Vectorizing PD for Machine Learning

Tao Hou

University of Oregon

Acknowledgment

Contents of the slide (including figures) are based on the book:

- Baris Coskunuzer and Cneyt Grcan Akora. Topological Methods in Machine Learning: A Tutorial for Practitioners

Background

Ultimate goal:

- To effectively use persistence diagrams (PDs) in ML framework to strengthen your ML task by harnessing the power of topological descriptors

Background

Ultimate goal:

- To effectively use persistence diagrams (PDs) in ML framework to strengthen your ML task by harnessing the power of topological descriptors

Problem:

- Directly inputting PDs into ML models is impractical because their variable-size points conflicts with the fixed-size input required by most ML algorithms.

Background

Ultimate goal:

- To effectively use persistence diagrams (PDs) in ML framework to strengthen your ML task by harnessing the power of topological descriptors

Problem:

- Directly inputting PDs into ML models is impractical because their variable-size points conflicts with the fixed-size input required by most ML algorithms.
- Even if we have PDs with fixed size, we don't have a "fixed" meaning for a "component" in your PD (because PD is a *set*):
  - e.g., "age", "height", "hobby" to describe a person

Background

Ultimate goal:

- To effectively use persistence diagrams (PDs) in ML framework to strengthen your ML task by harnessing the power of topological descriptors

Problem:

- Directly inputting PDs into ML models is impractical because their variable-size points conflicts with the fixed-size input required by most ML algorithms.
- Even if we have PDs with fixed size, we don't have a "fixed" meaning for a "component" in your PD (because PD is a *set*):
  - e.g., "age", "height", "hobby" to describe a person

Vectorization:

- Convert PDs into vectors where each component has a fixed meaning (e.g., an *image*)

- *Fixed* vectorization:
  - A *fixed* mapping from a PD to a vector (which may involve a *hyperparameter choice*)
  - All described vectorization methods are applicable to *any* type of data (they simply transform a given PD into a vector)
  - However, the effectiveness and common usage of certain vectorization methods can vary depending on the *data type* and the *density or sparsity* of the PDs

Outline

- *Fixed* vectorization:
  - A *fixed* mapping from a PD to a vector (which may involve a *hyperparameter choice*)
  - All described vectorization methods are applicable to *any* type of data (they simply transform a given PD into a vector)
  - However, the effectiveness and common usage of certain vectorization methods can vary depending on the *data type* and the *density or sparsity* of the PDs
- *Automated* vectorization:
  - Avoid the intricacies of hyperparameter tuning (*learned* from the data by the algorithm)

Outline

- *Fixed* vectorization:
  - A *fixed* mapping from a PD to a vector (which may involve a *hyperparameter choice*)
  - All described vectorization methods are applicable to *any* type of data (they simply transform a given PD into a vector)
  - However, the effectiveness and common usage of certain vectorization methods can vary depending on the *data type* and the *density or sparsity* of the PDs
- *Automated* vectorization:
  - Avoid the intricacies of hyperparameter tuning (*learned* from the data by the algorithm)

- *Notice*:
  - We will only show how to convert a PD_p into a vector v_p for a certain dimension p

Outline

- *Fixed* vectorization:
  - A *fixed* mapping from a PD to a vector (which may involve a *hyperparameter choice*)
  - All described vectorization methods are applicable to *any* type of data (they simply transform a given PD into a vector)
  - However, the effectiveness and common usage of certain vectorization methods can vary depending on the *data type* and the *density or sparsity* of the PDs
- *Automated* vectorization:
  - Avoid the intricacies of hyperparameter tuning (*learned* from the data by the algorithm)

- *Notice*:
  - We will only show how to convert a PD_p into a vector v_p for a certain dimension p
  - If you want the features in all the dimensions, you can concatenate all vectors $v_0 \| v_1 \| \dots \| v_d$ assuming the maximum dimension is d

Outline

- *Fixed* vectorization:
  - A *fixed* mapping from a PD to a vector (which may involve a *hyperparameter choice*)
  - All described vectorization methods are applicable to *any* type of data (they simply transform a given PD into a vector)
  - However, the effectiveness and common usage of certain vectorization methods can vary depending on the *data type* and the *density or sparsity* of the PDs
- *Automated* vectorization:
  - Avoid the intricacies of hyperparameter tuning (*learned* from the data by the algorithm)

- *Notice*:
  - We will only show how to convert a PD_p into a vector v_p for a certain dimension p
  - If you want the features in all the dimensions, you can concatenate all vectors $v_0 \| v_1 \| \ldots \| v_d$ assuming the maximum dimension is d
  - Alternatively, you can convert the PD of all dimensions, PD_*, into a single vector v_*

Betti vectors

- Consider a continuous filtration $\mathcal{F}^c : \{K_\alpha \mid \alpha \in [0, \infty)\}$, where a topological space K_α varies over the real line

Betti vectors

- Consider a continuous filtration $\mathcal{F}^c : \{K_\alpha \mid \alpha \in [0, \infty)\}$, where a topological space K_α varies over the real line
- Recall that $\beta_p(K_\alpha)$, the p-th Betti number of K_α, is the cardinality of the p-th homology basis of K_α

Betti vectors

- Consider a continuous filtration $\mathcal{F}^c : \{K_\alpha \mid \alpha \in [0, \infty)\}$, where a topological space K_α varies over the real line
- Recall that $\beta_p(K_\alpha)$, the p-th Betti number of K_α, is the cardinality of the p-th homology basis of K_α
- By selecting a set of the real values $\alpha_1, \alpha_2, \ldots, \alpha_n$, the p-th Betti vector $\vec{\beta}_p$ is defined as:

$$\vec{\beta}_p = [\beta_p(K_{\alpha_1}), \beta_p(K_{\alpha_2}), \ldots, \beta_p(K_{\alpha_n})]$$

Betti vectors

- Consider a continuous filtration $\mathcal{F}^c : \{K_\alpha \mid \alpha \in [0, \infty)\}$, where a topological space K_α varies over the real line
- Recall that $\beta_p(K_\alpha)$, the p-th Betti number of K_α, is the cardinality of the p-th homology basis of K_α
- By selecting a set of the real values $\alpha_1, \alpha_2, \ldots, \alpha_n$, the p-th Betti vector $\vec{\beta}_p$ is defined as:

$$\vec{\beta}_p = [\beta_p(K_{\alpha_1}), \beta_p(K_{\alpha_2}), \ldots, \beta_p(K_{\alpha_n})]$$

- Aka. by selecting the set of values $\alpha_1, \alpha_2, \ldots, \alpha_n$ we get a vector of size n which is a sample of the p-th Betti curve $B_p : [0, \infty) \to \mathbb{N}$ where $B_p(\alpha) = \beta_p(K_\alpha)$

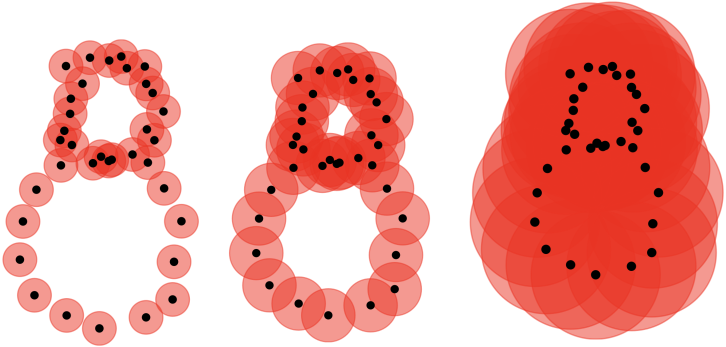
Betti vectors

- Consider a continuous filtration $\mathcal{F}^c : \{K_\alpha \mid \alpha \in [0, \infty)\}$, where a topological space K_α varies over the real line
- Recall that $\beta_p(K_\alpha)$, the p-th Betti number of K_α, is the cardinality of the p-th homology basis of K_α
- By selecting a set of the real values $\alpha_1, \alpha_2, \ldots, \alpha_n$, the p-th Betti vector $\vec{\beta}_p$ is defined as:

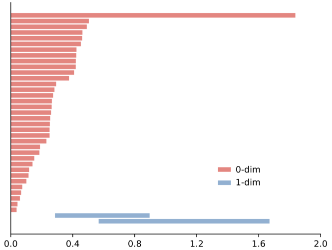
$$\vec{\beta}_p = [\beta_p(K_{\alpha_1}), \beta_p(K_{\alpha_2}), \ldots, \beta_p(K_{\alpha_n})]$$

- Aka. by selecting the set of values $\alpha_1, \alpha_2, \ldots, \alpha_n$ we get a vector of size n which is a sample of the p-th Betti curve $B_p : [0, \infty) \to \mathbb{N}$ where $B_p(\alpha) = \beta_p(K_\alpha)$
- $n, \alpha_1, \alpha_2, \ldots, \alpha_n$ are the *hyperparameters* that you need to determine in practice (e.g., we could let $n = 50$)

Example



(a) $\epsilon = 0.4$ (c) $\epsilon = 0.7$ (e) $\epsilon = 2$



Example

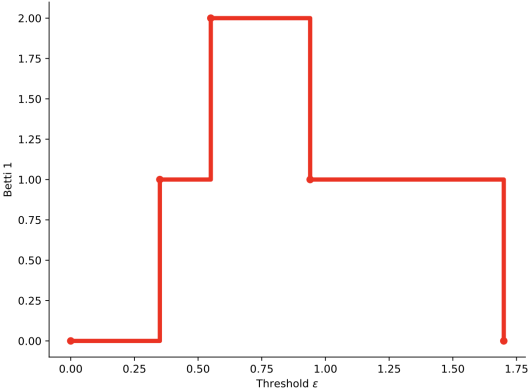
- Select the α values: $[0, 0.25, 0.75, 1.5, 1.75]$

Example

- Select the α values: $[0, 0.25, 0.75, 1.5, 1.75]$
- $\vec{\beta}_0 = [35, 20, 1, 1, 1]$

Example

- Select the α values: $[0, 0.25, 0.75, 1.5, 1.75]$
- $\vec{\beta}_0 = [35, 20, 1, 1, 1]$
- $\vec{\beta}_1 = [0, 0, 2, 1, 0]$



Remarks

Some advantages of Betti vectors:

- While being able to be derived from persistence diagrams, Betti vectors *do not* require the computation of persistence diagrams.
- Indeed, there are computationally more effective ways to produce Betti vectors.

Remarks

Some advantages of Betti vectors:

- While being able to be derived from persistence diagrams, Betti vectors *do not* require the computation of persistence diagrams.
- Indeed, there are computationally more effective ways to produce Betti vectors.
- Another favorable aspect of Betti vectors is their *ease of interpretation*: Simply put, $\beta_p(K_\alpha)$ is equal to the number of p-dimensional holes in K_α.

Persistence Landscapes

- Persistence Landscapes are one of the first vectorization methods in TDA, directly utilizing the lifespan information
- Using Persistence Landscapes, the points *away from the diagonal* (large features) are easily distinguished and promoted

Persistence Landscapes

- Persistence Landscapes are one of the first vectorization methods in TDA, directly utilizing the lifespan information
- Using Persistence Landscapes, the points *away from the diagonal* (large features) are easily distinguished and promoted
- Consider a persistence diagram $PD_p = \{(b_i, d_i)\}$
- For each $(b_i, d_i) \in PD_p$, first define its generating function

$$\Lambda_i : [0, \infty) \to \mathbb{R},$$

which is a piece-wise linear function obtained by two line segments connecting $(b_i, 0)$ and $(d_i, 0)$ to $(\frac{b_i+d_i}{2}, \frac{d_i-b_i}{2})$

Persistence Landscapes

- Persistence Landscapes are one of the first vectorization methods in TDA, directly utilizing the lifespan information
- Using Persistence Landscapes, the points *away from the diagonal* (large features) are easily distinguished and promoted
- Consider a persistence diagram $PD_p = \{(b_i, d_i)\}$
- For each $(b_i, d_i) \in PD_p$, first define its generating function

$$\Lambda_i : [0, \infty) \to \mathbb{R},$$

  which is a piece-wise linear function obtained by two line segments connecting $(b_i, 0)$ and $(d_i, 0)$ to $(\frac{b_i+d_i}{2}, \frac{d_i-b_i}{2})$
  - Note: It is 0 elsewhere

Persistence Landscapes

- Persistence Landscapes are one of the first vectorization methods in TDA, directly utilizing the lifespan information
- Using Persistence Landscapes, the points *away from the diagonal* (large features) are easily distinguished and promoted
- Consider a persistence diagram $PD_p = \{(b_i, d_i)\}$
- For each $(b_i, d_i) \in PD_p$, first define its generating function

$$\Lambda_i : [0, \infty) \to \mathbb{R},$$

  which is a piece-wise linear function obtained by two line segments connecting $(b_i, 0)$ and $(d_i, 0)$ to $(\frac{b_i+d_i}{2}, \frac{d_i-b_i}{2})$
  - Note: It is 0 elsewhere
- Observe that the longer the interval $[b_i, d_i)$ is, the "higher" and "wider" the generating function Λ_i is (so longer bars are emphasized)

Persistence Landscapes

- Given all generating functions $\{\Lambda_i\}$, we take the k-th largest value at each $\alpha \in [0, \infty)$

Persistence Landscapes

- Given all generating functions $\{\Lambda_i\}$, we take the k-th largest value at each $\alpha \in [0, \infty)$
- In particular, the k-th Persistence Landscape function $\lambda^k : [0, \infty) \to \mathbb{R}$ is defined as:

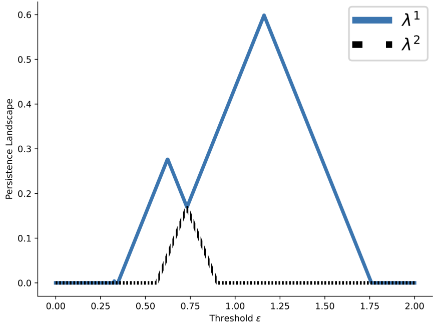
$$\lambda^k(\alpha) = k^{\text{th}}\text{-}\max\{\Lambda_i(\alpha)\}$$

Persistence Landscapes

- Given all generating functions $\{\Lambda_i\}$, we take the k-th largest value at each $\alpha \in [0, \infty)$
- In particular, the k-th Persistence Landscape function $\lambda^k : [0, \infty) \to \mathbb{R}$ is defined as:

$$\lambda^k(\alpha) = k^{\text{th}}\text{- max}\{\Lambda_i(\alpha)\}$$

- E.g.: λ^1 and λ^2 for PD_1 of the previous 8-shaped point cloud:



Remarks

- The previously defined λ^k is a continuous function from $[0, \infty)$ to $\mathbb{R}$

Remarks

- The previously defined λ^k is a continuous function from $[0, \infty)$ to $\mathbb{R}$
- We still need to vectorize (or discretize) it by selecting values

$$\alpha_1, \alpha_2, \ldots, \alpha_n,$$

and take the vector

$$\lambda^k(\alpha_1), \lambda^k(\alpha_2), \ldots, \lambda^k(\alpha_n)$$

Remarks

- The previously defined λ^k is a continuous function from $[0, \infty)$ to $\mathbb{R}$
- We still need to vectorize (or discretize) it by selecting values

$$\alpha_1, \alpha_2, \ldots, \alpha_n,$$

  and take the vector

$$\lambda^k(\alpha_1), \lambda^k(\alpha_2), \ldots, \lambda^k(\alpha_n)$$

- Also note that λ^1 and λ^2 are the most commonly used to produce the vectors, and the vectors are used with concatenation in the applications

Silhouette

- A practical modification for persistence landscapes by *eliminating* the k-th maxima in persistence landscapes and introducing a *tuning parameter* p to better utilize the lifespans of the bars in the barcode

Silhouette

- A practical modification for persistence landscapes by *eliminating* the k-th maxima in persistence landscapes and introducing a *tuning parameter* p to better utilize the lifespans of the bars in the barcode
- To differentiate from the tuning parameter p, we use q to denote the dimension of a PD

Silhouette

- A practical modification for persistence landscapes by *eliminating* the k-th maxima in persistence landscapes and introducing a *tuning parameter* p to better utilize the lifespans of the bars in the barcode
- To differentiate from the tuning parameter p, we use q to denote the dimension of a PD
- For a persistence diagram

$$PD_q = \{(b_i, d_i) \mid i = 1, \ldots, N\},$$

let Λ_i be the generating function for (b_i, d_i) as defined in the persistence landscapes

Silhouette

- A practical modification for persistence landscapes by *eliminating* the k-th maxima in persistence landscapes and introducing a *tuning parameter* p to better utilize the lifespans of the bars in the barcode
- To differentiate from the tuning parameter p, we use q to denote the dimension of a PD
- For a persistence diagram

$$PD_q = \{(b_i, d_i) \mid i = 1, \ldots, N\},$$

let Λ_i be the generating function for (b_i, d_i) as defined in the persistence landscapes

- The Silhouette function $\Psi_q : [0, \infty) \to \mathbb{R}$ is defined as the weighted sum of the Λ_i's:

$$\Psi_q(\alpha) = \frac{\sum_{i=1}^{N} w_i \Lambda_i(\alpha)}{\sum_{i=1}^{N} w_i},$$

where w_i is the weight of the function Λ_i

Silhouette

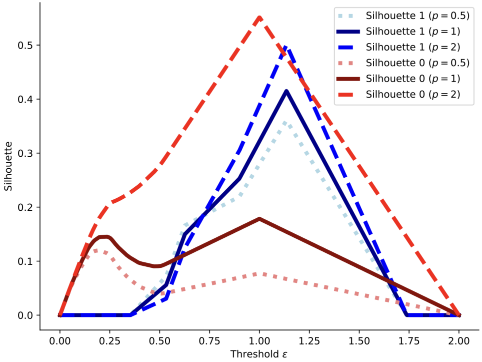
- The weight w_i for a Λ_i is typically taken as $(d_i - b_i)^p$, so Ψ_q becomes:
$$\Psi_q = \frac{\sum_{i=1}^{N}(d_i - b_i)^p \Lambda_i}{\sum_{i=1}^{N}(d_i - b_i)^p}$$

Silhouette

- The weight w_i for a Λ_i is typically taken as $(d_i - b_i)^p$, so Ψ_q becomes:

$$\Psi_q = \frac{\sum_{i=1}^{N}(d_i - b_i)^p \Lambda_i}{\sum_{i=1}^{N}(d_i - b_i)^p}$$

- E.g.: for the previous 8-shaped point cloud:



Remakrs

- The tuning parameter p is crucial as it adjusts the silhouette function's emphasis on topological features with varying lifespans

Remakrs

- The tuning parameter p is crucial as it adjusts the silhouette function's emphasis on topological features with varying lifespans
- For example, when $p > 1$, features with longer lifespans are assigned even larger weights, highlighting these longer features

Remakrs

- The tuning parameter p is crucial as it adjusts the silhouette function's emphasis on topological features with varying lifespans
- For example, when $p > 1$, features with longer lifespans are assigned even larger weights, highlighting these longer features
- When $p < 1$, shorter lifespans are given more weight compared to previous case, emphasizing shorter features

Remakrs

- The tuning parameter p is crucial as it adjusts the silhouette function's emphasis on topological features with varying lifespans
- For example, when $p > 1$, features with longer lifespans are assigned even larger weights, highlighting these longer features
- When $p < 1$, shorter lifespans are given more weight compared to previous case, emphasizing shorter features
- Common choices for p are $1/2$, 1, and 2
- If the persistence diagram has a few points and the goal is to emphasize these significant features, $p = 2$ would be a good choice

Remakrs

- The tuning parameter p is crucial as it adjusts the silhouette function's emphasis on topological features with varying lifespans
- For example, when $p > 1$, features with longer lifespans are assigned even larger weights, highlighting these longer features
- When $p < 1$, shorter lifespans are given more weight compared to previous case, emphasizing shorter features
- Common choices for p are $1/2$, 1, and 2
- If the persistence diagram has a few points and the goal is to emphasize these significant features, $p = 2$ would be a good choice
- If there are many points in PD and the key information comes from smaller features, $1/2$ can be used

Persistence Curve

- The Persistence Curve is a framework for vectorization PD which generalizes the previous vectorization techniques
- The core idea is to consider the intervals in a PD containing a real value α, denoted $PD(\alpha)$, and utilize a real-valued generating function Λ, a summary statistic T:

Persistence Curve

- The Persistence Curve is a framework for vectorization PD which generalizes the previous vectorization techniques
- The core idea is to consider the intervals in a PD containing a real value α, denoted $PD(\alpha)$, and utilize a real-valued generating function Λ, a summary statistic T:

$$pc_{\Lambda,T}(\alpha) = T\Big(\big[\Lambda(b,d,\alpha) \mid (b,d) \in PD(\alpha)\big]\Big)$$

Persistence Curve

- The Persistence Curve is a framework for vectorization PD which generalizes the previous vectorization techniques
- The core idea is to consider the intervals in a PD containing a real value α, denoted $PD(\alpha)$, and utilize a real-valued generating function Λ, a summary statistic T:

$$pc_{\Lambda,T}(\alpha) = T\Big(\big[\Lambda(b,d,\alpha) \mid (b,d) \in PD(\alpha)\big]\Big)$$

We have:

- "$[\cdots]$" encloses a "vector" which produces a real value for each $(b,d) \in PD(\alpha)$
- Given the vector, T then provides a "aggregate" (e.g., sum, mean, or max)

Persistence Curve

- Persistence Curves provide a general and unifying framework for vectorization methods
- By selecting different combinations of Λ and T, one can generate various functional summaries of the PD, each potentially highlighting different aspects of the data

Persistence Curve

- Persistence Curves provide a general and unifying framework for vectorization methods
- By selecting different combinations of Λ and T, one can generate various functional summaries of the PD, each potentially highlighting different aspects of the data
- Specifically, all previous vectorization methods can be considered as special cases of Persistence Curves by choosing a certain Λ and T
- E.g., let

$$\Lambda_{(b,d)}(t) = \begin{cases} 0 & \text{if } t \notin [b,d] \\ t-b & \text{if } t \in [b, \frac{b+d}{2}] \\ d-t & \text{if } t \in (\frac{b+d}{2}, d] \end{cases}$$

and T be the k^{th}-max function, then we get the persistence landscape

Persistence Images

- Unlike previous vectorizations, Persistence Images, as the name suggests, produce 2D-arrays
- The idea is to capture the location of the points in the PDs by using the *2D Gaussian functions* centered at these points

Persistence Images

- Unlike previous vectorizations, Persistence Images, as the name suggests, produce 2D-arrays
- The idea is to capture the location of the points in the PDs by using the *2D Gaussian functions* centered at these points
- Recall (1D) Gaussian (normal) distribution (where μ is the mean and σ^2 the variance)

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Persistence Images

- Unlike previous vectorizations, Persistence Images, as the name suggests, produce 2D-arrays
- The idea is to capture the location of the points in the PDs by using the *2D Gaussian functions* centered at these points
- Recall (1D) Gaussian (normal) distribution (where μ is the mean and σ^2 the variance)

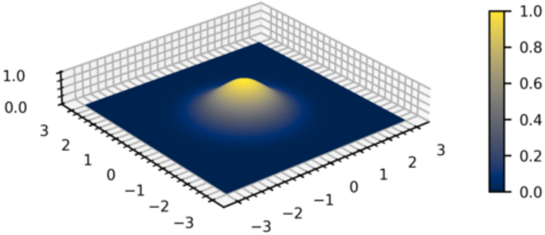
$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

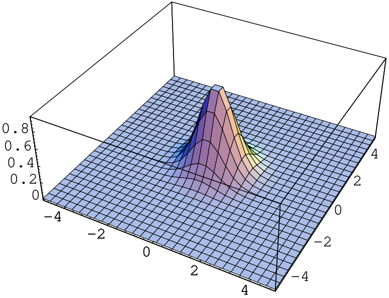
- A Gaussian function is a relaxation of it but in 2D. We don't need the integral to be 1 and the function always has the same height:

$$\phi(x) = \exp\left(-\frac{\|x-\mu\|_2^2}{\sigma^2}\right)$$

where μ is still the "mean" (center) and σ^2 is the "width" (spread) now

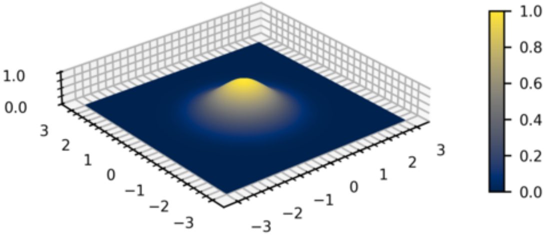
Gaussian functions

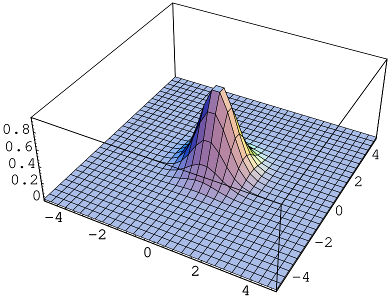




(Figure from handwiki and wolfram mathworld)

Gaussian functions





(Figure from handwiki and wolfram mathworld)

- So the function value is basically just an indication of how far a point x is from the center μ (instead of directly using the inverse distance $1/\|x-\mu\|_2$)

Persistence Images

- Consider
$$PD = \{(b_i, d_i) \mid i = 1, \ldots, N\}$$

Persistence Images

- Consider
$$PD = \{(b_i, d_i) \mid i = 1, \dots, N\}$$
- For each (b_i, d_i), we have a Gaussian function ϕ_i centered at (b_i, d_i)
- Notice that the spread σ is hyperparameter shared by all ϕ_i's

Persistence Images

- Consider
$$PD = \{(b_i, d_i) \mid i = 1, \ldots, N\}$$
- For each (b_i, d_i), we have a Gaussian function ϕ_i centered at (b_i, d_i)
- Notice that the spread σ is hyperparameter shared by all ϕ_i's
- The Persistence Image of PD is a 2D function $pi : \mathbb{R}^2 \to \mathbb{R}$:
$$pi(x) = \sum_i w_i \phi_i(x),$$

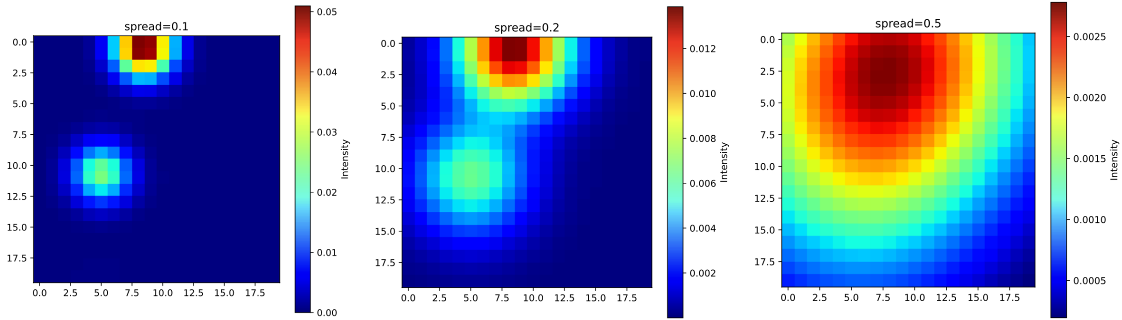
which is nothing but the weighted sum of the Gaussian functions of all the PD points

Persistence Images

- Consider
$$PD = \{(b_i, d_i) \mid i = 1, \ldots, N\}$$
- For each (b_i, d_i), we have a Gaussian function ϕ_i centered at (b_i, d_i)
- Notice that the spread σ is hyperparameter shared by all ϕ_i's
- The Persistence Image of PD is a 2D function $pi : \mathbb{R}^2 \to \mathbb{R}$:
$$pi(x) = \sum_i w_i \phi_i(x),$$
which is nothing but the weighted sum of the Gaussian functions of all the PD points
- w_i is typically $(d_i - b_i)^p$

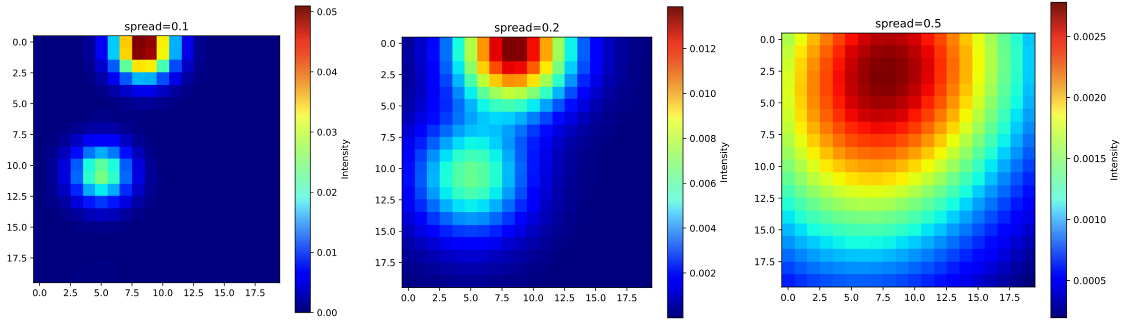
Example

- For the two points in the 1st PD of the previous 8-shaped point cloud:



Example

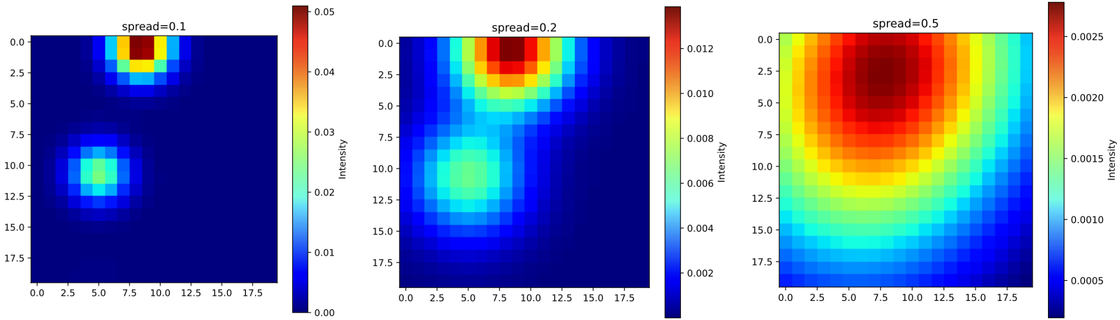
- For the two points in the 1st PD of the previous 8-shaped point cloud:



- So it's just a 2D function indicating the "positions" of the points in a PD

Example

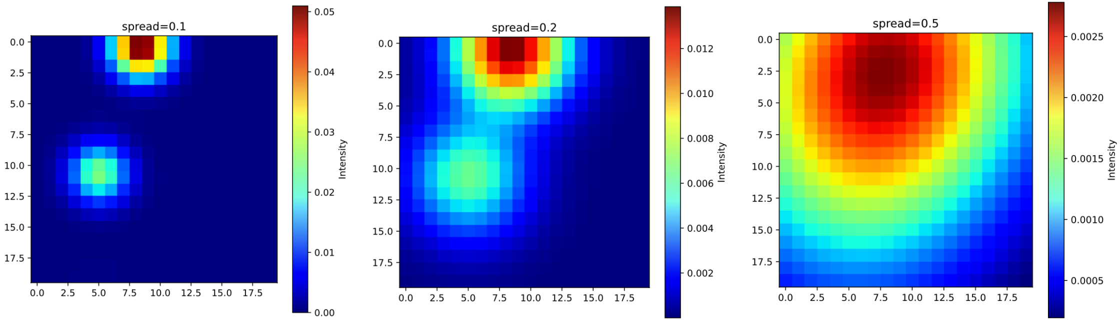
- For the two points in the 1st PD of the previous 8-shaped point cloud:



- So it's just a 2D function indicating the "positions" of the points in a PD
- Notice: $pi : \mathbb{R}^2 \rightarrow \mathbb{R}$ as defined is still a continuous function
- As in the previous vectorizations, we still need to discretize it into a 2D image: by doing some sampling on a 2D grid (which is another hyperparameter)

Question

- For the two points in the 1st PD of the previous 8-shaped point cloud:



- Why don't (can't) we use a 2D image to directly encode a PD (which is in 2D)?

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)
- Rely on the concept of Reproducing Kernel Hilbert Spaces (RKHS), which draws upon a “kernel” function K

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)
- Rely on the concept of Reproducing Kernel Hilbert Spaces (RKHS), which draws upon a "kernel" function K
- Using a kernel function K, you data (e.g., PDs) don't need to be vectorized

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)
- Rely on the concept of Reproducing Kernel Hilbert Spaces (RKHS), which draws upon a "kernel" function K
- Using a kernel function K, you data (e.g., PDs) don't need to be vectorized
- What K does is that, given two data items, say two PDs, D_1 and D_2, the function value $K(D_1, D_2)$ measures how "similar" the two PDs are

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)
- Rely on the concept of Reproducing Kernel Hilbert Spaces (RKHS), which draws upon a "kernel" function K
- Using a kernel function K, you data (e.g., PDs) don't need to be vectorized
- What K does is that, given two data items, say two PDs, D_1 and D_2, the function value $K(D_1, D_2)$ measures how "similar" the two PDs are
- The persistence weighted Gaussian kernel (PWGK) for PDs:

$$K(D_1, D_2) = \sum_{x_1 \in D_1} \sum_{x_2 \in D_2} w_{x_1} w_{x_2} \phi(x_1, x_2)$$

Kernel Methods for Integrating PD into ML framework

- Kernel methods used to be a major branch of methods (besides deep neural networks) in ML which was very successful (e.g., SVM)
- Rely on the concept of Reproducing Kernel Hilbert Spaces (RKHS), which draws upon a "kernel" function K
- Using a kernel function K, you data (e.g., PDs) don't need to be vectorized
- What K does is that, given two data items, say two PDs, D_1 and D_2, the function value $K(D_1, D_2)$ measures how "similar" the two PDs are
- The persistence weighted Gaussian kernel (PWGK) for PDs:

$$K(D_1, D_2) = \sum_{x_1 \in D_1} \sum_{x_2 \in D_2} w_{x_1} w_{x_2} \phi(x_1, x_2)$$

  where $\phi(x_1, x_2)$ is the Gaussian kernel:

$$\phi(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|_2^2}{\sigma^2}\right)$$

Kernel Methods for Integrating PD into ML framework

- "Kernel methods" is indeed a *framework* for machine learning on non-vectorized data
- So, you can design different kernel functions that can be used in the framework which produces a different learning algorithm

Kernel Methods for Integrating PD into ML framework

- "Kernel methods" is indeed a *framework* for machine learning on non-vectorized data
- So, you can design different kernel functions that can be used in the framework which produces a different learning algorithm
- Another commonly used kernel function for PDs is the sliced Wasserstein kernel (SWK), which we will not cover in this course

Kernel Methods for Integrating PD into ML framework

- “Kernel methods” is indeed a *framework* for machine learning on non-vectorized data
- So, you can design different kernel functions that can be used in the framework which produces a different learning algorithm
- Another commonly used kernel function for PDs is the sliced Wasserstein kernel (SWK), which we will not cover in this course
- Designing kernel functions is non-trivial as you need to map your data into Reproducing Kernel Hilbert Spaces (RKHS) and prove that your kernel function is an inner product in the RKHS (which needs a lot of functional analysis :-( )
- But using existing kernels already designed is not so hard

Kernel Methods for Integrating PD into ML framework

- "Kernel methods" is indeed a *framework* for machine learning on non-vectorized data
- So, you can design different kernel functions that can be used in the framework which produces a different learning algorithm
- Another commonly used kernel function for PDs is the sliced Wasserstein kernel (SWK), which we will not cover in this course
- Designing kernel functions is non-trivial as you need to map your data into Reproducing Kernel Hilbert Spaces (RKHS) and prove that your kernel function is an inner product in the RKHS (which needs a lot of functional analysis :-( )
- But using existing kernels already designed is not so hard
- Although kernel methods can yield better results in some settings, they can be computationally intensive and impractical for large datasets due to the high computational costs associated with computing the kernel matrix

Kernel Methods for Integrating PD into ML framework

- "Kernel methods" is indeed a *framework* for machine learning on non-vectorized data
- So, you can design different kernel functions that can be used in the framework which produces a different learning algorithm
- Another commonly used kernel function for PDs is the sliced Wasserstein kernel (SWK), which we will not cover in this course
- Designing kernel functions is non-trivial as you need to map your data into Reproducing Kernel Hilbert Spaces (RKHS) and prove that your kernel function is an inner product in the RKHS (which needs a lot of functional analysis :-( )
- But using existing kernels already designed is not so hard
- Although kernel methods can yield better results in some settings, they can be computationally intensive and impractical for large datasets due to the high computational costs associated with computing the kernel matrix
- In particular, computing kernels takes quadratic time in the number of diagrams, while vectorizing PDs takes only linear time

Stability of vectorizations

- In most applications, the stability of vectorization is vital for statistical and inferential tasks
- Essentially stability means that a small change in the persistence diagram (PD) should not lead to a significant change in its vectorization
- In particular, if two PDs, D_1 and D_2, are close, their corresponding vectorizations, $\vec{v}(D_1)$ and $\vec{v}(D_2)$, should also be close
- This ensures that the vectorization process preserves the structural properties of the data
- Therefore, when two persistence diagrams are similar, it implies that the datasets share similar shape characteristics (due to the stability of PD we learned before)
- If these datasets are intuitively expected to belong to the same class, their vectorizations should likewise remain close

Stability of vectorizations

- To measure the stability, we utilize the Wasserstein distance (which is more general)
- A vectorization technique $\vec{v}$ is said to be stable if it satisfies:

$$\|\vec{v}(D_1), \vec{v}(D_2)\| \leq W_p(D_1, D_2)$$

Stability of vectorizations

- To measure the stability, we utilize the Wasserstein distance (which is more general)
- A vectorization technique $\vec{v}$ is said to be stable if it satisfies:

$$\|\vec{v}(D_1), \vec{v}(D_2)\| \leq W_p(D_1, D_2)$$

- Among the methods described earlier, persistence landscapes, silhouettes, persistence images, and most kernel methods are stable vectorizations, while Betti functions are generally unstable

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand
- If your data contains a few prominent topological features that are crucial to the task, Silhouettes with $p \geq 2$ or Persistence Images may be the most suitable options

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand
- If your data contains a few prominent topological features that are crucial to the task, Silhouettes with $p \geq 2$ or Persistence Images may be the most suitable options
- These methods are also effective when dealing with noisy data, allowing you to filter out less significant features

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand
- If your data contains a few prominent topological features that are crucial to the task, Silhouettes with $p \geq 2$ or Persistence Images may be the most suitable options
- These methods are also effective when dealing with noisy data, allowing you to filter out less significant features
- Conversely, if your data generates a high number of small features, where the task hinges on their location and density – in other words, when the noise itself carries important information – Betti curves, Silhouettes with $p \leq 0.5$ and kernel methods are likely to yield strong performance

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand
- If your data contains a few prominent topological features that are crucial to the task, Silhouettes with $p \geq 2$ or Persistence Images may be the most suitable options
- These methods are also effective when dealing with noisy data, allowing you to filter out less significant features
- Conversely, if your data generates a high number of small features, where the task hinges on their location and density – in other words, when the noise itself carries important information – Betti curves, Silhouettes with $p \leq 0.5$ and kernel methods are likely to yield strong performance
- Lastly, if interpretability is a priority, Betti Curves stands out as the most interpretable vectorization method

Choice of Vectorization

- The choice of vectorization method should align with the characteristics of your data and the problem at hand
- If your data contains a few prominent topological features that are crucial to the task, Silhouettes with $p \geq 2$ or Persistence Images may be the most suitable options
- These methods are also effective when dealing with noisy data, allowing you to filter out less significant features
- Conversely, if your data generates a high number of small features, where the task hinges on their location and density – in other words, when the noise itself carries important information – Betti curves, Silhouettes with $p \leq 0.5$ and kernel methods are likely to yield strong performance
- Lastly, if interpretability is a priority, Betti Curves stands out as the most interpretable vectorization method
- Note that most vectorizations are computationally efficient and require minimal time compared to the computation of PDs

Choice of Hyperparameters

- Each vectorization method comes with its own set of hyperparameters that need to be carefully tuned to maximize performance

Choice of Hyperparameters

- Each vectorization method comes with its own set of hyperparameters that need to be carefully tuned to maximize performance
- Many vectorization methods have tuning parameters that are used to adjust sensitivity to topological noise

Choice of Hyperparameters

- Each vectorization method comes with its own set of hyperparameters that need to be carefully tuned to maximize performance
- Many vectorization methods have tuning parameters that are used to adjust sensitivity to topological noise
- In persistence landscapes, the number of landscapes specifies how many maxima are included in the representation
- A higher number of landscapes provides a richer depiction of topological features but increases computational complexity

Choice of Hyperparameters

- Each vectorization method comes with its own set of hyperparameters that need to be carefully tuned to maximize performance
- Many vectorization methods have tuning parameters that are used to adjust sensitivity to topological noise
- In persistence landscapes, the number of landscapes specifies how many maxima are included in the representation
- A higher number of landscapes provides a richer depiction of topological features but increases computational complexity
- For persistence images, the spread σ controls the width of the Gaussian kernels applied to each persistence point
- A smaller spread results in sharper, more localized features, whereas a larger spread yields smoother images

Choice of Hyperparameters

- Each vectorization method comes with its own set of hyperparameters that need to be carefully tuned to maximize performance
- Many vectorization methods have tuning parameters that are used to adjust sensitivity to topological noise
- In persistence landscapes, the number of landscapes specifies how many maxima are included in the representation
- A higher number of landscapes provides a richer depiction of topological features but increases computational complexity
- For persistence images, the spread σ controls the width of the Gaussian kernels applied to each persistence point
- A smaller spread results in sharper, more localized features, whereas a larger spread yields smoother images
- The resolution parameter sets the number of pixels in the persistence image, thus determining the output dimension of the vectorization
- Higher resolution captures finer detail but at a higher computational cost