Understanding the Power of Persistence Pairing via Permutation Test

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Recently many efforts have been made to incorporate persistence diagrams, one of major tools in topological data analysis (TDA), into machine learning pipelines. To better understand the power and limitation of persistence diagrams, we carry out a range of experiments on both graph data and shape data, aiming to decouple and inspect the effects of different factors involved. To this end, we also propose the so-called *permutation test*[1] for persistence diagrams to delineate critical values and pairings of critical values. For graph classification tasks, we note that while persistence pairing yields consistent improvement over various benchmark datasets, it appears that for various filtration functions tested, most discriminative power comes from critical values. For shape segmentation and classification, however, we note that persistence pairing shows significant power on most of the benchmark datasets, and improves over both summaries based on merely critical values, and those based on permutation tests. Our results help provide insights on when persistence diagram based summaries could be more suitable.

1 Introduction

Topological data analysis (TDA) is an emerging field that aims to characterize the shape of low and high dimensional data via methods steming from algebraic topology. One of the major tools of TDA is persistence diagrams (PDs). In recent years, many efforts have been made to utilize PDs as features for downstream machine learning tasks, such as material science [4], signal analysis [27] , cellular data [6] and shape recognition [23]. However, the geometry of the PD does not lend itself easily to well-adopted classifiers due to the lack of Hilbert structure.

To handle this issue, a natural way is to apply vectorization [3, 10, 11, 17] or kernelization [9, 28] to PDs, i.e., embedding PDs either to a Euclidean space $\mathbb{R}^d$ or a reproducing kernel Hilbert space (RKHS) associated with certain kernels. However, these approaches still have limitations. First, it has been shown that finite-dimension embedding can miss information about PDs [7]. Second, the time of computing a kernel is quadratic in the number of PDs, which is prohibitive for large scale applications. Third, choosing right vectorization/kernelization and its associated hyper-parameters is not straightforward and usually requires multiple rounds of trial and error.

Due to these extra complexities, one may wonder whether the benefits of using PDs outweigh the extra cost. Specifically, PDs have two major components: 1) filtration function and 2) persistence pairing (decomposition of persistence module). In this paper, we ask a simple yet fundamental question: *How much extra power can persistence pairing bring in?*

[1]The term shall not be confused with the permutation test in the statistical hypothesis testing.

Submitted to the Topological Data Analysis and Beyond Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

In this work, we propose a simple sanity check named "permutation test" that can shield light on the power of persistence diagram. Specifically, we permute the persistence points in such a way that only coordinates of PDs remain the same but the original pairing is completely destroyed. These fake PDs have the same form as the original PDs and therefore the same kernels for true diagrams can also be applied to fake ones. We use these fake diagrams as the input for various tasks and check their effectiveness. As we will see, this simple trick brings various insights on the use of PDs for different problems.

Our Contributions. To the best of our knowledge, our paper is the first work systematically quantifying (empirically) the power of persistence pairing for various applications. Specifically, our contributions are the following.

- We propose the permutation test for PDs that decouples the statistics of the critical values of filtration function and the persistence pairing. Using the proposed permutation test, we find that in graph classification, even fake diagrams perform quite well compared to original PDs . We believe this is due to the noisy nature of graph datasets (PDs are not stable against random insertion and deletion of edges). As a byproduct of our extensive experiments, we also provide some rules of thumb for using PDs in graph classification.
- For shape segmentation and classification, we find the power of persistence pairing depends on the particular featurization chosen. With the right choice of featurization, utilizing persistence pairing brings in significant improvement. Intuitively, we think that the shape models have more prominent geometric features in them, which are effectively captured by PDs. In contrast, PDs seem to be less effective at capturing features for graphs, partly due to the choice of descriptor functions as well as the nature of noise in graph (e.g., random insertions) which makes PDs less stable.

2 Experiment

2.1 Setup

On a high level, we use PDs obtained from different filtration functions as features. Depending on the task, we choose a proper featurization method (we use kernel methods when possible since they tend to perform better, but for large scale applications, computing kernels is not feasible so we use vector methods), followed by SVM for final classification. We maintain the same experiment settings for original PDs and permuted ones. For a comprehensive evaluation, we perform experiments on various tasks (graph classification, shape segmentation, and object classification) and diverse datatypes including social networks, molecules/proteins, and shapes of different categories.

We test degree (deg), Ricci Curvature [24] (Ricci), closeness centrality (cc) and square of Fiedler vector (the eigenvector corresponding to the second smallest eigenvalue of graph Laplacian, denoted as Fiedler-s) for graph classification. On shape datasets, we use geodesic distance as filtration function for shape segmentation and closeness centrality for shape classification We use Dionysus[2] to compute PDs and sklean_tda [3] for kernel computation. See details about kernels and datasets at appendix A.2.

2.2 Permutation Test and Baselines

In this section, we introduce permutation test that aims to preserve statistics of filtration function but destroy the persistence pairing of critical values of the filtration function. For a diagram P of n persistence points $P = \{p_1, p_2, ..., p_n\}$, denote the coordinates of point p_i by (x_i, y_i). Take the multiset $P_{\mathit{multiset}} = \{x_1, y_1, x_2, y_2, ..., x_n, y_n\}$ as input. We then randomly sample two values without replacement from P_{multiset} as the coordinates of the persistence point in the fake diagram. Repeat the same procedure n times and get a fake diagram called P_{fake} of size n.

To better quantify the power of persistence pairing, we also introduce two baselines. $\mathbf{Pervec}$ (vector obtained from coordinates of points in the $\mathbf{per}$muted diagram) is a histogram vector of the coordinates of PD, i.e., the histogram of P. $\mathbf{Filvec}$ is a histogram vector of all the $\mathbf{fil}$tration function values on the graph. The length of Pervec and Filvec is a hyper-parameter chosen from $\{100, 200, 300\}$ by cross-validation.

[2]http://mrzv.org/software/dionysus2/

[3]https://github.com/MathieuCarriere/sklearn_tda

3 Permutation Test for Graphs

In this section, we perform graph classification on common benchmark datasets with different filtration functions and featurizations. Through extensive experiments (see full table on synthetic and real graphs at appendix A.3), we draw the following conclusions.

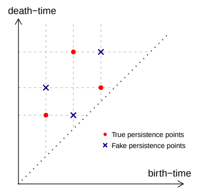

Figure 1: Visualize true and fake diagrams.

Choice of the Filtration Function. The choice of filtration function clearly matters. For datasets such as using Ricci curvature as the filtration function yields the best accuracy as opposed to other filtration functions.

If we fix method to be Sliced Wasserstein kernel, for graphs like `IMDB-B` (69.5 for degree vs. 69.2 for Ricci) and `IMDB-M` (43.1 vs. 43.7), `PROTEINS` (73.6 vs. 73.8), `D&D` (76.1 vs. 76.9), even degree performs as well as Ricci. This is consistent with the findings in the paper [5] where it is shown simple statistics based on node degree can perform on par with the state of the art. This raises the concern that the current benchmark datasets might be limited in evaluating different methods.

graph	BZR	COX2	D&D	DHFR	FRANK	IMDB-B	IMDB-M	NCI1	PROTEIN	PTC	REDDIT 5K
sw wo/ Ext_1^-	86.6	80.3	76.2	81.9	70.8	66.2	42.6	77.1	72.7	58.8	53.1
sw w/ Ext_1^-	**88.4**	80.5	76.9	**82.8**	**72.0**	69.5	45.7	77.8	74.0	58.7	54.1
pervec wo/ Ext_1^-	85.4	81.6	74.4	80.8	70.1	65.7	42.7	74.4	70.9	59.3	49.5
pervec w/ Ext_1^-	87.6	**81.6**	77.4	80.0	70.8	67.1	42.9	74.3	72.6	59.3	49.7
Perslay	87.2	**81.6**	-	81.8	70.7	70.9	48.7	72.8	74.8	-	56.6
WKPI-kM	-	-	**82.0**	-	-	70.7	46.4	**87.5**	**78.5**	62.7	59.1
WKPI-kC	-	-	80.3	-	-	**75.1**	**49.5**	84.5	75.2	**68.1**	**59.5**

Table 1: Fix featurlization as sw and look at the whether adding Ext_1^- in the PDs helps. The accuracy takes the maximum over different filtration function.

Sliced Wasserstein Kernel + Ricci is Powerful. Choosing the best accuracy for sw among different filtrations yields decent performance. `COX2`: 80.5 (best accuracy when using sw) vs. 82.0 (best accuracy among all filtration functions and featurlizations for a single dataset). `BZR`: 88.4 vs. 88.4. `DD`: 76.9 vs. 77.4. `DHFR`: 82.8 vs. 82.8. `FRANKENSTEIN`: 72.0 vs. 72.3. `IMDB-B`: 69.2 vs. 69.5. `IMDB-M`: 45.2 vs. 46.5. `NCI1`: 77.8 vs. 77.8. `PROTEINS`: 73.8 vs. 74.0. `REDDIT 5K`: 54.1 vs. 54.1.

As a corollary, to achieve good performance for graph classification, a rule of thumb is to use Ricci curvature as filtration plus Sliced Wasserstein kernel.

Permutation Test. Looking at the mean accuracies over four different filtration functions, sw performs better than sw-p/pervec/filvec, although the amount of improvement depends on the dataset. For `BZR`, `DHFR`, `FRANKENSTEIN`, `IMDB-B`, `NCI1`, `PROTEIN`, `REDDIT 5K` sw is better than filvec, pervec, and sw-p. For `COX2`, `dd`, sw is no better than the best of filvec/pervec/sw-p, but the gap (0.2 for `COX2` and 1.05 for `DD`) is small.

Pervec and filvec are used as baselines. The performance of pervec, filvec and sw-p is expected to be close to each other since none of them is using persistence pairing. This is indeed the case. The best of pervec and filvec is close to sw-p for all the graphs. (The difference is less than 2.5 percent.) Note hyper-parameter choice for pervec/filvec and sw-p will also result in different performance. After all, the input for pervec/filvec are vectors while for sw-p the input is fake PDs. We believe that the difference between pervec, filvec and sw-p is reasonable, and the conclusion that most discriminative power comes from function values is robust.

Learning for PDs. We compare the accuracy obtained from sw + best filtration function with Perslay/WKPI where learning is involved. As shown in Table 1, our method is comparable with Perslay. We conjecture replacing original filtration (based on heat kernel signature) used in Perslay with Ricci curvature may improve its performance for some datasets. WKPI (WKPI-kM and WKPI-kC differs in how they initialize weights.) learns the weights of different points in diagrams, which results in much better performance. This confirms the belief that to fully utilize the power of PDs, a proper weighting scheme is crucial.

The Effect of Adding Loops. We analyze the effect of adding loops (Ext_1^-) in extended PDs for graph classification. Since extended PDs capture loops in graphs with respect to the filtration function, the hope is that adding Ext_1^- in the PD will make it more discriminative. But one can perhaps argue the improvement may come from the coordinate values of extended PDs, so we fix featurization as pervec (as opposed to sw) to see the difference. Table 1 shows that adding coordinates of extended PDs increase the accuracy, no matter whether we utilize pairing (sw) or not (pervec).

4 Permutation Test for Shapes

We now perform permutation test on the problem of supervised 3D shape segmentation and object classification (see appendix A.4 due to space limitation). As we will see, the choice of permuting diagrams makes a much bigger difference here.

For each vertex x, we use the geodesic distance (distance to x) as the filtration function and compute the super-level 0-PDs as features. We convert PDs into feature vectors via either persistence landscape [3] or persistence image [1] and use SVM [25] as our classifier.

We use Princeton shape benchmark as dataset. This benchmark contains several different ground truth segmentations for each shape. For each shape in the training set, we use the same ground truth segmentation as [18]. For each category, we use 50% data for training and the rest for testing. The results are shown in Table 6, from which we draw following conclusions.

The effects of permutation test clearly depend on the vectorization selected. For Persistence Landscape (PL), there are some categories (such as cup, glasses, vase...) where permuting PDs yields better results than the original diagrams. We find corresponding diagrams for those categories all have very few persistence points (less than 2) far away from the diagonal on average. See diagram statistics in the appendix. In the formulation of persistence landscape, points close to diagonal are treated less important and thus play a less important role in final shape segmentation. In contrast, permuting PDs will "pull" those points away from diagonal with high probability and make them more important under PL's framework, therefore explaining the fact that permuting diagrams coupled with PL yields better results for those categories.

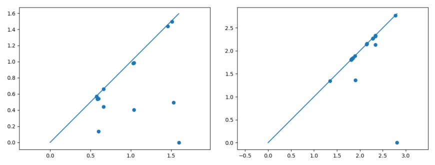

Figure 2: Two corresponding diagrams for Human and Cup. On average, diagrams in Human/Cup has 4.4/1.5 points away from diagonal.

In contrast, Persistence Image (PI) takes a very different way to vectorize PDs . In particular, PI assigns a weight for each point and points near diagonal are not necessarily assigned small weight. We use the same weight (proportional to death time) as in the original paper. It turns out that 1) permuting PDs for PI always gives worse result except for categories bearing and bust where a very small gap exists and 2) PI yields better results on all shapes compared to PL. Interestingly, given that PL gives smaller weights to points closer to the diagonal, while PI does not necessarily do (as in this experiments), we think this suggests that while PD can identify "features" of input shapes in a canonical way, the importance of these features (especially in terms of the tasks they are used for) may not be consistent with their persistence. This point has been made earlier in [1, 21, 34], which allow assigning different weights to persistence points. In our case, permutation test is proved to be a handy trick that can be applied to quickly test if the downstream featurization is effective.

5 Conclusion and Future Work

By introducing permutation test on PDs , we find persistence pairing is crucial for shape datasets. For graph datasets, although the small (yet consistent) improvements pairing brings may be unexpected and unsatisfying, we interpret this due to the challenging nature of graph classification and dataset problem. In the future, we are interested in understanding the discriminative power of PDs in a principle way. We believe developing a connection between the structure of persistence module (persistence pairing) and generalization error is important for the application of PDs in machine learning.

References

[1] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252, 2017.

[2] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[3] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015.

[4] Mickaël Buchet, Yasuaki Hiraoka, and Ippei Obayashi. Persistent homology and materials informatics. In *Nanoinformatics*, pages 75–95. Springer, Singapore, 2018.

[5] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attribute graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

[6] Pablo G Cámara. Topological methods for genomics: Present and future directions. *Current opinion in systems biology*, 1:95–101, 2017.

[7] Mathieu Carriere and Ulrich Bauer. On the metric distortion of embedding persistence diagrams into separable hilbert spaces. *arXiv preprint arXiv:1806.06924*, 2018.

[8] Mathieu Carriere, Frederic Chazal, Yuichi Ike, ThÃl'o Lacombe, Martin Royer, and Yuhei Umeda. Perslay: A simple and versatile neural network layer for persistence diagrams. *arXiv preprint arXiv:1904.09378*, 2019.

[9] Mathieu Carriere, Marco Cuturi, and Steve Oudot. Sliced wasserstein kernel for persistence diagrams. *arXiv preprint arXiv:1706.03358*, 2017.

[10] Mathieu Carrière, Steve Y Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. In *Computer Graphics Forum*, volume 34, pages 1–12. Wiley Online Library, 2015.

[11] Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. Stochastic convergence of persistence landscapes and silhouettes. In *Proceedings of the thirtieth annual symposium on Computational geometry*, page 474. ACM, 2014.

[12] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. In *Acm transactions on graphics (tog)*, volume 28, page 73. ACM, 2009.

[13] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[14] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[15] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[16] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, pages 1634–1644, 2017.

[17] Sara Kališnik. Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, 19(1):101–129, 2019.

[18] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. In *ACM Transactions on Graphics (TOG)*, volume 29, page 102. ACM, 2010.

[19] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.

[20] Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

[21] Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted gaussian kernel for topological data analysis. In *International Conference on Machine Learning*, pages 2004–2013, 2016.

[22] Tam Le and Makoto Yamada. Persistence fisher kernel: A riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, pages 10007–10018, 2018.

[23] Chunyuan Li, Maks Ovsjanikov, and Frederic Chazal. Persistence-based structural recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1995–2002, 2014.

[24] Yong Lin, Linyuan Lu, and Shing-Tung Yau. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series*, 63(4):605–627, 2011.

[25] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, pages 3778–3787, 2017.

[26] Steve Y Oudot. *Persistence theory: from quiver representations to data analysis*, volume 209. American Mathematical Society Providence, RI, 2015.

[27] Jose A Perea and John Harer. Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15(3):799–838, 2015.

[28] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4741–4748, 2015.

[29] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[30] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003.

[31] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[32] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[33] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[34] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. *arXiv preprint arXiv:1904.12189*, 2019.

A Missing details

A.1 Background

A.1.1 Persistent Homology

The definition of our proposed method relies on the so-called persistence diagram induced by a scalar function. We refer readers to resources such as [14, 26] for formal discussions on persistent homology and related developments. Below we only provide an intuitive and informal description of the persistent homology induced by a function under a simple setting. Let $f : X \to \mathbb{R}$ be a continuous real-valued function defined on a topological space X. We want to understand the structure of X from the perspective of f. Specifically, let $X_\alpha := \{x \in X | f(x) < \alpha\}$ denote the sublevel set of X w.r.t. $\alpha \in \mathbb{R}$. Now as we sweep X bottom-up (top down) by increasing the value, the sequence of sublevel (superlevel) sets connected by natural inclusion maps gives rise to a filtration of X induced by f:

$$X_{\alpha_1} \subset X_{\alpha_2} \subset ... \subset X_{\alpha_m} = X, \alpha_1 < \alpha_2 < ... < \alpha_m \quad (1)$$

We track how the topological features of sublevel sets change in terms of homology classes. In particular, as α increases, sometimes new topological features are born at time α, that is, new families of homology classes are created in $H_k(X_\alpha)$, the k-th homology group of X. Sometimes, existing topological features disappear, i.e, some homology classes become trivial in $H_k(X_\beta)$ for some $\beta > \alpha$. The persistent homology captures such birth and death events, and summarizes them in the so-called persistence diagram $Dg_k(f)$ (We will call it diagram for short when no ambiguity is raised). Specifically, $Dg_k(f)$ consists of a set of persistence points $(\alpha, \beta) \in \mathbb{R}^2$, where each (α, β) indicates a k-th homological feature created at α and killed at β. we also call (α, β) persistence pairing since it pairs two critical values α and β of the filtration function.

In particular, 0 homology is just connected components and can be computed efficiently in $O(\alpha(n)n)$ using union-find data structure where $\alpha(n)$ is an extremely slow-growing inverse Ackermann function.

A.1.2 Extended Persistence Homology

Ordinary persistence sometimes may be insufficient to encode the topology of an object X. For example, when X is a graph, the loops persist forever since they are not filled during the sublevel filtration. Similarly, upfork branching points (w.r.t. the filtration function f) are not captured (while those pointing downwards are detected), since they do not create connected components in the sublevel filtration.

To address the issues above, extended persistence refines the analysis by also including the super-level set $X^\alpha = \{x \in X : f(x) \geq \alpha\}$ into the filtration in Eqn (1). Similarly, letting α decrease from ∞ to $-\infty$ gives a sequence of increasing subsets, for which structural changes can be recorded. In particular, assuming we have a sequence of reals $\alpha_1 < \alpha_2 < \cdots < \alpha_m$ such that $X_{\alpha_1} = \emptyset$ $(X^{\alpha_1} = X)$ and $X_{\alpha_m} = X$ $(X^{\alpha_m} = \emptyset)$, we consider the following extended sequence:

$$\begin{aligned} \emptyset =& X_{\alpha_1} \subseteq X_{\alpha_2} \subseteq \cdots \subseteq X_{\alpha_m} = (X; X^{\alpha_m}) \\ \subseteq& (X; X^{\alpha_{m-1}}) \subseteq \cdots \subseteq (X; X^{\alpha_2}) \subseteq (X; X^{\alpha_1}) = \emptyset. \end{aligned} \tag{2}$$

where $(A;B)$ denotes a pair of space, and at the homology level, note that the natural map from $X_{\alpha_m} \to (X; X^{\alpha_m})$ induces an isomorphism. One can then consider the resulting persistent homology induced by the above extended sequence, and the resulting PDs are called extended PDs. Persistence pairings in such diagrams have four types, depending on whether the birth and death happen during the upward filtration (first line in Eqn (2)) or downward filtration (second line in Eqn (2)). In the context of graphs, these types are denoted as Ord_0, Rel_1, Ext_0^+ and Ext_1^- for downwards branches, upwards branches, connected components and loops respectively. Overall, we denote $Dg(G, f) = Ord_0(G, f) \cup Rel_1(G, f) \cup Ext_0^+(G, f) \cup Ext_1^-(G, f)$.

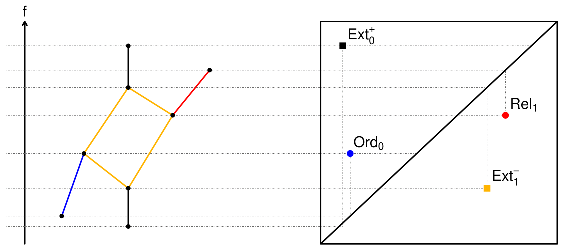

Figure 3: Given the filtration f as height for graph X, the four types of features of graphs and their corresponding persistence points in the extended persistence diagram.

A.1.3 PDs for Machine Learning

PDs have been proposed as features for machine learning in a series of work (e.g, [1, 11]), starting from persistence landscapes [3]. We briefly review related methods in this section. The detailed information can be found in the appendix.

Kernel Method. Given a set $\mathcal{X}$ (PDs in our case, a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a positive definite kernel if for all integers n and all families $x_1, ..., x_n$ of n elements in X, the matrix $[k(x_i; x_j)]_{i,j}$ itself is positive semi-definite. It is known that kernels generalize scalar products, in the sense that, given a kernel k, there exists a RKHS $\mathcal{H}_k$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}_k$ such that $k(x_1, x_2) =<$

$\phi(x_1), \phi(x_2) >_{\mathcal{H}_k}$. A kernel k also induces a distance d_k that can be computed as the Hilbert norm of the difference between two embeddings: $d_k^2(x_1, x_2) = k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2)$.

In this paper, we try four common kernels for diagrams, i.e. Persistence Weighted Gaussian Kernel (pwg) [21], Persistence Scale Space Kernel (pss) [28], Sliced Wasserstein Kernel (sw) [9] and Persistence Fisher Kernel (pf) [22].

Vector Method. Another way to use PDs for machine learning is to convert them into vectors. Persistence Landscape (PL) and Persistence Image (PI) are two examples. One advantage of vectorization over kernelization is that computing kernels takes quadratic time in the number of diagrams while vectorizing PDs takes only linear time. Thus, we only use vector methods for shape segmentation where the number of diagrams is roughly 20k-50k.

A.1.4 PDs for Graphs

For graph classification, [16] is the first work introducing a neural network framework to convert PDs into feature vectors for graph classification in an end-to-end data-dependent way. Perslay [8] unifies many existing featurization such as persistence landscape, persistence silhouette [11] and persistence surface as different instances of a single permutation invariant neural network based on DeepSets [33]. Weighted Persistence Image Kernel [34] (WKPI) is a recently proposed weighted kernel based on persistence image. WKPI assigns weights for different locations in PDs where weights are learned from data via gradient descent. Empirically, improved performance over other kernels is shown for the task of graph classification.

A.2 Missing Details at Section 2: Datasets and Choice of Kernels

We apply permutation test on diverse data types, whose quantitative summaries are in the appendix.

Graph Datasets: We perform experiments on common benchmark datasets for graph classification. `IMDB-B`, `IMDB-M`, `REDDIT 5K` are composed of social networks. `BZR`, `COX2`, `DD`, `DHFR`, `FRANKENSTEIN`, `NCI1`, `PROTEINS`, `PTC` are graphs from medical or biological applications.

Shape Datasets: We use Princeton Shape Benchmark [12] for shape segmentation. This benchmark contains 19 categories of different objects (human, cup, glasses...) with 20 shapes for each category. For shape classification, we use ModelNet10/ModelNet40 [31] where there are 4899/12,311 CAD models from 10/40 man-made object categories (bathtub, bed, chair...) respectively.

Choice of Kernels: We test four kernels for PDs i.e., sw, pss, pwg and pf for different filtration functions on `PROTEINS`. As shown in Table 2 in the appendix, the performance of different kernels are close in terms of accuracy. We prefer Sliced Wasserstein kernel mainly because it is fast ($O(mlogm)$ as opposed to $O(m^2)$ for pss), easy to tune (search over 5 values for bandwidth as opposed to 45 hyper-parameter combinations for pwg), and still yields decent performance. Note that in this paper we focus on understanding the extra power persistence pairing brings in, not achieving the state of the art.

Table 2: Performance of different filtration function and persistence kernels for `PROTEINS` dataset.

method	cc	deg	Fiedler-s	Ricci	mean
pf	70.3	73.4	72.8	70.6	71.78
pss	74.3	72.5	73.9	73.6	73.58
sw	74.0	73.6	73.5	73.8	73.72
pwg	74.7	72.4	68.4	73.8	72.32

A.3 Missing Details at Section 3: Synthetic Graph Data

In this section, we perform permutation test on synthetic graph dataset where graphs sampled from 2 stochastic block models are classified. In particular, denote $sbm(n_1, n_2, p, q)$ as stochastic block model of two blocks of size n_1 and n_2, and within each block, the edge probability is p, while between blocks, the edge probability is q. We sample 1000 graphs for classification from $sbm(100, 50, 0.5, 0.1)$ and $sbm(75, 75, 0.4, 0.2)$. This is a simple classification problem and there are many ways to achieve perfect results.

To make the classification harder, we randomly flip some labels. For example, when label noise is 0.1, we randomly flip 10% of labels. We are interested in the behaviors of whether to perform permutation test or not under different label noise levels. As shown in Figure 4, independent from filtration functions used, applying permutation test has rather small effects on the final performance for different noise levels.

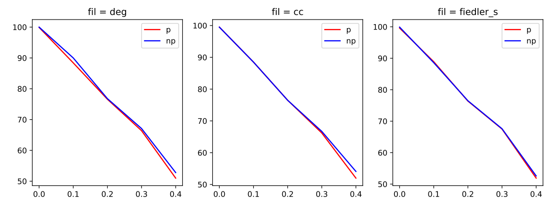

Figure 4: Results on synthetic graph data where the difference between permuting diagram versus not permuting is rather small.

A.4 Missing Details at Section 4: 3D Object classification

Next, we evaluate our method for shape classification on ModelNet10/ModelNet40. For each shape, 1024 points are uniformly sampled on mesh faces according to face area and normalized into a unit sphere. We then construct a 8-neighborhood graph from sampled points and use closeness centrality as our filtration function. For each shape, we use the resulting PDs as the shape representations and use sw/pf/PI/PL for object classification.

Table 5 shows that if we do not utilize persistence pairing, we get very similar results for three methods (P, Pervec, Filvec), which are much worse than result using PDs (NP), regardless of featurlizations. A detailed performance breakdown on ModelNet 10 is also shown in Table 4 where permuting diagrams yields much worse results on all shape categories except dresser.

Results both in Table 6 and 5 suggest that persistence pairings are effective and meaningful for 3D models, which partly could be due to that 3D models tend to have clear geometric features that are also stable under the typical types of (Hausdorff) noise added to these models.

In contrast, persistent homology seems to be less effective at capturing features for graphs: this could partly be due to the choice of descriptor functions used for graphs are not effective at capturing features. Another potential reason could be that PDs are more sensitive to the common types of noise in graphs (random insertions), and hence resulting persistence-based features are less stable and meaningful.

A.5 Analysis of True and Fake Diagrams

A.5.1 Separation from True and Fake Diagrams

Due to the decent result without using persistence pairing on graph datasets, one may wonder whether there is any useful structure contained in persistence pairing for graph classification. To better understand the implications of the permutation test, we try to separate true PDs from fake PDs.

For each dataset and filtration function, we compute PDs and generate fake diagrams by applying permutation test. We compute the Sliced Wasserstein kernel and train SVM to discriminate the true diagrams from the fake ones. We observe in Table **??** consistently that regardless of the datasets and filtration functions, we can easily get 85%-100% accuracy. This suggests that true diagrams have some structure that is very different from fake ones.

A.5.2 Confusion Matrix Analysis

We also examine the confusion matrix in cases where we need to additionally differentiate true/fake diagrams. In particular, for each graph G_i with label y_i (assume all labels are represented as positive numbers), we compute the true PD of each graph and generate a fake diagram. In scenario 1) we assign both diagrams as same label y_i and in scenario 2), we assign two diagram with different label y_i and $-y_i$. In other words, in the second scenario, a classifier has to differentiate both true/fake

Table 3: The accuracy obtained from filvec, pervec, sw (Sliced Wasserstein Kernel) and sw-p (with permutation) for different filtration functions and datasets.

graph	method	cc	deg	Fiedler-s	Ricci	mean
BZR	filvec	80.7(0.6)	83.4(0.5)	80.5(0.7)	87.6(0.8)	83.05
	pervec	85.4(0.5)	82.5(0.7)	80.5(0.6)	87.6(0.5)	84.00
	sw	86.2(0.5)	83.4(0.6)	81.0(0.4)	88.4(0.6)	84.75
	sw-p	85.1(0.5)	82.2(0.6)	82.2(0.7)	82.7(0.7)	83.05
COX2	filvec	78.8(0.5)	78.2(0.6)	78.8(0.7)	82.0(0.6)	79.45
	pervec	78.8(0.6)	78.2(0.6)	79.4(0.7)	81.6(0.5)	79.50
	sw	79.9(0.7)	78.6(0.5)	78.2(0.7)	80.5(0.5)	79.30
	sw-p	78.2(0.6)	78.6(0.6)	78.8(0.5)	80.1(0.7)	78.93
DD	filvec	75.0(0.4)	72.5(0.4)	67.0(0.6)	76.8(0.6)	72.82
	pervec	75.1(0.4)	70.5(0.6)	66.4(0.5)	77.4(0.5)	72.35
	sw	76.1(0.6)	76.1(0.5)	71.1(0.5)	76.9(0.5)	75.05
	sw-p	76.7(0.5)	76.0(0.6)	74.3(0.4)	77.4(0.5)	76.10
DHFR	filvec	75.1(0.4)	67.3(0.6)	72.1(0.5)	80.1(0.6)	73.65
	pervec	75.5(0.6)	71.3(0.5)	73.3(0.5)	80.8(0.4)	75.23
	sw	79.0(0.7)	73.4(0.6)	74.7(0.4)	82.8(0.5)	77.48
	sw-p	78.7(0.5)	74.7(0.5)	76.6(0.6)	76.5(0.6)	76.62
FRANK	filvec	65.3(0.2)	66.8(0.3)	62.8(0.3)	72.3(0.2)	66.80
	pervec	65.2(0.2)	65.4(0.2)	63.0(0.3)	70.8(0.2)	66.10
	sw	67.8(0.3)	67.3(0.4)	67.1(0.3)	72.0(0.2)	68.55
	sw-p	65.6(0.2)	66.8(0.3)	65.0(0.2)	69.0(0.3)	66.60
IMDB-B	filvec	66.4(0.5)	64.6(0.6)	60.4(0.6)	65.6(0.6)	64.25
	pervec	65.7(0.5)	67.1(0.6)	63.1(0.6)	63.7(0.5)	64.90
	sw	69.5(0.6)	69.5(0.5)	66.5(0.6)	69.2(0.5)	68.68
	sw-p	66.1(0.5)	67.8(0.5)	65.2(0.7)	67.5(0.5)	66.65
IMDB-M	filvec	46.0(0.3)	46.1(0.3)	43.5(0.4)	46.5(0.3)	45.52
	pervec	42.5(0.3)	42.7(0.3)	42.1(0.3)	42.9(0.3)	42.55
	sw	42.6(0.4)	42.6(0.4)	45.7(0.2)	45.2(0.3)	44.03
	sw-p	42.3(0.3)	42.3(0.3)	45.0(0.3)	42.6(0.2)	43.05
NCI1	filvec	69.3(0.2)	64.7(0.3)	67.0(0.3)	74.3(0.3)	68.82
	pervec	69.7(0.2)	63.4(0.3)	64.2(0.3)	74.4(0.1)	67.93
	sw	74.9(0.2)	67.2(0.2)	72.1(0.2)	77.8(0.3)	73.00
	sw-p	69.9(0.3)	64.9(0.2)	69.3(0.2)	71.1(0.2)	68.80
PRO-	filvec	72.4(0.4)	68.3(0.4)	71.7(0.3)	71.2(0.4)	70.90
TEINS	pervec	72.6(0.4)	69.4(0.4)	71.1(0.4)	70.9(0.3)	71.00
	sw	74.0(0.4)	73.6(0.2)	73.5(0.3)	73.8(0.3)	73.72
	sw-p	73.1(0.3)	72.1(0.3)	72.8(0.3)	73.0(0.2)	72.75
REDDIT	filvec	47.0(0.1)	45.8(0.2)	40.3(0.1)	51.0(0.2)	46.03
5K	pervec	49.2(0.2)	48.9(0.1)	39.0(0.1)	49.7(0.2)	46.70
	sw	52.6(0.2)	49.1(0.1)	42.5(0.1)	54.1(0.1)	49.58
	sw-p	50.4(0.1)	49.2(0.1)	41.9(0.1)	53.3(0.2)	48.70

Table 4: The performance ($F1$ score) breakdown of using PDs for 3D object classification on ModelNet10.

	mean	bathtub	bed	chair	desk	dresser	monitor	night stand	sofa	table	toilet
# of shapes	-	156	615	989	286	286	565	286	780	492	444
$F1$ score wo/ permutation	57.2	54.1	56.4	76.7	21.2	36.3	64.5	46.8	62.1	54.2	74.0
$F1$ score w/ permutation	44.3	24.3	37.3	63.7	0.0	46.1	44.7	31.2	48.8	16.1	15.2

Table 5: Classification accuracy on ModelNet. Four numbers under NP (no permutation) and P (permutation) are accuracies for sw, pf, PI and PL.

	NP	P	Pervec	Filvec
ModelNet-10	55.2/53.6/53.1/53.2	42.7/42.9/41.5/41.5	43.5	44.5
ModelNet-40	43.4/35.8/35.7/34.2	28.1/27.5/27.1/26.9	28.5	30.1

Table 6: The performance (error) of Persistence Landscape (PL) and Persistence Image (PI) for permuting diagrams (P) and not permuting diagrams (NP).

	PL + NP	PL + P	PI + NP	PI + P
Human	7.2	20.9	3.2	8.2
Cup	8.8	5.3	2.6	3.6
Glasses	2.6	2.4	1.6	2.0
Airplane	9.7	17.6	3.5	10.7
Ant	2.5	5.0	1.6	4.6
Chair	3.7	5.0	1.1	4.3
Octopus	2.8	4.5	1.1	3.8
Table	1.6	1.7	0.3	0.9
Teddy	18.6	16.6	4.0	11.3
Hand	10.8	19.0	1.8	10.3
Plier	3.2	4.9	2.9	4.4
Fish	14.4	10.3	7.7	8.1
Bird	8.4	10.8	3.0	9.4
Armadillo	17.1	39.7	4.6	23.6
Bust	44.0	32.5	16.7	15.9
Mech	23.8	18.3	11.0	13.6
Bearing	13.6	6.7	3.3	2.9
Vase	24.3	16.1	7.0	9.2
Fourleg	13.0	20.7	3.5	11.5

Table 7: The accuracy of seprating true PDs from fake ones for different graphs and filtration functions.

	deg	Ricci	cc
`IMDB-B`	99.8	99.5	99.6
`IMDB-M`	99.8	99.8	99.7
`REDDIT 5K`	95.7	87.6	91.7
`DD`	99.5	99.3	99.6

diagrams and diagrams of different types of graphs. We want to know whether scenario 2 will make problem harder.

In particular, we use node degree as filtration function and Sliced Wasserstein kernel. It can be seen in Table 8 that fake diagrams never get confused with true diagrams. We can recover the confusion matrix on the left from the matrix on the right by merging true and fake diagrams.

B Concepts

Due to the space limitation, we list definitions of some concepts needed for the paper in the appendix. We refer readers to [14, 26] for more details.

Table 8: Confusion matrices obtained from kernel SVM without (left)/with (right) fake PDs on DD. I/II: graph types. T/F: whether PDs used are true or fake.

	I	II
I	121	22
II	38	55

	I+T	I+F	II+T	II+F
I+T	65	0	8	0
II+F	0	56	0	14
I+T	25	0	29	0
II+F	0	13	0	26

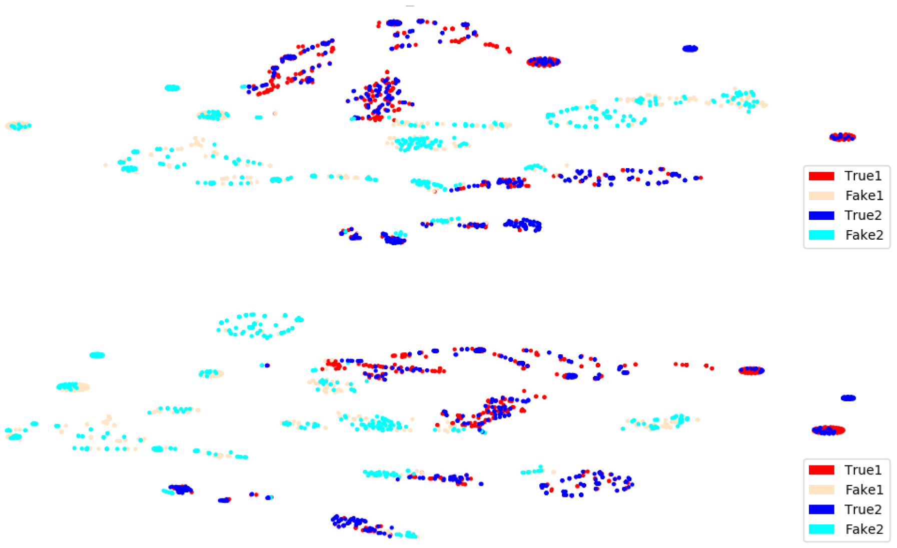

Figure 5: The visualization of true/fake PDs in dark/bright color for IMDB-B. Top: TSNE with kernel distance induced from Sliced Wasserstein kernel. Bottom: TSNE with bottleneck distance (see appendix).

B.1 Homology

The key concept of homology theory is to study the properties of some object X by means of (commutative) algebra. In particular, we assign to X a sequence of modules $C_0, C_1, ...$ which are connected by homomorphisms $\partial_n : C_n \to C_{n-1}$ such that im$\partial_n \subseteq$ ker∂_n A structure of this form is called a chain complex and by studying its homology groups $H_n = \ker \partial_n / \operatorname{im} \partial_{n+1}$ we can derive properties of X.

B.2 Bottleneck Distance

Given two PDs D_1 and D_2, let $\Gamma : D_1 \supseteq A \to B \subseteq D_2$ be a partial bijection between D_1 and D_2. Then for any point $x \in A$, the *p-cost* of x is defined as $c_p(x) = \|x - \Gamma(x)\|_\infty^p$ and for $y \in (D_1 \cup D_2) \backslash (A \cup B)$, the p-cost of y is defined as $c_p'(y) = \|y - \pi_\Delta(y)\|_\infty^p$, where π_Δ the projection onto the diagonal $\Delta = \{(x, x) : x \in \mathbb{R}\}$. The cost of this partial bijection Γ is defined as $c_p(\Gamma) = \left(\sum_x c_p(x) + \sum_y c_p'(y)\right)^{1/p}$. Finally, define the p-th diagram distance d_p as the cost of the best partial bijection:

$$d_p(D_1, D_2) = \inf_\Gamma c_p(\Gamma), \tag{3}$$

where Γ ranges over all partial bijections between D_1 and D_2. In the particular case when $p = \infty$, the cost of Γ is therefore $c(\Gamma) = \max\{\max_x c_1(x) + \max_u c_1'(y)\}$. The corresponding distance d_∞ is often called the bottleneck distance between diagrams D_1 and D_2.

C Diagram Statistics

We list diagram statistics for the shape segmentation task in Table 9. Ave # of PD points (first row) stands for the average number of persistence points in the diagram for a shape category. A persistence point is considered as near diagonal if its lifetime is less than one-tenth of the lifetime of the furthest point in the same diagram.

As we can see in Table 9, most persistence points for mech, bust, cup, bearing, glasses, fish, vase, teddy are concentrated near diagonal. Those are exactly the same categories on which permuting diagram yields much better results than not permuting diagrams when PI is used.

Table 9: The diagram statistics for different shape categories in Princeton Shape Benchmark.

Categroy	Mech	Bust	Cup	Bearing	Glasses	Fish	Vase	Teddy	Bird	Plier	Airplane	Table	Human	Armadillo	Chair	Hand	Fourleg	Octopus	Ant
Ave # of PD points	11	22.8	15.9	16.9	7.5	15.1	15.7	18.3	18	6.6	12.5	16.6	58.2	78.6	18.8	19.7	31.4	14.6	15.9
# of points near diagonal	9.9	21.5	14.4	15.3	5.6	13.2	13.7	15.1	14.3	2.9	8.5	12.3	53.7	74.1	14.1	14.5	25.9	6.7	7.3
Difference	1.1	1.3	1.5	1.6	1.9	1.9	2	3.2	3.6	3.7	4	4.3	4.4	4.6	4.7	5.1	5.4	7.9	8.6

D Vector Methods for PDs

Persistence Landscape [3] is the first proposed vectorization method for PDs to overcome some undesirable property of the space of PDs , such as lacking a unique Frechet mean. This construction is mainly intended for statistical computations, enabled by the vector space structure of L_p. Given a PD $D = \{(b_i, d_i)\}_{i=1}^m$, persistence landscape can be thought of as a sequence of functions $\lambda_k : \mathbb{R} \to \bar{\mathbb{R}}$ where $\lambda_k(t) = k$th largest value of min $(t - b_i, d_i - t)_+$.

Persistent Image [1] produces a persistence surface ρ_B from a PD by taking a weighted sum of Gaussians centered at each point. The vector representation, named by persistence image, is created by integrating persistence surface over a grid. In particular, they fix a grid in the plane with n pixels and assign to each the integral of ρ_B over that region.

There are at least three parameters involved in the construction of persistence image: 1) a non-negative weighting function 2) the bandwidth of Gaussian kernel (many other functions can be chosen but in the original paper only Gaussian is considered) and 3) the resolution of the grid put over the persistence surface. The authors report that in classification experiments they conducted, the accuracy is insensitive to the choice of resolution and bandwidth.

E Kernels Methods for PDs

Persistence Weighted Gaussian Kernel [21] essentially utilizes the idea of kernel mean embedding of distribution, where persistence diagram, treated as a special case of distribution, can be embedded into RKHS. In particular, Let $K, \rho > 0$ and D_1 and D_2 be two PDs. Let K_ρ be the Gaussian kernel with parameter $\rho > 0$. Let H_ρ be the RKHS associated to k_ρ .

Let $\mu_1 = \Sigma_{x \in D_1} \text{arctan}(Kpers(x)^p k_\rho(*, x)) \in H_p$ be the kernel mean embedding of D_1 weighted by the diagonal distances. Let μ_2 be defined similarly. Let $\tau > 0$, the persistence weighted gaussian kernel K_{pwg} is defined as the gaussian kernel with bandwidth τ on H_p :

$$K_{pwg}(D_1, D_2) = e^{-\frac{\|\mu_1 - \mu_2\|_{H_p}}{2t^2}} \tag{4}$$

Persistence Scale Space Kernel [28] represents persistence diagram as sum of Dirac's delta measure. The persistence scale space kernel is defined as the scalar product of the solution of the heat diffusion equation with the persistence diagram as an initial value.

The closed form

$$K_{pss}(D_1, D_2) = \frac{1}{8} \sum_{p \in D_1} \sum_{q \in D_2} e^{(-\frac{\|p-q\|^2}{8t})} - e^{(-\frac{\|p-\bar{q}\|^2}{8t})} \tag{5}$$

can be computed exactly in $O(|D_1| * |D_2|)$ time where $\bar{q} = (y, x)$ is the symmetric of $q = (x, y)$ along the diagonal. $|D_1|$ and $|D_2|$ denote the cardinality of the multisets D_1 and D_2

Sliced Wasserstein Kernel [9] uses Sliced Wasserstein approximation of the Wasserstein distance to define a new kernel for PDs. Different from previous multiple kernels, it is provable not only *stable* but also *discriminative* (with a bound depending on the number of points in the PDs) w.r.t. the first diagram distance w_1^∞ between PDs.

In particular, the kernel has the following closed-form:

$$K_{sw}(D_1, D_2) = e^{\frac{-SW(D_1,D_2)}{2\sigma^2}} \tag{6}$$

where $SW(D_1, D_2)$, is defined as the sliced Wasserstein distance between PDs.

Persistence Fisher Kernel [22] differs from slice Wasserstein kernel in the sense that the Wasserstein geometry is replaced by Fisher information geometry (metric), which induces a negative definite distance. The form of Persistence fisher kernel is

$$k_{PF}(D_1, D_2) = e^{-td_{FIM}(D_1,D_2)} \tag{7}$$

where t is a positive scalar and d_{FIM} is the Fisher information metric.

Weighted Persistence Image Kernel (WKPI) [34] is recently proposed weighted kernel that is based on persistence image. In particular, WKPI assigns weight for different location in the diagram where the weight is learned via gradient descent. The form of WKPI is

$$k_w(PI, PI') = \Sigma_{s=1}^N w(p_s) e^{-\frac{(PI(s)-PI'(s))^2}{2\sigma^2}} \tag{8}$$

where PI, PI' are persistence image, s is the location for each pixel in PI, w is the weight function that will be learned from data. Note that majority of time to compute WKPI is spent on learning $w(p_s)$ via stochastic gradient descent.

Table 10: Summary of different kernels for PDs. Here n is the number of diagrams. m is the number of persistence points in the diagram of largest size. M_1 is the number of random Fourier feature used for approximating Gaussian kernel. M_2 is the number of directions for approximating Sliced Wasserstein kernel.

	pss	pwg	sw	pf	wkpi
# of Param	1	3	1	2	2
Exact Comp. Time	$O(m^2n^2)$	$O(m^2n^2)$	$O(m^2log(m)n^2)$	$O(m^2n^2)$	-
Approx. Time	-	$O(M_1mn + M_1n^2)$	$O(M_2mlog(m)n^2)$	$O(mn^2)$	-

F Hyper-parameters Choice

We chose our hyper-parameters by 10-fold cross-validation on training set. We list the specific search range for all methods below. All experiments are performed on a single

Persistence Landscape: the number of $\lambda_k(t)$ are chosen from {3, 4, 5, 6, 7, 8} and each function $\lambda_k(t)$ is discretized into {50, 100, 200, 300} bins.

Persistence Image: we use the same weight function (Gaussian) in the original paper, the resolution chosen from {20*20, 30*30}, and bandwidth of Gaussian is selected from {0.01, 0.1, 1, 10, 100}.

Persistence Scale Space Kernel: the parameter t is chosen from 13 values: {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100, 500, 1000}.

Sliced Wasserstein Kernel: following the original paper [9], we grid search bandwidth from the 5 values: {0.01, 0.1, 1, 10, 100} and the number of slices, i.e., M_2 is set to be 10.

Persistence Weighted Gaussian Kernel: we try all the combinations of 5 values from {0.01, 0.1, 1, 10, 100} for bandwidth, and 3 values from {0.1, 1, 10} for both K and ρ, leading to 45 different sets of parameters.

Persistence Fisher Kernel: there are two hyper-parameters t and τ (for smoothing persistence diagram), both of which are selected from {0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100}.

Pervec/Filvec: The length of the vector is chosen from {100, 200, 300}.

Choice of SVM hyper-parameter: For kernel SVM, the only hyper-parameter C is selected from $\{0.01, 1, 10, 100, 1000\}$. For Pervec and Filvec, Gaussian kernel is utilized where the bandwidth is selected from $\{0.01, 0.1, 1, 10, 100\}$.

Graph Classification: We follow the standard protocol in graph classification literature, i.e., 10-fold cross-validations, using 9 folds for training and the rest for testing, and repeat the experiments 10 times. We report the average classification accuracies.

G Datasets Description

The statistics of the benchmark graph datasets used in the paper are reported in Table 6. Due to space limitation, we only list the dataset statistics here. We provide detailed dataset description in the data appendix.

Table 11: Statistics of the benchmark graph datasets.

Datasets	graph #	class #	average_nodes #	average edges #	label #
BZR	405	2	35.75	38.36	+
COX2	467	2	41.22	43.45	+
DD	1178	2	284.32	715.66	+
DHFR	467	2	42.43	44.54	+
FRANKSTEIN	4337	2	16.90	17.88	-
IMDB BINARY	1000	2	19.77	96.53	-
IMDB MULTI	1500	3	13.00	65.94	-
NCI1	4110	2	29.87	32.30	+
PROTEINS	1113	2	39.06	72.82	+
PTC	344	2	14.29	14.69	+
REDDIT 5K	4999	5	508.82	594.87	-

G.1 Non-attributed Graph Datasets

IMDB-BINARY [32] is a movie collaboration dataset that consists of the ego-networks of 1,000 actors/actresses who played roles in movies in IMDB. In each graph, nodes represent actors/actresses, and there is an edge between them if they appear in the same movie. These graphs are derived from the Action and Romance genres.

IMDB-MULTI is generated in a similar way to IMDB-BINARY. The difference is that it is derived from three genres: Comedy, Romance, and Sci-Fi.

REDDIT-BINARY consists of graphs corresponding to online discussions on Reddit. In each graph, nodes represent users, and there is an edge between them if at least one of them responds to the other's comment. There are four popular subreddits, namely, IAmA, AskReddit, TrollXChromosomes, and atheism. IAmA and AskReddit are two question/answer based subreddits, and TrollXChromosomes and atheism are two discussion-based subreddits. A graph is labeled according to whether it belongs to a question/answer-based community or a discussion-based community.

REDDIT-MULTI (5K) is generated in a similar way to REDDIT-BINARY. The difference is that there are five subreddits involved, namely, worldnews, videos, AdviceAnimals, aww, and mildlyinteresting. Graphs are labeled with their corresponding subreddits.

G.2 Attributed Graphs

PTC [15] consists of graph representations of chemical molecules. In each graph, nodes represent atoms, and edges represent chemical bonds. Graphs are labeled according to carcinogenicity on rodents, divided into male mice (MM), male rats (MR), female mice (FM), and female rats (FR).

PROTEINS [2] consist of graph representations of proteins. Nodes represent secondary structure elements (SSE), and there is an edge if they are neighbors along the amino acid sequence or one of three nearest neighbors in space. The discrete attributes are SSE types. The continuous attributes are the 3D length of the SSE. Graphs are labeled according to which EC top-level class they belong to.
DD [13] consists of graph representations of 1,178 proteins. In each graph, nodes represent amino acids, and there is an edge if they are less than six Angstroms apart. Graphs are labeled according to whether they are enzymes or not.

NCI1 [20, 29] consists of graph representations of 4,110 chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively.

FRANK [19] is a chemical molecule dataset that consists of 2,401 mutagens and 1,936 nonmutagens. Originally, nodes are associated with chemical atom symbols.

BZR, **COX2**, and **DHFR** [30] all are chemical compound datasets. Still, in each graph, nodes represent atoms, and edges represent chemical bonds. The discrete attributes correspond to atom types. The continuous attributes are 3D coordinates.