
PROTOTYPE SELECTION USING TOPOLOGICAL DATA ANALYSIS (PREPRINT)

Jordan Eckert

Department of Mathematics & Statistics
Auburn University
Auburn, AL 36849
jpe0018@auburn.edu

Elvan Ceyhan

Department of Mathematics & Statistics
Auburn University
Auburn, AL 36849
ceyhan@auburn.edu

Henry Schenck

Department of Mathematics & Statistics
Auburn University
Auburn, AL 36849
hks0015@auburn.edu

November 10, 2025

ABSTRACT

Recently, there has been an explosion in statistical learning literature to represent data using topological principles to capture structure and relationships. We propose a topological data analysis (TDA)-based framework, named Topological Prototype Selector (TPS), for selecting representative subsets (prototypes) from large datasets. We demonstrate the effectiveness of TPS on simulated data under different data intrinsic characteristics, and compare TPS against other currently used prototype selection methods in real data settings. In all simulated and real data settings, TPS significantly preserves or improves classification performance while substantially reducing data size. These contributions advance both algorithmic and geometric aspects of prototype learning and offer practical tools for parallelized, interpretable, and efficient classification.

Keywords Prototype Selection · Topological Data Analysis · Persistent Homology · Bifiltration

1 Introduction

The *prototype selection problem* is a machine learning task critical in reducing data complexity, enhancing computational efficiency, and improving the interpretability of machine learning models [1]. The prototype selection problem arises from several practical considerations such as computational complexity where classification becomes prohibitive for large datasets, storage requirements where storing the entire dataset in memory is infeasible, and / or for noise sensitivity or removing redundancies within the data set [2, 3, 4]. The exponential growth of data complexity and dimensionality in modern applications has necessitated the development of novel prototype frameworks. *Topological data analysis* (TDA) has emerged as a method that leverages the mathematical machinery of algebraic topology to extract robust, noise-invariant features from complex dataset topologies [5]. However, using TDA as a prototype selection tool has seen limited exploration as a standalone prototype selection method.

This work proposes a method entirely based on TDA to select representative prototype points from a dataset. The fundamental idea is to leverage the topology between (inter-class) and within (intra-class) labeled classes to identify the most topologically significant points for a class, using topological invariants computed through persistent homology (PH) to detect these points. These methods introduce user-defined geometric regularization hyperparameters to control

prototype selection near the boundary.

PH is a powerful tool in TDA that captures snapshots of topological features in a nested family of simplicial complexes across incremental threshold values [6]. These topological features are encoded, considering those values when they are “born” (appear) and when they “die” (disappear or merge). The numerical difference between birth and death scales is called a topological feature’s *persistence*. The evolution of the simplicial structure is encoded using high-level representations [6, 5]. Despite the rich topological information gained from TDA, it has not been used in prototype selection. Rather, TDA is used primarily as a visualization tool [5], or topological descriptors are commonly built as inputs into other machine learning model [7, 8], or even as a standalone classification strategy [9, 10].

Prototype selection is often done as a data preprocessing or reduction step for classification tasks. These data reduction techniques aim to identify and retain only the most informative training instances while eliminating redundant internal points and noisy examples that degrade performance [11].

1.1 Contributions

The main contributions of this work are as follows:

- The development and detailing of a topological prototype selector (TPS) algorithm.
- We conduct nine simulated experiments to see the effectiveness of TPS on different class imbalance and class overlap settings.
- We perform Monte Carlo simulations to see the effect of different hyperparameters on both classification performance and prototype set cardinality.
- We perform Monte Carlo simulations to show computational time of TPS.
- We evaluate the performance of TPS on eight real world datasets comparative to other prototype selection methods.
- Lastly, we show the effectiveness of TPS compared to other prototype selection methods using different metrics on text-based classification settings.

The fundamental concepts, algorithms, and methodology of the proposed prototype selection method are detailed in Section 2. Section 3 explains the concepts, algorithms, and methodology of the proposed prototype selector. Next, Section 4 describes the experimental protocols to assess the proposed and baseline algorithms, and an analysis of the results is provided. We end with conclusions and further discussion in Section 5.

2 Foundational Concepts

In this section, we provide some background on both the prototype selection problem and persistent homology. The concepts discussed are intended to be brief, high-level overviews. For more detailed discussion of the prototype selection problem we recommend [11]. For more detailed discussion of persistent homology and topological data analysis we recommend [6], [12], [13], and [14].

2.1 Prototype Selection Problem

Formally, the prototype selection problem is defined as:

Definition 1 Given a training data $\mathcal{X} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ where class labels $y_i \in \mathcal{Y}$ and $\mathcal{Y} = 1, \dots, K$, the objective of the prototype selection problem is to find $S \subset \mathcal{X}$ that minimizes $|S|$ subject to maintaining classification performance on a validation set.

Prototype algorithms are typically classified either by *objectives* or by *approaches*. When classifying by objectives, prototype selection algorithms are seen as either

1. *Condensation Methods*. Condensation methods try to remove redundant class observations. An example of condensation method algorithms is condensed nearest neighbor (CNN) which incrementally builds S by adding misclassified examples. However, condensation method performance is not robust to order-dependencies and tends to retain unnecessary prototypes near class centers.

2. *Edition Methods*. Edition methods try to remove noisy or mislabeled points. Approaches like edited nearest neighbor (ENN) construct $\mathcal{S}$ by removing points from $\mathcal{X}$ that disagree with their neighborhood majority. While effective for noise removal, these methods often achieve limited reduction rates and suffer from accuracy loss.
3. or *Hybrid Methods*. Hybrid methods combine condensation and edition, which typically achieve better performance but at increased computational cost. An example of a hybrid prototype selection classifier would be CNN+ENN which combines sequentially condensed nearest neighbor with edited nearest neighbor [1].

Alternatively, prototype selection algorithms are classified by approaches as either,

1. *Heuristic Methods*. These methods are based on simple rules or distance criteria. Both CNN and ENN are examples of heuristic methods.
2. *Optimization Methods*. These methods formulate prototype selection as an optimization problem. Commonly, this is the minimization of the *set cover problem* where the goal is to find minimal cardinality prototypical objects that cover the dataset of interest. One variation is the Bien-Tibshirani set-covering prototype selector (BienTib) which uses metric balls centered at class observations to form covers [4]. A similarly related problem is the *class cover problem* where metric balls are constructed to cover each class specifically, such as in class cover catch digraphs (CCCDs) [15, 16]. When the optimization problem is explicitly tied to the performance of a downstream model, these methods are called *wrapper methods*. Examples of wrapper methods might be forward or backward selection of prototypes under k -NN while optimizing under downstream model accuracy such as in the All K -Nearest Neighbors (AllKNN) prototype selection algorithm [17].
3. *Clustering-Based Methods*. Cluster-based methods use clustering such as K -Means or K -Medoids to find representative centroids which then act as a prototype set [11].
4. or *Information-Theoretic and Learning-Based Methods*. These strategies are *filtered methods* which select prototypes based on intrinsic properties of the data without explicitly evaluating model performance [11].

The central principle shared among all prototype methods is the *boundary-proximity principle*, which states that patterns near classification boundaries are considered to have higher classification contribution [11, 18]. From a topological viewpoint, classification boundaries can be characterized as *submanifolds* separating different classes. Capture the underlying topology and density information can provide a promising framework for characterizing these submanifolds. Additionally, topological based prototype selection promises a more robust prototype based on genuine structural features comparative to other geometric selection strategies based on distances, which can be affected by outliers.

2.2 Simplicial Complexes

Computational topology requires discrete representations called *simplices* [19]. A *simplex* is defined as follows:

Definition 2 A q -simplex, σ , is the generalization of a tetrahedral region of space to q dimensions. Given a q -simplex σ , a d -simplex τ with $0 \leq d \leq q$ and vertex set $|\mathcal{V}(\tau)| = d + 1$ is called a d -face of σ denoted by $\tau \leq \sigma$. We also call σ a q -coface of τ denoted by $\sigma \geq \tau$.

Simplices provide discrete representations by encoding topological information through combinatorial structure. The vertex set $\mathcal{V}(\sigma)$ is the set of *vertices* of σ and the simplex σ is said to be *generated* by $\mathcal{V}(\sigma)$. The dimension of a simplex σ is one less than the cardinality of its vertex set, that is $\dim(\sigma) = |\mathcal{V}(\sigma)| - 1$. Geometrically, simplices correspond to either a vertex point (0-simplex), edges between vertices, (1-simplex), triangles (2-simplex), tetrahedra (3-simplex), and their higher-dimensional analogues as seen in Figure (1).

In order to define homology groups of topological spaces, the notion of a *simplicial complex* is central:

Definition 3 A *simplicial complex* $\mathcal{K}$ in $\mathbb{R}^n$ is a finite collection of simplices in $\mathbb{R}^n$ such that

- (i) Given a q -simplex $\sigma \in \mathcal{K}$ and $\tau \leq \sigma$ then $\tau \in \mathcal{K}$, and
- (ii) If $\sigma_1, \sigma_2 \in \mathcal{K}$ then $\sigma_1 \cap \sigma_2$ is either a face of both σ_1 and σ_2 or is empty.

Essentially, collections of simplices that are glued together in a specific orientation are called *simplicial complexes*. The orientation and boundary operations for each simplex is assigned using matrices [14, 19].

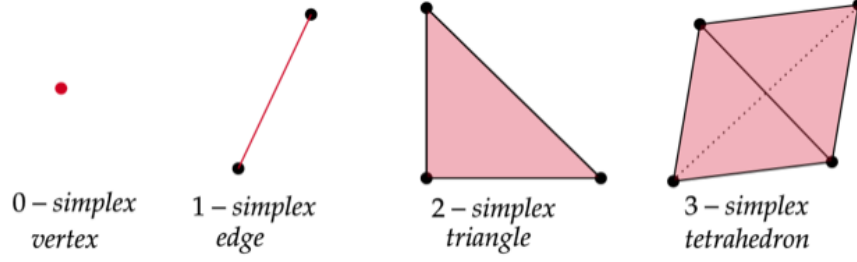


Figure 1: Geometric representations of q -simplices. Each corresponds to their respective tetrahedral dimensional analogues.

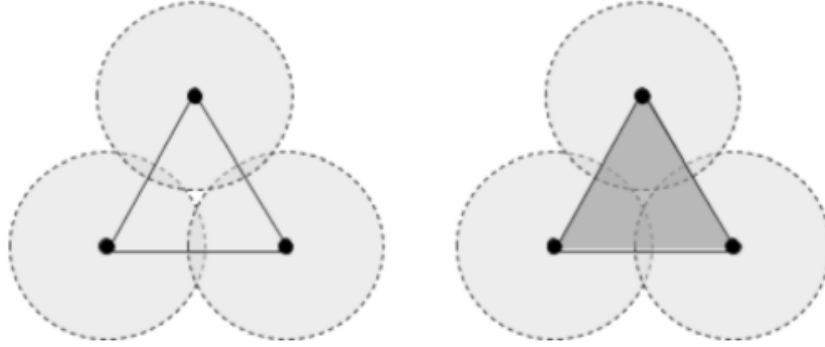


Figure 2: Comparison of Čech complex (left) with Rips complex (right) at the same radius. The simplicial Čech complex consists of a hollow triangle since all three balls do not overlap, where the Rips complex is the solid triangle.

2.3 Persistent Homology

Homology provides an algebraic framework for quantifying topological features using simplicial complexes. The *homology group*, $H_k(\mathcal{X})$, is calculated by defining boundary maps between the *chain* of different groups of formal linear combinations of k -simplices, finding the kernels and images of these maps, and then taking the quotient of the kernels by the images at each stage. For each dimension k , $H_k(\mathcal{X})$ characterizes the k -dimensional “holes” in the topological space $\mathcal{X}$ [5, 19]. $H_0(\mathcal{X})$ measures connected components, $H_1(\mathcal{X})$ measures “loops” or 1-dimensional holes, $H_2(\mathcal{X})$ measures “voids” or 2-dimensional holes, and so on with $H_k(\mathcal{X})$ measuring k -dimensional analogues. One *topological invariant*, is the rank of $H_k(\mathcal{X})$, called the *Betti number* which measures the number of independent k -dimensional holes [14]. For a complete theoretical basis, see [6].

As a general rule, the objective of persistent homology is to track how homology changes across scale values which vary incrementally, in a process known as a *filtration*.

Definition 4 Let $\mathcal{K}$ be a simplicial complex. A *filtration*, $\mathcal{F}$, on $\mathcal{K}$ is a succession of increasing sub-complexes of $\mathcal{K}$, $\emptyset \subseteq \mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_2 \cdots \subseteq \mathcal{K}_n = \mathcal{K}$. In this case, $\mathcal{K}$ is called a *filtered simplicial complex*.

The filtration $\mathcal{F}$ on a filtered simplicial complex $\mathcal{K}$ is obtained by taking a collection of $\mathcal{E}_{\mathcal{K}}$ values $0 < \epsilon_0 < \epsilon_1 < \cdots < \epsilon_n$, and the complex $\mathcal{K}_i$ corresponds to the value ϵ_i . The set $\mathcal{E}_{\mathcal{K}}$ is called the *filtration value collection* associated to $\mathcal{F}$. In general, scales along a filtrations correspond to metric covering balls with increasing radii that form covers of the set of data, $\mathcal{X}$.

Filtrations in TDA applications are dominated by two primary simplicial complex constructions, the *Čech complex* and the *Vietoris-Rips (Rips) complex* [14].

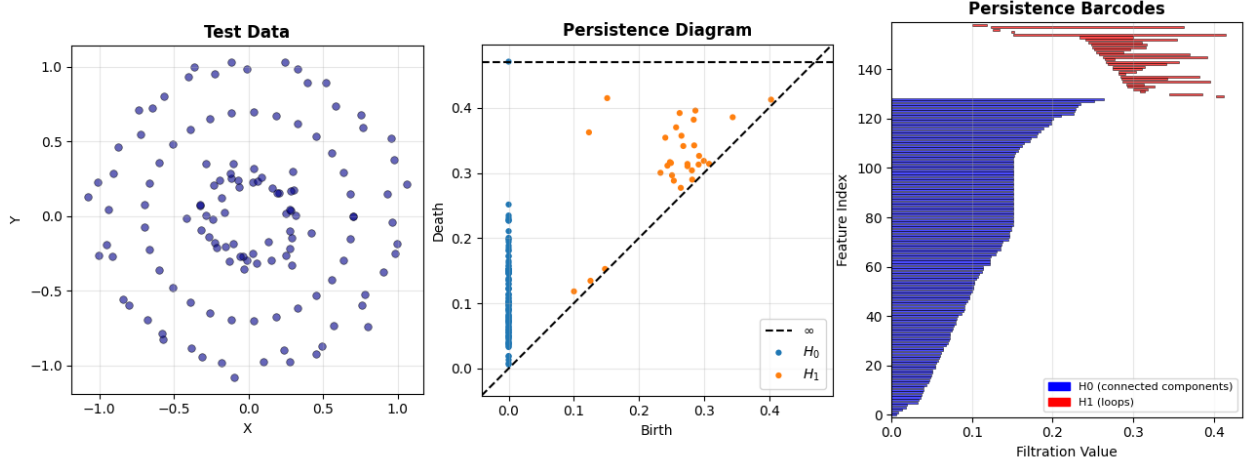


Figure 3: Example of barcode and persistent homology visualizations. Left is the original data, middle is the corresponding persistent homology diagram (birth, death) pairs plotted, and right is the barcodes for specific filtration values.

Definition 5 The Čech complex, $\check{C}_\epsilon(\mathcal{X})$, is a simplicial complex with q -simplices if and only if there exists $\{x_i\}_{i=1}^{q+1} \in \mathcal{X}$ tuples of points such that $\bigcap_{i=1}^{q+1} \overline{B_\epsilon(x_i)} \neq \emptyset$. That is, there exists a q -simplex if the intersections of the closed ϵ -distanced balls centered at each of the $q + 1$ points is non-empty.

Čech complexes act as a *nerve complex* [6]. Nerve complexes are abstract complexes that record the pattern of intersections between the points in $\mathcal{X}$. The *nerve theorem* states that that under conditions where Čech complexes form *good covers*, the simplicial complex is homotopy equivalent to $\mathcal{X}$ [14]. We define good covers as:

Definition 6 Let $\mathcal{X}$ be a topological space. Let $\mathcal{U}$ be a non-empty and finite collection of sets $U = \{U_i\}_{i \in I}$ such that $\mathcal{U}$ covers $\mathcal{X}$. That is, $\mathcal{X} = \bigcup_{i \in I} U_i$. If all $U_i \in \mathcal{U}$ and all finite, nonempty intersections of U_i are able to be shrunk to a single point we say $\mathcal{U}$ is a *good cover*.

Rips complexes do not require that require all ϵ -radius balls to intersect together to form a simplicial complex:

Definition 7 The Vietoris-Rips complex, $R_\epsilon(\mathcal{X})$, is a simplicial complex with q -simplices if and only if there exists $q + 1$ sets of points of $\mathcal{X}$ such that $d(x_i, x_j) \leq \epsilon$ for all $x_i, x_j \in \mathcal{X}$ such that $i \neq j$ for distance metric $d(\cdot, \cdot)$.

The Rips complex only needs to check the pairwise distances to determine if higher-dimensional simplices exist. If all edges exist, the higher-dimensional simplex is automatically included, making Rips complexes more computationally efficient to compute compared to Čech complexes. Rips complexes provide only approximate topological accuracy to the underlying $\mathcal{X}$. We provide a comparison of both simplicial complexes in Figure (2). We can “trap” exact topological information from Čech complexes using Rips complexes. The inclusion maps $\check{C}_\epsilon(\mathcal{X}) \subseteq R_{2\epsilon}(\mathcal{X}) \subseteq \check{C}_{2\epsilon}(\mathcal{X})$ demonstrate this [14]. Therefore, it is computationally feasible to determine the Rips complex at some scale that captures the essential topological features of the Čech complex at a related scale and vice-versa [14].

Given $\{\mathcal{K}_i\}_{i=0}^n$, the inclusion maps $\mathcal{K}_i \hookrightarrow \mathcal{K}_j$ induce homomorphisms $H_k(\mathcal{K}_i) \rightarrow H_k(\mathcal{K}_{i'})$ on homology groups. Essentially, each level change from i to i' has the potential to correspond to either a “birth” or a “death” of a topological feature. This yields a sequence of vector spaces connected by linear maps called a *persistence module*. Persistence modules decompose into “birth” and “death” pairs, each representing the “lifetime” of a topological feature [14, 5, 10]. The length of the interval of a (birth, death) pair, is called its *persistence*. Long persistence correspond to underlying geometric features in the data, while short persistence is likely noise. Persistence can be graphically represented through *barcodes* [12]. We can also visualize the lifetimes of topological features indicating when it first appears and when it disappears in the filtration by plotting (birth, death) pairs on *persistence diagrams* [5]. We display both styles of visualizations on an example data set in Figure (3). A crucial property of persistent homology is its stability, as small perturbations in the input data produce small changes in barcodes and persistence diagrams.

2.4 Multi-Parameter Persistent Homology

Many applications, especially those in biological fields, require simultaneous consideration of filtrations across multiple scale parameters [20, 21]. For instance, in analyzing medical imaging, one might vary both distance between points on a manifold and the curvature of the manifold to get a more in-depth geometric understanding [22, 23]. This motivates the study of *bifiltrations*, a two-parameter extension of a filtration. Specifically, bifiltrations allow for the examination of multi-dimensional persistence lifetimes, providing a more comprehensive view of the structure of data.

For a bifiltration, sub-complex $\mathcal{K}_{i,j} \subset \mathcal{K}$ is defined where i and j correspond to different filtration values in the filtration value collection $\mathcal{E}_{i,j}$ where (i, j) belongs to a partially ordered set [13]. Formally, we define it as:

Definition 8 A bifiltration of a topological space X is a family of subsets $\{\mathcal{K}_{i,j}\}_{i,j \in \mathcal{E}_{i,j}}$ such that the subsets satisfy the following:

- For any fixed $i, j \mapsto \mathcal{K}_{i,j}$ is a filtration (increasing with i), and
- For any fixed $j, i \mapsto \mathcal{K}_{i,j}$ is also a filtration (increasing with j).

Figure (4) illustrates an example of a bifiltration commutative diagram for $i = 0, 1, 2$ and $j = 0, 1$. We see that bifiltrations are a collection of two parameter nested simplicial complexes that capture how topology changes across the different varying scales.

$$\begin{array}{ccccccc}
 \emptyset & \longrightarrow & \mathcal{K}_{0,0} & \longrightarrow & \mathcal{K}_{1,0} & \longrightarrow & \mathcal{K}_{2,0} \\
 & & \downarrow & & \downarrow & & \downarrow \\
 & & \mathcal{K}_{0,1} & \longrightarrow & \mathcal{K}_{1,1} & \longrightarrow & \mathcal{K}_{2,1} \longrightarrow \mathcal{K}
 \end{array}$$

Figure 4: An example of a commutative diagram for a bifiltration of $\mathcal{K}$ across two parameters. Each $\mathcal{K}_{i,j}$ represents a sub-complex.

Many of the fundamental theorems for filtrations do not have analogous variants for bifiltrations. The classification of finitely generated modules over multivariable polynomial rings is much more complicated than the corresponding result for single variable polynomial rings [14, 22]. Therefore, unlike one-parameter persistence, multi-parameter persistence modules do not admit a complete discrete invariant. In general, the goal for multi-parameter persistent homology is to stratify or filter the modules to obtain an analog of the barcode or persistence lifetime. *Fibered barcodes* are collections of a persistence module obtained by considering restrictions of the persistence module to affine lines across positive slopes. Therefore, single-parameter persistent homology can be recovered from the bifiltration by *slicing* along one parameter at a fixed level.

3 Proposed Prototype Selection Method

We assume training data $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$ with class labels $\mathcal{Y} = 1, \dots, K$. Our goal is to find a prototype set $\mathcal{S} \subset \mathcal{X}$ using topology in such a way that $|\mathcal{S}|$ is minimized while maintaining classification performance on some validation set. In this section, we detail a prototype selection method to help identify $\mathcal{S}$. Overall, a bifiltered simplicial complex $\mathcal{K}$ is built over $\mathcal{X}$. The proposed method is based on the supposition that on the bifiltration some “meaningful” sub-complex $\mathcal{K}_{i,j} \subset \mathcal{K}$ exists that provides relevant inter- and intra-class relationships to serve as prototypical points for $\mathcal{X}$. In the context of prototype selection, these points should lie along the classification boundary in order to minimize potential classification performance loss. Specifically, for a q -simplex $\sigma \in \mathcal{K}_{i,j}$ we take the vertex set $\mathcal{V}(\sigma)$ to act as the prototype set $\mathcal{S}$. The rest of this section is dedicated to identifying the sub-complex $\mathcal{K}_{i,j}$ used for prototype selection.

The algorithm begins by defining a target class $c \in \mathcal{Y}$. The nontarget class is the collection of other class points whose y_i value satisfies $y_i \in \mathcal{Y} \setminus c$. We define a bifiltered simplicial complex $\mathcal{K}$ using both a *radius filtration parameter* ϵ_i ,

and a *neighbor filtration parameter* ϵ_j . The neighbor filtration parameter ϵ_j is the filtration based on the distances from target class observations to the non-target classes, while the radius filtration parameter ϵ_i is the filtration based on the distances of only target class observations. The neighbor filtration shows the inter-class topological relationships of the target and k -closest non-target class(es), and the radius filtration shows the intra-class topological relationships of the target class. Both parameters ϵ_i and ϵ_j correspond to the indices i and j of the sub-complex $\mathcal{K}_{i,j}$ respectively. The next section details finding a sub-complex $\mathcal{K}_{i,j} \subset \mathcal{K}$ which serves as an approximate for the underlying data point cloud.

3.1 Prototypical Sub-Complexes for Target Class

We can think of a bifiltration as a collection of radius filtrations along the level sets of each neighborhood filtration value. We leverage this fact, and calculate the neighbor filtration then sequentially the radius filtration on the resultant subset of vertices selected from the neighbor filtration. This is done rather than computing full two-parameter invariants as it provides computational speedup. Calculation for both filtrations are done using the **Ripser** library in Python [24]. In calculation of (co)homology, a maximal dimension of $2 < q \ll |\mathcal{X}|$ is used to control simplicial complexes exponential growth.

The distance and proximity functions used in calculating the filtrations can be problem specific, making TPS flexible on the metric equipped data space. The neighbor filtration is built using the sum of K nearest non-target class observations. We care about leveraging inter-class topological information for selection of points close to the boundary, therefore we only care about points that are locally close to each target class point as the boundary is subjected to geometrically change as points are far away.

Consider the collection of all ϵ_j values $\{\epsilon_j\}_{j=1}^m$. Since the bifiltration is constructed with the neighbor filtration first without the radius filtration, we can consider as a sliced filtration along the radius filtration along $\epsilon_{i=0}$. That is, this filtration is treated as a one-parameter filtration. Each inter-class topological feature is represented by an element in a homology group of a given dimension d by the interval $(birth, death) \subset \mathbb{R}$.

Let D^j be the set of persistence intervals of non-trivial j -cycles along the filtration of $\mathcal{K}_{0,j}$ where $i = 0$ indicates that no radius filtration has occurred yet. The chain of inclusion maps induced by the different values of the neighbor filtration values $\{\epsilon_j\}_{j=1}^m$ would therefore be $\mathcal{K}_{0,0} \rightarrow \mathcal{K}_{0,1} \rightarrow \dots \rightarrow \mathcal{K}_{0,m}$. The collection of all persistence intervals is then represented as $D = \bigcup_{j>0} D^j$. For each topological feature $d \in D$, we truncate the immortal value to be the maximum value in the neighbor filtration values, that is $\max(\mathcal{E}_{0,j})$. We define

$$int(d) = \min\{d[death], \max(\mathcal{E}_{0,j})\} - d[birth] \quad (1)$$

Equation [1] serves as a way to calculate the lifetime of all inter-class topological features [10]. To pick which neighborhood persistence interval $d \in D$ will serve as the slice for the radius filtration, define the following:

$$d_q = QuantInt(D) = \begin{cases} d_{(1)} & \text{if } q = 0 \\ d_{(n)} & \text{if } q = 1 \\ (1 - \gamma) \cdot d_{(k)} + \gamma \cdot d_{(k+1)} & \text{otherwise} \end{cases} \quad (2)$$

where $d_{(k)}$ represents the k^{th} ordered lifetimes, and

$$h = (n - 1) \cdot q + 1 \quad (3)$$

$$k = \lfloor h \rfloor \quad (4)$$

$$\gamma = h - k. \quad (5)$$

Here, Equations [2] - [5] are used to find the persistence lifetime of the q^{th} quantile of all possible lifetime values. Once d_q is identified, the filtration value closest to $\epsilon_{j^*} = d_q[death]$ is selected. The simplicial complexes at this filtration are collected, and the original data points corresponding to the resultant sub-complex $\mathcal{K}_{0,j^*}$ are recovered.

If a tie occurs during calculation of $QuantInt(D)$, all tied persistence lifetimes are used to extract potential vertices for the next filtration, rather than focusing on a lifetime with the earliest birth or latest death. This might seem counterintuitive, as it could potentially lead to larger prototype sizes, however, the exclusion of such points might be detrimental to classification performance. Additionally, there is no guarantee that these points will be included after the final radius filtration. We leave additional studies using earliest birth or latest death persistence lifetimes in cases of tied

Table 1: Descriptions of the nine simulated datasets used for testing prototype selection performance of TPS.

Dataset	Number of Observations (n)	Classes	Class Ratio	Imbalance Ratio	Baseline G-Mean			
					1-NN	3-NN	5-NN	Linear SVM
Well-Separated Blobs (3 classes)	600	3	200:200:200	1.00	1.000	1.000	1.000	1.000
Overlapping Blobs (4 classes)	800	4	200:200:200:200	1.00	0.836	0.880	0.872	0.853
Overlapping Clusters (3 classes)	600	3	200:200:200	1.00	0.678	0.749	0.767	0.789
Two Moons (Moderate Noise)	500	2	250:250	1.00	0.993	1.000	1.000	0.880
Two Moons (High Noise)	500	2	250:250	1.00	0.880	0.899	0.899	0.827
Concentric Circles (Moderate Noise)	500	2	250:250	1.00	0.973	0.987	0.973	0.473
Concentric Circles (High Noise)	500	2	250:250	1.00	0.800	0.798	0.833	0.480
Imbalanced Classes (80/20%)	500	2	400:100	4.00	0.769	0.745	0.683	0.715
Mixed Multi-class (3 classes, 40/30/10%)	800	3	400:300:100	4.00	0.941	0.943	0.940	0.959

quantiles for later work.

In practice, we can filter out topological noise by considering lifetimes above some user-defined threshold τ_{min} . Both q and τ_{min} acts as geometric regularization for the boundary-proximity principle. As q decreases, $|\mathcal{S}|$ will also decrease as the resultant sub-complex $\mathcal{K}_{0,j^*}$ will include less points for the next filtration. The tradeoff is that classification performance can potentially worsen as q decreases. This allows users to tune the tradeoff between dataset reduction and potential classification performance reduction as necessary.

We now build the sliced radius filtration along the level set provided at ϵ_{j^*} . Similarly, let D^i be the set of persistence intervals of non-trivial i -cycles along the filtration of $\mathcal{K}_{i,j^*}$ where j^* corresponds to the selected neighbor filtration scale ϵ_{j^*} . The chain of inclusion maps induced by the different values of the sliced radius filtration values $\{\epsilon_i\}_{i=1}^n$ would therefore be $\mathcal{K}_{0,j^*} \rightarrow \mathcal{K}_{1,j^*} \rightarrow \dots \rightarrow \mathcal{K}_{n,j^*}$. The collection of all persistence intervals is then represented as $D = \bigcup_{i>0} D^i$. For each topological feature $d \in D$, we truncate the immortal value to be the maximum value in the radius filtration value $\max(\mathcal{E}_{i,j^*})$. We can define the interval in a similar way to Equation [6] using $\max(\mathcal{E}_{i,j^*})$:

$$int(d) = \min\{d[death], \max(\mathcal{E}_{i,j^*})\} - d[birth]. \quad (6)$$

Equation [6] serves as a way to calculate the lifetime of all intra-class topological features. A desired persistence interval $d \in D$ is selected by using:

$$d_a = AvgInt(D) = \operatorname{argmin}_{d \in D} |int(d) - avg(D)| \quad (7)$$

where $avg(D) = \frac{1}{|D|} \sum_{d_i \in D} int(d_i)$. Equation [7] represents the persistence lifetime closest to the persistence lifetime average. The rationale for using the mean persistence lifetime interval for radius scale selection is the mean is more representative of the overall underlying class topology distribution. However, additional functions for radius persistence selection such as using the median persistence lifetime, could also be considered instead of the average. Once the persistence lifetime is selected, we can identify the sub-complex $\mathcal{K}_{i,j^*} \subset \mathcal{K}$ that corresponds to that interval. In the event of a tie, all persistence lifetimes corresponding to the average are used as before.

This resultant sub-complex at the intersection of the slices of the neighbor filtration and radius filtration is acts as an approximate representation of the most important underlying inter- and intra-class topological structures. The vertex set for $\mathcal{K}_{i,j^*}$ is extracted as the prototype set for the original dataset. Once this operation is completed, a new target class is selected, and the steps are repeated for all classes within $\mathcal{X}$. Complete algorithms are provided in the Appendix.

4 Results

In this section we evaluate the performance of the topological prototype selector (TPS) algorithm. We evaluate the performance of TPS on simulated data, as well as study the different effects the hyperparameters have on the performance of TPS. This section also presents computational time trial results, and performance of TPS against other state-of-the-art prototype selection algorithms in real data settings.

4.1 Performance on Simulated Datasets

We first evaluate the performance on nine different simulated datasets under different data intrinsic characteristics. Before using prototype selection, data was preprocessed using a stratified 70/30% training and testing split to ensure

class imbalance was maintained in the training set. A baseline classifier is trained on the full training split for comparison of classification performance. Afterwards, we run our prototype selection algorithm on the training split and a new classifier is trained on the resultant prototypes only. Both baseline and prototype-only trained models are then evaluated on the remaining 30% testing split. We used 1-nearest neighbor (1-NN), 3-nearest neighbor (3-NN), 5-nearest neighbor (5-NN), and linear kernel support vector machine (SVM) as different baselines for model comparisons.

Since some of the data is imbalanced, another metric than accuracy should be used to evaluate classification performance of the prototype sets [25]. We instead opt for comparison using the geometric mean (G-Mean) as it provides a class-wide balance of both the sensitivity and specificity of the classification model [26, 27]. Optimal configurations for TPS hyperparameters are determined by the configuration that causes the smallest decrease (or in some instances the largest increase) in G-Mean. In the event of a tie, the optimal configuration is the one that also corresponds to the highest reduction percentage.

The nine artificial datasets were constructed with different underlying distributions and intrinsic data characteristics, to evaluate the reduction performance of TPS on difficult settings. All relevant information on the simulated datasets is provided in Table (1). We note that all simulated datasets are simulated in $\mathbb{R}^2$ to allow for visualization of prototype selected, which are provided in the Appendix.

While TPS can be performed over any $H_k(\mathcal{X})$ homology group, we focus exclusively on the $H_0(\mathcal{X})$ homology group for this experiment. Representing a 1-dimensional hole in H_1 requires more vertex points than its corresponding H_0 analog. Meaning that H_1 -based prototype sets must retain more points by design, thus limiting reduction capabilities [6]. We acknowledge that for specialized data with high dimensional characteristics that must be maintained (such as voids in porous media for material sciences, volumetric physical simulations, etc.) exploration of higher homology groups can, and should be, considered. Selection for hyperparameter q is chosen between $q = 0.05, 0.10, 0.15, 0.20$, and 0.25 . We let the K -neighbors for the neighbor filtration range from $K = 1, 3, 5$, and 10 . The minimum persistence threshold for calculation of the quantile lifetime and mean lifetime for the two filtrations was $\tau = 0.001, 0.01$, and 0.1 . Graphical representations of the simulated datasets and resultant prototypes are provided as figures in the Appendix.

Results for each baseline classifier model are presented in Table (2). Under optimal configurations, the linear SVM baseline has the highest average across all datasets improvement in G-Mean (+1.98%) and the highest reduction percentage (80.21%). The 1-NN baseline also shows positive average G-Mean improvement (+1.06%) with 78.04% reduction. Both 3- and 5-nearest neighbor baselines show only a slight decreases in average G-Mean (−0.60% and −0.86% respectively), with high reduction percentages (76.36% and 76.04%). The prototype set selected by TPS seems to maintain underlying features of the original dataset, as evidenced by the prototype ratios being imbalanced and visualizations showing class overlap. Ideal configurations vary wildly depending on the dataset and model used for evaluation, which might indicate that tuning is important for using TPS.

4.2 Parameter Effects on Prototype Selection

To understand the effects of the hyperparameters q , K and τ_{min} on the prototype selection algorithm, we perform stratified 10-fold cross validation on a simulated binary classification clusters of points normally distributed with $\sigma_1 = \sigma_2 = 1.0$ for each class about vertices of an 4-dimensional hypercube with sides of length 2. The data is generated in such a way that the imbalance ratio is 4.0. The simulated dataset has 2500 observations generated from $\mathbb{R}^4$.

To assess the effects of prototype selection on classification under class imbalance, we chose just to use the 1-NN baseline model due to its non-robust nature in this data setting. For each fold, a 1-NN model was trained on the entire training set. TPS was applied, and a new 1-NN model was trained on just the prototype set. Classification evaluation was done using G-Mean, with results being recorded.

We use $q = 0.05$, $K = 1$, and $\tau_{min} = 0.001$ as the baseline parameter value for TPS. We vary only a single variable of q , K , or τ_{min} at a time and compare it to the baseline. Configurations of TPS were considered with the range of possible values being:

- $q = 0.05, 0.10, 0.15, 0.20, 0.25$, and 0.50
- $K = 1, 28, 56, 84, 111, 139, 167, 194, 222$, and 250
- $\tau_{min} = 0.001, 0.01, 0.05, 0.10, 0.25, 0.50$ and 10.0 .

Table 2: Performance comparison of TPS against baseline classifiers on simulated datasets.

Dataset	Δ G-Mean (%)	Prototypes (#)	Reduction (%)	Prototype Ratio	Best Config (q, K, τ_{min})
<i>1-Nearest Neighbor Baseline</i>					
Well-Separated Blobs (3 classes)	0.00	175	58.33	53:38:84 (30.3%, 21.7%, 48.0%)	(0.05, 1, 0.001)
Overlapping Blobs (4 classes)	-1.54	134	76.07	31:30:39:34 (23.1%, 22.4%, 29.1%, 25.4%)	(0.25, 5, 0.001)
Overlapping Clusters (3 classes)	8.10	79	81.19	17:31:31 (21.5%, 39.2%, 39.2%)	(0.1, 1, 0.1)
Two Moons (Moderate Noise)	-2.00	55	84.29	32:23 (58.2%, 41.8%)	(0.15, 3, 0.001)
Two Moons (High Noise)	0.04	66	81.14	29:37 (43.9%, 56.1%)	(0.2, 1, 0.001)
Concentric Circles (Moderate Noise)	-0.65	59	83.14	28:31 (47.5%, 52.5%)	(0.1, 5, 0.001)
Concentric Circles (High Noise)	1.59	116	66.86	56:60 (48.3%, 51.7%)	(0.25, 1, 0.1)
Imbalanced Classes (80/20%)	3.22	74	78.86	58:16 (78.4%, 21.6%)	(0.2, 10, 0.001)
Mixed Multi-class (3 classes, 40/30/10%)	0.77	42	92.50	17:16:9 (40.5%, 38.1%, 21.4%)	(0.05, 1, 0.001)
<i>3-Nearest Neighbors Baseline</i>					
Well-Separated Blobs (3 classes)	0.00	175	58.33	53:38:84 (30.3%, 21.7%, 48.0%)	(0.05, 1, 0.001)
Overlapping Blobs (4 classes)	-4.95	135	75.89	33:29:36:37 (24.4%, 21.5%, 26.7%, 27.4%)	(0.25, 10, 0.001)
Overlapping Clusters (3 classes)	1.66	94	77.62	28:35:31 (29.8%, 37.2%, 33.0%)	(0.25, 1, 0.001)
Two Moons (Moderate Noise)	-3.34	102	70.86	52:50 (51.0%, 49.0%)	(0.25, 10, 0.001)
Two Moons (High Noise)	-3.31	66	81.14	29:37 (43.9%, 56.1%)	(0.2, 1, 0.001)
Concentric Circles (Moderate Noise)	0.00	105	70.00	54:51 (51.4%, 48.6%)	(0.2, 1, 0.001)
Concentric Circles (High Noise)	3.19	116	66.86	56:60 (48.3%, 51.7%)	(0.25, 1, 0.1)
Imbalanced Classes (80/20%)	1.00	21	94.00	17:4 (81.0%, 19.0%)	(0.05, 1, 0.001)
Mixed Multi-class (3 classes, 40/30/10%)	0.38	42	92.50	17:16:9 (40.5%, 38.1%, 21.4%)	(0.05, 1, 0.001)
<i>5-Nearest Neighbors Baseline</i>					
Well-Separated Blobs (3 classes)	0.00	175	58.33	53:38:84 (30.3%, 21.7%, 48.0%)	(0.05, 1, 0.001)
Overlapping Blobs (4 classes)	-4.08	134	76.07	39:29:34:32 (29.1%, 21.6%, 25.4%, 23.9%)	(0.25, 1, 0.001)
Overlapping Clusters (3 classes)	-0.32	79	81.19	17:31:31 (21.5%, 39.2%, 39.2%)	(0.1, 1, 0.1)
Two Moons (Moderate Noise)	-4.04	57	83.71	29:28 (50.9%, 49.1%)	(0.15, 10, 0.001)
Two Moons (High Noise)	-3.31	66	81.14	29:37 (43.9%, 56.1%)	(0.2, 1, 0.001)
Concentric Circles (Moderate Noise)	0.00	105	70.00	54:51 (51.4%, 48.6%)	(0.2, 1, 0.001)
Concentric Circles (High Noise)	0.56	107	69.43	56:51 (52.3%, 47.7%)	(0.2, 1, 0.1)
Imbalanced Classes (80/20%)	3.18	68	80.57	52:16 (76.5%, 23.5%)	(0.2, 1, 0.001)
Mixed Multi-class (3 classes, 40/30/10%)	0.29	90	83.93	39:33:18 (43.3%, 36.7%, 20.0%)	(0.15, 1, 0.001)
<i>Linear SVM Baseline</i>					
Well-Separated Blobs (3 classes)	0.00	175	58.33	53:38:84 (30.3%, 21.7%, 48.0%)	(0.05, 1, 0.001)
Overlapping Blobs (4 classes)	-2.56	132	76.43	34:30:37:31 (25.8%, 22.7%, 28.0%, 23.5%)	(0.25, 3, 0.001)
Overlapping Clusters (3 classes)	-1.11	49	88.33	17:16:16 (34.7%, 32.7%, 32.7%)	(0.1, 1, 0.001)
Two Moons (Moderate Noise)	-1.32	73	79.14	37:36 (50.7%, 49.3%)	(0.2, 10, 0.001)
Two Moons (High Noise)	0.00	57	83.71	29:28 (50.9%, 49.1%)	(0.2, 10, 0.001)
Concentric Circles (Moderate Noise)	7.10	132	62.29	65:67 (49.2%, 50.8%)	(0.25, 10, 0.001)
Concentric Circles (High Noise)	7.60	28	92.00	14:14 (50.0%, 50.0%)	(0.05, 5, 0.001)
Imbalanced Classes (80/20%)	8.44	21	94.00	17:4 (81.0%, 19.0%)	(0.05, 1, 0.001)
Mixed Multi-class (3 classes, 40/30/10%)	-0.29	69	87.68	27:22:20 (39.1%, 31.9%, 29.0%)	(0.1, 5, 0.001)

Increasing each of the three variables increases the number of prototypes selected. For q specifically, we see in Figure (5) that the cardinality of the final prototype set increases at what appears to be a linear rate. Using the median quantile ($q = 0.50$) results in just above 900 total prototypes from the 2250 training observations ($\approx 59\%$ reduction). G-Mean increases as q and the cardinality increases, but not at the same rate. A lower q threshold focuses on regions closest to the decision boundary, as points far from the boundary are less likely to appear in the neighbor filtration slice. Results seem to indicate that geometrically, lower q values introduce a form of regularization that suppresses local fluctuations and topological noise, smoothing the underlying density estimate [28]. This can potentially lead to not only a cleaner, more stable decision boundary, but also gains to G-Mean performance.

Increasing K also increases the number of prototypes selected as seen in Figure (6). However, the effect is much more gradual than increasing q . Additionally, increasing K does not lead to a significant gain in G-Mean performance for the model trained on the resultant prototypes compared to the baseline; in fact, the performance when tuning just K lead to worse performance than the baseline. Larger values of K look at points that are further away from the initial target class points, and thus potentially further away from the local boundary between classes, obscuring local boundary topological information. This potentially leads to selection of prototypes that do not contribute as significantly to inter-class topology, and thus worse classification performance. We end by noting that tuning K is likely also data-dependent, as the boundary between two different datasets can be vastly different in geometric structure.

Figure (7) shows the effect of increasing τ_{min} on the cardinality of the selected prototype set. Initial increases to τ_{min} seem to have little effect compared to the baseline TPS model, however we see that the cardinality rises at an exponential rate as τ_{min} increases. When $\tau_{min} = 10.00$, we see the final prototype cardinality was around 1600 of the original 2250 training observations ($\approx 27\%$ reduction) with a G-Mean equal to the 1-nearest neighbor model

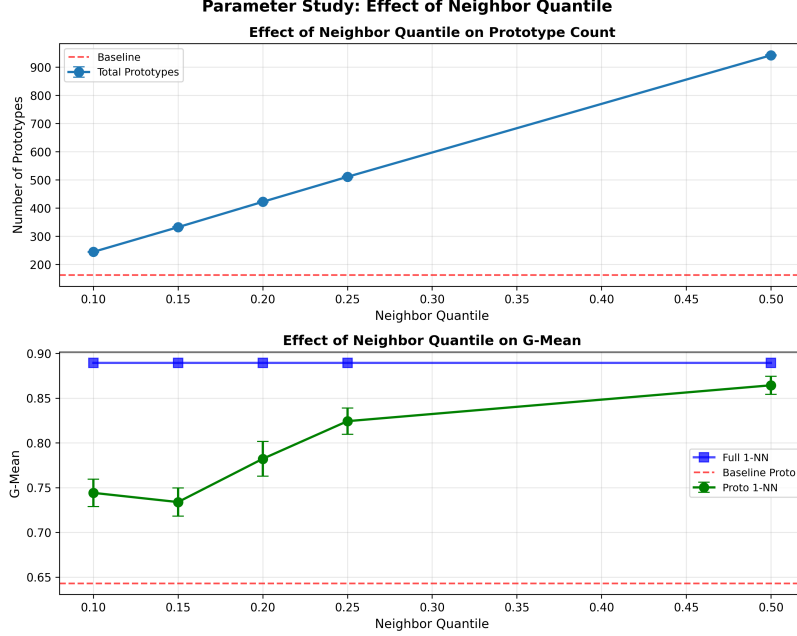


Figure 5: Effect of neighbor quantile hyperparameter (q) selection on prototype count and G-Mean compared to baseline TPS.

trained on the entire dataset. Parameter τ_{min} filters out potentially noisy persistence lifetimes before calculation of Equation (2). In essence a high τ_{min} will remove lifetimes considered as “topological noise” before calculation of the neighbor quantile slice. If too many lifetimes are removed, the selection of the q^{th} -quantile lifetime will be much higher than if those lifetimes remained. The inflated value acts as a deterrent to the geometric regularization effects of q as more points further away from the boundary are retained in the neighbor filtration slice and thus potentially become prototypes after the radius filtration.

Overall, we advocate for the following when tuning TPS hyperparameters:

- Use a lower value of K in order to better capture local boundary geometry.
- Tuning for both q and τ_{min} are more important than K for downstream model classification performance.
- Use lower values of τ_{min} if reduction percentage is a priority as it deters the geometric regularization of q .

A more extensive study into the effect of tuning both simultaneously is left for future work.

We close by looking at the effect of prototype selection on class imbalance under tuning each of these hyperparameters. Figure (44) found in the Appendix shows the distribution of Class 0 to Class 1 points selected as prototypes. Tuning all three hyperparameters results in a similar class imbalance structure as in the original dataset. This highlights a possible advantage of TPS over other prototype selection methods, such as condensation and edition methods, as many alternative prototype selection methods balance classes. Balancing class sizes improves performance of downstream modeling, but can be undesirable when the primary goal is data condensation for memory storage. In such cases, keep the original class imbalance and overlap structure can be important for later analysis. Additionally, methods that balance classes must retain a disproportionately large number of prototypes from minority classes, which are not representative of the original data structure.

4.3 Time Trials for TPS

A potential key computational bottleneck remains in the combinatorial calculation of persistent homology. This opens a potential criticism to TDA-based prototype selection over other distance-based or optimization-based selection methods. Ongoing research into more efficient and stable algorithms for computing persistence homology have been promising, however [29, 30]. To assess the computational time performance of TPS, 2500 total observations

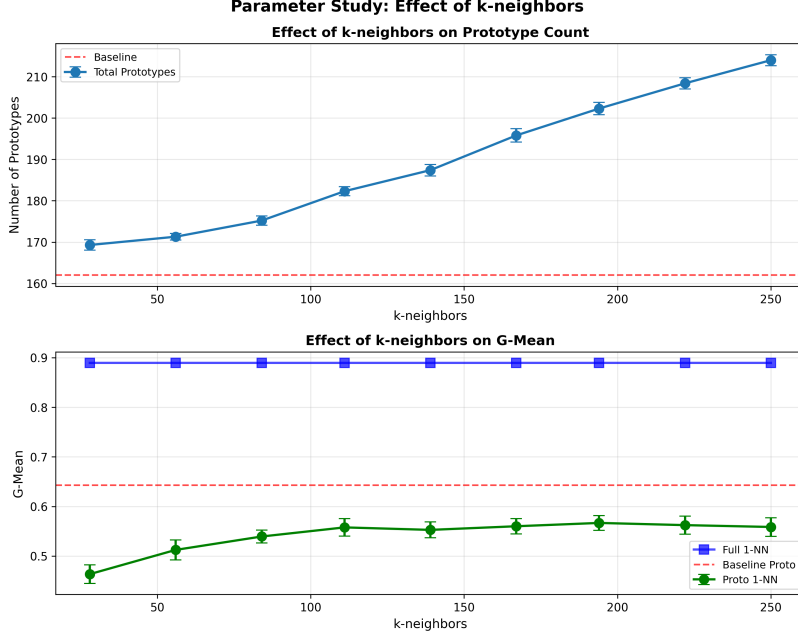


Figure 6: Effect of K neighbors selected for the neighbor filtration hyperparameter on prototype count and G-Mean compared to baseline TPS.

were simulated from a multivariate Gaussian distribution in $\mathbb{R}^4$. TPS was applied to the simulated dataset and the selection time was recorded in seconds (s). We used a total of 8 different configurations for TPS when calculating prototypes, $q = 0.05, 0.25$, $K = 1, 10$, and $\tau_{min} = 0.0001, 0.1$. We repeated this for 100 iterations. Afterwards, the mean and standard error (SE) were recorded, and the 95% confidence intervals for each configuration were calculated.

The results are presented in Table (3). Larger values of q , which result in more points being selected during the neighbor filtration, result in longer computational times. Given earlier results, higher q (and by extension much higher τ_{min}) will lead to a longer computational time. Using TPS when calculating higher homology groups $H_k(\mathcal{X})$, potentially will have similar effects as the neighbor filtration will select more potential points before doing the radius filtration.

From a practical standpoint, TPS may not yet offer complete end-to-end computational savings in all contexts (especially if the dataset cardinality is quite large). However, we note that at the time of writing prototypes are calculated for each class sequentially. There is opportunity to speed up TPS by running prototype selection of classes in parallel.

q	K	τ_{min}	Mean (s)	SE (s)	95% CI	
					Lower	Upper
0.05	1	0.001	1.9987	0.0142	1.9704	2.0269
0.05	1	0.100	1.9428	0.0128	1.9174	1.9682
0.05	10	0.001	1.8252	0.0026	1.8200	1.8304
0.05	10	0.100	1.8869	0.0087	1.8697	1.9042
0.25	1	0.001	2.0665	0.0086	2.0494	2.0835
0.25	1	0.100	2.0314	0.0086	2.0144	2.0484
0.25	10	0.001	1.9974	0.0054	1.9867	2.0080
0.25	10	0.100	2.1008	0.0572	1.9873	2.2143

Table 3: Prototype selection computational time results across simulated Gaussian data for different configurations.

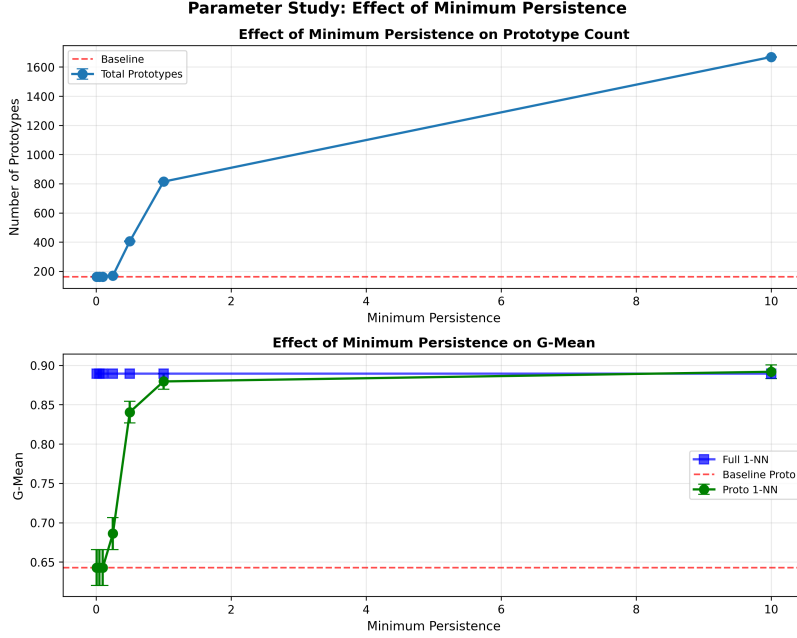


Figure 7: Effect of the minimum persistence required before calculation of neighbor filtration lifetime quantile (τ_{min}) hyperparameter selection on prototype count and G-Mean compared to baseline TPS.

4.4 Performance on Real Datasets

We benchmark the performance of TPS on eight datasets obtained from the UCI Machine Learning Repository [31]. Detailed characteristics of each dataset, including number of features, classes, and imbalance ratios, are provided in Table (4). The selected datasets are: Iris, Wine, Breast Cancer Wisconsin (Wisconsin), Glass Identification (Glass), Pima Indians Diabetes (Pima), Liver Disorders (BUPA), Cleveland Heart Disease (Cleveland), and Optical Recognition of Handwritten Digits (Digits). These datasets were chosen for their diversity in size, number of classes, feature dimensionality, class distribution, and real-world domain representation.

Each dataset was split into 70/30% training and testing split using stratified random sampling. Baseline 1-NN, 3-NN, 5-NN, and linear kernel SVM were trained on the training set. TPS, CNN+ENN, AllKNN, K -Means, and the BienTib prototype selectors were applied to the training data. Grid searches were conducted for each hyperparameter configurations to determine the best performing prototype set. Respective models were created for each of the resultant prototype set. Both baseline and prototype-trained models were evaluated on the 30% testing data using G-Mean. Optimal configuration for each was considered to be the one that caused the smallest change in G-Mean difference; in the event of a tie, the one that also had the largest reduction percentage was used. Results are presented in Table (5).

For each of the real datasets, TPS achieves a reduction percentage around 60 – 85%. In the Iris, Wisconsin, BUPA, and Cleveland datasets TPS had the highest average reduction percentage across all different considered baseline models. For TPS, the lowest average reduction across experiments was Digits (49.6%) with the largest reduction being Wisconsin (86.8%). On average across all experiments, both TPS and BienTib resulted in an average increase in G-Mean (+0.013 and +0.020 respectively) compared to the baseline models. Overall, both achieved a positive G-Mean difference in 22 out of 32 (69%) of the tested experiments. However, it is worth noting that TPS prototypes had a larger reduction than BienTib prototype in all but the Wine dataset, and were computed faster in all scenarios.

We also note disparity in performance of TPS versus other heuristic prototype selection methods such as CNN+ENN and AllKNN, with TPS performing better than both. We believe the reason for the difference in performance is TPS is only concerned with how the topology changes, not about the geometric scaling of the different features, which can vary wildly within datasets. Since the datasets were not first preprocessed by normalizing or scaling, other distance-only based methods such as CNN+ENN and AllKNN will be dominated by features with larger scales, making

the algorithms essentially blind to smaller-scale features when selecting prototypical points.

We provide below a quick summary of the average performances of TPS against all other prototype selection methods:

- TPS vs CNN+ENN:
 - TPS reduction percentage average of 69.3% was higher than CNN+ENN’s 59.2%.
 - TPS resulted in higher average G-Mean difference (+0.013 vs −0.040).
 - TPS is generally 2 – 3x faster on medium/large datasets.
- TPS vs. AllKNN:
 - TPS was far more aggressive with reduction percentage compared to AllKNN (24.5%).
 - TPS resulted in higher average G-Mean difference compared to AllKNN (0.001).
 - AllKNN was faster on smaller datasets, but was comparable to TPS on larger datasets.
- TPS vs. BienTib:
 - TPS had a higher reduction percentage on average compared to BienTib prototypes (49.8%).
 - TPS resulted in lower average G-Mean difference compared to BienTib (+0.020).
 - TPS was dramatically faster to compute prototypes (e.g. on the Digits dataset 0.19s vs 41s).
- TPS vs. K -Means:
 - TPS reduction percentage is higher on average than K -Means (59.3%).
 - TPS and K -Means resulted in the same average G-Mean difference.
 - TPS generally is faster, especially on small datasets.

Persistent homology is known to be *metric dependent*, meaning that using different metrics on the same dataset can lead to different persistence results [32]. Therefore while many prototype selection methods are reliant only on distance calculations, we hypothesize that TPS will be more effective in leveraging the new topological information gleaned in certain topological spaces where different metrics are more appropriate for learning. One such example is the use of cosine similarity over Euclidean distance for the analysis of text data [33], or using Manhattan distance over Euclidean distance in high dimensional spaces [34].

To test TPS performance against other prototype selection methods under different metrics we performed prototype selection on a collection of text data from the Kaggle Dataset repository [35]. The dataset consists of strings of text that are either classified as “ham” messages or “spam” messages with an imbalance ratio of 3.0. To process text data we must first convert it to a numerical form through *vectorization*. While there are many different methods for vectorization, we chose to use Doc2Vec vectorization as it creates dense vector embeddings. Doc2Vec is often used in spam classification tasks because unlike traditional bag-of-words approaches that discard word order and semantic relationships, Doc2Vec better learns distributed representations by predicting words in a document given both the word context and a unique document identifier [36, 37]. This effectively captures semantic and syntactic patterns within the text. Further information of Doc2Vec can be found in [37].

After vectorization, the data is split into a 70/30% stratified sample, and the same baseline classification models as in the real data study are calculated. TPS, CNN+ENN, AllKNN, K -Means, and the Bien-Tibshirani prototypes are calculated on the 70% training data, and a new model is trained on the prototype sets. Both the baseline and prototype trained models are evaluated on the remaining testing dataset. This process is repeated for both Euclidean and cosine similarity metrics were appropriate. The optimal configuration was considered to be the

Dataset	Number of Classes	Number of Observations	Features	Class Distribution	Imbalance	Avg. CV
Iris	3	150	4	50:50:50	1.00	0.346
Wine	3	178	13	59:71:48	1.48	0.294
Wisconsin	2	569	30	212:357	1.68	0.486
Glass	6	214	9	70:76:17:13:9:29	8.44	0.773
Pima	2	768	8	500:268	1.87	0.618
BUPA	2	345	6	145:200	1.38	0.558
Cleveland	2	303	13	164:139	1.18	0.752
Digits	10	1797	64	178:182:177:183:181:182:181:179:174:180	1.05	4.291

Table 4: Descriptions of the real datasets used for testing prototype selection performance of TPS.

Table 5: Real Data Experimental Results Across Multiple Classifiers

Dataset	Method	1-NN				3-NN				5-NN				Linear SVM			
		N_p	Red.%	ΔG	Time	N_p	Red.%	ΔG	Time	N_p	Red.%	ΔG	Time	N_p	Red.%	ΔG	Time
Iris	Baseline (G-Mean)			0.928				0.953				0.977				1.000	
	TPS	38	63.8	+0.025	0.006	29	72.4	+0.024	0.007	29	72.4	+0.000	0.006	31	70.5	-0.023	0.005
	CNN+ENN	39	62.9	-0.021	0.028	43	59.0	-0.953	0.031	43	59.0	-0.977	0.029	39	62.9	-0.093	0.028
	AIKNN	100	4.8	+0.049	0.004	100	4.8	+0.024	0.004	100	4.8	+0.000	0.004	100	4.8	+0.000	0.003
	BienTib	3	97.1	+0.003	0.142	74	29.5	+0.000	0.105	74	29.5	-0.049	0.101	74	29.5	+0.000	0.106
	KMeans	20	81.0	+0.049	0.012	30	71.4	+0.002	0.016	100	4.8	+0.000	0.046	9	91.4	+0.000	0.024
Wine	Baseline (G-Mean)			0.693				0.632				0.708				0.938	
	TPS	17	86.3	+0.098	0.008	27	78.2	+0.124	0.007	27	78.2	+0.062	0.009	32	74.2	-0.040	0.006
	CNN+ENN	41	66.9	+0.047	0.056	37	70.2	+0.093	0.060	37	70.2	+0.016	0.064	35	71.8	-0.025	0.054
	AIKNN	81	34.7	+0.071	0.004	81	34.7	+0.152	0.004	81	34.7	+0.055	0.005	92	25.8	-0.055	0.004
	BienTib	4	96.8	+0.070	0.105	12	90.3	+0.084	0.096	33	73.4	+0.016	0.102	33	73.4	-0.057	0.098
	KMeans	9	92.7	+0.046	0.011	9	92.7	+0.092	0.011	100	19.4	+0.036	0.057	50	59.7	+0.021	0.049
Wisconsin	Baseline (G-Mean)			0.920				0.912				0.917				0.939	
	TPS	76	80.9	-0.016	0.062	55	86.2	-0.010	0.058	55	86.2	-0.009	0.062	40	89.9	-0.032	0.090
	CNN+ENN	170	57.3	+0.001	0.033	166	58.3	+0.014	0.032	170	57.3	+0.009	0.032	166	58.3	-0.001	0.065
	AIKNN	365	8.3	-0.014	0.005	365	8.3	-0.015	0.006	365	8.3	-0.010	0.006	365	8.3	-0.011	0.217
	BienTib	37	90.7	-0.006	0.508	110	72.4	+0.002	0.576	110	72.4	+0.014	0.498	109	72.6	-0.030	0.544
	KMeans	200	49.7	-0.020	0.134	200	49.7	+0.000	0.135	200	49.7	-0.004	0.134	50	87.4	+0.022	0.037
Glass	Baseline (G-Mean)			0.636				0.509				0.494				0.014	
	TPS	32	78.5	+0.007	0.012	27	81.9	+0.105	0.051	64	57.0	-0.008	0.009	64	57.0	+0.650	0.010
	CNN+ENN	10	93.3	-0.623	0.024	10	93.3	-0.509	0.025	50	66.4	-0.135	0.098	8	94.6	-0.005	0.033
	AIKNN	86	42.3	-0.624	0.005	86	42.3	-0.496	0.006	93	37.6	-0.054	0.004	73	51.0	-0.014	0.006
	BienTib	60	59.7	+0.092	0.199	122	18.1	+0.010	0.190	136	8.7	+0.028	0.222	122	18.1	-0.001	0.188
	KMeans	100	32.9	+0.070	0.060	100	32.9	-0.004	0.057	100	32.9	+0.004	0.059	50	66.4	-0.001	0.035
Pima	Baseline (G-Mean)			0.622				0.636				0.853				0.649	
	TPS	180	66.5	+0.053	0.106	180	66.5	+0.048	0.106	155	71.1	-0.065	0.110	180	66.5	+0.073	0.121
	CNN+ENN	268	50.1	+0.042	0.773	271	49.5	+0.056	0.722	267	50.3	-0.009	0.375	261	51.4	+0.055	1.162
	AIKNN	243	54.7	+0.035	0.011	243	54.7	+0.041	0.010	502	6.5	-0.014	0.006	332	38.2	+0.106	0.487
	BienTib	10	98.1	+0.008	0.658	137	74.5	+0.029	0.514	533	0.7	+0.000	0.672	10	98.1	+0.043	0.681
	KMeans	50	90.7	+0.034	0.048	100	81.4	+0.073	0.092	200	62.8	-0.022	0.147	50	90.7	+0.044	0.515
BUPA	Baseline (G-Mean)			0.635				0.575				0.800				0.698	
	TPS	80	66.8	+0.020	0.035	87	63.9	+0.055	0.028	46	80.9	-0.050	0.022	36	85.1	+0.030	0.023
	CNN+ENN	138	42.7	-0.023	0.197	138	42.7	+0.057	0.197	146	39.4	+0.055	0.094	126	47.7	+0.020	0.233
	AIKNN	101	58.1	-0.022	0.006	72	70.1	+0.040	0.007	228	5.4	-0.025	0.003	195	19.1	-0.050	0.093
	BienTib	232	3.7	+0.008	0.241	40	83.4	+0.028	0.189	228	5.4	+0.028	0.202	127	47.3	+0.031	0.194
	KMeans	50	79.3	-0.044	0.036	200	17.0	+0.035	0.129	200	17.0	+0.036	0.132	200	17.0	+0.009	0.151
Cleveland	Baseline (G-Mean)			0.589				0.652				0.901				0.813	
	TPS	42	80.2	+0.049	0.021	63	70.3	-0.013	0.019	67	68.4	-0.011	0.018	57	73.1	+0.001	0.025
	CNN+ENN	122	42.5	+0.076	0.151	127	40.1	+0.050	0.169	121	42.9	+0.000	0.106	126	40.6	+0.056	0.209
	AIKNN	99	53.3	+0.085	0.006	118	44.3	+0.022	0.004	181	14.6	+0.011	0.005	164	22.6	-0.066	0.041
	BienTib	10	95.3	+0.137	0.200	207	2.4	+0.009	0.195	69	67.5	+0.011	0.148	25	88.2	+0.019	0.207
	KMeans	200	5.7	+0.009	0.119	200	5.7	+0.009	0.117	100	52.8	-0.022	0.063	100	52.8	+0.014	0.143
Digits	Baseline (G-Mean)			0.986				0.986				0.986				0.975	
	TPS	712	43.4	-0.062	0.179	475	62.2	-0.064	0.216	558	55.6	-0.060	0.178	790	37.2	-0.058	0.201
	CNN+ENN	230	81.7	-0.069	0.170	230	81.7	-0.091	0.166	230	81.7	-0.105	0.142	207	83.5	-0.046	0.169
	AIKNN	1244	1.0	+0.000	0.015	1244	1.0	+0.000	0.017	1244	1.0	+0.000	0.013	1244	1.0	+0.006	0.029
	BienTib	1257	0.0	+0.000	40.704	1257	0.0	+0.000	41.573	1257	0.0	+0.000	41.011	1257	0.0	+0.000	41.146
	KMeans	200	84.1	-0.015	0.291	200	84.1	-0.026	0.289	200	84.1	-0.036	0.397	200	84.1	-0.022	0.313

one that caused the smallest decrease (or largest gain) in G-Mean from the baseline model. Since vectorization results in high dimensional vectors for each document, we allow TPS filtrations to be calculated over both H_0 and H_1 homology groups. For any prototype configuration to be considered optimal, we require at least a 10% reduction of the vectorized data. In the event of a tie, the one that also had the largest reduction percentage was used.

We note that for BienTib prototypes, the optimization requires L_p -metric balls and is incompatible with the cosine similarity metric [4]. There is a similar issue with the K -Means prototype selector, where the use of the cosine similarity metric requires transformations to spherical coordinate space and using a variation known as Spherical K -Means algorithms instead [38]. For both algorithms we report only the results under the Euclidean metric.

We compare each prototype selector against the respective baselines, and present the results in Table (6). We note that baseline performance was higher under the cosine similarity metric compared to the Euclidean. Under the Euclidean metric, TPS resulted in a decrease in G-Mean from the baseline models with above 50% reduction in all cases. However, both TPS and CNN+ENN resulted in higher G-means than the baseline models when using cosine similarity. The gains in G-mean for CNN+ENN were slightly higher under cosine similarity than TPS, however, we note that CNN+ENN had a significantly lower reduction percentage than TPS across all baseline models. AIKNN showed only a decrease from baseline G-mean across all models and metrics, and all reductions were lower than 25%.

Under the Euclidean distance metric, K -Means performed worse than BienTib. Under the Euclidean metric, BienTib prototypes yielded significantly higher G-Mean improvements over baselines compared to TPS prototypes with higher reduction percentages. However, comparing TPS prototypes using cosine similarity to the Euclidean BienTib, we see that the TPS prototypes resulted in a larger increase in baseline G-Means and a greater reduction percentage. These results show that TPS is indeed not robust to the selection of metric, as changing the metric will change the

underlying inter- and intra-class topological structures. However, TPS is able to better leverage certain metrics to improve performance past other heuristic, distance based prototype selectors or other prototype selectors that require certain metrics (or metric properties).

Table 6: Comprehensive comparison of prototype selection methods across different classifiers

Method	Metric	G-Mean				Prototypes (% Reduction)			
		1-NN	3-NN	5-NN	SVM	1-NN	3-NN	5-NN	SVM
Baseline	Euclidean	0.845	0.825	0.814	0.819	-	-	-	-
	Cosine	0.865	0.875	0.857	0.819	-	-	-	-
TPS	Euclidean	0.771	0.797	0.759	0.714	140 (90.0)	332 (76.3)	362 (74.1)	639 (54.4)
	Cosine	0.872	0.892	0.869	0.852	456 (67.4)	127 (90.9)	96 (93.1)	529 (62.2)
CNN+ENN	Euclidean	0.851	0.851	0.847	0.892	470 (66.4)	436 (68.9)	450 (67.9)	458 (67.3)
	Cosine	0.879	0.891	0.876	0.892	385 (72.5)	373 (73.4)	339 (75.8)	458 (67.3)
AllKNN	Euclidean	0.844	0.840	0.826	0.859	1204 (14.0)	1101 (21.4)	1237 (11.6)	1069 (23.6)
	Cosine	0.771	0.746	0.752	0.859	1242 (11.3)	1217 (13.1)	1237 (11.6)	1069 (23.6)
BienTib	Euclidean	0.860	0.856	0.867	0.886	282 (79.9)	260 (81.4)	280 (80.0)	271 (80.6)
	Cosine	-	-	-	-	-	-	-	-
K-Means	Euclidean	0.818	0.799	0.791	0.802	200 (85.7)	200 (85.7)	200 (85.7)	200 (85.7)
	Cosine	-	-	-	-	-	-	-	-

5 Conclusions

The selection of topologically relevant prototypes is a promising novel area of research. In this paper, we proposed a topological data analysis (TDA)-based framework for selecting prototypes which minimize classification performance loss. The fundamental idea is using sequentially calculated filtrations to identify a meaningful bifiltration sub-complex that best represents the underlying dataset inter- and intra-class topologies. Once such a sub-complex is recovered, the prototype points are taken to be the resultant vertex set.

To verify the topological prototype selector (TPS) we used different simulated and real data settings. In simulated studies,

- 1-NN baseline showed positive average G-Mean percent improvement (+1.98%) with a reduction percentage of 80.21% on average.
- Both 3-NN and 5-NN baselines showed high reduction percentages (76.36% and 76.04%, respectively) with moderate percent decreases (−0.60% and −0.86% respectively) on average.
- The linear kernel SVM baseline has the highest average improvement in G-Mean (+1.98%) and the highest reduction percentage (80.21%) on average.

A Monte Carlo simulation study showed that hyperparameter tuning is important for both downstream model performance and cardinality of the resultant prototype set. TPS uses three hyperparameters: K which selects the number of non-target class observations to build the neighbor filtration, q which selects the q^{th} quantile lifetime to serve as the slice along the bifiltration, and τ_{min} which filters topologically noisy lifetimes. The study showed that both q and τ_{min} act as geometric regularization terms for the boundary-proximity principle, while K helps control locality of the inter-class topology. Tuning of q and τ_{min} allows users to find an optimal balance between classification performance and cardinality, adding a level of flexibility to TPS prototype selection. We advocate that tuning for both q and τ_{min} hyperparameters should take priority, with the selection of K being set arbitrarily low.

In real data settings, we compared TPS against other commonly used prototype selection algorithms. In summary,

- TPS reduction percentage average of 69.3% was higher than CNN+ENN’s 59.2% and TPS resulted in higher average G-Mean difference (+0.013 vs −0.040).
- TPS was far more aggressive with reduction percentage compared to AllKNN (24.5%), and TPS resulted in higher average G-Mean difference compared to AllKNN (0.001).
- TPS had a higher reduction percentage on average compared to BienTib prototypes (49.8%), but TPS resulted in lower average G-Mean difference compared to BienTib (+0.020).

- TPS reduction percentage is higher on average than K -Means (59.3%) with TPS and K -Means resulted in the same average G-Mean difference.

TPS is flexible in the choice of metric, while other optimization prototype selectors are not. Examples of this include the Bien-Tibshirani prototype selection method, which requires minimization of the set cover problem over L_p -metric balls, and K -Means. We note that other optimization-based prototype selectors such the selection of prototypes using optimal transport (SPOT) algorithm might not require specific L_p metrics, but rather metrics that exhibit characteristics such as submodularity to guarantee theoretical results [39].

Given this, we compared TPS to the other prototype selectors using text classification data where it is known the preferred metric for machine learning tasks is the cosine similarity, not Euclidean metric. Under the cosine similarity metric, TPS outperformed AllKNN, Bieñ-Tibshirani prototypes, and K -Means across both classification performance from baseline and reduction percentage. Both TPS and CNN+ENN improved G-Mean performance from baseline models, but TPS was able to do so with a significantly less prototype set cardinality. These results establish TPS as a particularly efficient prototype selection method for high-dimensional text embeddings, where computational cost and storage requirements are critical considerations.

Calculation for TPS can be performed under any homological group $H_k(\mathcal{X})$. As we noted earlier, when using higher k , potentially more points will be retained, so choice of homology group should be data-dependent and selected prior to prototype selection. Since TPS is based off of topological persistence, it is *label aware* and tends to preserve class imbalance structures in its selection of prototypes. This means that TPS is able to preserve boundary-informative samples without explicit rebalancing of classes or arbitrarily inflating the minority class representation, which is not always true for other heuristic methods such as CNN+ENN and AllKNN. While this may not be as beneficial for downstream classification performance, it is a benefit for storage of data where keeping these characteristics is important.

Throughout the paper we already discussed some research directions for topological-based prototype selection should explore. We conclude the paper by providing those recommendations, and other directions that we think are relevant, below.

- Currently, in the event of a tie during calculation of Equation (2), all persistence lifetimes are used for extracting potential vertex candidates. There is evidence of using, instead, the persistence interval with the longer birth time or some other similar method for breaking the tie [10]. Alternatively, instead of using Equation (7), alternative measures of centers should be considered and studied. Using median lifetime instead allows for a radius filtration lifetime selection that is robust to outlier lifetimes, which can potentially provide more meaningful topological information and a richer prototype set.
- Additional Monte Carlo studies should be considered to get a more extensive parameter study. Examples include hyperparameter tuning using simulated data with different data intrinsic characteristics, different baseline models, and under different metrics.
- Lastly, calculation of persistent homology is currently done using **Ripser** and the Vietoris Rips Complex. However, alternative simplicial complex constructions such as the Witness Complex which uses a small subset of “landmark” points and the remaining “witness” points to build the filtrations can potentially improve both downstream classification performance and dataset reduction.

6 Appendix

6.1 Algorithms

Algorithm 1 Topological Prototype Selector (TPS)

Input: Dataset $\mathcal{X}$, Bifiltration Prototype Selection required parameters θ

Output: Prototype dictionary $\mathcal{P}_{dict}$

```

 $\mathcal{C} \leftarrow$  unique classes in  $\mathcal{X}$ 
 $\mathcal{X}_{dict} \leftarrow \emptyset$ 
for each class  $c \in \mathcal{C}$  do
   $\mathcal{P}_c \leftarrow \text{BPS}(\mathcal{X}, c, \theta)$ 
   $\mathcal{P}_{dict}[c] \leftarrow \mathcal{P}_c$ 
end for
return  $\mathcal{P}_{dict}$ 

```

Algorithm 2 Extract Vertices from Persistent Features (ExtractVertices)

Require: Persistence Features $\mathcal{F} = \{(b_1, d_1, \ell_1), \dots, (b_m, d_m, \ell_m)\}$

Require: Vertex weights $\{w_i\}_{i=1}^n$, distance matrix $D \in \mathbb{R}^{n \times n}$

Require: Homology dimension h

Ensure: Vertex set $V \subseteq \{1, \dots, n\}$

```

1:  $V \leftarrow \emptyset$ 
2: for each  $(b, d, \ell) \in \mathcal{F}$  do
3:    $\epsilon_b \leftarrow 0.1 \cdot \ell$  ▷ Birth tolerance
4:    $\epsilon_d \leftarrow 0.1 \cdot \ell$  ▷ Death tolerance
5:   if  $h = 0$  then ▷ Connected components ( $H_0$ )
6:      $V \leftarrow V \cup \{i : |w_i - b| \leq \epsilon_b\}$  ▷ Vertices at birth
7:     for  $i = 1$  to  $n$  do
8:       for  $j = i + 1$  to  $n$  do
9:          $e_{ij} \leftarrow \max(D_{ij}, w_i, w_j)$  ▷ Edge filtration value
10:        if  $|e_{ij} - d| \leq \epsilon_d$  then
11:           $V \leftarrow V \cup \{i, j\}$  ▷ Vertices on merging edges
12:        end if
13:      end for
14:    end for
15:   else if  $h = 1$  then ▷ Loops ( $H_1$ )
16:      $V \leftarrow V \cup \{i : |w_i - b| \leq \epsilon_b\}$  ▷ Vertices at birth
17:     for  $i = 1$  to  $n$  do
18:       for  $j = i + 1$  to  $n$  do
19:          $e_{ij} \leftarrow \max(D_{ij}, w_i, w_j)$ 
20:         if  $|e_{ij} - b| \leq \epsilon_b$  then
21:            $V \leftarrow V \cup \{i, j\}$  ▷ Vertices forming the loop
22:         end if
23:       end for
24:     end for
25:   else ▷ Higher dimensional homology
26:      $V \leftarrow V \cup \{i : |w_i - b| \leq \epsilon_b\}$ 
27:   end if
28: end for
29: return  $V$ 

```

Algorithm 3 Bifiltration Prototype Selection (BPS) Algorithm**Require:** Training data $X \in \mathbb{R}^{n \times d}$, labels $y \in \{0, 1\}^n$, target class c **Require:** Parameters: K (neighbors), h (homology dimension), τ_{min} (min persistence), q (neighbor quantile)**Ensure:** Prototype indices $P \subseteq \{1, \dots, n\}$

```

1:  $X_{target} \leftarrow \{x_i : y_i = c\}$  ▷ Target class samples
2:  $X_{other} \leftarrow \{x_i : y_i \neq c\}$  ▷ Non-target class samples
3:  $n_{target} \leftarrow |X_{target}|$ 
4:
5:  $D \leftarrow$  compute pairwise distance matrix for  $X_{target}$  ▷  $n_{target} \times n_{target}$  distance matrix
6: for  $i = 1$  to  $n_{target}$  do
7:    $r_i \leftarrow \sum_{j \neq i} D_{ij}$  ▷ Radius: sum of all same-class distances
8: end for
9:
10: for  $i = 1$  to  $n_{target}$  do
11:    $d_i \leftarrow$  compute distance to other class  $d(x_i, X_{other})$ 
12:    $\{d_i^{(1)}, \dots, d_i^{(k)}\} \leftarrow$  Select  $K$  Nearest neighbors
13:    $n_i \leftarrow \sum_{j=1}^k d_i^{(j)}$  ▷ Neighbor: sum of  $K$  nearest other-class distances
14: end for
15:
16:  $D^{(n)} \leftarrow (D, \{n_i\})$  ▷ Edge  $(i, j)$  appears at  $\max(D_{ij}, n_i, n_j)$ 
17:  $PD^{(n)} \leftarrow$  Rips filtration on  $(D^{(n)}, h)$  ▷ Persistence diagram
18:  $\mathcal{F}^{(n)} \leftarrow \{(b, d, \ell) \in PD^{(n)} : d < \infty, \ell = d - b \geq \tau_{min}\}$  ▷ Lifetimes above minimum persistence threshold
19:
20:  $L^{(n)} \leftarrow \{\ell : (b, d, \ell) \in \mathcal{F}^{(n)}\}$  ▷ Lifetimes
21:  $\theta^{(n)} \leftarrow \text{Quantile}(L^{(n)}, q)$  ▷ Neighbor threshold
22:  $\mathcal{F}_{sel}^{(n)} \leftarrow \arg \min_{(b, d, \ell) \in \mathcal{F}^{(n)}} |\ell - \theta^{(n)}|$ 
23: ▷ Select feature(s) closest to threshold
24:
25:  $V^{(n)} \leftarrow \text{ExtractVertices}(\mathcal{F}_{sel}^{(n)}, \{n_i\}, D, h)$  ▷ Vertices participating in selected features
26:
27:  $\{r'_i\} \leftarrow \{r_i : i \in V^{(n)}\}$  ▷ Subset radius values
28:  $D' \leftarrow D[V^{(n)}, V^{(n)}]$  ▷ Subset distance matrix
29:  $D^{(r)} \leftarrow (D', \{r'_i\})$ 
30:
31:  $PD^{(r)} \leftarrow$  Rips filtration on  $(D^{(r)}, h)$ 
32:  $\mathcal{F}^{(r)} \leftarrow \{(b, d, \ell) \in PD^{(r)} : d < \infty, \ell = d - b \geq \tau_{min}\}$ 
33:  $L^{(r)} \leftarrow \{\ell : (b, d, \ell) \in \mathcal{F}^{(r)}\}$ 
34:  $\theta^{(r)} \leftarrow \text{Mean}(L^{(r)})$ 
35:  $\mathcal{F}_{sel}^{(r)} \leftarrow \arg \min_{(b, d, \ell) \in \mathcal{F}^{(r)}} |\ell - \theta^{(r)}|$ 
36:  $V_{local}^{(r)} \leftarrow \text{ExtractVertices}(\mathcal{F}_{sel}^{(r)}, \{r'_i\}, D', h)$ 
37:  $P \leftarrow$  indexes for vertices  $V_{local}^{(r)}$  in original dataset  $X$ 
38:
39: return  $P$  ▷ Final prototype indices

```

6.2 Figures

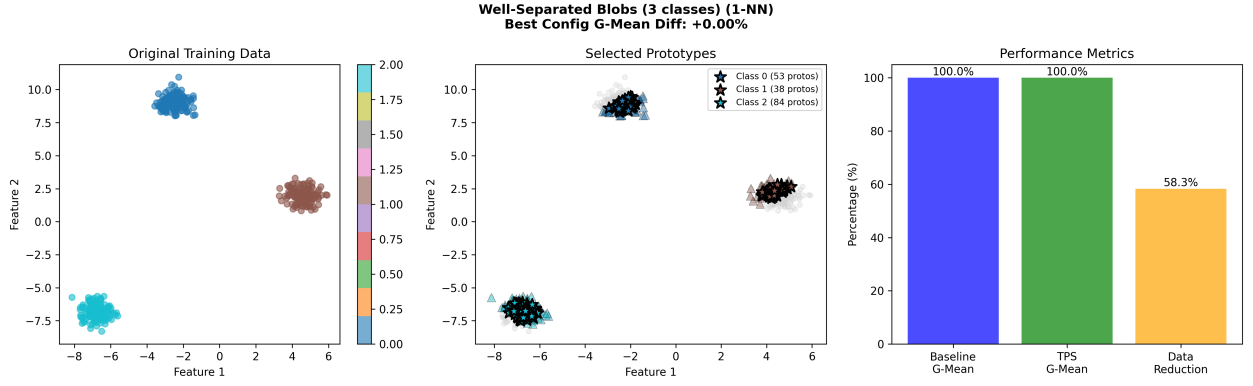


Figure 8: TPS selected prototypes on well-separated blobs simulated data for the 1-NN baseline.

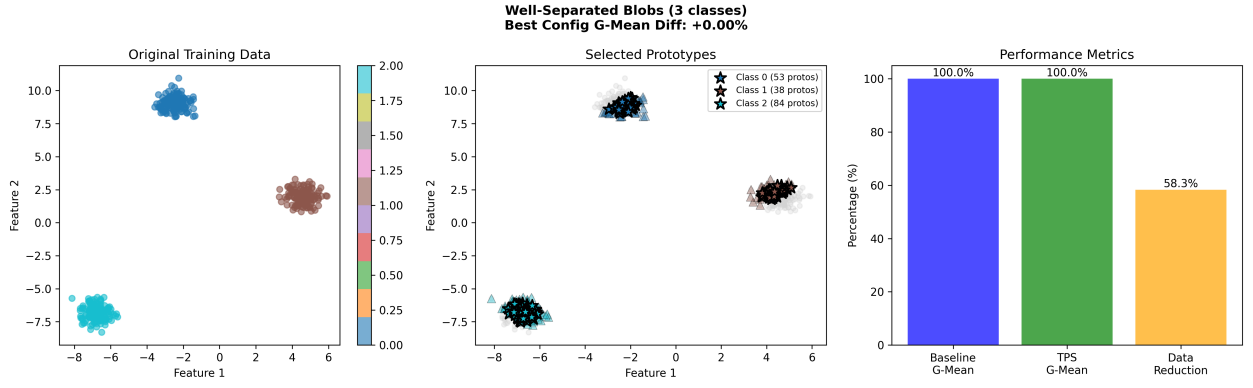


Figure 9: TPS selected prototypes on well-separated blobs simulated data for the 3-NN baseline.

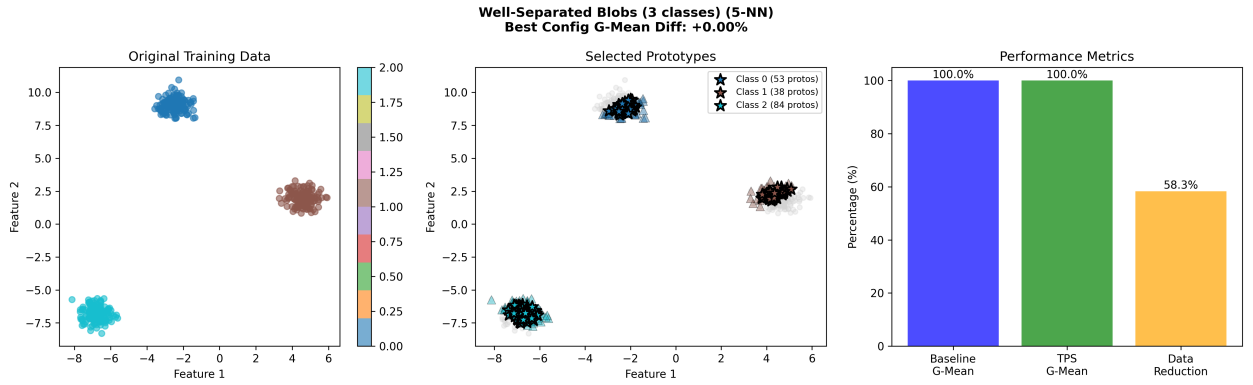


Figure 10: TPS selected prototypes on well-separated blobs simulated data for the 5-NN baseline.

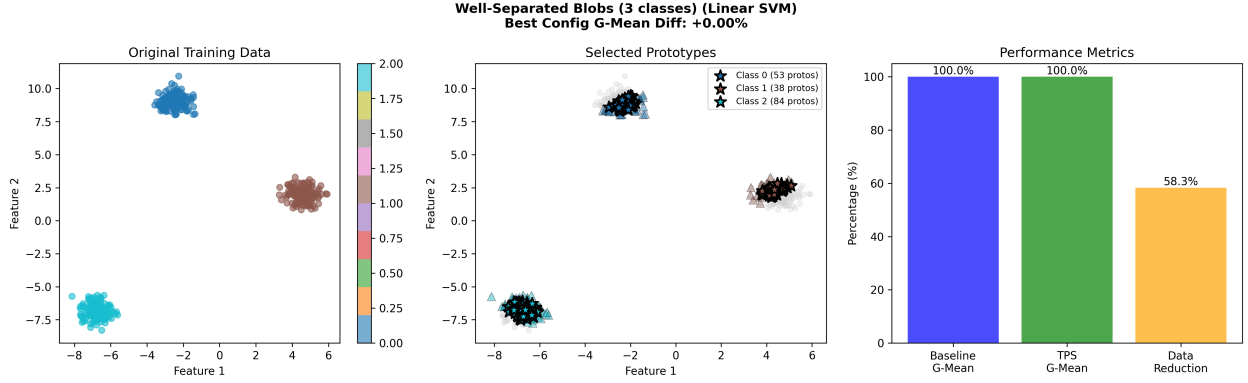


Figure 11: TPS selected prototypes on well-separated blobs simulated data for the SVM baseline.

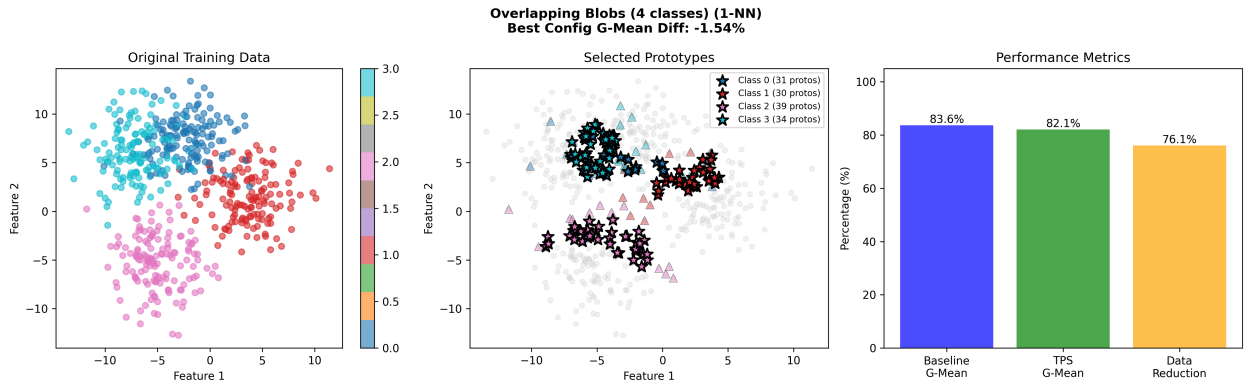


Figure 12: TPS selected prototypes on overlapping blobs simulated data for the 1-NN baseline.

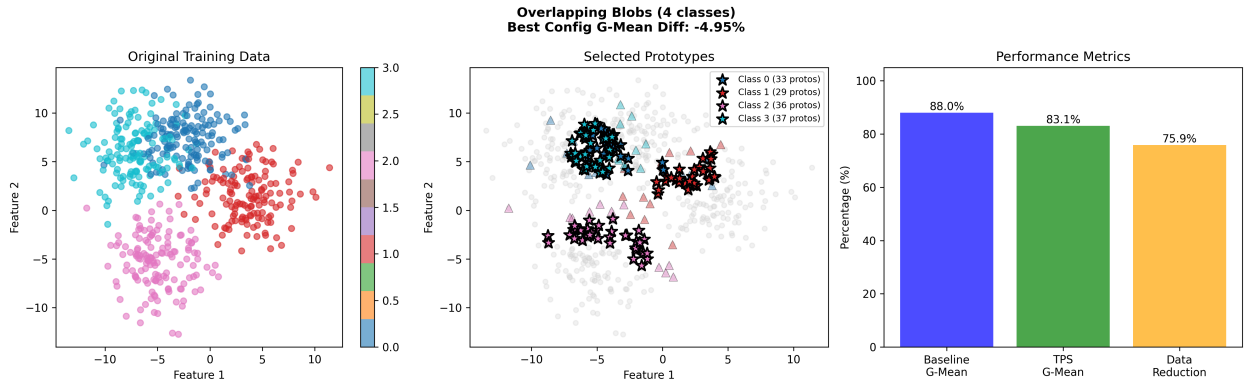


Figure 13: TPS selected prototypes on overlapping blobs simulated data for the 3-NN baseline.

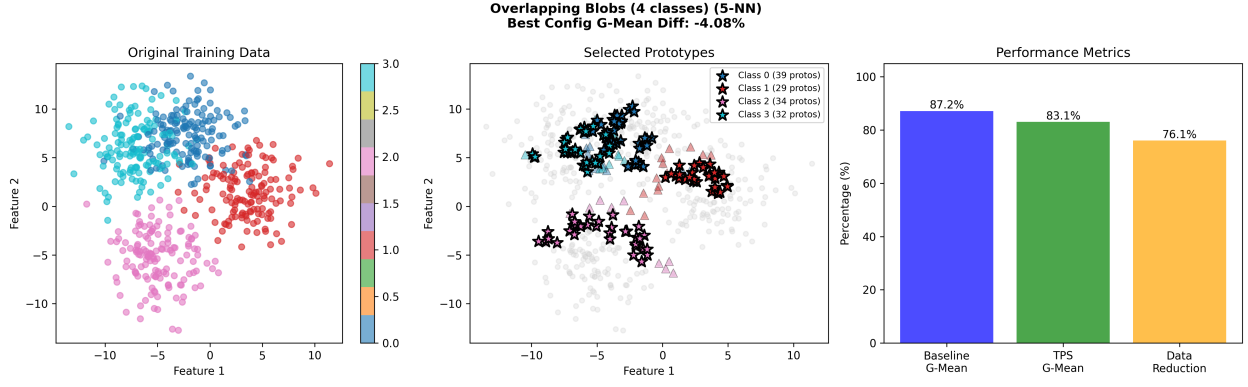


Figure 14: TPS selected prototypes on overlapping blobs simulated data for the 5-NN baseline.

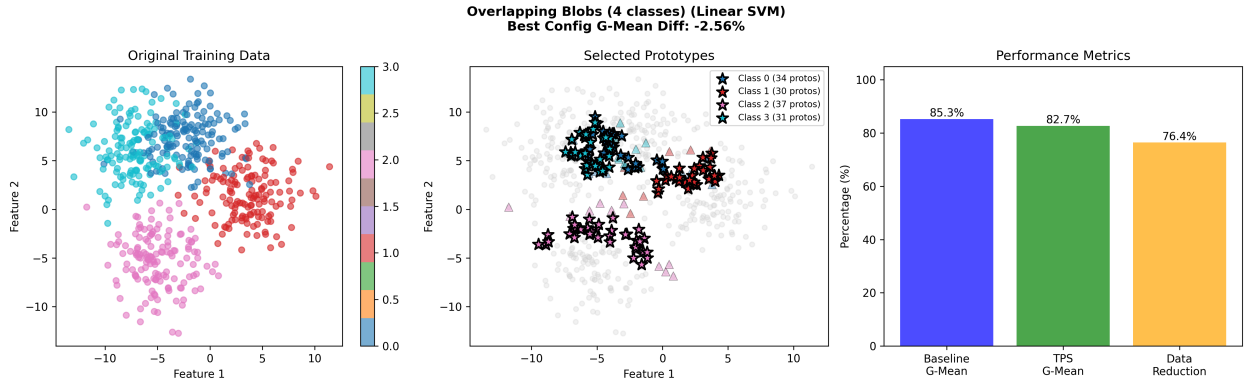


Figure 15: TPS selected prototypes on overlapping blobs simulated data for the SVM baseline.

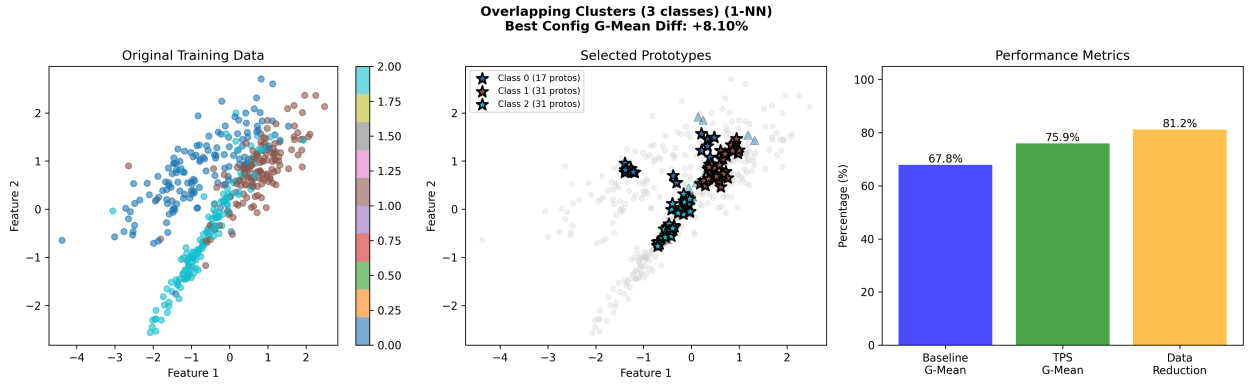


Figure 16: TPS selected prototypes on overlapping clusters simulated data for the 1-NN baseline.

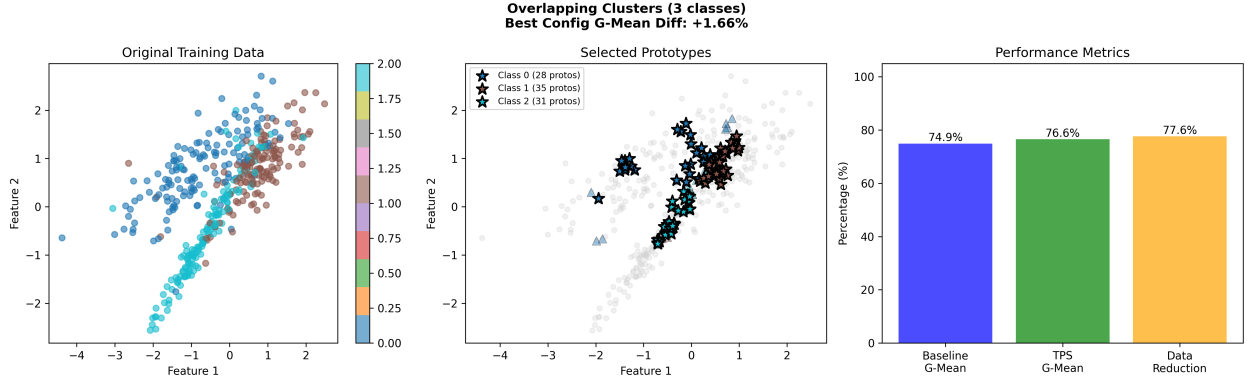


Figure 17: TPS selected prototypes on overlapping clusters simulated data for the 3-NN baseline.

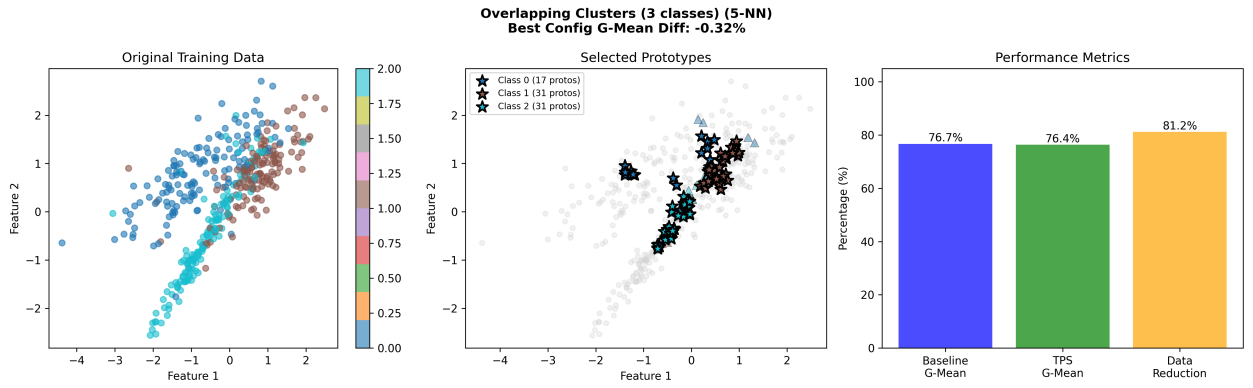


Figure 18: TPS selected prototypes on overlapping clusters simulated data for the 5-NN baseline.

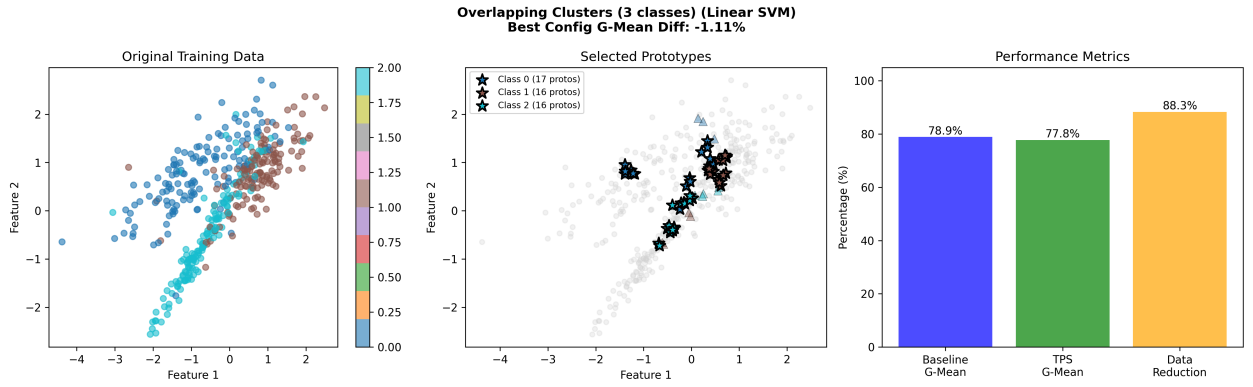


Figure 19: TPS selected prototypes on overlapping clusters simulated data for the SVM baseline.

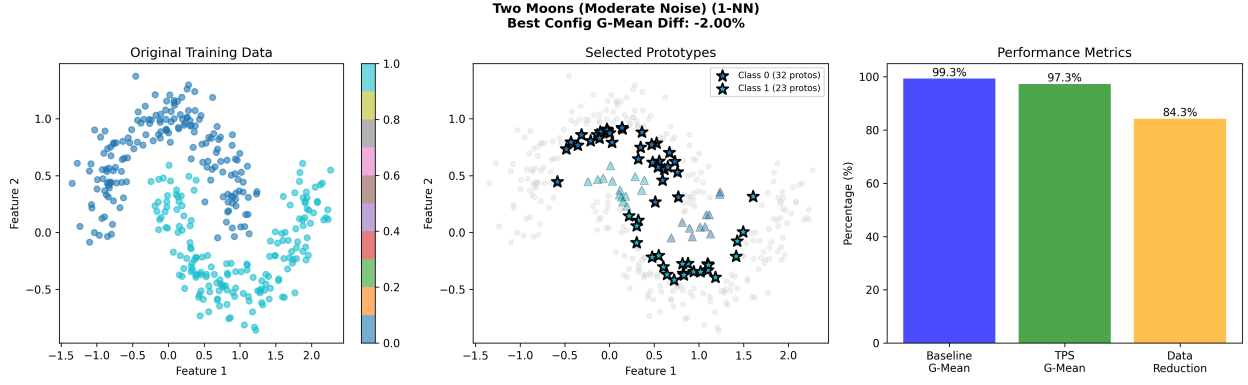


Figure 20: TPS selected prototypes on two moons with moderate noise simulated data for the 1-NN baseline.

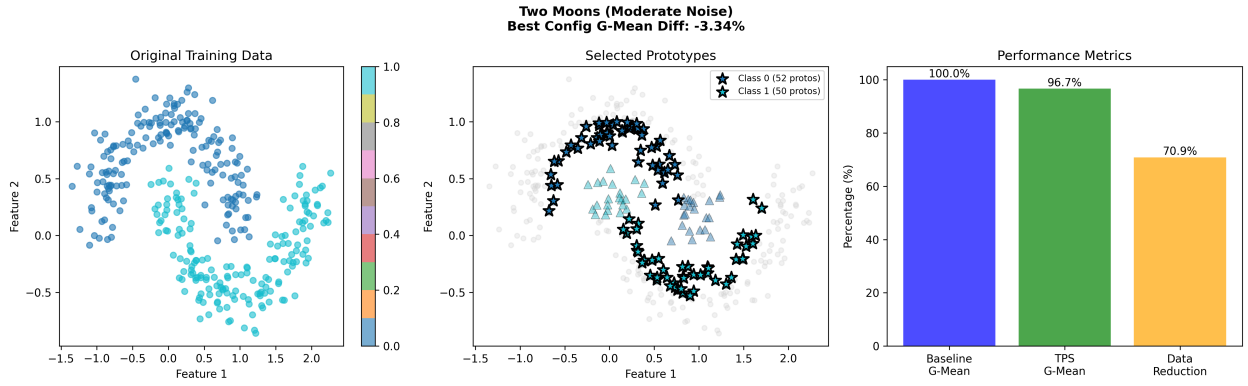


Figure 21: TPS selected prototypes on two moons with moderate noise simulated data for the 3-NN baseline.

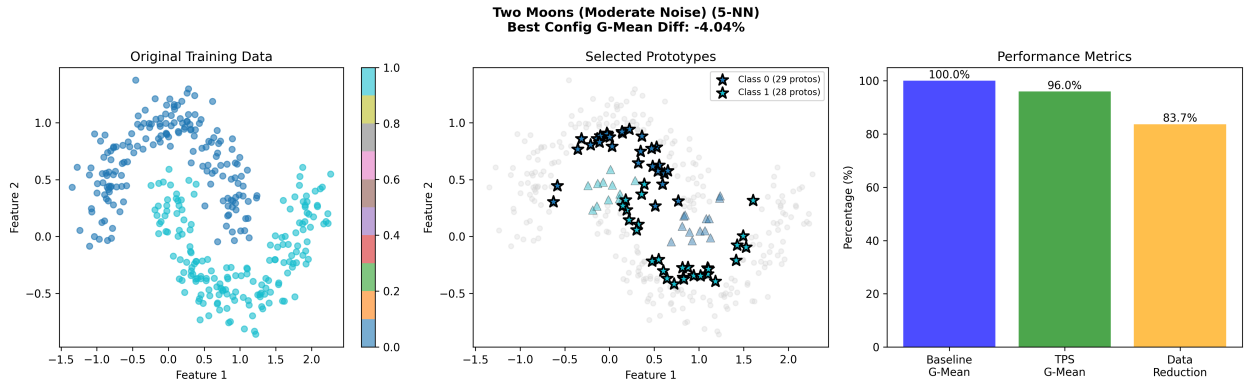


Figure 22: TPS selected prototypes on two moons with moderate noise simulated data for the 5-NN baseline.

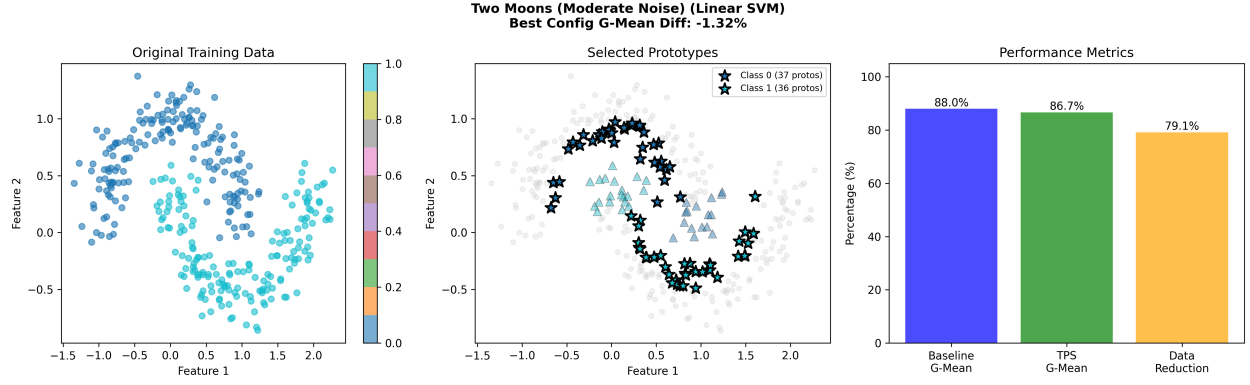


Figure 23: TPS selected prototypes on two moons with moderate noise simulated data for the SVM baseline.

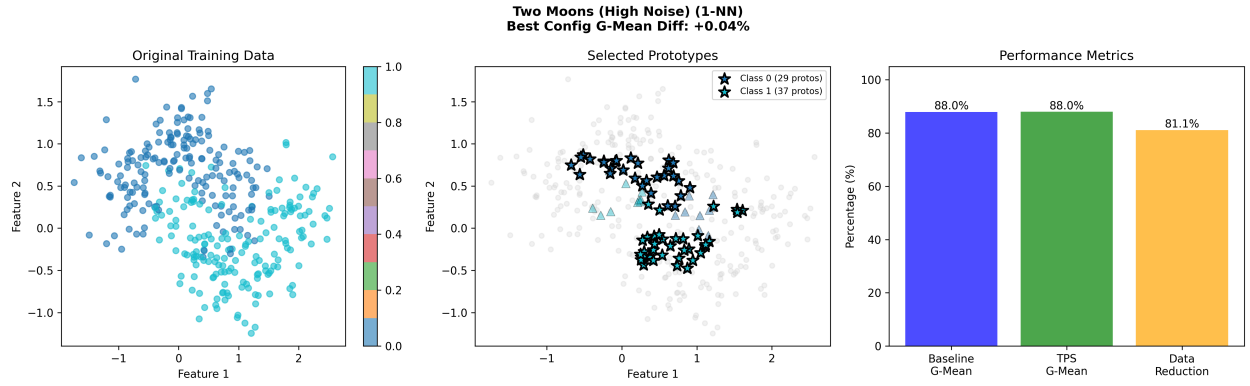


Figure 24: TPS selected prototypes on two moons with high noise simulated data for the 1-NN baseline.

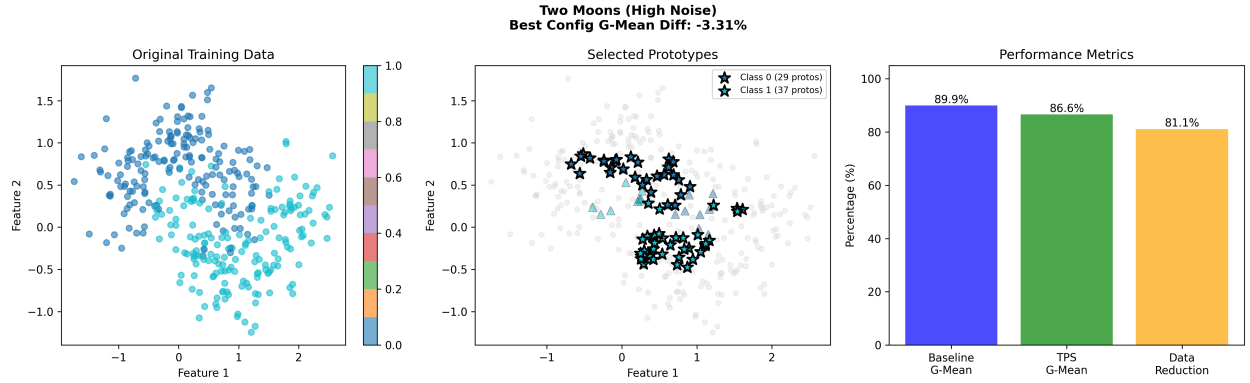


Figure 25: TPS selected prototypes on two moons with high noise simulated data for the 3-NN baseline.

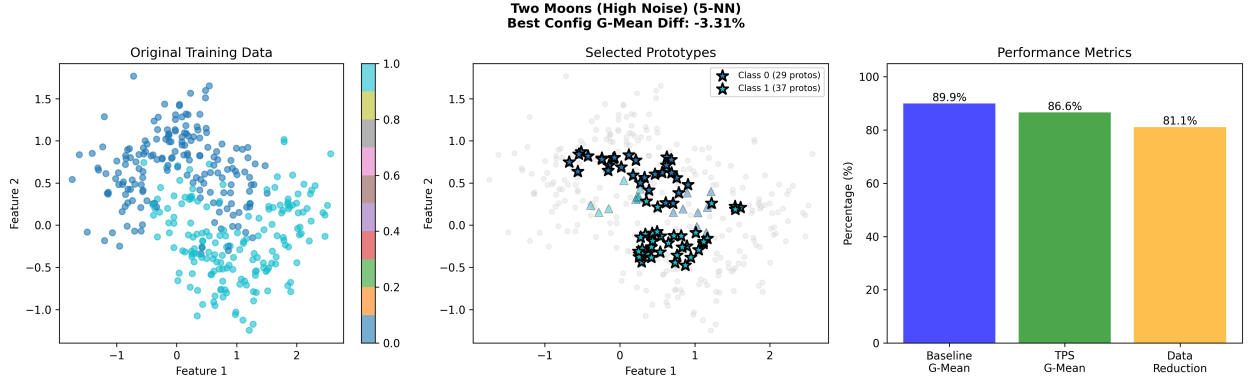


Figure 26: TPS selected prototypes on two moons with high noise simulated data for the 5-NN baseline.

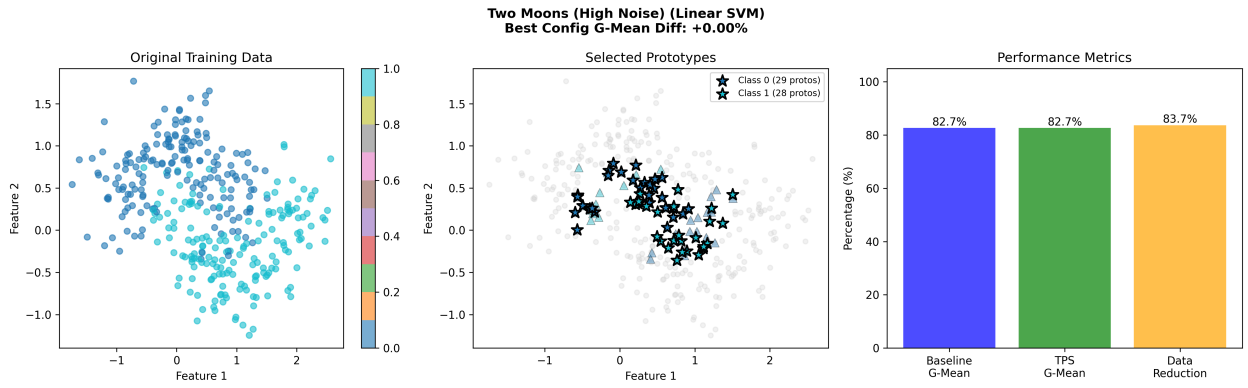


Figure 27: TPS selected prototypes on two moons with high noise simulated data for the SVM baseline.

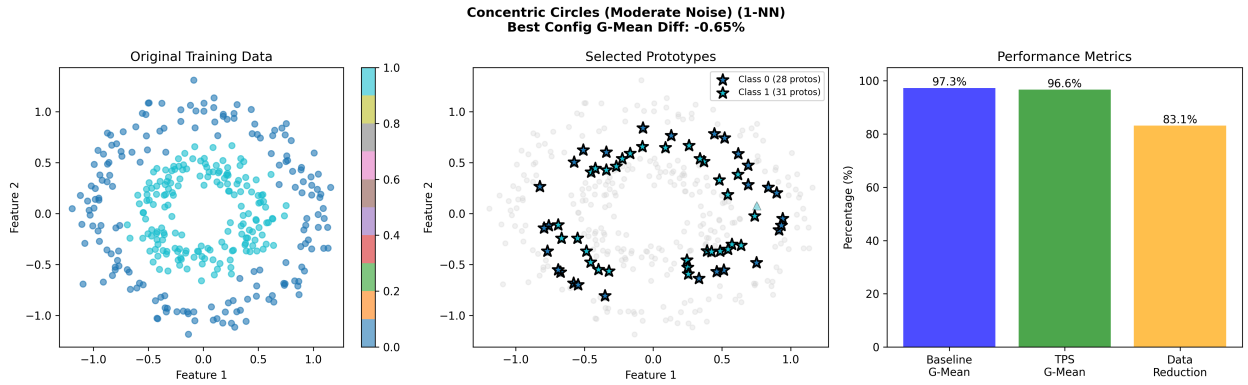


Figure 28: TPS selected prototypes on concentric circles with moderate noise simulated data for the 1-NN baseline.

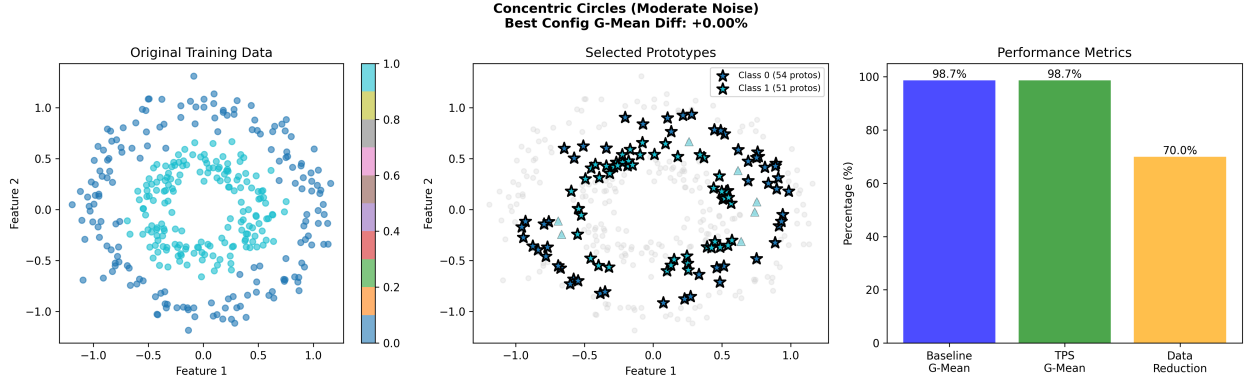


Figure 29: TPS selected prototypes on concentric circles with moderate noise simulated data for the 3-NN baseline.

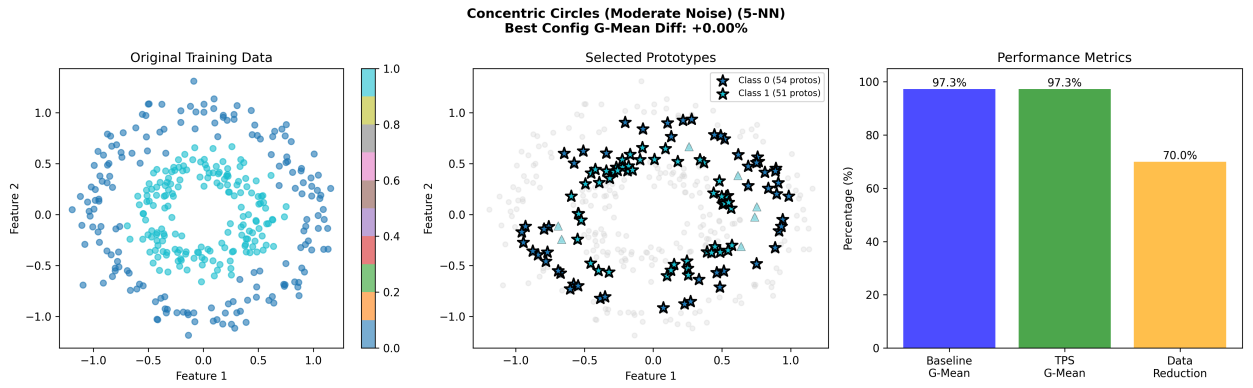


Figure 30: TPS selected prototypes on concentric circles with moderate noise simulated data for the 5-NN baseline.

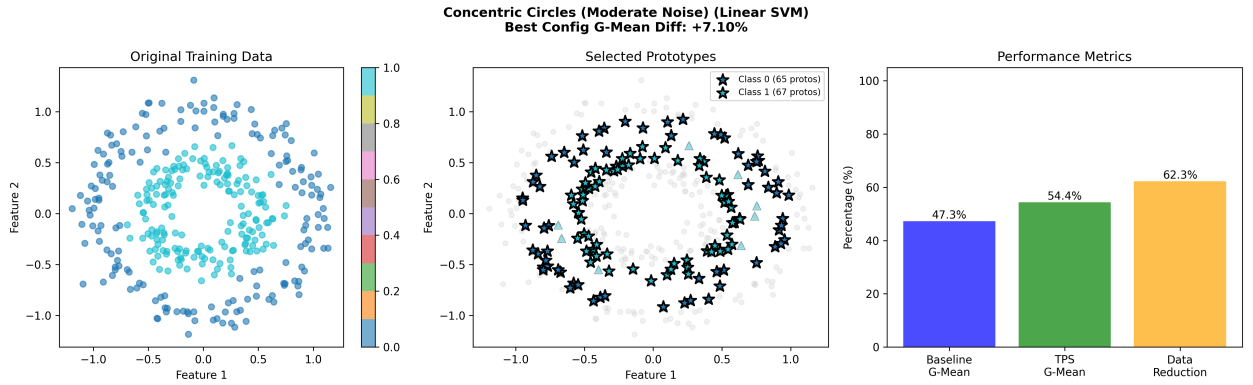


Figure 31: TPS selected prototypes on concentric circles with moderate noise simulated data for the SVM baseline.

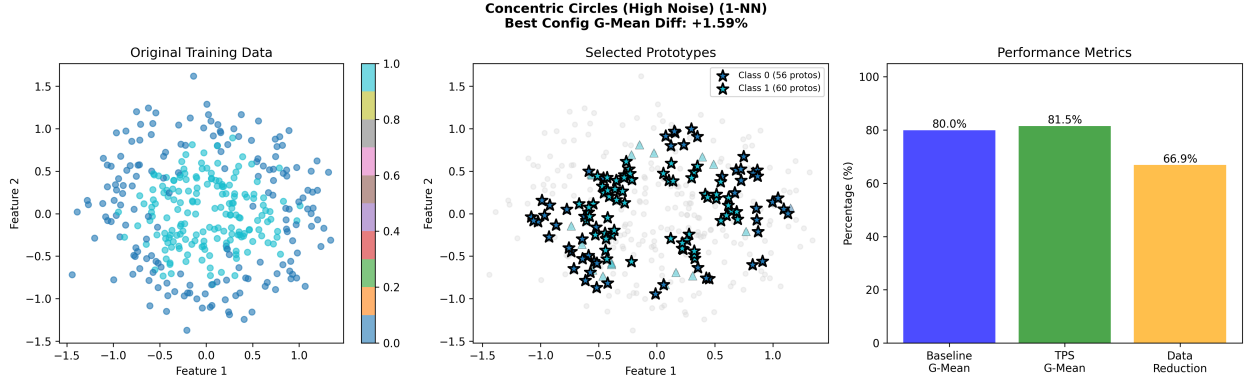


Figure 32: TPS selected prototypes on concentric circles with high noise simulated data for the 1-NN baseline.

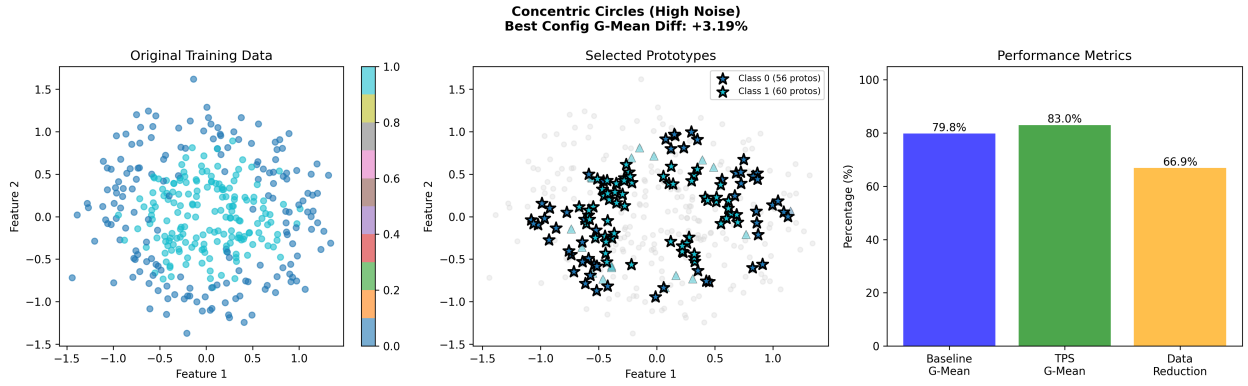


Figure 33: TPS selected prototypes on concentric circles with high noise simulated data for the 3-NN baseline.

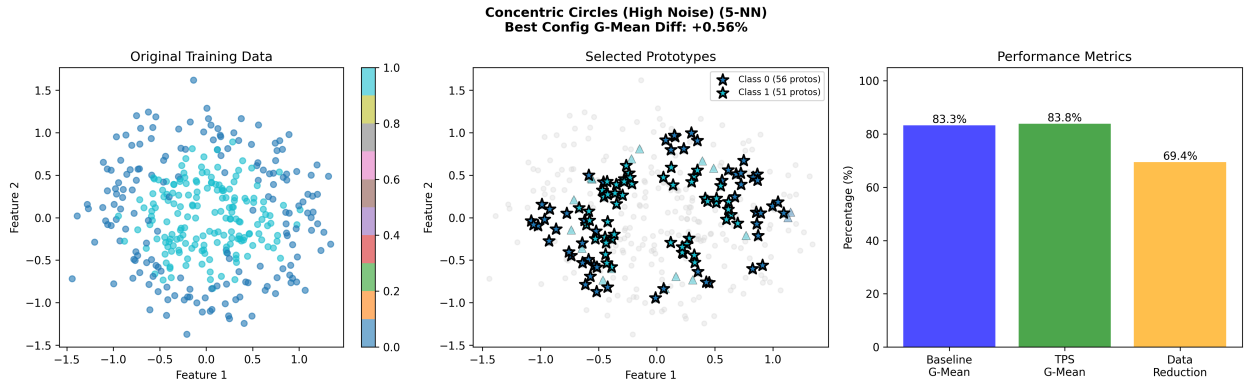


Figure 34: TPS selected prototypes on concentric circles with high noise simulated data for the 5-NN baseline.

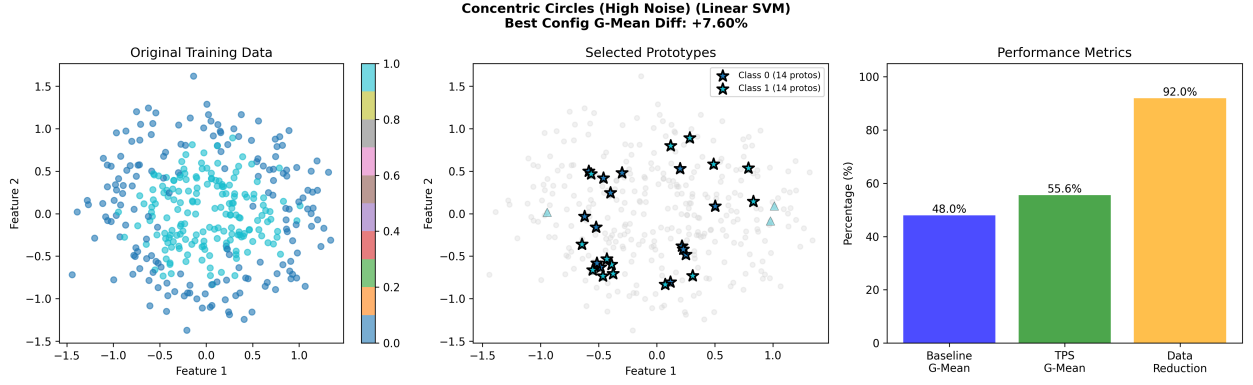


Figure 35: TPS selected prototypes on concentric circles with high noise simulated data for the SVM baseline.

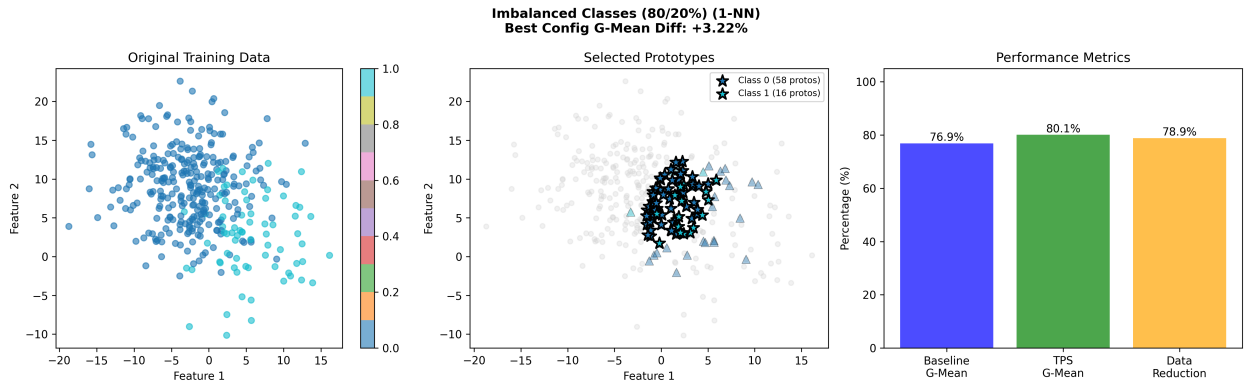


Figure 36: TPS selected prototypes on imbalanced simulated data for the 1-NN baseline.

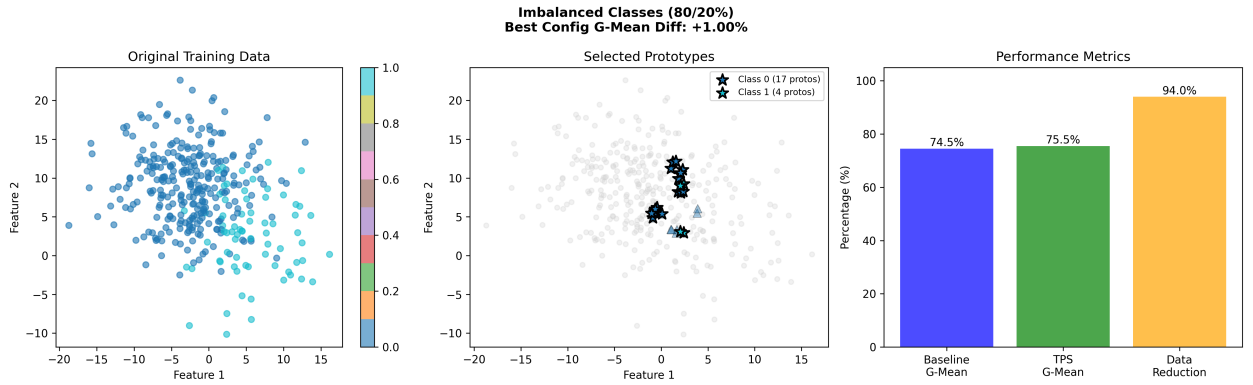


Figure 37: TPS selected prototypes on imbalanced simulated data for the 3-NN baseline.

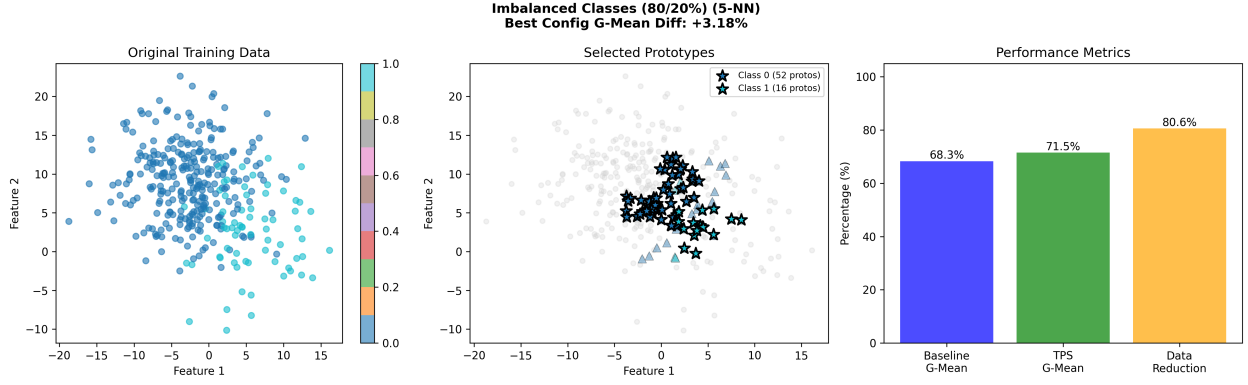


Figure 38: TPS selected prototypes on imbalanced simulated data for the 5-NN baseline.

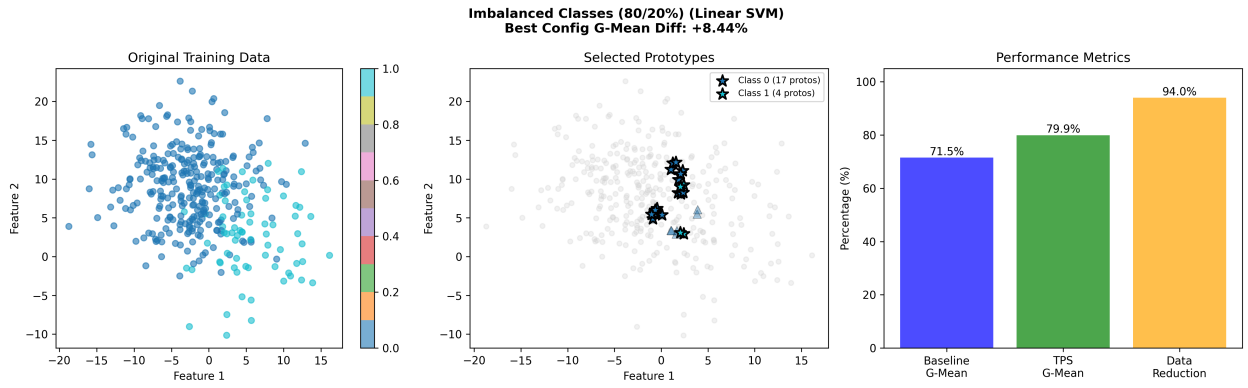


Figure 39: TPS selected prototypes on imbalanced simulated data for the SVM baseline.

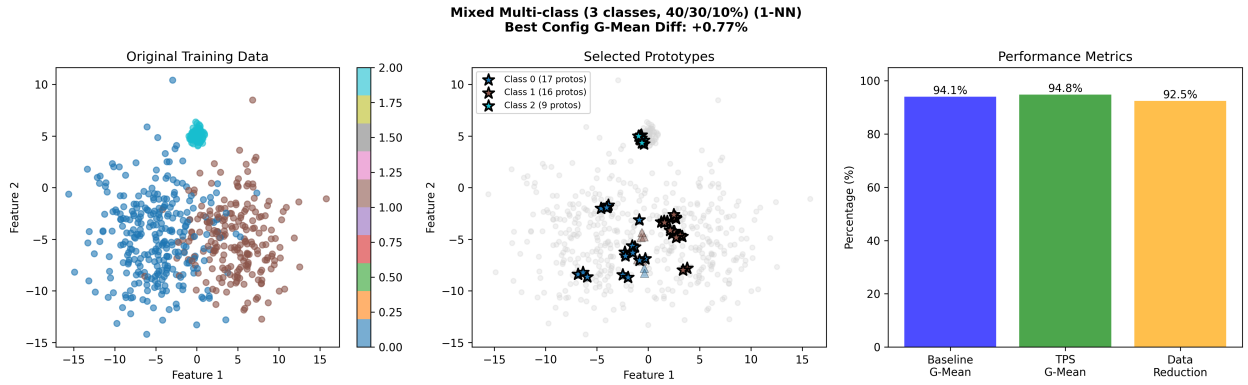


Figure 40: TPS selected prototypes on mixed imbalanced and overlapping simulated data for the 1-NN baseline.

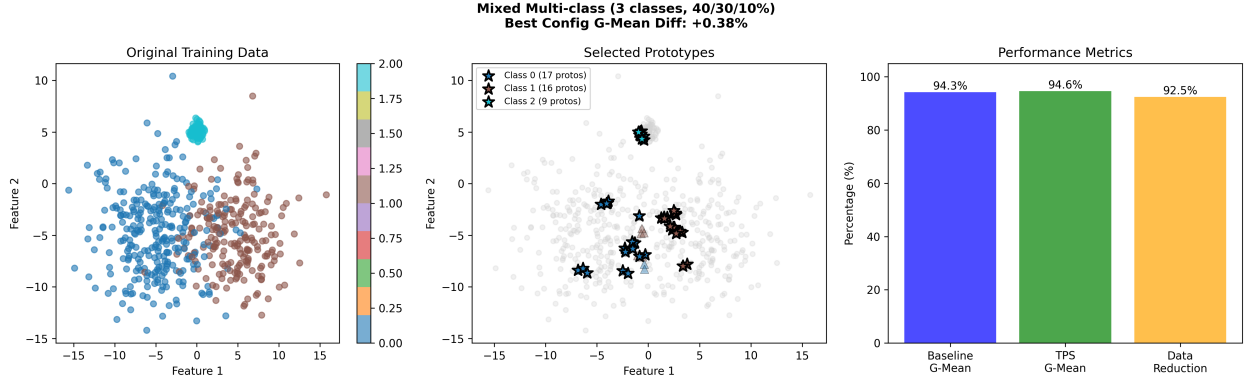


Figure 41: TPS selected prototypes on mixed imbalanced and overlapping simulated data for the 3-NN baseline.

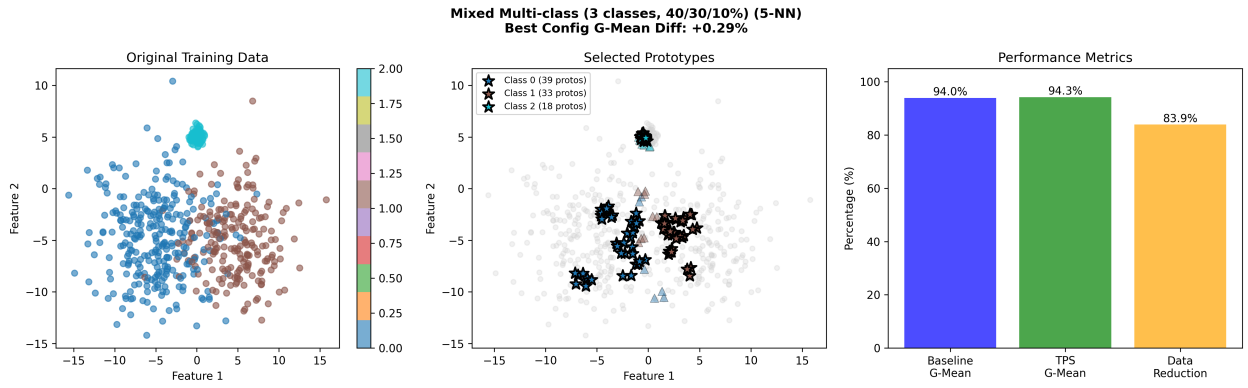


Figure 42: TPS selected prototypes on mixed imbalanced and overlapping simulated data for the 5-NN baseline.

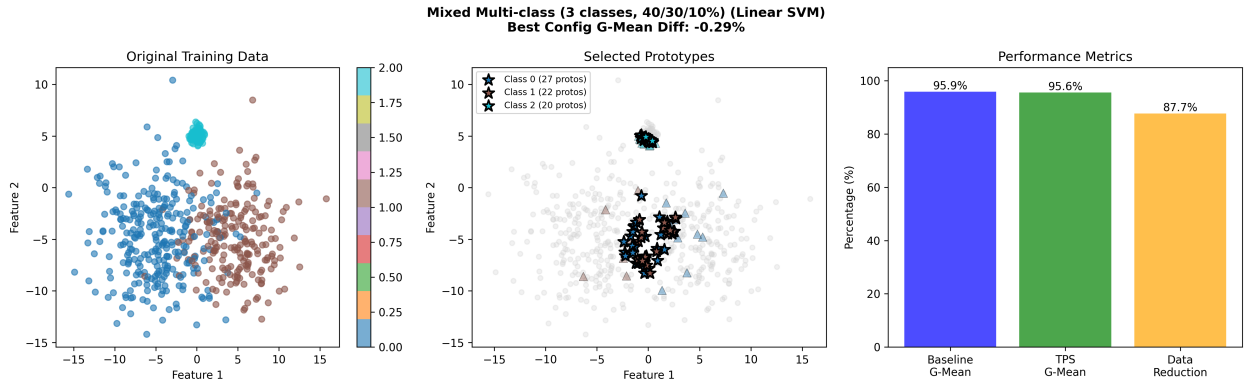


Figure 43: TPS selected prototypes on mixed imbalanced and overlapping simulated data for the SVM baseline.

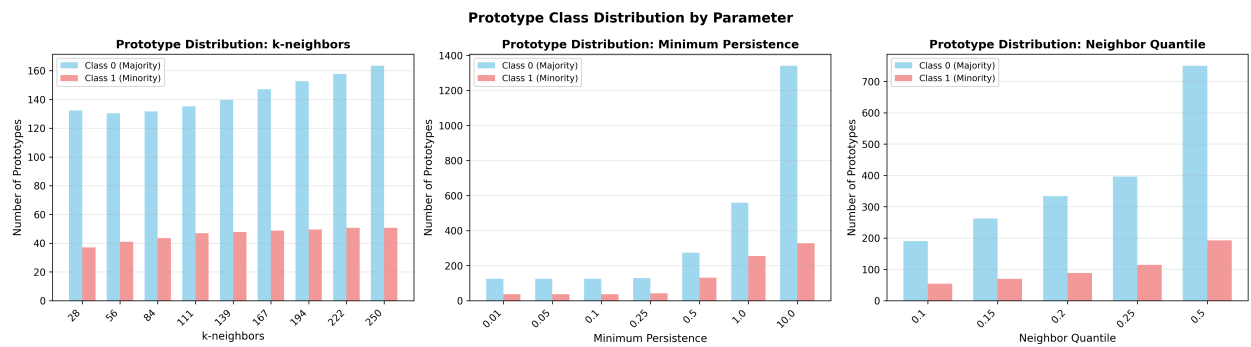


Figure 44: Prototype class distribution by parameter

7 References

References

- [1] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, J. Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143, August 2010.
- [2] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.
- [3] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, and J. Fco. Martínez-Trinidad. Prototype Selection Via Prototype Relevance. In José Ruiz-Shulcloper and Walter G. Kropatsch, editors, *Progress in Pattern Recognition, Image Analysis and Applications*, volume 5197, pages 153–160. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. Series Title: Lecture Notes in Computer Science.
- [4] Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, 5(4), December 2011. Publisher: Institute of Mathematical Statistics.
- [5] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, January 2009.
- [6] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.
- [7] Frédéric Chazal and Bertrand Michel. An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists, 2021.
- [8] Ali Zia, Abdelwahed Khamis, James Nichols, Usman Bashir Tayab, Zeeshan Hayder, Vivien Rolland, Eric Stone, and Lars Petersson. Topological deep learning: a review of an emerging paradigm. *Artificial Intelligence Review*, 57(4):77, February 2024.
- [9] Henri Riihimäki, Wojciech Chachólski, Jakob Theorell, Jan Hillert, and Ryan Ramanujam. A topological data analysis based classification method for multiple measurements. *BMC Bioinformatics*, 21(1):336, December 2020.
- [10] Rolando Kindelan, José Frías, Mauricio Cerda, and Nancy Hitschfeld. A topological data analysis based classifier. *Advances in Data Analysis and Classification*, 18(2):493–538, June 2024.
- [11] José Arturo Olvera López, Jesús Ariel Carrasco Ochoa, and José Francisco Martínez Trinidad. Prototype selection methods. *Computación y Sistemas*, 13(4):449–462, 2010. Publisher: Instituto Politécnico Nacional, Centro de Investigación en Computación.
- [12] Robert Ghrist. Barcodes: The persistent topology of data. *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY*, 45, February 2008.
- [13] Faustine Janes, Marco Mpimbo, and Makungu Mwanzalima. On theoretical construction of bifiltrations. *Research in Mathematics*, 12(1):2474773, December 2025.
- [14] H. Schenck. *Algebraic Foundations for Applied Topology and Data Analysis*. Mathematics of Data. Springer International Publishing, 2022.
- [15] Artür Manukyan and Elvan Ceyhan. Classification of Imbalanced Data with a Geometric Digraph Family. *The Journal of Machine Learning Research*, 17(1):6504–6543, October 2019.
- [16] Carey E. Priebe, Jeffrey L. Solka, David J. Marchette, and B. Ted Clark. Class cover catch digraphs for latent class discovery in gene expression monitoring by DNA microarrays. *Computational Statistics & Data Analysis*, 43(4):621–632, 2003.
- [17] Ivan Tomek. Two Modifications of CNN. *IEEE Transactions on Systems Man and Communications*, 6:769–772, 1976.
- [18] Elżbieta Pękalska, Robert P.W. Duin, and Pavel Paclík. Prototype selection for dissimilarity-based classifiers. *Part Special Issue: Complexity Reduction*, 39(2):189–208, February 2006.
- [19] Tamal Krishna Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022.
- [20] Kelin Xia and Guo-Wei Wei. Multidimensional persistence in biomolecular data. *Journal of Computational Chemistry*, 36(20):1502–1520, 2015. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.23953>.

- [21] Oliver Vipond, Joshua A. Bull, Philip S. Macklin, Ulrike Tillmann, Christopher W. Pugh, Helen M. Byrne, and Heather A. Harrington. Multiparameter persistent homology landscapes identify immune cell spatial patterns in tumors. *Proceedings of the National Academy of Sciences*, 118(41):e2102166118, 2021. _eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.2102166118>.
- [22] Sara Scaramuccia, Federico Iuricich, Leila De Florian, and Claudia Landi. Computing multiparameter persistent homology through a discrete Morse-based approach. *Computational Geometry*, 89:101623, 2020.
- [23] Madjid Allili, Tomasz Kaczynski, Claudia Landi, and Filippo Masoni. Acyclic Partial Matchings for Multidimensional Persistence: Algorithm and Combinatorial Interpretation. *Journal of Mathematical Imaging and Vision*, 61(2):174–192, February 2019.
- [24] Ulrich Bauer. Ripser: efficient computation of Vietoris–Rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, September 2021.
- [25] Nathalie Japkowicz. Assessment Metrics for Imbalanced Learning. In *Imbalanced Learning*, pages 187–206. John Wiley & Sons, Ltd, 2013. Section: 8 _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118646106.ch8>.
- [26] Mohamed Bekkar, Hassiba Kheliouane Djemaa, and Taklit Akrouf Alitouche. Evaluation measures for models assessment over imbalanced data sets. *J Inf Eng Appl*, 3(10), 2013.
- [27] Hossin M and Sulaiman M.N. A Review On Evaluation Metrics For Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 2015.
- [28] Alessandro Rinaldo and Larry Wasserman. Generalized density clustering. *The Annals of Statistics*, 38(5), October 2010. Publisher: Institute of Mathematical Statistics.
- [29] Matija Cufar and Ziga Virk. Fast computation of persistent homology representatives with involuted persistent homology. *Foundations of Data Science*, 5(4):466–479, 2023. Publisher: American Institute of Mathematical Sciences (AIMS).
- [30] Jean-Daniel Boissonnat and Siddharth Pritam. Edge Collapse and Persistence of Flag Complexes. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969.
- [31] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.
- [32] Vincent P. Grande and Michael T. Schaub. Non-isotropic Persistent Homology: Leveraging the Metric Dependency of PH, 2023. _eprint: 2310.16437.
- [33] Alfirna Rizqi Lahitani, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1–6, 2016.
- [34] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [35] Bagavathy Priya. Spam Ham Dataset.
- [36] Hechen Gong, Fucheng You, and Shaomei Wang. A Study of the Chinese spam Classification with Doc2vec and CNN. *IOP Conference Series: Materials Science and Engineering*, 563(4):042026, July 2019. Publisher: IOP Publishing.
- [37] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents, 2014. _eprint: 1405.4053.
- [38] Yasunori Endo and Sadaaki Miyamoto. Spherical k-Means++ Clustering. In Vicenc Torra and Torra Narukawa, editors, *Modeling Decisions for Artificial Intelligence*, pages 103–114, Cham, 2015. Springer International Publishing.
- [39] Karthik S. Gurumoorthy, Pratik Jawanpuria, and Bamdev Mishra. SPOT: A framework for selection of prototypes using optimal transport, 2021. _eprint: 2103.10159.