Introduction to Topology-Based Graph Classification

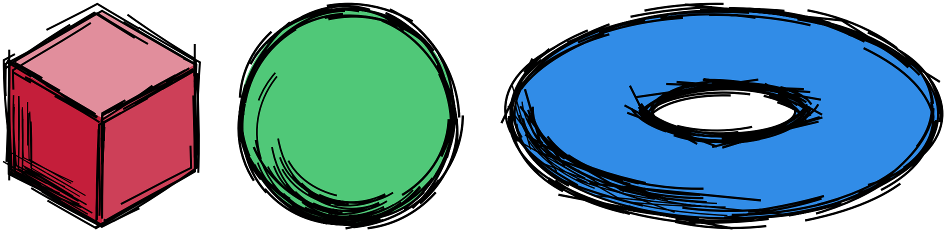
Bastian Rieck

MLCB DBSSE ETH zürich

Pseudomanifold

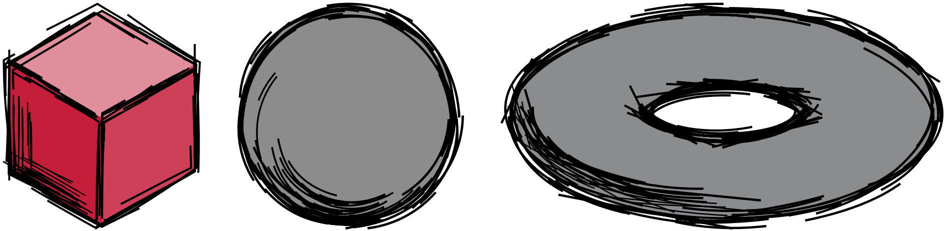
What is topology?

Studying the abstract shape of objects



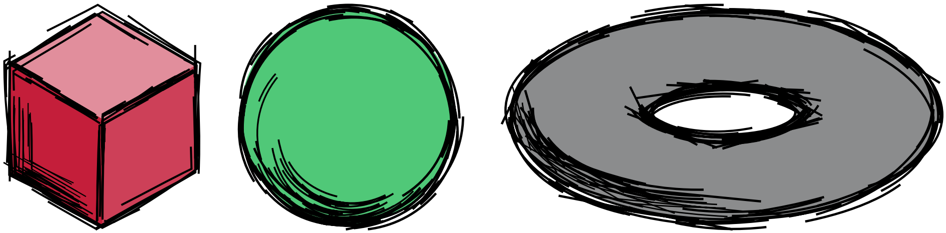
What is topology?

Studying the abstract shape of objects



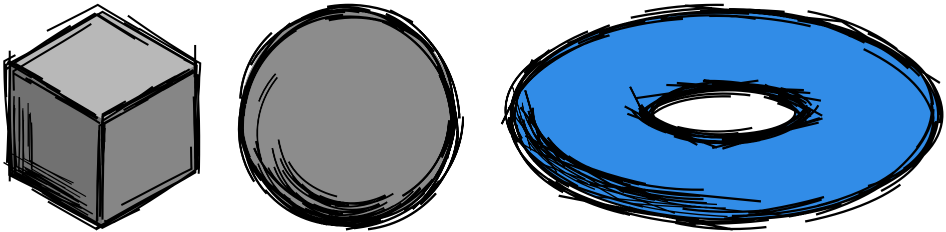
What is topology?

Studying the abstract shape of objects



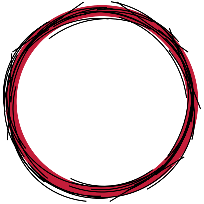
What is topology?

Studying the abstract shape of objects



Betti numbers

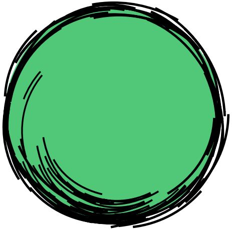
Counting d-dimensional holes



$$\beta_0 = 1, \beta_1 = 1$$

Betti numbers

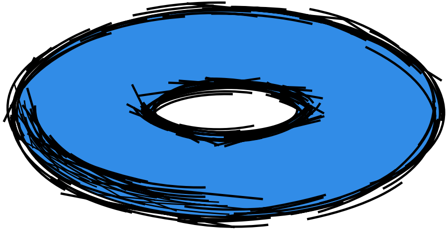
Counting d-dimensional holes



$$\beta_0 = 1, \beta_1 = 0, \beta_2 = 1$$

Betti numbers

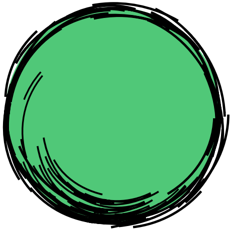
Counting d-dimensional holes



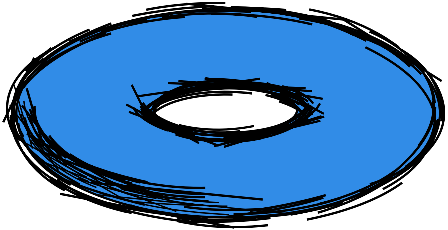
$$\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$$

Betti numbers

Counting d-dimensional holes

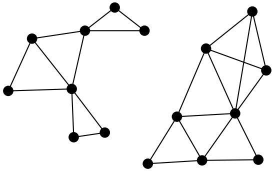


$\beta_0 = 1, \beta_1 = 0, \beta_2 = 1$



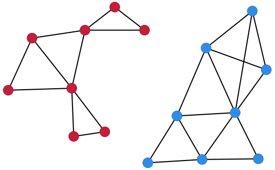
$\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$

The shape of a graph



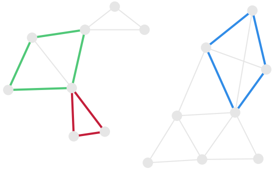
A simple graph

The shape of a graph



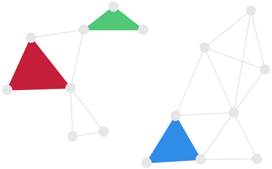
Connected components

The shape of a graph



Some cycles

The shape of a graph

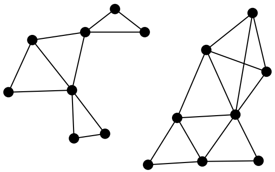


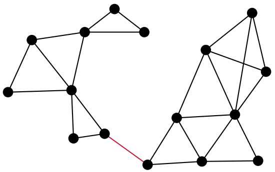
Some 2-cliques

The shape of a graph

A 3-clique

Comparing two graphs

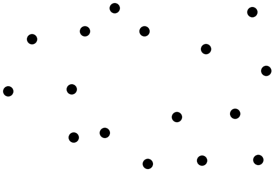




Clearly, the graphs are very *similar* (they are of course not isomorphic), but if we blindly calculate Betti numbers, we will judge them to be different.

Persistent homology

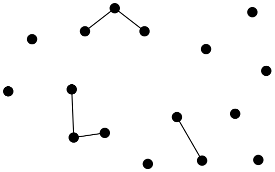
Calculating multi-scale Betti numbers



$\epsilon = 0.00$: 16 connected components

Persistent homology

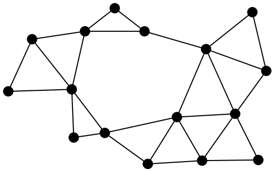
Calculating multi-scale Betti numbers



$\epsilon = 0.25$: 11 connected components

Persistent homology

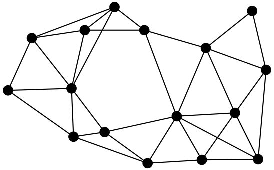
Calculating multi-scale Betti numbers



$\epsilon = 0.50$: 1 connected component, 12 cycles

Persistent homology

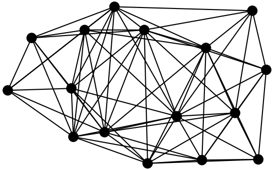
Calculating multi-scale Betti numbers



$\epsilon = 0.75$: 1 connected component, 19 cycles

Persistent homology

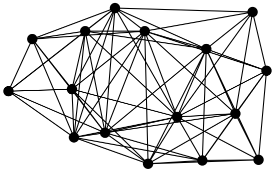
Calculating multi-scale Betti numbers

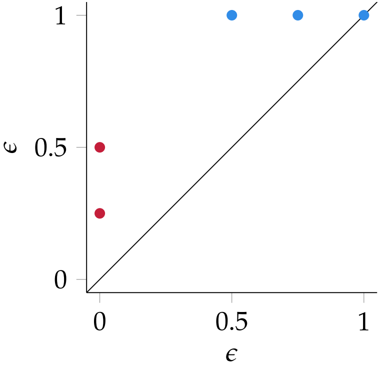


$\epsilon = 1.00$: 1 connected component, 57 cycles

Persistence diagram

Multi-scale topological descriptor

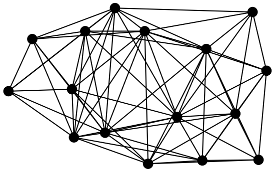


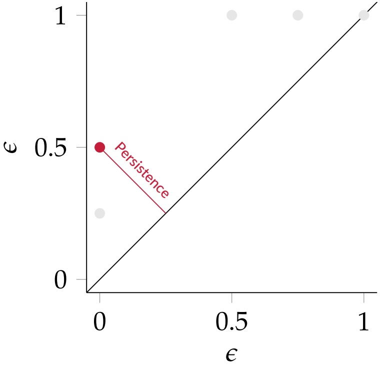


There is a one-to-one correspondence between topological features in a persistence diagram and vertices/edges of the graph.

Persistence diagram

Multi-scale topological descriptor





There is a one-to-one correspondence between topological features in a persistence diagram and vertices/edges of the graph.

Classifying weighted, unlabelled graphs

1. Calculate set of persistence diagrams for each graph
2. Calculate *kernel* or *distance* between corresponding persistence diagrams
3. Train classifier on kernel matrix

Step 2 needs to be fast, so we need vectorisation methods!

Simple feature vector representation

Persistence images


Journal of Machine Learning Research 18 (2017) 1-35 Submitted 7/16; Published 2/17

Persistence Images: A Stable Vector Representation of Persistent Homology

Henry Adams ADAMS@MATH.COLOSTATE.EDU
Tegan Emerson EMERSON@MATH.COLOSTATE.EDU
Michael Kirby KIRBY@MATH.COLOSTATE.EDU
Rachel Neville NEVILLE@MATH.COLOSTATE.EDU
Chris Peterson PETERSON@MATH.COLOSTATE.EDU
Patrick Shipman SHIPMAN@MATH.COLOSTATE.EDU
Department of Mathematics
Colorado State University
1874 Campus Delivery
Fort Collins, CO 80523-1874

Sofya Chepushtanova SOFYA.CHEPUSHTANOVA@WILKES.EDU
Department of Mathematics and Computer Science
Wilkes University
84 West South Street
Wilkes-Barre, PA 18766, USA

Eric Hanson ERIC.HANSON@TCU.EDU
Department of Mathematics
Texas Christian University
Box 298900
Fort Worth, TX 76129

Francis Motta MOTTA@MATH.DUKE.EDU
Department of Mathematics
Duke University
Durham, NC 27708, USA

Lori Ziegelmeier LZIEGEL1@MACALESTER.EDU
Department of Mathematics, Statistics, and Computer Science
Macalester College
1600 Grand Avenue
Saint Paul, MN 55105, USA

Editor: Michael Mahoney

Abstract

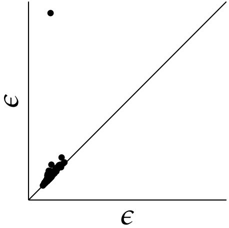
Many data sets can be viewed as a noisy sampling of an underlying space, and tools from topological data analysis can characterize this structure for the purpose of knowledge discovery. One such tool is persistent homology, which provides a multiscale description of the homological features within a data set. A useful representation of this homological information is a *persistence diagram* (PD). Efforts have been made to map PDs into spaces with additional structure valuable to machine learning tasks. We convert a PD to a finite-dimensional vector representation which we call a *persistence image* (PI), and prove the stability of this transformation with respect to small perturbations in the inputs. The

- Two-dimensional binning of a persistence diagram (grid)
- Perform Gaussian smoothing over all grid cells
- Obtain a feature vector of a fixed size

Source: *H. Adams et al., 'Persistence images: A stable vector representation of persistent homology'*

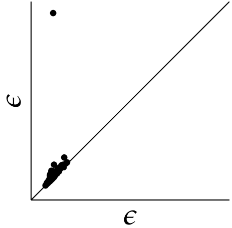
Example

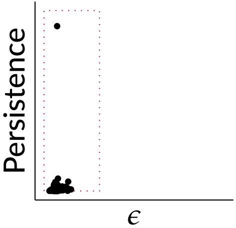
Persistence image transformation



Example

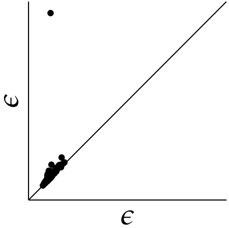
Persistence image transformation

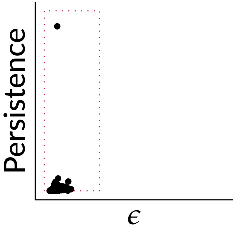




Example

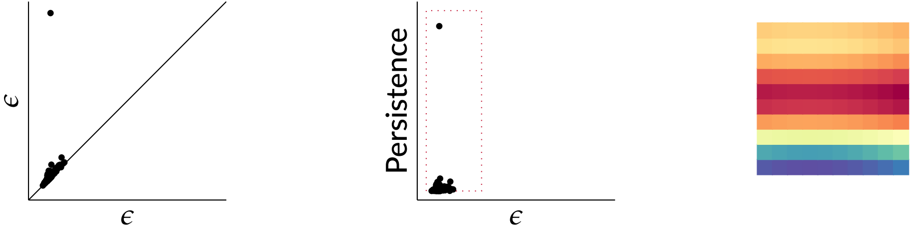
Persistence image transformation





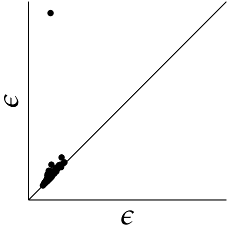
Example

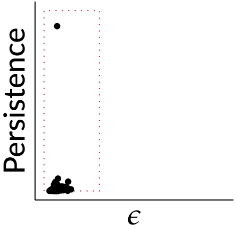
Persistence image transformation

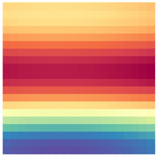


Example

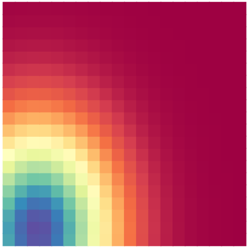
Persistence image transformation

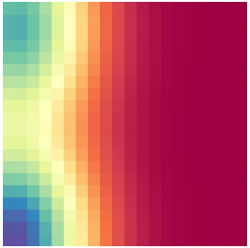


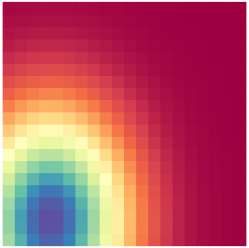




Calculating *mean* descriptors







How to use this for graph classification

Learning metrics for persistence-based summaries and applications for graph classification

Qi Zhao*
zhao.2017@osu.edu

Yusu Wang *
yusu@cse.ohio-state.edu

Abstract

Recently a new feature representation framework based on a topological tool called persistent homology (and its persistence diagram summary) has gained much momentum. A series of methods have been developed to map a persistence diagram to a vector representation so as to facilitate the downstream use of machine learning tools. In these approaches, the importance (weight) of different persistence features are usually *pre-set*. However often in practice, the choice of the weight-function should depend on the nature of the specific data at hand. It is thus highly desirable to *learn* a best weight-function (and thus metric for persistence diagrams) from labelled data. We study this problem and develop a new weighted kernel, called *WKPI*, for persistence summaries, as well as an optimization framework to learn the weight (and thus kernel). We apply the learned kernel to the challenging task of graph classification, and show that our WKPI-based classification framework obtains similar or (sometimes significantly) better results than **the best results** from a range of previous graph classification frameworks on benchmark datasets.

1 Introduction

In recent years a new data analysis methodology based on a topological tool called persistent homology has started to attract momentum. The persistent homology is one of the most important developments in the field of topological data analysis, and there have been fundamental developments both on the theoretical front (e.g, [24, 11, 14, 9, 15, 6]), and on algorithms / implementations (e.g, [46, 5, 16, 21, 30, 4]). On the high level, given a domain X with a function $f : X \to \mathbb{R}$ on it, the persistent homology summarizes "features" of X across multiple scales simultaneously in a single summary called the *persistence diagram* (see the second picture in Figure 1). A persistence diagram consists of a multiset of points in the plane, where each point $p = (b, d)$ intuitively corresponds to the birth-time (b) and death-time (d) of some (topological) features of X w.r.t. f. Hence it provides a concise representation of X, capturing *multi-scale features* of it simultaneously. Furthermore, the persistent homology framework can be applied to complex data (e.g, 3D shapes, or graphs), and different summaries could be constructed by putting different descriptor functions on input data.

Due to these reasons, a new persistence-based feature vectorization and data analysis framework (Figure 1) has become popular. Specifically, given a collection of objects, say a set of graphs modeling chemical compounds, one can first convert each shape to a persistence-based representation. The input data can now be viewed as a set of points in a persistence-based feature space. Equipping this space with appropriate distance or kernel, one can then perform downstream data analysis tasks (e.g, clustering).

The original distances for persistence diagram summaries unfortunately do not lend themselves easily to machine learning tasks. Hence in the last few years, starting from the persistence landscape [8], there have been a series of methods developed to map a persistence diagram to a vector representation to facilitate

*Computer Science and Engineering Department, The Ohio State University, Columbus, OH 43221, USA.

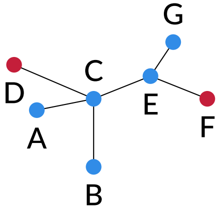
1

- Obtain persistence images from graph filtration
- *Learn* a weight function on the persistence image
- Calculate weighted distance between images
- Use this as a *kernel* in an SVM

Source: *Q. Zhao and Y. Wang, 'Learning metrics for persistence-based summaries and applications for graph classification'*

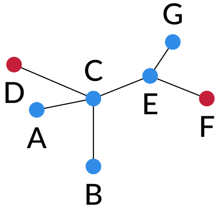
Classifying *labelled* graphs

Weisfeiler–Lehman iteration & subtree feature vector



Classifying *labelled* graphs

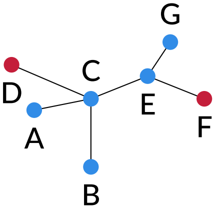
Weisfeiler–Lehman iteration & subtree feature vector



Node	Own label	Adjacent labels
A	🔵	🔵
B	🔵	🔵
C	🔵	🔵 🔵 🔵 🔴
D	🔴	🔵
E	🔵	🔵 🔵 🔴
F	🔴	🔵
G	🔵	🔵

Classifying *labelled* graphs

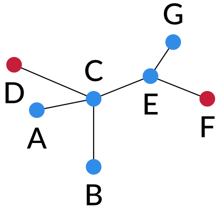
Weisfeiler–Lehman iteration & subtree feature vector



Node	Own label	Adjacent labels	Hashed label
A	●	●	●
B	●	●	●
C	●	● ● ● ●	●
D	●	●	●
E	●	● ● ●	●
F	●	●	●
G	●	●	●

Classifying *labelled* graphs

Weisfeiler–Lehman iteration & subtree feature vector



Label				
Count	3	1	2	1

$$\Phi(\mathcal{G}) := (3, 1, 2, 1)$$

Compare $\mathcal{G}$ and $\mathcal{G}'$ by evaluating a kernel between $\Phi(\mathcal{G})$ and $\Phi(\mathcal{G}')$ (linear, RBF, ...).

Improving the Weisfeiler–Lehman procedure

A Persistent Weisfeiler–Lehman Procedure for Graph Classification

Bastian Rieck [*1] Christian Bock [*1] Karsten Borgwardt [1]

Abstract

The Weisfeiler–Lehman graph kernel exhibits competitive performance in many graph classification tasks. However, its subtree features are not able to capture connected components and cycles, topological features known for characterising graphs. To extract such features, we leverage propagated node label information and transform unweighted graphs into metric ones. This permits us to augment the subtree features with topological information obtained using *persistent homology*, a concept from topological data analysis. Our method, which we formalise as a generalisation of Weisfeiler–Lehman subtree features, exhibits favourable classification accuracy and its improvements in predictive performance are mainly driven by including cycle information.

1. Introduction

Graph-structured data sets are ubiquitous in a variety of different application domains, each of them posing a separate challenge while also requiring different tasks to be solved. A common task involves *graph classification*, for which a variety of methods exists. These methods comprise convolutional neural networks (Duvenaud et al. 2015), recurrent neural networks (Lei et al. 2017), or Hilbert space methods (Vishwanathan et al. 2010), the latter also being referred to as *graph kernels*. While several approaches for defining graph kernels exist, the most common one uses the $\mathcal{R}$-convolution framework (Haussler 1999), which makes it possible to define the similarity between two graphs as a function of the similarity of their substructures.

Substructures that have been used for graph classification range from graphlets (Shervashidze et al. 2009), i.e. small non-isomorphic graphs of fixed size, over shortest paths (Borgwardt & Kriegel 2005), to random walks (Gärtner et al. 2003, Kashima et al. 2003, Sugiyama & Borgwardt 2015). One of the most powerful substructures is the set of *subtree patterns* (Ramon & Gärtner 2003), i.e. patterns based on rooted subgraphs of a graph. Their computational complexity made their applicability somewhat limited, until the Weisfeiler–Lehman (WL) graph kernel framework (Shervashidze & Borgwardt 2009, Shervashidze et al. 2011) was developed. Properly trained, it still constitutes the state-of-the-art method for many graph classification tasks. The framework is based on the idea of iteratively propagating (node) label information through a graph, leading to a feature vector representation that can be used to assess the dissimilarity of two graphs.

One of the disadvantages of this framework is that its relabelling step, i.e. the step in which subtree patterns are being *compressed*, is somewhat “brittle”: labels are only compared with a Dirac kernel, making their dissimilarity a coarse function. Moreover, the subtree feature vector only contains counts of compressed labels and can neither account for their relevance with respect to the topology of the graph nor capture connected components and cycles, both of which are important and interpretable features for characterising graphs (Rieck et al. 2018, Sizemore et al. 2017). We thus propose an enhancement of the original WL stabilisation procedure that uses recent advances in topological data analysis (Munch 2017) to alleviate these issues. Our contributions are as follows:

- We measure the relevance of topological features (connected components and cycles) in graphs and use them to define a novel set of WL subtree features, which we show to be a generalised version of the original ones.
- We develop a topology-based kernel that uses an iterative variant of the WL stabilisation procedure to classify non-attributed graphs.
- We demonstrate that our proposed features perform favourably on a range of graph classification benchmark data sets. In particular, we empirically show that the inclusion of *cycle* information yields classification accuracy improvements over state-of-the-art methods.

[*]Equal contribution [1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland. Correspondence to: Bastian Rieck <bastian.rieck@bsse.ethz.ch>, Karsten Borgwardt <karsten.borgwardt@bsse.ethz.ch>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

- The Weisfeiler–Lehman iteration vectorises labels in graphs
- Persistent homology assess the relevance of topological features
- We can *combine* both of them!
- This requires a distance between multisets

Source: *B. Rieck et al., ‘A persistent Weisfeiler–Lehman procedure for graph classification’*

A distance between label multisets

Let $A = \{l_1^{a_1}, l_2^{a_2}, \ldots\}$ and $B = \{l_1^{b_1}, l_2^{b_2}, \ldots\}$ be two multisets that are defined over the same label alphabet $\Sigma = \{l_1, l_2, \ldots\}$.

Transform the sets into count vectors, i.e. $\mathbf{x} := [a_1, a_2, \ldots]$ and $\mathbf{y} := [b_1, b_2, \ldots]$.

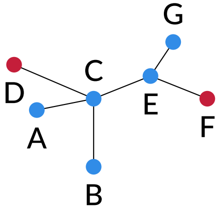
Calculate their *multiset distance* as

$$\operatorname{dist}(\mathbf{x}, \mathbf{y}) := \left(\sum_i |a_i - b_i|^p\right)^{\frac{1}{p}},$$

i.e. the p Minkowski distance, for $p \in \mathbb{R}$. Since nodes and their multisets are in one-to-one correspondence, we now have a metric on the graph!

Multiset distance

Example for $p = 1$



$$
\begin{aligned}
\mathrm{dist}(C, E) &= \mathrm{dist}\Big(\{{\color{blue}\bullet}^3, {\color{red}\bullet}^1\}, \{{\color{blue}\bullet}^2, {\color{red}\bullet}^1\}\Big)\\
&= \mathrm{dist}([3,1],[2,1])\\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{dist}(C, A) &= \mathrm{dist}\Big(\{{\color{blue}\bullet}^3, {\color{red}\bullet}^1\}, \{{\color{blue}\bullet}^1\}\Big)\\
&= \mathrm{dist}([3,1],[1,0])\\
&= 3
\end{aligned}
$$

Vertex distance

Use vertex label from *previous* Weisfeiler–Lehman iteration, i.e. $\mathrm{l}_{v_i}^{(h-1)}$, as well as $\mathrm{l}_{v_i}^{(h)}$, the one from the *current* iteration:

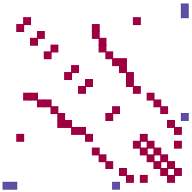
$$\mathrm{dist}(v_i, v_j) := \left[\mathrm{l}_{v_i}^{(h-1)} \neq \mathrm{l}_{v_j}^{(h-1)}\right] + \mathrm{dist}\left(\mathrm{l}_{v_i}^{(h)}, \mathrm{l}_{v_j}^{(h)}\right) + \tau$$

$\tau \in \mathbb{R}_{>0}$ is required to make this into a proper metric. This turns *any* labelled graph into a weighted graph whose persistent homology we can calculate!

Vertex distance

Example

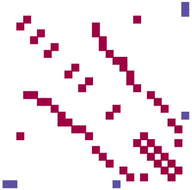
$h = 0$



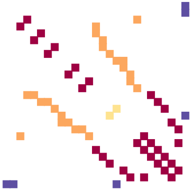
Vertex distance

Example

$h = 0$



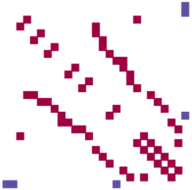
$h = 1$



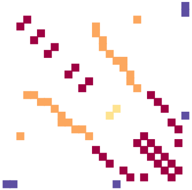
Vertex distance

Example

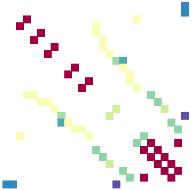
$h = 0$



$h = 1$

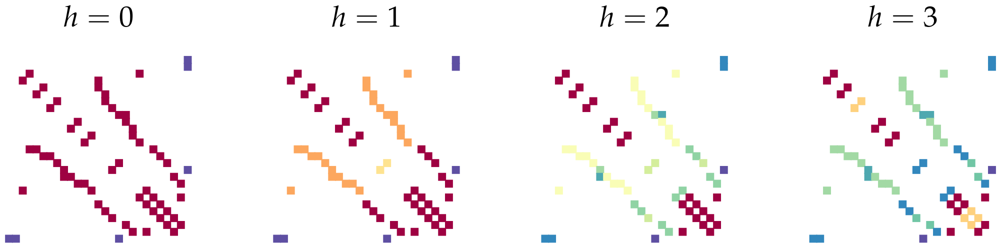


$h = 2$



Vertex distance

Example



Persistence-based Weisfeiler–Lehman feature vectors

Connected components

$$\Phi^{(h)}_{\textsf{P-WL}} := \left[\mathfrak{p}^{(h)}(l_0), \mathfrak{p}^{(h)}(l_1), \dots\right]$$
$$\mathfrak{p}^{(h)}(l_i) := \sum_{\mathrm{l}(v)=l_i} \mathrm{pers}(v)^p,$$

Cycles

$$\Phi^{(h)}_{\textsf{P-WL-C}} := \left[\mathfrak{z}^{(h)}(l_0), \mathfrak{z}^{(h)}(l_1), \dots\right]$$
$$\mathfrak{z}^{(h)}(l_i) := \sum_{l_i \in \mathrm{l}(u,v)} \mathrm{pers}(u,v)^p,$$

Persistence-based Weisfeiler–Lehman feature vectors

Connected components

$$\Phi^{(h)}_{\text{P-WL}} := \left[\mathfrak{p}^{(h)}(l_0), \mathfrak{p}^{(h)}(l_1), \dots\right]$$
$$\mathfrak{p}^{(h)}(l_i) := \sum_{\mathrm{l}(v)=l_i} \operatorname{pers}(v)^p,$$

Cycles

$$\Phi^{(h)}_{\text{P-WL-C}} := \left[\mathfrak{z}^{(h)}(l_0), \mathfrak{z}^{(h)}(l_1), \dots\right]$$
$$\mathfrak{z}^{(h)}(l_i) := \sum_{l_i \in \mathrm{l}(u,v)} \operatorname{pers}(u,v)^p,$$

Bonus

We can re-define the vertex distance to obtain the original Weisfeiler–Lehman subtree features (plus information about cycles):

$$\operatorname{dist}(v_i, v_j) := \begin{cases} 1 & \text{if } v_i \neq v_j \\ 0 & \text{otherwise} \end{cases}$$

Results

	D & D	MUTAG	NCI1	NCI109	PROTEINS	PTC-MR	PTC-FR	PTC-MM	PTC-FM
V-Hist	78.32 ± 0.35	85.96 ± 0.27	64.40 ± 0.07	63.25 ± 0.12	72.33 ± 0.32	58.31 ± 0.27	68.13 ± 0.23	66.96 ± 0.51	57.91 ± 0.83
E-Hist	72.90 ± 0.48	85.69 ± 0.46	63.66 ± 0.11	63.27 ± 0.07	72.14 ± 0.39	55.82 ± 0.00	65.53 ± 0.00	61.61 ± 0.00	59.03 ± 0.00
RetGK*	81.60 ± 0.30	90.30 ± 1.10	84.50 ± 0.20		75.80 ± 0.60	62.15 ± 1.60	67.80 ± 1.10	67.90 ± 1.40	63.90 ± 1.30
WL	79.45 ± 0.38	87.26 ± 1.42	85.58 ± 0.15	84.85 ± 0.19	76.11 ± 0.64	63.12 ± 1.44	67.64 ± 0.74	67.28 ± 0.97	64.80 ± 0.85
Deep-WL*		82.94 ± 2.68	80.31 ± 0.46	80.32 ± 0.33	75.68 ± 0.54	60.08 ± 2.55			
P-WL	79.34 ± 0.46	86.10 ± 1.37	85.34 ± 0.14	84.78 ± 0.15	75.31 ± 0.73	63.07 ± 1.68	67.30 ± 1.50	68.40 ± 1.17	64.47 ± 1.84
P-WL-C	78.66 ± 0.32	90.51 ± 1.34	85.46 ± 0.16	84.96 ± 0.34	75.27 ± 0.38	64.02 ± 0.82	67.15 ± 1.09	68.57 ± 1.76	65.78 ± 1.22
P-WL-UC	78.50 ± 0.41	85.17 ± 0.29	85.62 ± 0.27	85.11 ± 0.30	75.86 ± 0.78	63.46 ± 1.58	67.02 ± 1.29	68.01 ± 1.04	65.44 ± 1.18

Try it out

- Favourable performance
- Can make use of *cycles*
- Code is open-source

A neural network approach

Deep Learning with Topological Signatures

Christoph Hofer
Department of Computer Science
University of Salzburg, Austria
chofer@cosy.sbg.ac.at

Roland Kwitt
Department of Computer Science
University of Salzburg, Austria
Roland.Kwitt@sbg.ac.at

Marc Niethammer
UNC Chapel Hill, NC, USA
mn@cs.unc.edu

Andreas Uhl
Department of Computer Science
University of Salzburg, Austria
uhl@cosy.sbg.ac.at

Abstract

Inferring topological and geometrical information from data can offer an alternative perspective on machine learning problems. Methods from topological data analysis, e.g., persistent homology, enable us to obtain such information, typically in the form of summary representations of topological features. However, such topological signatures often come with an unusual structure (e.g., multisets of intervals) that is highly impractical for most machine learning techniques. While many strategies have been proposed to map these topological signatures into machine learning compatible representations, they suffer from being agnostic to the target learning task. In contrast, we propose a technique that enables us to input topological signatures to deep neural networks and *learn* a task-optimal representation during training. Our approach is realized as a novel input layer with favorable theoretical properties. Classification experiments on 2D object shapes and social network graphs demonstrate the versatility of the approach and, in case of the latter, we even outperform the state-of-the-art by a large margin.

1 Introduction

Methods from algebraic topology have only recently emerged in the machine learning community, most prominently under the term *topological data analysis (TDA)* [7]. Since TDA enables us to infer relevant topological and geometrical information from data, it can offer a novel and potentially beneficial perspective on various machine learning problems. Two compelling benefits of TDA are (1) its versatility, i.e., we are not restricted to any particular kind of data (such as images, sensor measurements, time-series, graphs, etc.) and (2) its robustness to noise. Several works have demonstrated that TDA can be beneficial in a diverse set of problems, such as studying the manifold of natural image patches [8], analyzing activity patterns of the visual cortex [28], classification of 3D surface meshes [27, 22], clustering [11], or recognition of 2D object shapes [29].

Currently, the most widely-used tool from TDA is *persistent homology* [15, 14]. Essentially[1], persistent homology allows us to track topological changes as we analyze data at multiple "scales". As the scale changes, topological features (such as connected components, holes, etc.) appear and disappear. Persistent homology associates a *lifespan* to these features in the form of a *birth* and a *death* time. The collection of (birth, death) tuples forms a multiset that can be visualized as a persistence diagram or a barcode, also referred to as a *topological signature* of the data. However, leveraging these signatures for learning purposes poses considerable challenges, mostly due to their

[1] We will make these concepts more concrete in Sec. 2.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

- Obtain persistence diagrams from graph filtration
- Define layer to *project* persistence diagrams to 1D
- Learn parameters for multiple projections
- Stack projected diagrams and use as features

Source: *C. Hofer et al., 'Deep learning with topological signatures'*

Open questions

Learning appropriate filtrations

arXiv:1905.10996v1 [cs.LG] 27 May 2019

Graph Filtration Learning

Christoph Hofer
Department of Computer Science
University of Salzburg, Austria
chofer@cosy.sbg.ac.at

Roland Kwitt
Department of Computer Science
University of Salzburg, Austria
Roland.Kwitt@sbg.ac.at

Marc Niethammer
UNC Chapel Hill, NC, USA
mn@cs.unc.edu

Abstract

We propose an approach to learning with graph-structured data in the problem domain of graph classification. In particular, we present a novel type of *readout* operation to aggregate node features into a graph-level representation. To this end, we leverage persistent homology computed via a real-valued, learnable, filter function. We establish the theoretical foundation for differentiating through the persistent homology computation. Empirically, we show that this type of readout operation compares favorably to previous techniques, especially when the graph connectivity structure is informative for the learning problem.

1 Introduction

We consider the problem of learning a function from the space of (finite) undirected graphs, $\mathbb{G}$, to a (discrete/continuous) target domain $\mathbb{Y}$. Additionally, graphs might have discrete, or continuous attributes attached to each node. Prominent examples for this class of *learning* problem appear in the context of classifying molecule structures, chemical compounds or social networks.

A substantial amount of research has been devoted to developing techniques for supervised learning with graph-structured data, ranging from kernel-based methods [23, 22, 9, 15], to more recent approaches based on graph neural networks (GNN) [20, 11, 31, 18, 26, 28]. Most of the latter works use an iterative message passing scheme [10] to learn node representations, followed by a graph-level pooling operation that aggregates node-level features. This aggregation step is typically referred to as a *readout* operation. While research has mostly focused on variants of the message passing function, the readout step may have a significant impact, as it aims to capture properties of the entire graph. Importantly, both simple and more refined readout operations, such as summation, differentiable pooling [28], or sort pooling [30], are inherently coupled to the amount of information carried over via multiple rounds of message passing. Hence, architectural GNN choices are typically guided by dataset characteristics, e.g., requiring to tune the number of message passing rounds to the expected size of graphs.

Contribution. We propose a *homological* readout operation that captures the full global structure of a graph while relying only on node representations that are learned (end-to-end), from immediate neighbors. This not only alleviates the aforementioned design challenge, but potentially also offers additional discriminative information.

The *main idea* is to consider a graph, G, as a simplicial complex, K, i.e., the main structure in simplicial homology. While this view would allow us to study, e.g., the ranks of homology groups, revealing the number of connected components or loops, the information is quite coarse. Alternatively, we can *construct* K, one part at a time, and keep track of the induced homological changes. To do

Preprint. Under review.

- Learn initial node representation on a graph
- Calculate corresponding persistence diagram
- Apply differentiable coordinate function
- Adjust learned representation and repeat

Source: *C. Hofer et al., 'Graph filtration learning'*

Summary

Three ways for TDA-based graph classification

1. Filtration plus feature vectors
2. Filtration plus ‘hybrid’ feature vectors
3. Filtration plus differentiable feature vectors

Join our Slack community ‘TDA in ML’ to discuss papers, ideas, and collaborations!