is applied, according to the specified probabilities. The dataset comprises two classes: Class 1 (triangle points) and Class 0 (random non-triangle points).

7.12.1 Decision Tree:



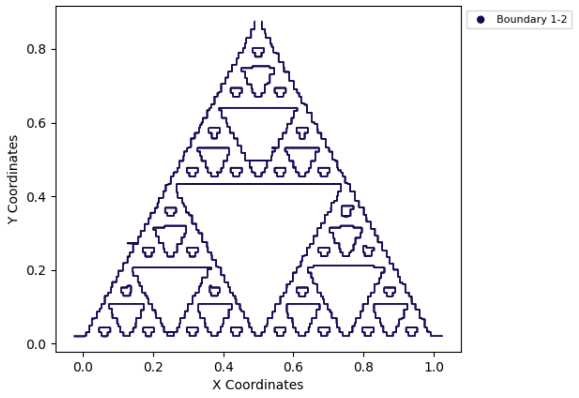


Figure 19: Plots of Decision Tree on STD Dataset corresponding to Table 19

Fractal Dimension (Boundary)	1.0176
Connectivity Factor (Boundary)	0.2599
Topological Dimension (CF Based)	1
Topological Dimension (Weight Based)	1

Weight (Topology 0, Boundary)	0.0001
Weight (Topology 1, Boundary)	0.9864
Weight (Topology 2, Boundary)	0.0133
Fractal Dimension (Object 1)	1.6232

Table 20: Boundary characteristics results

7.12.2 KNN:

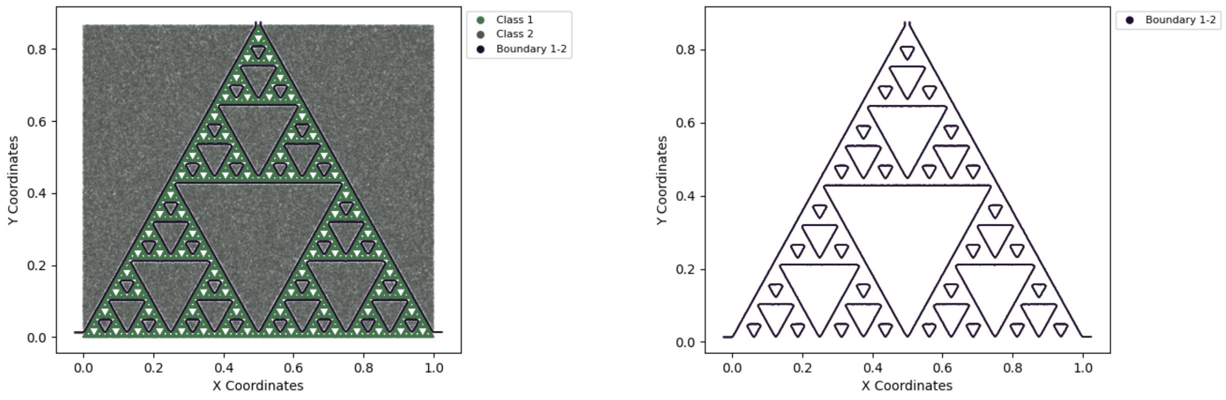


Figure 20: Plots of KNN on STD Dataset corresponding to Table 19

Fractal Dimension (Boundary)	1.0292	Weight (Topology 0, Boundary)	6.3985
Connectivity Factor (Boundary)	0.4222	Weight (Topology 1, Boundary)	0.7702
Topological Dimension (CF Based)	1	Weight (Topology 2, Boundary)	0.2297
Topological Dimension (Weight Based)	1	Fractal Dimension (Object 1)	1.6232

Table 21: KNN boundary characteristics results