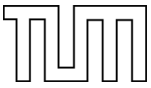


Technische Universität München

Department of Mathematics

Master Thesis

Topological Kernels for Gaussian Processes

Jonathan Clancy

Supervisor: Dr. Bastian Rieck

Advisor: Dr. Bastian Rieck

Submission Date: 13 June 2024

I assure the single handed composition of this master's thesis only supported by declared resources.

Garching, 13.06.24 J. Clancy

Abstract

Persistence diagrams are one of the most important objects occurring when employing techniques from topological data analysis (TDA). They occur when analyzing plain point clouds and investigating their global structure across various dimensions. They provide a concise summary of the topological information contained in that point cloud, assuming that it is a good subsample from some manifold. However, processing these diagrams on a large scale has proven to pose some challenges. For this, researchers proposed kernel methods in order to use kernel based models like support vector machines (SVM). Four kernels have been proposed up to now, which all used SVMs for performing classification tasks. In this thesis, we employ Gaussian processes (GP) for investigating these kernels on classification tasks. Although primarily a regression model, GPs can be used for classification tasks as well. Firstly, we test this model on basic topological objects and provide recommendations on how to best train all kernels via e.g. likelihood maximization. As the GP is a probabilistic model, we analyze how meaningful the obtained probabilities of the GP are when performing classification. We find that all kernels return different confidences, but the returned probability can indeed be interpreted as confidences by performing probability calibration. We replicate various classification tests that used SVMs and show that for all four kernels, performance varies slightly when using either a GP or SVM. When selecting the best performing kernel, however, we mostly have the same performance, showing that GPs provide a valuable addition to SVMs. We find that two kernels prove not only sightly superior, but also easier to train and emerge with a clear recommendation of kernels for future experiments.

Contents

1 Introduction

In recent years, Topological data analysis (TDA) has established itself as a fundamentally new perspective on many data analysis tasks and as powerful addition to many machine learning models and problems. It originates from topology, a theory that investigates properties of geometric objects which are preserved under continuous transformations. Informally, topology is also called "rubber-sheet-geometry", highlighting its global, high-level perspective, neglecting information like curvature and size. More specifically, it originates from algebraic topology, a branch in pure mathematics, aiming at finding invariants of topological spaces in order to categorize and distinguish those spaces up to continuous transformations. This can be done by investigating how those spaces differ with respect to their connectivity, loops, holes and higher-dimensional analogous of holes via homology theory. Methods from e.g. geometry have enabled to use ideas from a particular intuitive and geometric homology theory, simplicial homology, to apply this branch of pure mathematics to many problems occurring in practical applications. Its fundamental idea of detecting global features, rather than many local features, is a fundamental difference to other existing models aiming at solving geometric problems. Since the introduction of fast algorithms for applying these ideas like Ripser by Ulrich Bauer, see [3], or the development of the GUDHI library [25], TDA has seen a wide range of applications. For instance, TDA has been used to simplify complex neural data [33], analyze periodic patterns in gene expression [24], and ensure coverage in sensor networks [11]. It has also been used to analyze the structure of human brain networks [34], study the topological properties of materials [17], and investigate the spread of diseases through social networks [36].

An object whose central importance cannot be highlighted enough in the setting of topological data analysis, is a so called persistence diagram. These diagrams occur when computing the persistent homology of point clouds. Persistent homology can compute and detect the importance and persistence of many topological features which are present in a point cloud, provided this point cloud is a good subsample of some underlying manifold. Persistent diagrams are then defined as multi-sets of points, representing this topological information. Whilst for point clouds representing basic topological objects, an intuitive interpretation of these objects is easily possible for a human, processing these diagrams efficiently in a supervised or unsupervised learning setting for large scale point clouds and data sets is still an area of active research. Also, processing these objects is not straightforward as persistent diagrams are defined as sets of varying size, without any inherent ordering. Recent advances include using vectorization techniques such as persistence landscapes [5] and persistence images [2], which transform persistence diagrams into feature vectors suitable for machine learning tasks. In the context of neural networks, persistent diagrams have been incorporated into convolutional neural networks (CNNs) for image classification [16], graph neural networks (GNNs) for graph classification [40] or recurrent neural networks (RNNs) for time-series analysis [38].

In 2014, Reininghaus et al. were the first to construct a kernel for persistence diagrams, by introducing the Persistence Scale Space Kernel (PSSK), see [30]. They use ideas from scale space theory in order to define the kernel via an L_2-valued feature map. Kernels can be very powerful as one can use the added flexibility of a higher dimensional space

without explicitly accessing it and thus suffering from the higher computational costs that those higher dimensional spaces exhibit. These kernels enable the usage of many kernel based machine learning models like support vector machines, kernel PCA and Gaussian processes. Three additional kernels for persistence diagrams were constructed following [30], all of which are based on different mathematical and geometric ideas. Carriere et al. used the sliced wasserstein distance, see [8], to construct the so called Sliced Wasserstein Kernel (SWK). Kusano et al. used kernel embeddings to construct the Persistence Weighted Gaussian kernel (PWGK), [19], and Le and Yamada used ideas from statistical geometry, see [20], in order to construct the Persistent Fisher Kernel (PFK).

Until now, there has not been an investigation of how well Gaussian processes can be used in combination with topological kernels. Albeit a regression model from non-parametric statistics, it can also be used for classification tasks by using ideas similar to applying linear regression to classification problems via logistic regression. A particular motivation for using GPs is that they are a fully probabilistic model and thus are able to return confidence intervals for regression tasks and probabilities for classification tasks instead of black box decisions. They also enable convenient model training via likelihood maximization as they are a fully probabilistic model. One of the most important (if not the most important) challenge when using GPs for machine learning tasks is the selection and training of an appropriate kernel. Investigating how well these topological kernels work in the context of GPs for solving machine learning problems is the main objective of this thesis.

In particular, we aim at analyzing how well GPs perform in general on classification tasks in comparison to SVMs and what advantages and disadvantages we can identify for employing GPs. We also aim at analyzing in detail how to best train all kernels by testing various optimization techniques. Since training four kernels for each experiment is very costly, we want to give explicit recommendations about which kernel kernels are superior to others. As one of the key motivations for using GPs is the availability of a probability vector when making predictions, we also want to thoroughly analyze how to interpret this probability vector.

This thesis is structured as follows: we first present the necessary mathematical background, relevant literature and our methodology. Then we present our experiments and findings in two main parts. The first one focuses on developing and testing ideas, whereas the second part aims at a direct comparison of the performance of GPs and SVMs on real world data. More specifically, we start by conducting many experiments on simple, synthetic topological data, in a controlled and computationally manageable setting. We will provide a first baseline and make a series of thorough investigations on how to train the kernels. We will not only investigate likelihood maximization, but also use ideas from other areas of statistic like Bayesian optimization and kernel density estimation. We investigate the probability vector several times by checking how it changes under noisy data, increased number of training points and unseen objects. We employ Markov Chain Monte Carlo simulation in order to investigate the PWGK in more detail. In the second part, we then continue to replicate more elaborate experiments conducted in [20], [19] and [28], which all used a SVM for performing classification tasks. The main goal of this part

is to directly asses the GPs performance in comparison with a SVM and identify strengths and weaknesses of this model. We will encounter some practical difficulties with the PFK and will come up with an intuitive explanation for this behaviour. After conducting these experiments, we also address the challenge of interpreting the returned probabilities of the GP in a meaningful way by using probability calibration. Finally, we will investigate a particularly large data set, the Modelnet10 data set, in order to analyze how well GPs work an large data sets, also in comparison to a SVM.

2 Background

This chapter focuses on introducing the mathematical principles required for employing Gaussian processes with topological kernels. We aim at giving both the main mathematical formulation of the models, as well as some intuition and basic considerations necessary for using these methods. The content is three folded: We introduce Gaussian processes, as they are the main machine learning model we are using in this thesis. We will present the main theory behind topological data analysis (TDA), in particular persistent diagrams, which constitute the data we are going to train the GP on. Finally, we will provide some notions revolving around Reproducing kernel Hilbert spaces and support vector machines.

2.1 Gaussian Processes for Machine Learning

Albeit not being as famous and widely used as artificial neural-networks nowadays, Gaussian processes still provide an interesting and flexible framework for supervised machine learning, which we are going to elaborate in detail in this section. We will explore the probabilistic and non-parametric formulation of this model. In particular we will start by considering how to do regression tasks with this model. We will observe that all formulations take place in the realm of normal distributions, as the core assumption about the data is being jointly normally distributed. The key idea of gaining information about test points is by considering conditional distributions, which incidentally can be calculated very well in the realm of multivariate Gaussian distributions. Conditioning will yield a normal distribution for each test point again. Albeit these reformulations work out very nicely analytically, they involve matrix inversion. This is why GPs in its pure and original form do not scale well to larger data sets. The key for defining a multivariate normal distribution is to define a covariance matrix via a kernel function which defines how different observations depend on each other. Selecting and tuning these kernel functions lies at the core of performing supervised learning tasks with a Gaussian process. This great flexibility is both a strength and weakness for the model, as it is a source for errors on the one hand but offers great flexibility to adapt to different data sets on the other hand. In the final model formulation, we will observe a key difference of a GP compared to a SVM, namely that it uses all training points for making a new prediction instead of support vectors. We will then explore how to turn this regressor into a classifier, as classification tasks are what we are mainly interested in this thesis. This is done in exactly the same way as for instance linear regression is turned into logistic regression. While for logistic regression the obtained likelihood is not analytically tractable, it can be optimized easily via Iteratively reweighted least squares. As we will see, it is not as easy to work with the resulting model when using a GP as a regressor, as the latent space is not a simple linear model anymore but a Gaussian process. We then present the main approximation scheme used in this thesis to solve this problem, the Laplace approximation which, somewhat ironically, approximates the intractable posterior distribution with a normal distribution again.

2.1.1 Gaussian Process Regression

We closely follow the standard work of Rasmussen [29] in order to state a more formal model description. A *Gaussian process* (GP) is defined as a collection of random variables where any finite number of those random variables have a joint Gaussian distribution. Formally, a Gaussian process $f(\mathbf{x})$ can be written as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$$

A GP then is fully specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. Often, stochastic process (or Gaussian processes $f(\cdot)$) are indexed over time, but we follow a more general formulation allowing arbitrary index sets $\mathcal{X}$ as the set of possible inputs, which could be more general than $\mathbb{R}$, e.g. $\mathbb{R}^D$ or even arbitrary sets. The covariance function (also known as the kernel function) defines the covariance between pairs of random variables (i.e., function evaluations at different points). We already can see that the freedom to choose an arbitrary kernel function offers great flexibility. This is both an advantage and disadvantage and much effort for training Gaussian Processes revolves around suitable selection and fine tuning of these kernel function and their corresponding hyperparameters.

In a supervised learning setting, we first consider a noise free dataset $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$, where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ are the input points and $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$ are the corresponding outputs. To make predictions at new input points $\mathbf{X}^*$, we consider the joint distribution of the observed outputs $\mathbf{y}$ and the function values at the new test points $\mathbf{f}^* = f(\mathbf{X}^*)$. We use the notation that $f_i \triangleq f(\mathbf{x}_i)$, the random variable corresponding to the train case $(\mathbf{x}_i, y_i)$ and $\mathbf{f}_i^* \triangleq f(\mathbf{x}_i^*)$ for test cases.
The joint distribution is given by:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}^* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^\top & \mathbf{K}_{**} \end{pmatrix}\right)$$

where $\mathbf{K}$, $\mathbf{K}_*$, $\mathbf{K}_{**}$ are the covariance matrices defined as:

$$\mathbf{K} = k(\mathbf{X}, \mathbf{X}), \quad \mathbf{K}_* = k(\mathbf{X}, \mathbf{X}^*), \quad \mathbf{K}_{**} = k(\mathbf{X}^*, \mathbf{X}^*)$$

The predictive distribution for $\mathbf{f}^*$, given the observed data $\mathbf{y}$ (we could say $\mathbf{f}$ as well), is obtained by conditioning the joint Gaussian distribution. This results in

$$\mathbf{f}^* | \mathbf{X}, \mathbf{y}, \mathbf{X}^* \sim \mathcal{N}(\bar{\mathbf{f}}^*, \mathrm{cov}(\mathbf{f}^*))$$

where

$$\bar{\mathbf{f}}^* = \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{y}$$

$$\mathrm{cov}(\mathbf{f}^*) = \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{K}_*$$

As were are only dealing with Gaussian distributions, the calculations work out very nicely and we can work with an analytical solution for the predictive distribution. This distributions again Gaussian, given some test inputs. As we will later see, the predictive distribution is not Gaussian in the case of Gaussian process classification. We can also observe a first weakness of the GP formulation; the need for inverting the covariance matrix of the training points. This corresponds to a running time which scales cubically in the number of training points. Matrix inversion is usually done via Cholesky-factorization for increased speed and numerical stability.

In practice, observations are often corrupted by noise. Assuming Gaussian noise with variance σ_n^2, the observed outputs are given by:

$$\mathbf{y} = f(\mathbf{X}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2 \mathbf{I})$$

The covariance matrix of the noisy observations becomes:

$$\mathbf{K}_y = \mathbf{K} + \sigma_n^2 \mathbf{I}$$

The predictive distribution for the noisy case is then:

$$\mathbf{f}^* | \mathbf{X}, \mathbf{y}, \mathbf{X}^* \sim \mathcal{N}(\bar{\mathbf{f}}^*, \mathrm{cov}(\mathbf{f}^*))$$

where:

$$\bar{\mathbf{f}}^* = \mathbf{K}_*^\top \mathbf{K}_y^{-1} \mathbf{y}$$

$$\mathrm{cov}(\mathbf{f}^*) = \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}_y^{-1} \mathbf{K}_*$$

The noise parameter is a fundamental hyperparameter to consider. It models the uncertainty we have about our predictions and directly affects the uncertainties of predictions we are going to make. Also note that adding $\sigma_n^2 \mathbf{I}$ to $\mathbf{K}$ can serve as regularization in case $\mathbf{K}$ is not invertible and is also used to avoid overfitting.

For a single test point $\mathbf{x}^*$, we can give a more concise predictive distribution. Let $\mathbf{k}_* = k(\mathbf{X}, \mathbf{x}^*)$ be the covariance vector between the training inputs $\mathbf{X}$ and the test point $\mathbf{x}^*$, and let $k_{**} = k(\mathbf{x}^*, \mathbf{x}^*)$ be the variance at $\mathbf{x}^*$. The predictive mean and variance are given by:

$$\bar{f}^* = \mathbf{k}_*^\top \mathbf{K}_y^{-1} \mathbf{y}$$

$$\mathrm{var}(f^*) = k_{**} - \mathbf{k}_*^\top \mathbf{K}_y^{-1} \mathbf{k}_*$$

Thus, the predictive distribution for $f(\mathbf{x}^*)$ is:

$$f(\mathbf{x}^*) | \mathbf{X}, \mathbf{y}, \mathbf{x}^* \sim \mathcal{N}(\bar{f}^*, \mathrm{var}(f^*))$$

We can observe that the mean prediction $\bar{f}^*$ is a linear combination of all observations $\mathbf{y}$. This is a more memory-expensive model formulation as for a SVM, where only the supporting vectors need to be stored for future usage of the model.
The log marginal likelihood of the data is an important quantity for hyperparameter estimation. It is given by:

$$\log p(\mathbf{y} | \mathbf{X}, \theta) = -\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log \det \mathbf{K}_y - \frac{n}{2} \log 2\pi$$

where θ represents the hyperparameters of the covariance function. The term marginal steems from the fact that we are marginalizing over all possible function values $\mathbf{f}$.

We present one of the most commonly used kernels, the Squared Exponential (SE) kernel, also known as the Radial Basis Function (RBF) kernel. Other commonly used kernels are the Matern kernel and the Rational Quadratic kernel. The SE kernel is defined, in its most general form (incorporating noise in the input data), as

$$k_{\mathrm{SE}}(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) + \sigma_n^2 \delta_{xx'},$$

where σ_f^2 is the variance parameter, ℓ is the lengthscale parameter, σ_n^2 is the uncertainty in the prediction and $\delta_{xx'}$ evaluates to one if $x = x'$ and to zero otherwise.
The *lengthscale* parameter ℓ controls the smoothness of the function. It determines how quickly the correlation between the values of the function decreases with distance. A small lengthscale implies that the function values change rapidly, allowing the GP to model more complex and wiggly functions. Conversely, a large lengthscale results in a smoother function, where changes are more gradual. The lengthscale is crucial in defining the behavior of the GP and how it generalizes from the training data.
The *variance* parameter σ_f^2 represents the variability of the function values. It controls the vertical variation of the function. A larger variance means that the function values can vary significantly, while a smaller variance indicates that the function values are more tightly clustered around the mean. The variance parameter impacts the confidence intervals of the predictions, with larger values leading to wider intervals, reflecting greater uncertainty. The hyperparameters of the kernel can be learned by maximizing the log marginal likelihood of the observed data. For an illustration of the above hyperparameters, see fig. 1

2.1.2 Gaussian Process Classification

To motivate Gaussian process classification (GPC), it is helpful to first understand logistic regression, a widely used technique for binary classification. Logistic regression is a simple method that illustrates how a regressor can be turned into a classifier.
Logistic regression models the probability that a given input x belongs to a particular class. For binary classification, where the output y can be either 0 or 1, logistic regression uses the logistic (or sigmoid) function to map the linear combination of input features to the interval $[0, 1]$. The model is defined as follows:

$$p(y = 1 \mid x) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + \exp(-(\mathbf{w}^T \mathbf{x} + b))},$$

where $\mathbf{w}$ is the weight vector, b is the bias term, and $\sigma(z) = \frac{1}{1+\exp(-z)}$ is the sigmoid function.

The parameters $\mathbf{w}$ and b are learned by maximizing the likelihood of the observed data. The likelihood for a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ is given by:

$$\mathcal{L}(\mathbf{w}, b) = \prod_{i=1}^{n} \sigma(\mathbf{w}^T \mathbf{x}_i + b)^{y_i} (1 - \sigma(\mathbf{w}^T \mathbf{x}_i + b))^{1-y_i}.$$

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood, also known as the cross-entropy loss:

$$\mathcal{J}(\mathbf{w}, b) = -\sum_{i=1}^{n} \left[y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i + b)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i + b)) \right].$$

This optimization problem is typically solved using gradient descent or other numerical optimization methods like the famous iteratively reweighted least squares (IRLS) algorithm as it can be shown that the target function is concave and admits a unique maximum.
Logistic regression provides a straightforward approach to binary classification by modeling the probability of class membership using a linear decision boundary. However, in many real-world problems the relationship between the input features and the output class is not linear. Gaussian Process Classification (GPC) addresses this limitation.
Similar to logisitic regression, we regard our original model, the Gaussian process regressor, as a latent function which is mapped through a link function to obtain a probability estimate. For binary classification, the sigmoid (logistic) function is commonly used:

$$p(y = 1 \mid f(x)) = \sigma(f(x)) = \frac{1}{1 + \exp(-f(x))}.$$

given some training point x.
Inference is now done analogously as for gaussian process regression, by first marginalizing the latent function out:

$$p\left(f_* \mid X, \mathbf{y}, \mathbf{x}_*\right) = \int p\left(f_* \mid X, \mathbf{x}_*, \mathbf{f}\right) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f} \tag{2.1}$$

where $p(\mathbf{f} \mid X, \mathbf{y}) = p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid X) / p(\mathbf{y} \mid X)$ is the posterior over the latent variables and subsequently using this distribution over the latent function to produce a probabilistic prediction

$$\bar{\pi}_* \triangleq p\left(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*\right) = \int \sigma\left(f_*\right) p\left(f_* \mid X, \mathbf{y}, \mathbf{x}_*\right) df_*$$

Equation eq. (2.1) is not analytically tractable due to the non-Gaussian likelihood. Therefore, approximation techniques are needed for being able to make predictions.

2.1.3 Laplace Approximation

The need for approximating a posterior distribution is a common problem, which does not only turn up in the case of Gaussian process classification. The Laplace approximation is a powerful technique used to approximate posterior distributions when dealing with non-Gaussian likelihoods. This subsection presents the main notion and idea behind this approximation scheme, where we again follow closely [29].

We recall that for binary classification we model the probability of the output y given the latent function f using a Bernoulli distribution. The likelihood can be as usual be rewritten as:

$$p(y_i \mid f_i) = \sigma(f_i)^{y_i}(1 - \sigma(f_i))^{1-y_i},$$

where $\sigma(f)$ is the sigmoid function defined by $\sigma(f) = \frac{1}{1+\exp(-f)}$ but other choices are possible and do not fundamentally change the following calculations. The posterior distribution of the latent function $\mathbf{f}$ given the observations $\mathbf{y}$, $p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})$, is intractable in our case due to the non-Gaussian nature of the likelihood. The main idea of the Laplace approximation is to approximate the posterior with a Gaussian distribution. Specifically, the posterior distribution $p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})$ is approximated around its mode $\hat{\mathbf{f}}$ via a second order Taylor Expansion.

To find the mode $\hat{\mathbf{f}}$ of the posterior, we maximize the posterior distribution $p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})$. This is equivalent to maximizing the log-posterior:

$$\log p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) = \log p(\mathbf{y} \mid \mathbf{f}) + \log p(\mathbf{f} \mid \mathbf{X}) + \text{const.}$$

The log-posterior consists of the log-likelihood and the log-prior. For Gaussian processes, the prior is Gaussian, $p(\mathbf{f} \mid \mathbf{X}) = \mathcal{N}(\mathbf{f} \mid 0, \mathbf{K})$, where $\mathbf{K}$ is the covariance matrix. Thus, the log-posterior becomes:

$$\log p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) = \sum_{i=1}^{n} [y_i \log \sigma(f_i) + (1 - y_i) \log(1 - \sigma(f_i))] - \frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi.$$

This maximum is usually found using Newtons's method.

Once the mode $\hat{\mathbf{f}}$ is found, we approximate the log-posterior using a second-order Taylor expansion around $\hat{\mathbf{f}}$:

$$\log p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) \approx \log p(\hat{\mathbf{f}} \mid \mathbf{y}, \mathbf{X}) + (\mathbf{f} - \hat{\mathbf{f}})^\top \nabla \log p(\hat{\mathbf{f}} \mid \mathbf{y}, \mathbf{X}) + \frac{1}{2}(\mathbf{f} - \hat{\mathbf{f}})^\top \mathbf{H}(\mathbf{f} - \hat{\mathbf{f}}),$$

where $\mathbf{H}$ is the Hessian matrix of the log-posterior evaluated at $\hat{\mathbf{f}}$:

$$\mathbf{H} = \nabla^2_{\mathbf{f}} \log p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})\big|_{\mathbf{f}=\hat{\mathbf{f}}}.$$

Since $\hat{\mathbf{f}}$ is a mode, the first derivative term $\nabla \log p(\hat{\mathbf{f}} \mid \mathbf{y}, \mathbf{X})$ is zero. Thus, the log-posterior simplifies to:

$$\log p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) \approx \log p(\hat{\mathbf{f}} \mid \mathbf{y}, \mathbf{X}) + \frac{1}{2}(\mathbf{f} - \hat{\mathbf{f}})^\top \mathbf{H}(\mathbf{f} - \hat{\mathbf{f}}).$$

Exponentiating both sides, we obtain the Gaussian approximation:

$$p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) \approx q(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) := \mathcal{N}(\mathbf{f} \mid \hat{\mathbf{f}}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\Sigma} = -\mathbf{H}^{-1}$. Rasmussen notes, that one problem with the Laplace approximation is that it may give a poor approximation to the true shape of the posterior with respect to its shape. The peak could be much broader or narrower than the Hessian indicates, or it could be a skew peak, while the Laplace approximation assumes it has elliptical

contours. Finally, posterior mean and variance for f_* using methods used for Gaussian Process Regression, with which one can either calculate an averaged predictive probability of the form

$$\bar{\pi}_* \simeq \mathbb{E}_q \left[\pi_* \mid X, \mathbf{y}, \mathbf{x}_* \right] = \int \sigma \left(f_* \right) q \left(f_* \mid X, \mathbf{y}, \mathbf{x}_* \right) df_*$$

or as a MAP, the sigmoid of the expectation of $\mathbf{f} : \hat{\pi}_* = \sigma \left(\mathbb{E}_g \left[f_* \mid \mathbf{y} \right] \right)$. In case of binary classification, choosing predicted labels according to the class of highest probability yields identical results if either using the former or latter probabilities. For an explicit algorithm we refer to chapter 3.4.3 in [29].

2.2 Topological Data Analysis

In the following section, we will present the main mechanisms for applying topological data analysis and persistent homology in particular. We typically start by considering data in the form of plain point clouds, and then follow a data processing pipeline which eventually converts a plain point cloud into one concise object that contains topological information, a persistence diagram. We start by constructing a sequence of simplicial complexes, which are higher-dimensional analogs of graphs, built from the data points. By increasing a scale parameter, we obtain a filtration, a nested sequence of simplicial complexes, where each complex captures the topology of the data at a particular scale. Persistent homology then computes the homology groups of these complexes, tracking the birth and death of topological features as the scale parameter changes. These homology groups originate from a branch of pure mathematics, algebraic topology, and detect, very broadly speaking, n-dimensional holes in our object and are what can tell objects apart. This is because from a topology perspective, voids and holes can not appear from nowhere and vice versa. Persistent homology now generalizes this idea so that we can consider the evolution of these holes along a scale parameter. This output is often visualized using persistence diagrams or barcodes. The key concept of persistence diagrams is to consider features that persist over a long range of scales as significant, while those with short lifespans are considered noise. We can then distinguish point clouds by comparing their most persistent topological features. It is important to note, however, that points of low persistence can in general contain valuable information about the point cloud under consideration as well.

2.2.1 Simplicial Complexes

We start by considering simplicial complexes, which provide a discrete perspective for studying topological spaces. They allow for the application of combinatorial methods to solve topological problems, which can be run efficiently on a computer. The basic building block of a simplicial complex is the *simplex*. An n-simplex is the convex hull of $n + 1$ affinely independent points in an Euclidean space. These points are called the *vertices* of the simplex. The simplices of lower dimensions are well known objects:

- A 0-simplex is a point.
- A 1-simplex is a line segment.

- A 2-simplex is a triangle.
- A 3-simplex is a tetrahedron.

Formally, an n-simplex σ can be represented as:

$$\sigma = [v_0, v_1, \ldots, v_n], \tag{2.2}$$

where $v_0, v_1, \ldots, v_n$ are the vertices of the simplex.
A *face* of an n-simplex is any simplex formed by a subset of its vertices. If $\sigma = [v_0, v_1, \ldots, v_n]$ is an n-simplex, then any k-simplex $\tau = [v_{i_0}, v_{i_1}, \ldots, v_{i_k}]$ with $\{i_0, i_1, \ldots, i_k\} \subseteq \{0, 1, \ldots, n\}$ is a face of σ. The faces of a simplex include the simplex itself and the empty set. The *dimension* of a face is the number of vertices minus one.
A *simplicial complex* K is a finite collection of simplices that satisfies two conditions:

1. Every face of a simplex in K is also a simplex in K.
2. The intersection of any two simplices in K is either empty or a face of both.

A simplicial complex can be visualized as a combination of points, line segments, triangles, and higher-dimensional analogues. For example, a triangle with its interior, edges, and vertices forms a simplicial complex. Other well known mathematical objects can also be seen as simplicial complexes:

- **Graph**: A graph can be viewed as a 1-dimensional simplicial complex where the vertices represent 0-simplices and the edges represent 1-simplices.
- **Triangulation of a Polygon**: A triangulated polygon, where the interior is divided into non-overlapping triangles, is a 2-dimensional simplicial complex.

An *abstract simplicial complex* is a generalization of a simplicial complex, defined purely in combinatorial terms. It is a set K of finite, non-empty subsets of a set V, such that if $\sigma \in K$ and $\tau \subseteq \sigma$ and $\tau \neq \emptyset$, then $\tau \in K$ if . Here, V represents the set of vertices, and elements of K are the simplices.
For example, if $V = \{a, b, c\}$, an abstract simplicial complex could be
$K = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$, corresponding to a filled triangle.
The *geometric realization* of an abstract simplicial complex K is a topological space constructed from K by mapping its simplices to geometric simplices in a Euclidean space. This process assigns a point to each 0-simplex, a line segment to each 1-simplex, a triangle to each 2-simplex, and so on.
Formally, given an abstract simplicial complex K with a vertex set $V = \{v_0, v_1, \ldots, v_n\}$, the geometric realization $|K|$ is a subset of $\mathbb{R}^{n+1}$ formed by:

$$|K| = \bigcup_{\sigma \in K} \left\{ \sum_{i=0}^{n} \lambda_i v_i \mid \sum_{i=0}^{n} \lambda_i = 1, \lambda_i \geq 0 \right\}, \tag{2.3}$$

where each $\sigma \in K$ is mapped to the convex hull of its vertices in $\mathbb{R}^{n+1}$. Note that a geometric realization is even possible for $\mathbb{R}^{2d+1}$ by mapping the vertices to points in general linear position.

2.2.2 Homology

Homology is a fundamental concept in algebraic topology that provides a way to associate algebraic invariants to topological spaces, capturing their essential features such as holes and voids. The intuition behind homology is to measure the n-dimensional holes in a space. For example, in a surface like a torus, homology can detect the one-dimensional hole (the loop around the hole) the two-dimensional void (the interior of the torus), and a second one-dimensional hole.
We define homology for simplicial complexes, although theories like singular homology allow for a more general study of topological objects. As for simplicial complexes, both theories yield the same homology, we restrict to simplicial homology. We begin with a *simplicial complex* K. The combinatorial structure of a simplicial complex allows us to define algebraic objects that encode topological information.
An *n-simplex* in K is denoted as σ. An *n-chain* is a formal sum of n-simplices, expressed as $c = \sum_i a_i \sigma_i$, where a_i are coefficients (typically integers or elements of a ring although for persistent homology $\mathbb{F}_2 := \mathbb{Z}/2\mathbb{Z}$ is used), and σ_i are n-simplices. The set of all n-chains forms a free abelian group $C_n(K)$.
The *boundary operator* $\partial_n : C_n(K) \to C_{n-1}(K)$ is a homomorphism that maps an n-simplex to its $(n-1)$-dimensional faces. For an n-simplex $\sigma = [v_0, v_1, \ldots, v_n]$, the boundary operator is defined as:

$$\partial_n(\sigma) = \sum_{i=0}^{n} (-1)^i [v_0, \ldots, \hat{v_i}, \ldots, v_n]$$

where $\hat{v_i}$ denotes the omission of the vertex v_i.
For example, for a 1-simplex $[v_0, v_1]$, a line segment, the boundary is $\partial_1[v_0, v_1] = [v_1] - [v_0]$ and if using coefficients from $\mathbb{F}_2$ this would be equal to $[v_1] + [v_0]$.
An *n-cycle* is an n-chain $c \in C_n(K)$ with $\partial_n(c) = 0$. An *n-boundary* is an n-chain that is the boundary of an $(n+1)$-chain, i.e., $c = \partial_{n+1}(b)$ for some $b \in C_{n+1}(K)$. The set of n-cycles forms a subgroup $Z_n(K) \subseteq C_n(K)$, and the set of n-boundaries forms a subgroup $B_n(K) \subseteq C_n(K)$. An important observation, making the definition of homology valid, is that all boundaries are already cycles, i.e. $B_n(K) \subseteq Z_n(K)$ which can be proven by straightforward calculations.
The n-th *homology group* $H_n(K)$ of a simplicial complex K is defined as the quotient group:

$$H_n(K) = \frac{Z_n(K)}{B_n(K)}$$

This group measures the n-dimensional holes in the simplicial complex K. Elements of $H_n(K)$ are equivalence classes of n-cycles modulo n-boundaries. For example, H_0 captures connected components, H_1 detects loops or 1-dimensional holes, and H_2 identifies voids or 2-dimensional holes.
The key insight from algebraic topology is that the homology groups $H_n(K)$ are topological invariants, meaning they do not change under homeomorphisms (continuous deformations). Thus, homology provides a tool for distinguishing between different topological spaces based on their homology. If we would now obtain another object which does not have the exact same homology groups, we could assert that these two objects can not be

transformed continuously into one another. A famous consequence of these distinctions is for example Brouwer's fixed-point theorem. For a more thorough examination of the above material we refer to the standard book of Hatcher on Algebraic Topology, see [14].

2.2.3 Filtrations

Filtrations are a crucial tool for understanding the multi-scale structure of topological spaces. A *filtration* is an increasing sequence of simplicial complexes, each nested within the next. Three important filtrations are the Čech-filtration, the Vietoris-Rips filtration and alpha complexes, all of which are widely used in topological data analysis.
Given a point cloud $P \subset \mathbb{R}^d$ and a parameter $\epsilon \geq 0$, the *Čech complex* $\mathcal{C}_\epsilon$ is constructed as follows: For each subset $\sigma \subseteq P$, form a simplex if the closed ϵ-balls centered at the points of σ have a non-empty common intersection. Formally,

$$\sigma \in \mathcal{C}_\epsilon \iff \bigcap_{p \in \sigma} B(p, \epsilon) \neq \emptyset$$

where $B(p, \epsilon)$ denotes the closed ball of radius ϵ centered at p. Note that this defines an abstract simplicial complex.
As ϵ increases, the Čech complex grows, forming an increasing sequence of simplicial complexes:

$$\mathcal{C}_0 \subseteq \mathcal{C}_{\epsilon_1} \subseteq \mathcal{C}_{\epsilon_2} \subseteq \cdots$$

This sequence is known as the *Čech filtration*.
The *Vietoris-Rips complex* $\mathcal{R}_\epsilon$ is another important filtration. For a given point cloud $P \subset \mathbb{R}^d$ and a parameter $\epsilon \geq 0$, form a k-simplex for each subset $\sigma \subseteq P$ of $k+1$ points if the pairwise distances between points in σ are all less than or equal to ϵ. Formally,

$$\sigma \in \mathcal{R}_\epsilon \iff d(p_i, p_j) \leq \epsilon \quad \forall p_i, p_j \in \sigma$$

where $d(p_i, p_j)$ denotes the Euclidean distance between points p_i and p_j.
Similar to the Čech complex, as ϵ increases, the Vietoris-Rips complex forms an increasing sequence of simplicial complexes:

$$\mathcal{R}_0 \subseteq \mathcal{R}_{\epsilon_1} \subseteq \mathcal{R}_{\epsilon_2} \subseteq \cdots$$

This sequence is called the *Vietoris-Rips filtration*. The Vietoris-Rips filtration approximates the Čech filtration, but is often computationally simpler to construct.

2.2.4 Persistent Homology

After having constructed nested simplicial complexes, where we know how to compute the homology on each single one of them, persistent homology now allows us to generalize the ideas from homology to not only a single simplicial complex, but a nested sequence of simplicial complexes. Let K be a simplicial complex with an associated filtration:

$$\emptyset = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_m = K$$

Each K_i is a simplicial complex, and $K_{i-1} \subseteq K_i$ for all $i \in [m]$. The filtration parameter typically represents a scale, time, or some other meaningful metric but we leave it as

indices here. Note that one can find suitable transformations to switch between scale indices (measuring distance) and conventional indices by using monotonicity properties of the filtration.
For each K_i, we form a chain complex $C_*(K_i)$, where $C_n(K_i)$ is the free abelian group generated by the n-simplices of K_i. The boundary operator $\partial_n^i : C_n(K_i) \to C_{n-1}(K_i)$ is a homomorphism satisfying $\partial_{n-1}^i \circ \partial_n^i = 0$.
The n-cycles $Z_n(K_i) = \ker(\partial_n^i)$ and n-boundaries $B_n(K_i) = \mathrm{im}(\partial_{n+1}^i)$ form subgroups of $C_n(K_i)$. The n-th homology group $H_n(K_i)$ is defined as:

$$H_n(K_i) = \frac{Z_n(K_i)}{B_n(K_i)}$$

For $K_{i-1} \subseteq K_i$, the inclusion map induces, by functoriality, a homomorphism on chain complexes $\iota_*^i : C_*(K_{i-1}) \to C_*(K_i)$. This extends, again by functoriality, to homomorphisms on homology groups $\iota_*^i : H_n(K_{i-1}) \to H_n(K_i)$. Thus, we obtain a sequence of homology groups connected by inclusion-induced homomorphisms:

$$H_n(K_0) \xrightarrow{\iota_*^1} H_n(K_1) \xrightarrow{\iota_*^2} \cdots \xrightarrow{\iota_*^m} H_n(K_m)$$

Studying these induced homomorphism now is the key for computing persistent homology, which tracks the changes in homology classes across the filtration. An n-homology class $[c] \in H_n(K_i)$ is said to be *born* at K_i if it is not the image of any homology class in $H_n(K_{i-1})$ under ι_*^i. It *dies* entering K_j if its image under ι_*^j maps to zero in $H_n(K_j)$. The *persistence* of the homology class $[c]$ is the difference $j-i$, representing how long the class persists across the filtration.
The collection of birth and death pairs (i, j) for each homology class is summarized in a *persistence diagram*, where each point (i, j) corresponds to a homology class born at K_i and dying at K_j. Alternatively, a *barcode* is a visual representation of persistence intervals $[i, j)$, with each interval corresponding to a persistent homology class. For a more detailed treatment of persistent homology we refer to [12].

2.2.5 Stability Theorem of Persistent Homology

We now present one of the most important theorems relating to persistent homology, its stability result. For this, we need some preliminary definitions and a slight change of perspective. An important notion is that the construction of filtrations on a simplicial complex can be encoded by a so called *piecewise linear function* on the geometric realizations of the simplicial complex. Then, by considering the sublevel sets of this function on the geometric realization, the filtration can be obtained, see chapter 12.1 in [13] for details. We can then associate a piecewise linear function $f : |K| \to \mathbb{R}$ to a persistence diagram $D(f)$ originating from the filtration of some simplicial complex K. For both objects, piecewise linear functions f and persistence diagrams $D(f)$, we can define distance measures:

Definition 1 (Bottleneck Distance)**.** *The* bottleneck distance d_B *between two persistence diagrams* D_1 *and* D_2 *is defined as*

$$d_B(D_1, D_2) = \inf_{\varphi} \sup_{x \in D_1} \|x - \varphi(x)\|_\infty,$$

where φ ranges over all bijections from $D_1 \cup \Delta_1$ to $D_2 \cup \Delta_2$, and $\|\cdot\|_\infty$ denotes the L^∞-norm on $\mathbb{R}^2$.

Definition 2 (Supremum Norm)**.** *The* supremum norm *(or* infinity norm*) of the difference between two functions $f, g : X \to \mathbb{R}$ is defined as*

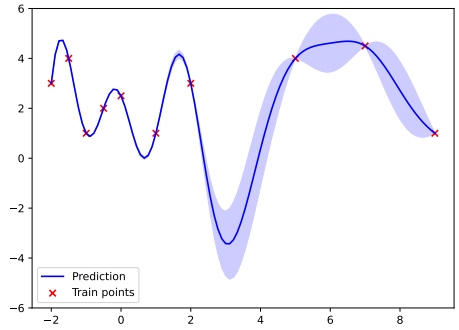
$$\|f - g\|_\infty = \sup_{x \in X} |f(x) - g(x)|.$$

We can now state the stability result:

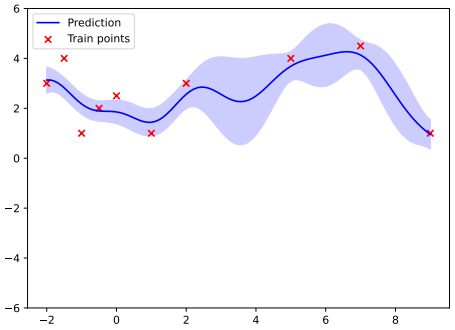
Theorem 1 (Stability Theorem)**.** *Let $f, g : X \to \mathbb{R}$ be two piecewise linear functions on a topological space X. Then the bottleneck distance between their persistence diagrams $D(f)$ and $D(g)$ satisfies:*

$$d_B(D(f), D(g)) \leq \|f - g\|_\infty,$$

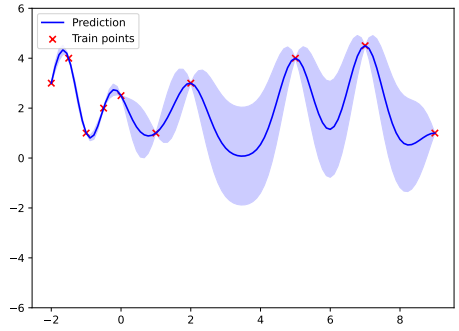
The theorem states that if two functions f and g are close to each other (in the sense of the supremum norm), then their persistence diagrams will also be close (in the sense of the bottleneck distance). This implies that small changes or noise in the input data will not drastically change the topological features detected by persistent homology. This stability result is crucial for applications in data analysis, where data may be subject to noise. It guarantees that the persistence diagrams provide a reliable summary of the topological features of the data, even when the data is slightly perturbed. For an illustration of persistence diagrams and the stability theorem, see fig. 2.



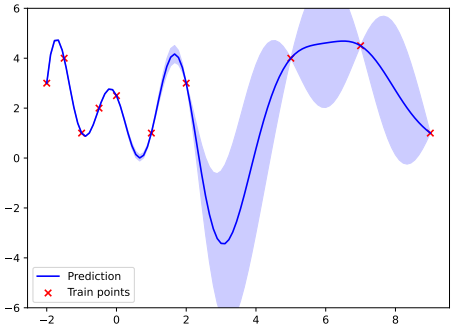
(a) $\sigma_f^2 = 1.0$, $\ell = 1.0$, $\sigma_n^2 = 10^{-10}$



(b) $\sigma_f^2 = 1.0$, $\ell = 1.0$, $\sigma_n^2 = 0.1$

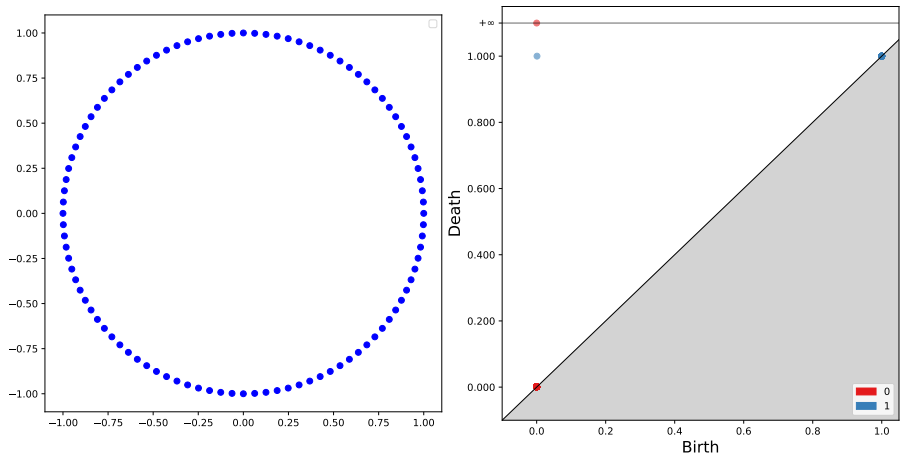


(c) $\sigma_f^2 = 1.0$, $\ell = 0.5$, $\sigma_n^2 = 10^{-10}$

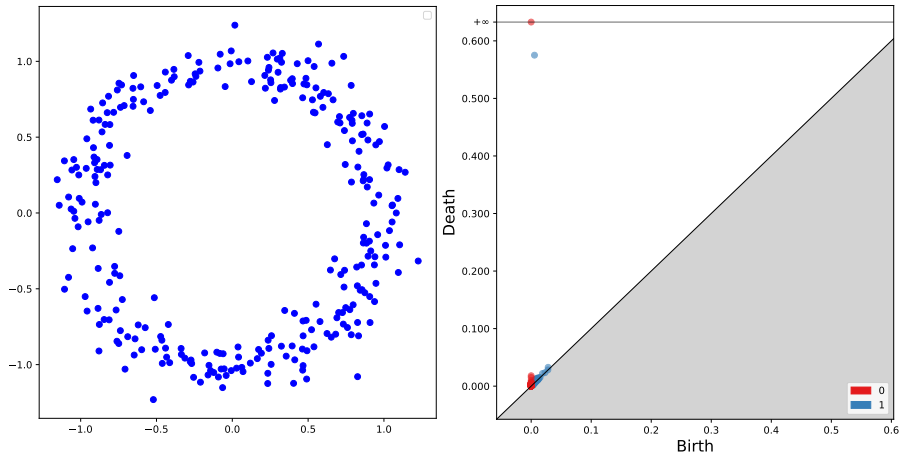


(d) $\sigma_f^2 = 5.0$, $\ell = 1.0$, $\sigma_n^2 = 10^{-10}$

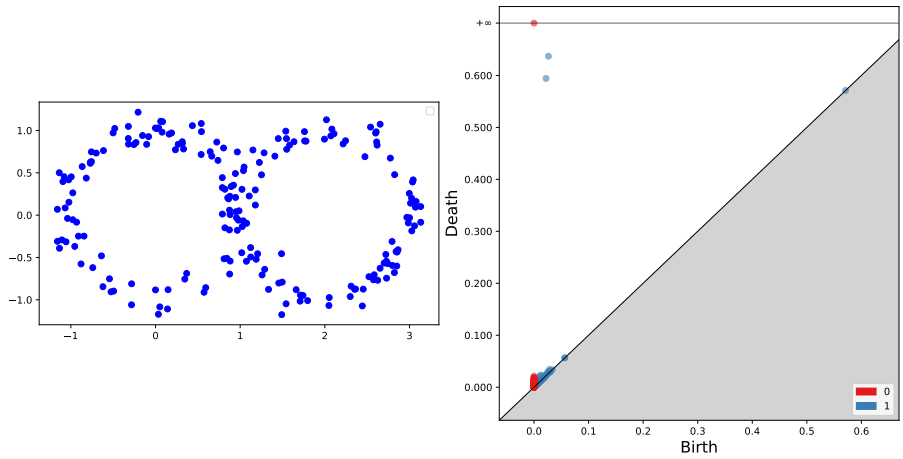
Figure 1: Sample plots for Gaussian process regression with different hyperparameters, illustrating the effect of varying the noise, variance and lenghtscale. Note that we keep the y-axis fixed for illustration purposes. Considering fig. 1a as baseline, we see that an increase of the noise results in the GP not interpolating the train points exactly, see fig. 1b, resulting in a smoother curve. It serves as regularization and is therefore able to better capture global trends of the data and can reduce overfitting. A decrease of lengthscale changes how quickly the prediction changes or how curved the predictions is, see fig. 1c and finally an increase of the variance simply increases the confidence around the prediction, see fig. 1d. The expected value does not change, compare to fig. 1a.



(a) Perfectly sampled circle with evenly spaced points on a circle. The corresponding persistence diagram has one point of very high persistence corresponding the fundamental group of the circle, which is isomorphic to $\mathbb{Z}$.



(b) Points sampled on a circle with noise. In the spirit of the stability theorem, we can see a similar structure to the above persistence diagram but with plenty of added point close to the diagonal which denote the noise.



(c) Samples from a figure eight with added noise. Note how we now have two points of high persistence. This is also expected as the fundamental group of this object is isomorphic to $\mathbb{Z} \bigoplus \mathbb{Z}$, which could for instance be proven via the Mayer-Vietoris sequence.

Figure 2: Three samples of topological spaces, illustrating persistent homology and the stability theorem.

2.3 Reproducing Kernel Hilbert Spaces

Reproducing Kernel Hilbert Spaces (RKHS) are a fundamental concept in kernel theory, providing a framework for various machine learning algorithms like Gaussian processes and support vector machines.
Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a non-empty set. A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *reproducing kernel* of $\mathcal{H}$ if it satisfies the following conditions:

1. For all $x \in \mathcal{X}$, $K(x, \cdot) \in \mathcal{H}$.

2. For all $x \in \mathcal{X}$ and $f \in \mathcal{H}$, the reproducing property holds:

$$f(x) = \langle f, K(x, \cdot) \rangle_{\mathcal{H}}. \tag{2.4}$$

The existence of such a reproducing kernel implies that $\mathcal{H}$ is an RKHS, and K is positive semi-definite.
The theory of RKHS is integral to both Support Vector Machines and GPs, as it provides the mathematical foundation for defining and working with kernel functions. RKHS enables the *kernel trick*, allowing algorithms to operate in high-dimensional feature spaces implicitly. The kernel function $K(x, x')$ is derived from an inner product in the feature space defined by the RKHS, enabling nonlinear classification boundaries in the original input space.
The covariance function in GPs is essentially the reproducing kernel of an RKHS, determining the prior distribution over functions. One of the most powerful aspects of RKHS is its ability to handle arbitrary data types, including non-numerical data, through the use of appropriate kernel functions. The kernel function $K(x, x')$ essentially measures the similarity between data points x and x', enabling the application of kernel methods to diverse types of data as long as a valid kernel is defined. RKHS can be utilized for various types of data such as text, graphs and images. The key is to design or choose a kernel function that captures the inherent similarity or structure in the data. Some examples include:

- **String Kernels**: Used for text data, such as the spectrum kernel, which compares the common substrings between two text sequences.

- **Graph Kernels**: Used for data represented as graphs, such as the random walk kernel or the Weisfeiler-Lehman kernel, which capture graph substructures and their similarities.

- **Image Kernels**: Used for image data, where kernels like the pyramid match kernel can compare sets of image features.

Choosing an appropriate kernel is crucial for the performance of both SVMs and GPs. Commonly used kernels in Euclidean space include the linear kernel (for linear decision boundaries), polynomial kernel, radial basis function (RBF) kernel (for highly non-linear decision boundaries), and Matern kernel. The selection depends on the nature of the data and the specific task.

A common example for applying kernels are Support Vector Machines (SVMs), which are a class of supervised learning models used for classification and regression tasks:

Primal Formulation Given a training set $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, the primal form of an SVM optimization problem is:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to} \quad y_i(w^\top \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \end{aligned} \tag{2.5}$$

where w is the weight vector, b is the bias term, ξ_i are slack variables, C is the regularization parameter, and $\phi(\cdot)$ is a feature mapping function.

Dual Formulation

Using Lagrange multipliers, the dual formulation of the SVM optimization problem is:

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ &\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{2.6}$$

where α_i are the Lagrange multipliers and $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ is the kernel function. The decision function for classification is then given by:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right). \tag{2.7}$$

3 Related Work

We touch upon two types of related work. On the one hand we present synergies of TDA and machine learning methods and we also present the most relevant models for point cloud classification in recent years.

3.1 TDA in Machine Learning

A question of particular importance is how to transform persistence diagrams into forms which can be feed into any kind of machine learning model, either through explicit vectorization, function representation or kernel evaluations. While there are approaches of calculating one-dimensional summary statistics of persistence diagrams, we present more extensive summary statistics. Bubenik proposed so called *persistence landscapes*, see [4], as a topological summary for persistence diagrams in form of a set of real valued functions. From these persistence landscape, various summary statistics can easily be derived and also be integrated into subsequent machine learning models. Building on the work of Bubenik, Kim et al. managed to incorporate persistence landscapes into a neural network architecture, reaching state of the art accuracies on the Orbit data set, see [18]. Another approach in the spirit of persistence landscapes were *persistence images*, introduced by Adams et al., see [2]. They highlight the importance of maintaining an interpretable connection to the original persistence diagram and those objects have found several usages in subsequent models. A particular advantage of their method is the ability conveniently provide the possibility to combine PDs of different homological dimensions into a single object.

As mentioned in the introduction, many ways have been found to incorporate topological information, by means of a persistence diagram or else, into neural networks. The first work to do so was done by Hofer et al., see [16]. They came up with a novel input layer for deep neural networks that takes a persistence diagram, and computes a parametrized projection that can be learned during network training. According to the authors, such a topological layer should be seen as a valuable addition to existing models but not as replacement. Carriere et al. introduced *PersLay*, see [7], a neural network layer that was inspired by *deep sets* [39] to enable permutation invariant processing of input data. Another particularly nice property of their model formulation is that many vectorization techniques such as persistent images or landscapes are encompased by this layer formulation and can be equipped with learnable parameters.

Not only has topological information been incorporated directly into the network architecture, but also into loss functions used for training the networks. Stucki et al. [35] have for instance defined a topological loss function via Betti number matching to improve performance with image segmentation. Moor et al. presented a so called *topological autoencoder*, see [23], which aims at calculating latent representations of data by calculating topological signatures of both the input and latent space and using a newly defined a topological loss term, trying to keep the topology of both spaces as similar as possible. For a more thorough investigation of the usage of topological method in machine learning, we refer to [15], reviewing relevant models using TDA in machine learning.

3.2 Point Cloud Classification

We briefly mention relevant work for state-of-the-art point cloud classification. One of the first models to achieve good results was PointNet [27], one of the first neural network architectures that directly consumes point clouds, simply using a series of multi-layer perceptrons. Whilst most previous works did some kind of feature engineering in order to process those feature with a subsequent machine learning model, Qi et al. designed neural network that is permutation invariant, which is crucial when processing point clouds which are essentially sets. The model was tested on various benchmarks for 3D shape classification like the ModelNet40 data set [37] and semantic segmentation, achieving state-of-the-art results. Following this work, the authors presented an improvement called PointNet++, see[26]. It builds on PointNet and introduces hierarchical feature learning, capturing local structures through a recursive grouping and feature extraction process. The core idea is to use a hierarchical approach to partition the point set into local neighborhoods and extract features in a hierarchical manner, thus also capturing global properties and remedy for instance for regions which differ in their sampling density. The model demonstrated superior performance on tasks involving point cloud classification and part segmentation compared to its predecessor. State of the Art performance on the ModelNet40 dataset is nowadays mostly achieved by CNNS, with the top three of the leaderboard having CNNs as building block at its core. The best performing model is *RS-CNN*, short for Relation-Shape Convolutional Neural Network, see [22]. For further models performing point cloud classification, we refer to the leaderboard of the ModelNet40 data set were many models reached accuracies of above 90%.

4 Methods

In this section we will present the methodology of this thesis. Most importantly, we provide a summarizing description of the four topological kernels for persistence diagrams mentioned in the introduction. Secondly, we will provide insights into the software we used and also about the implementations we did for using these kernels effectively for our experiments.

4.1 Persistence Scale Space Kernel

The Persistence Scale Space Kernel (PSSK) was introduced by Reininghaus et al. [30] and is the first work, presenting a connection of topological data analysis and kernel-based learning techniques such as support vector machines, kernel PCA and Gaussian Processes. More specifically, they propos a kernel for persistence diagrams using ideas from scale space theory. Denoting the space of persistence diagrams as $\mathcal{D}$, they define the kernel via a feature map $\Phi_\sigma : \mathcal{D} \to L_2(\Omega)$, where $\Omega \subset \mathbb{R}^2$ denotes the closed halfplane above the diagonal where the points of the persistence diagrams lie. They represent persistence diagrams as sum of dirac delta distributions, one for each point in a persistence diagram D. They then set the feature map Φ_σ as the solution of the heat equation at time $t = \sigma$, with initial conditions as exactly the sum of dirac delta distributions and finally set the kernel $k_\sigma(F, G) = \langle \Phi_\sigma(F), \Phi_\sigma(G) \rangle$. An evaluation of this integral yields the following explicit form of the kernel:

$$k_\sigma(F, G) = \frac{1}{8\pi\sigma} \sum_{\substack{p \in F \\ q \in G}} e^{-\frac{\|p-q\|^2}{8\sigma}} - e^{-\frac{\|p-q\|^2}{8\sigma}} \tag{4.1}$$

Note that the computational complexity of a kernel evaluation is $\mathcal{O}(|F||G|)$, where $|F|$ and $|G|$ denote the cardinality of the persistence diagrams, seen as multisets, F and G, respectively. Another important observation of this expression is that augmenting any persistence diagrams F and G with points on the diagonal does not change the kernel evaluation.
They further show that the kernel k_σ is 1-Wasserstein stable. This extends the important stability results from persistent homology to the kernel evaluation and shows that small perturbations in input data lead to small changes in the kernel evaluation. They compare the performance of the PSSK to persistence landscapes [4] and show that it proves superior to persistence landscapes both on tests with shape/image classification and texture classification and claim that the added flexibility of the scale parameter improves fine tuning of the model considerably.

4.2 Persistence Weighted Gaussian Kernel

The Persistence Weighted Gaussian Kernel (PWGK) was introduced by Kusano et al. [19] as another topological kernel. The main idea behind this kernel is to have the flexibility of controlling the influence of points of low persistence close to the diagonal. Also, to

vectorize the persistence diagrams such that common kernels from $\mathbb{R}^n$ like the linear or Gaussian kernel can be applied. To construct this kernel, they use the framework of embedding measures into reproducing kernel Hilbert spaces (RKHS), where persistence diagrams are regarded as a non-negative measure which can be embedded into a RKHS by the Bochner integral. For the explicit construction, we refer to [19]. Choosing the linear kernel to process the resulting feature vectors, yields

$$K_L(D,E) = \sum_{x\in\underline{D}}\sum_{y\in\underline{E}} w_{\text{arc}}(x)w_{\text{arc}}(y)k_G(x,y)$$

as kernel evaluation. Using the Gaussian kernel yields,

$$K_G(D,E) = \exp\left(-\frac{d_{k_G}^{w_{\text{arc}}}(D,E)^2}{2\tau^2}\right)$$

where τ is a positive parameter and

$$\begin{aligned} d_{k_G}^{w_{\text{arc}}}(D,E)^2 = &\sum_{x\in\underline{D}}\sum_{x'\in\underline{D}} w_{\text{arc}}(x)w_{\text{arc}}(x')\,k_G(x,x') \\ &+ \sum_{y\in\underline{E}}\sum_{y'\in\underline{E}} w_{\text{arc}}(y)w_{\text{arc}}(y')\,k_G(y,y') \\ &- 2\sum_{x\in\underline{D}}\sum_{y\in\underline{E}} w_{\text{arc}}(x)w_{\text{arc}}(y)k_G(x,y). \end{aligned}$$

for persistence diagrams D, E and with $w_{\text{arc}}(x) = \arctan\left(C\,\text{pers}(x)^p\right)$ $(C, p > 0)$, $k_G(x,y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$. We are only using the Gaussian version of this kernel throughout this thesis, as do the authors of [19]. They also prove a stability result for their kernel, which only holds if $p > d+1$, where $\mathbb{R}^d$ is the space where the point clouds lie in, from which then the persistence diagrams are being constructed. We follow the authors by setting $p := 5$ throughout this thesis.
The computational complexity of the PWGK is $\mathcal{O}(|F||G|)$. However, the authors provide a faster approximation scheme for approximating the entire Gram matrix $\left(K_G\left(D_i,D_j\right)\right)_{i,j=1,\dots,n}$. Using Random Fourier Features, the complexity can be reduced to $O\left(mnM + n^2M\right)$, where m denotes the maximal size of all persistence diagrams, n the size of the Gram matrix and M is the number of random variables used for the approximation. Finally, they state that they also used cross-validation for the choice of the appropriate parameters for supervised learning cases and present a heuristics for initializing the hyperparameters in an unsupervised learning setting.
They conduct various test on real world and synthetic data sets of which we are going to reproduce the protein classification experiment.

4.3 Sliced Wasserstein Kernel

The Sliced Wasserstein Kernel (SWK) by Carrière et al. [8] was the third kernel presented. They highlight that deriving a kernel is not straightforward, as the induced metric by the wasserstein distance is not negative semi-definite, thus inducing a positive-definite kernel with some appropriate transformation is not possible. To tackle this problem, they refer

to the so called *Sliced Wasserstein distance*, developed by Rabin et al. [28]. The definition is at follows:
Given $\theta \in \mathbb{R}^2$ with $\|\theta\|_2 = 1$, let $L(\theta)$ denote the line $\{\lambda\theta : \lambda \in \mathbb{R}\}$, and let $\pi_\theta : \mathbb{R}^2 \to L(\theta)$ be the orthogonal projection onto $L(\theta)$. Let $\mathrm{Dg}_1, \mathrm{Dg}_2$ be two persistence diagrams, and let $\mu_1^\theta = \sum_{p \in \mathrm{D}_{\mathrm{g}_1}} \delta_{\pi_\theta(p)}$ and $\mu_{1\Delta}^\theta = \sum_{p \in \mathrm{Dg}_1} \delta_{\pi_\theta \circ \pi_\Delta(p)}$, and similarly for μ_2^θ, where π_Δ is the orthogonal projection onto the diagonal. Then, the **Sliced Wasserstein distance** is defined as:

$$\mathrm{SW}\left(\mathrm{Dg}_1, \mathrm{Dg}_2\right) \overset{\text{def}}{=} \frac{1}{2\pi} \int_{\mathbb{S}_1} \mathcal{W}\left(\mu_1^\theta + \mu_{2\Delta}^\theta, \mu_2^\theta + \mu_{1\Delta}^\theta\right) \mathrm{d}\theta.$$

where we have that $\mathcal{W}(\mu, \nu) = \inf_{P \in \Pi(\mu,\nu)} \iint_{\mathbb{R}\times\mathbb{R}} |x - y| P(\,\mathrm{d}x,\ \mathrm{d}y)$ and μ and ν are non-negative measures on the real line. Knowing that this distance is negative definite, it is straighforeward to define the Sliced Wasserstein kernel as follows:

$$k_{\mathrm{SW}}\left(\mathrm{Dg}_1, \mathrm{Dg}_2\right) \overset{\text{def.}}{=} \exp\left(-\frac{\mathrm{SW}\left(\mathrm{Dg}_1, Dg_2\right)}{2\sigma^2}\right) \tag{4.2}$$

for $\sigma > 0$. They also present some stability result for their kernel. As an explicit computation of the SWK has running time of $O(N^2 \log N)$, the authors propose an approximation of the kernel with a running time of $O(NM \log N)$, where N denotes a bound on the cardinality of the persistence diagrams and M are directions sampled in the half circle $\mathbf{S}_1^+$. They also provide heuristics on how many directions to chose for reasonable performance, noting that in many cases very few directions suffice without any reduction of accuracies. For more details, we refer to [8]. Another important observations is that the kernel is *infinitely divisible*. This means that for the computation of several kernel evaluations, for different values of the parameter σ, only one evaluation of all pairs of sliced wasserstein distances is needed in order to efficiently compute all kernel evaluations. This is a major advantage when trying to find an optimal value for σ via cross-validation. They provide an explicit and straightforward strategy for choosing the best parameter. They calculate the first, second (i.e. the median) and third quantile of all pairwise sliced wasserstein distances and then additionally multiply each of these three quantiles with values from from the set $100, 10, 1, 0.1, 0.01, 0.01$, resulting in total of 15 tested parameters. On a general note, the key factor for time efficient model training when using cross-validation thus is the cost of the training the subsequent kernel based machine learning model. When using a GP, this means a cubic running time in the number of input points. Finally, they compare their proposed kernel to the PWGK and PSSK using various benchmark data sets which we will use in this thesis as well, showing superior performance of the SWK compared to the other two kernels.

4.4 Persistence Fisher Kernel

The Persistence Fisher kernel (PFK), introduced in [20], is the most recent of the proposed kernels. Its central idea is taken from information geometry, by using the so called *Fisher information metric* for the construction of a kernel for persistence diagrams. As already seen several times, the authors start by considering a persistence diagram D as a sum of dirac measures $\mu = \sum_{u \in \mathrm{Dg}} \delta_u$. Then, given some bandwidth $\sigma > 0$, they smooth and

normalize this measure in the following way:

$$\rho_{\mathrm{Dg}} := \left[\frac{1}{Z} \sum_{u \in \mathrm{Dg}} \mathrm{N}(x; u, \sigma I) \right]_{x \in \Theta}$$

where $Z = \int_\Theta \sum_{u \in \mathrm{Dg}} \mathrm{N}(x; u, \sigma I) \mathrm{d}x$ and N denotes the Gaussian density at point x with mean u and covariance matrix $\sigma I \in \mathbb{R}^2$ with I being the identity matrix. This parameter has a very nice intuitive meaning. A high variance makes the persistence diagrams look very similar and a small variance makes different persistence diagrams very dissimilar. A "good" σ thus will be a value that balances these two extremes well for objects from the same and different classes. Interpreting two persistence diagrams as probability measures ρ_i and ρ_j, one can define the Fisher information metric as

$$d_{\mathcal{P}}\left(\rho_i, \rho_j\right) = \arccos\left(\int \sqrt{\rho_i(x)\rho_j(x)} \mathrm{d}x \right)$$

where the integral is taken over the set Θ. To obtain the definition of the fisher kernel some more intricate steps are required. We define $\mathrm{Dg}_{i\Delta} := \{\Pi_\Delta(u) \mid u \in \mathrm{Dg}_i\}$, where $\Pi_\Delta(u)$ is the projection of the point u on the diagonal and we define

$$d_{FIM}\left(Dg_i, Dg_j\right) := d_{\mathcal{P}}\left(\rho_{\left(Dg_i \cup Dg_{j\Delta}\right)}, \rho_{\left(Dg_j \cup Dg_{j\Delta}\right)}\right)$$

The authors prove that this distance measure is indeed negative definite, which makes it possible to apply the Gaussian kernel to this distance measure, which in turn defines a valid kernel and gives rise to the following definition:

$$k_{\mathrm{PF}}\left(\mathrm{Dg}_i, \mathrm{Dg}_j\right) := \exp\left(-t d_{\mathrm{FIM}}\left(\mathrm{Dg}_i, \mathrm{Dg}_j\right)\right)$$

We note that for the actual computation, we set $\Theta := \mathrm{Dg}_i \cup \mathrm{Dg}_{j\Delta} \cup \mathrm{Dg}_j \cup \mathrm{Dg}_{i\Delta}$.
They also used grid search for finding the best hyperparameters (of which there are two, t and σ). For the former they calculated all three quantiles of all pairwise distances, similar to the SWK paper, and for finding an optimal sigma they simply iterated over the set $\{100, 10, ..., 0.001\}$ as possible candidates and combined all these choices for the choice of the optimal hyperparameters.
They also performed various tests and compare their proposed kernel with the three previously introduced kernels and claim to have reached superior performance.

4.5 Implementations

There are several libraries implementing various functionalities of GPs, most importantly GPy and scikit-learn. As GP Regression is straightforward to implement, they differ only in their implementation for classification tasks. GPy's implementation uses the Expectation propagation algorithm and Scikit implements the Laplace approximation, see [29] for more details of these algorithms. We chose to use the scikit-learn library for our basic implementations. However, GPy comes with an implementation of Hamiltonian Monte Carlo algorithm which we used as well. We also note that the most time consuming computations were the kernel evaluations. Therefore, we did not use libraries or algorithms

tailored for fast Gaussian process calculations such as sparse Gaussian processes, see [21].

We implemented the PSSK and PWGK and PFK from scratch in python and used the implementation of the SWK from the pytorch-topological library. By doing so, we could use automatic differentiation for the PGWK and PSSK for using gradients for training. Another important observation is that for the PSSK, persistence diagrams can be padded with points on the diagonal without altering the evaluation of the kernel. We can therefore provide an even even faster implementation, where we padded all persistence diagrams to the same length and could speed up computations by stacking all persistence diagrams of equal length into a large 4-dimensional tensor and then compute the entries of the Gram matrix on a GPU, enabling faster computation times. We used the approximation via fast Fourier features provided by the authors of the the PWGK. Finally, we note that we did not get the approximation scheme for the PFK running and therefore could not use this kernel effectively for very large data sets.

5 Basic Experiments

This section provides a first examination of our model on mostly synthetically generated topological data. It focuses more on testing and implementing basic concepts and ideas, rather than assessing the best possible performance of the model, which we will do in chapter 6.

5.1 Basic Topological Objects

We want to start our investigation of using topological kernels for Gaussian processes by considering their behaviour on five objects, commonly used in algebraic topology and persistent homology:

- **Circle**, denoted by S^1. Homology is given by $H_0(S^1) \cong \mathbb{Z}, H_1(S^1) \cong \mathbb{Z}, H_2(S^1) \cong 0$.
- **Sphere**, denoted by S^2. Homology is given by $H_0(S^2) \cong \mathbb{Z}, H_1(S^2) \cong 0, H_2(S^2) \cong \mathbb{Z}$
- **Unit cube**, denoted by C. Homology given by $H_0(C) \cong \mathbb{Z}, H_1(C) \cong 0, H_2(C) \cong 0$, contractible.
- **Torus** , denoted by T. Homology given by $H_0(T) \cong \mathbb{Z}, H_1(T) \cong \mathbb{Z}^2, H_2(T) \cong \mathbb{Z}$
- **Annulus**, denoted by A, Homology is given by $H_0(S^1) \cong \mathbb{Z}, H_1(S^1) \cong \mathbb{Z}, H_2(S^1) \cong 0$.

We set up a basic first test: We train a GP classifier with three objects from each of the five classes and test it with 10 objects from each class. We sample 400 points uniformly at random from each object. We ensure that the magnitude of all objects is the same in order to focus on purely topological information. We compute persistence diagrams using the Vietoris-Rips filtration in the first dimension. Tuning the hyperparameters with a simple grid search over multiples of 10 of the hyperparameters, we get perfect accuracy for all kernels, except the PWGK, where we reach an accuracy of 0.82. We make a first important observation: The three kernels with one hyperparameter are easier to tune than the PWGK, and in our basic test case, this makes it possible to get perfect accuracies for all kernels except the PWGK, where we, however, cannot rule out that it is not able to model the 5 objects in a better way. The added hyperparameters of the PWGK promise more flexibility and we will investigate in the following sections if it is possible to take advantage of this flexibility.

As classification with a GP is done by choosing classes of the highest probabilities, we want to analyze what these probabilities can tell us. We successively add several levels of Gaussian noise to all five test objects and monitor the average confidence/probability over the correct classes returned by the GP. One might hypothesize that an increase in noise in the test data results in less confidence of the GP for each correctly classified object. Remembering the stability theorem of persistence homology, we do not expect an immediate worsening of the accuracies. The evolution of the accuracies can be found in table 1. We notice that the PWGK performs similarly with all noise levels, whereas all other kernels seem very robust to slight perturbations. This is in full spirit of the stability

Noise Level	PSSK	SWK	PWGK	PFK
No noise	1.0	1.0	0.85	1.0
0.01	1.0	1.0	0.82	1.0
0.05	1.0	1.0	0.86	1.0
0.1	1.0	1.0	0.9	1.0
0.25	1.0	1.0	0.85	1.0
0.5	0.96	0.92	0.93	1.0

Table 1: Accuracies of all four topological kernels on the five classes with increasing noise. Noise was added only to test data, not the train data.

Noise Level	PSSK	SWK	PWGK	PFK
No noise	0.476	0.222	0.321	0.363
0.01	0.476	0.222	0.321	0.367
0.05	0.473	0.223	0.32	0.363
0.1	0.493	0.222	0.321	0.358
0.25	0.511	0.221	0.316	0.347
0.5	0.54	0.22	0.319	0.338

Table 2: Average confidences over all correctly classified points, with increasing levels of noise. Noise was added only to test data, not the train data. For increasing levels of noise, We can not observe a decrease of the probability/confidence for the average confidence of the correct classes. We can not read off the increasing uncertainty in the test data form the returned probabilities.

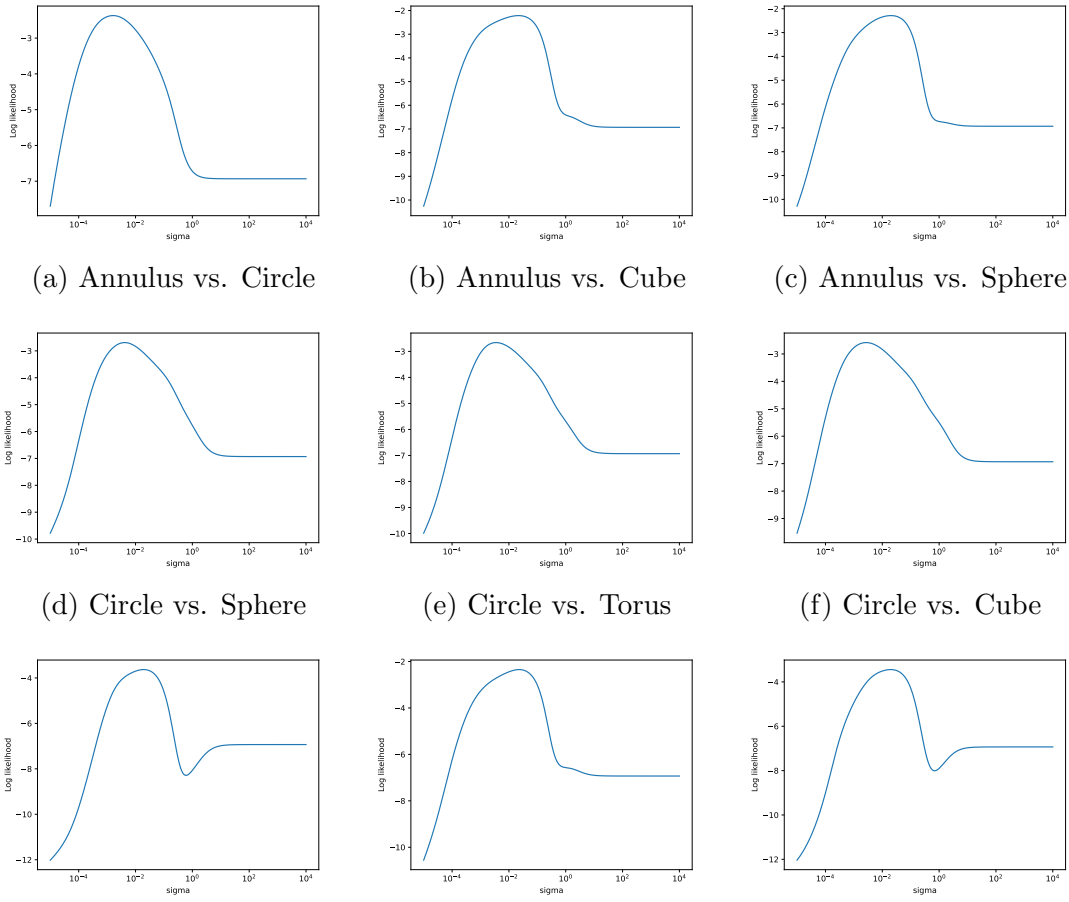
theorem of persistent homology. The evolution of the average confidences can be found in table 2. Interestingly, all kernels return different magnitudes of probabilities. The SWK barely returns more than 0.2 confidence, the necessary minimum for a perfect accuracy on five objects, while the PSSK shows comparably high confidence. Different kernels seem to produce different confidence levels, despite being able to make perfect predictions, so there does not seem to exist an immediate way to compare the confidence of different kernels. Also, the increased noise level is not reflected in the returned confidences at all, as they stay mostly unchanged for each kernel. As we will later see, there is another way to interpret the returned confidences through probability calibration.

5.2 Optimization of Kernels

For probabilistic models, there is usually a natural approach for finding the "best" model in case those models depend on one or several hyperparameters. This is done by searching for hyperparameters that maximize the likelihood of the posterior distribution of the model. Whilst some caution with this method is necessary, we will investigate this for Gaussian process classifiers using topological kernels. For the PSSK and PWGK, we test how well optimization of the likelihood via gradient ascent works in general. For the PFK we present a heuristic that might be useful for finding a near-optimal value. As the SWK is infinitely divisible, we can compute many Gram matrices by exponentiation of only one distance matrix, making simple grid search our preferred choice for this kernel.

PSSK: For the PSSK, there is only one hyperparameter to consider in the classification case; we are dealing with a simple one-dimensional problem. A closed-form derivative of the PSSK can easily be calculated and implemented, and we have gradient-based optimization at our disposal. We start by only comparing pairs of shapes with each other, e.g. sphere vs. torus and so on. For most pairs, the likelihood has a unique maximum, see fig. 3, and subsequent evaluation on test data shows that using this maximum suffices to obtain perfect accuracies between most pairs of classes. There is one test case, however, where the likelihood function does not exhibit this behaviour: sphere and torus, where the likelihood function can be seen in fig. 4a or fig. 4b. There are two problems: First, during gradient descent, the hyperparameter value explodes as the likelihood is mostly increasing. Secondly, using a large value as found during gradient descent, yields a worse accuracy than the parameter found by simply using cross-validation. Using only likelihood optimization instead of cross validation for all 5 classes combined yields an accuracy of only 0.9. We have thus found an example where maximizing the likelihood does not necessarily correspond to the best possible accuracy for the PSSK. We increased the number of sampled points and also used an alpha-complex instead, but retained the same behaviour. Interestingly, when considering the second dimension instead of the first, this behaviour does not persist and we obtain a nice likelihood with a unique maximum again, see fig. 4c. This maximum is able to perfectly distinguish the sphere and the torus. Finally, we observe that if choosing the initial value of the optimization procedure too large, gradient ascent does not converge for the well-behaved models neither, as for values larger than 10^1 the likelihood seems constant for all pairs shown in fig. 3. Therefore the quality of the solution also depends on the choice of the initial parameter. As we are considering a simple 1D maximization problem, simple optimization procedures like Golden Section Search prove more robust for finding the maximum of the log-likelihood. This method is less dependent on the initial choice of the parameter and converges very quickly for this simple data set. We conclude that maximizing the likelihood is always a very helpful option if computation resources admit it. However, for the PSSK, the obtained hyperparameters should be double-checked with cross-validation of some grid of parameters.

PWGK: The persistent weighted Gaussian kernel is slightly more involved as the number of hyperparameters is 3. We let the parameter p be always fixed, as the authors do in [19]. Gradients of these computations can be obtained via automatic differentiation. When starting gradient descent from the proposed heuristic for the five-class classification case, we reach an accuracy of 0.82. By starting with 10 restarts on the above data set (yielding a total of 55 optimization runs, as we have to fit 5 classifiers for each restart), we can get the performance up to 0.92. Interestingly, this is slightly better than the performance of the PSSK when using values obtained by maximizing the likelihood of the PSSK, which achieved 0.9 accuracy. However, this procedure is very time-consuming. As gradient descent is not satisfactory, we investigate different optimization procedures such as Bayesian optimization. This method is particularly well suited for problems where the cost function is very expensive to evaluate, which is clearly the case here. With this method, we can reach an accuracy of 0.958, while keeping the computing time comparably small compared to the gradient descent approach. So even though we do not reach perfect

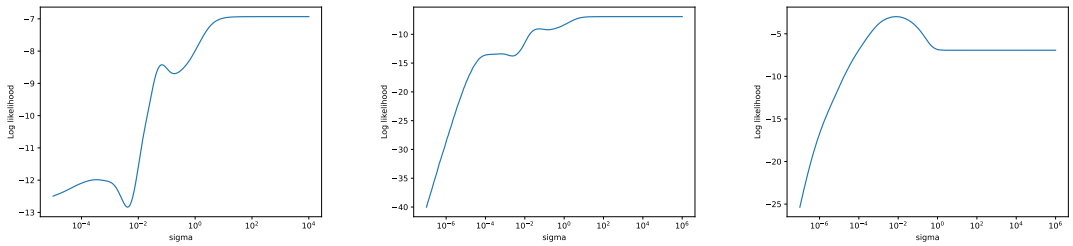


(a) Annulus vs. Circle (b) Annulus vs. Cube (c) Annulus vs. Sphere

(d) Circle vs. Sphere (e) Circle vs. Torus (f) Circle vs. Cube

(g) Cube vs. Torus (h) Annulus vs. Torus (i) Sphere vs. Torus

Figure 3: Log-likelihoods for Gaussian process classification for pairs of topological objects, using first-dimensional persistence diagrams. All likelihoods have a unique maximum and selecting this maximum for training also yielded perfect classification for all nine pairs. Note than when starting with a value larger than roughly 10^1, gradient ascent would not converge.



(a) First dimension, 400 points per object

(b) First dimension, 800 points per object.

(c) Second dimension, 400 points per object

Figure 4: Log-likelihoods for Gaussian process classification of sphere vs. torus using a VR-filtration. We can observe that the likelihood mostly increases for increasing σ, both in figs. 4a and 4b and does not admit a unique maximum. When using second dimensional PDs, see fig. 4c, we have a unique maximum again.

accuracy for the five-class classification task as the other kernels do, the PWGK seems well suited for applying Bayesian optimization to it. By doing so, it is able to almost fully describe these five topological objects and does this better than, for instance, the PSSK when simply maximizing the likelihood.

PFK: The PFK has two parameters, t and σ, where σ denotes the variance of the Gaussian distributions put over each point of the persistence diagrams. Initial experiments, using automatic differentiation for optimizing sigma, do not yield satisfactory results. Due do numerical reasons, discontinuities can occur during the computation. In this thesis, we will settle for grid search and in order to reduce computational complexity, we always choose t as the inverse median value of all computed distances, given some σ. Since the PFK still proves to be the computationally most expensive kernel, we present a heuristic for selecting the optimal σ. We make the following observation: The PFK puts Gaussian distributions on all points of the persistence diagram (and the projections of the paired PD's as well). After normalizing the sum of these Gaussians to a probability distribution, we have followed the same procedure as Kernel Density Estimation (KDE) does. KDE is a non-parametric tool from descriptive statistics where one seeks to obtain a distribution function, provided sample points, without any prior knowledge about the underlying distribution. For both methods it is important not to choose the variance of the Gaussian's too small but not too big either, as then different distributions become indistinguishable. Several ideas for balancing this trade-off exist, for example Silvermans Rule [32], where we set

$$\sigma_{\text{Silverman}} = \left(\frac{4}{(d+2)n}\right)^{\frac{1}{d+4}} \sqrt{\det(\mathbf{\Sigma})}$$

where d is the dimension, n the number of data points and Σ is the covariance matrix of the data. Similarly, we have Scotts Rule [31], given by

$$\sigma_{\text{Scott}} = \left(\frac{n}{\text{Vol}(D)}\right)^{\frac{1}{d+4}}$$

where D is the volume of the data domain. We conduct the following test: we sample 3 data points per topological object (15 in total) and test with 20 point clouds per class and add noise to all point clouds. We do grid search on 40 choices of sigma, chosen uniformly in the log space between 10^{-5} and 10, and plot the obtained accuracies by the PFK on the test data. Additionally, we mark the performance of the PFK for choosing σ as both KDE estimates. We get a promising result, see fig. 5. Both estimated sigmas are exactly among those for which the PFK attains perfect accuracy. We have thus obtained a promising rule of thumb for selecting a candidate for the variance which balances the trade off between making persistence diagrams from the same class appear similar and making persistence diagrams from different classes appear less similar.

For the same test case, we also plotted the accuracies for various hyperparameters of the PSSK and SWK, see fig. 6. We can see that for the SWK, a very wide range of choices for σ suffices to obtain a perfect accuracy, making this kernel very easy to tune. For the PSSK, the range of optimal choices is not as big. So at least for these simple

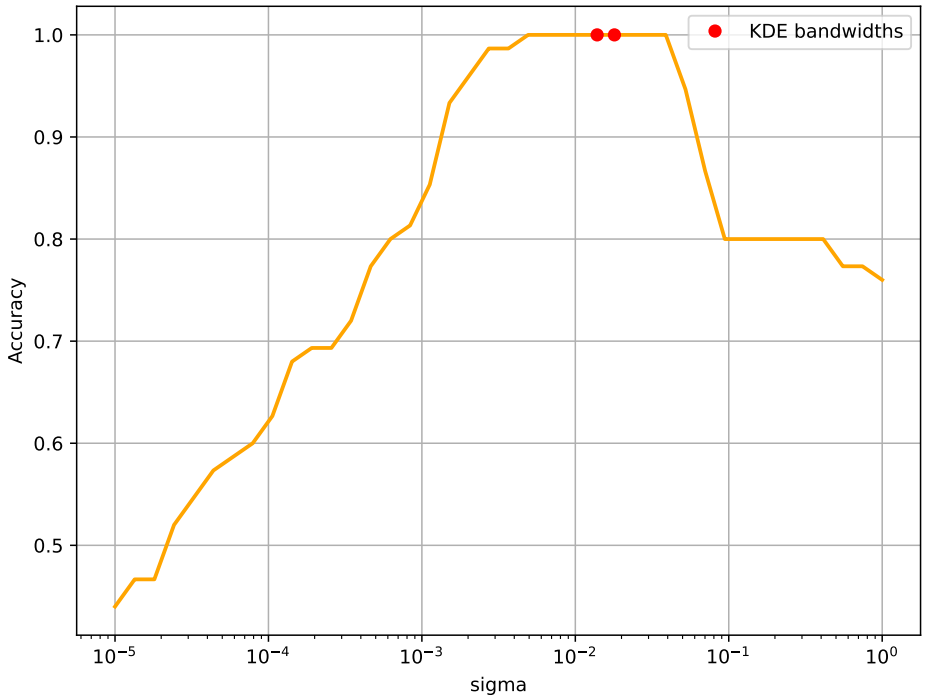
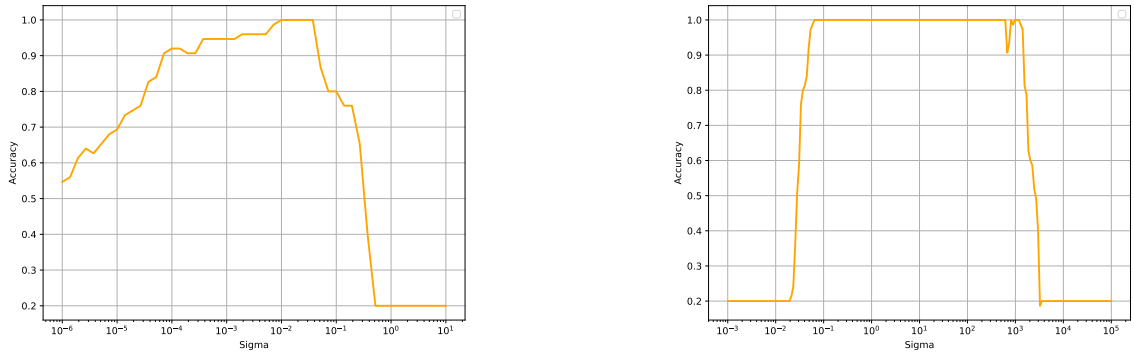


Figure 5: Sigma vs. Accuracy for PFK for classification of 5 topological objects, plotted on a log-scale. We highlighted both KDE estimates with red points and can observe that they are among those which achieve a perfect accuracy. Note also that at least for this simple case, a wide range of parameters reaches perfect accuracy.

test cases, simply cross validating several hyperparameters of varying magnitude seems to be justified in order to achieve the highest accuracy possible, in particular for the SWK.



(a) PSSK (b) SWK

Figure 6: Sigma vs. Accuracy for classification of 5 topological objects for two kernels, plotted on a log-scale. For the SWK, we see a wide range of parameters which achieve perfect accuracy, indicating a very easy training process if using cross-validation. For the PSSK, this interval is not as wide, however for the interval $[10^{-4}, 5 \cdot 10^{-2}]$ accuracy still is quite good with more than 90%

5.3 Markov Chain Monte Carlo

We want to further analyze the PWGK. Firstly, we want to investigate if performance of the kernel can be improved by MCMC simulations and secondly, if we can derive meaningful insights from the sampled posterior distributions. In particular, we consider the five topological objects from above with objects resized to similar magnitudes. As usual, we are considering only a binary classifier, thus we perform 5 simulations, one for each object against the remaining ones. We employ Hamiltonian Monte Carlo, choose suitable priors, stepsizes and number of steps and scale the hyperparameters to be of equal magnitude. We note that we choose a very conservative prior as we do not want to incorporate too many assumptions about the hyperparameters. We merely choose unimodal distributions for the prior such that the density at zero vanishes and such that the hyperparameters can not get too large. We also choose them to contain the heuristic provided of the authors in [19] in a region of high probability density, as we do note expect the values to deviate drastically from these values. We generate 600 samples with a burnin of 200 samples. The results of all objects can be seen in table 3 and one exemplary trace plot and pair plot can be seen in fig. 7 and fig. 8. First of all, we observe that the trace plots show no evidence against convergence. Secondly, taking the sample mean and the 75th quantile of the sampled posterior distribution, always improves the accuracy, showing that MCMC can indeed provide yet another valuable optimization procedure for the PWGK. The results are of course sensitive to the choice of the priors over the hyperparameters and different distributions can potentially also make the median or mode better choices. Thirdly, the pair plots also reveal that there are no immediate correlations between the hyperparameters, as the scatter plots of all pairs of variable seem uncorrelated. There is thus no evidence that the number of hyperparameters could be reduced and the model could be simplified. So despite being more expensive to optimize, the added flexibility of the PWGK can still be advantageous as we will later confirm for the orbit data set. We also note that the obtained distribution and the trace do not seem too complicated in our test case and fewer runs of the simulation would have sufficed to explore the full parameter space.

	Heuristic	Median	Mode	Mean	25th quantile	75th quantile
Torus	0.8	0.8	0.8	0.84	0.8	0.8
Sphere	0.8	0.8	0.8	0.88	0.8	0.87
Cube	0.9	0.89	0.88	1.0	0.83	1.0
Circle	0.8	0.8	0.8	1.0	0.8	1.0
Annulus	0.8	0.8	0.8	1.0	0.8	1.0

Table 3: Accuracies for Hamiltonian MCMC simulation for GP Classifier posterior distribution for five topological objects. We trained One vs. Rest for each object and report accuracies for some heuristic, median, mode, average and quartiles of the sampled points.

5.4 Interpolation

We try to come up with a transformation between topological objects in order to test if the GP classification model is able to detect “intermediate” objects along these transfor-

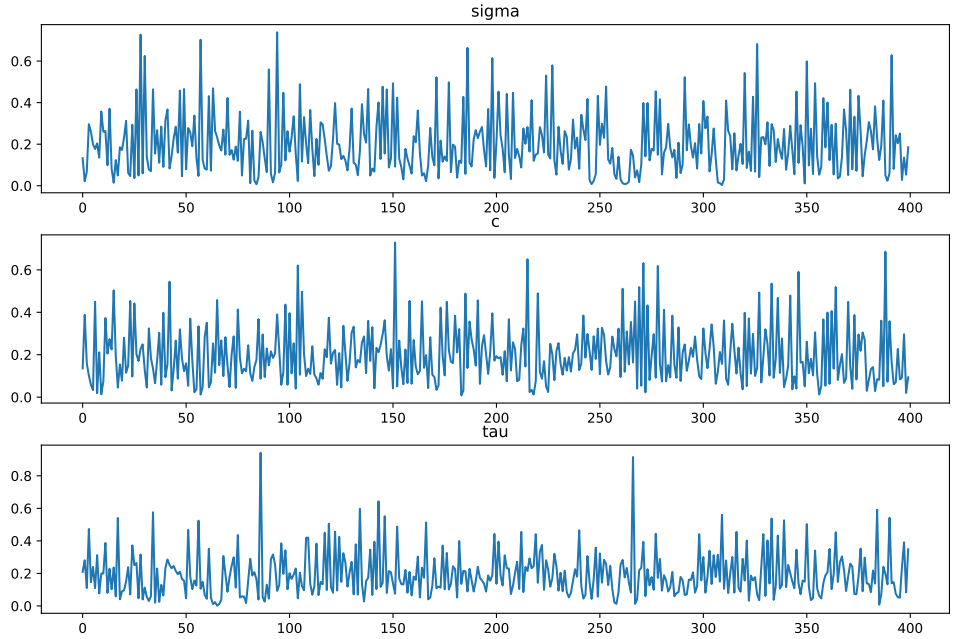


Figure 7: Trace plot for hyperparameters of PWGK when training Torus vs all other objects. There is no evidence against convergence.

mations it was not trained on. This means, that if the GP sees an objects that is similar to two objects but not any other the GP was trained on, the GP returns a higher and similar probability for those two classes and lower probabilities for the remaining ones. For simplicity, we always train on objects from three classes and test on objects which we want to see "in between." Training on only two classes would not be useful, as in this case an equal probability would not show if the GP either does not know what object it is dealing with or detects similarities between both objects.

We start with a very simple idea by generating three two-dimensional spheres S^2 with radii of 1, 5, and 9 cm, upon which the GP is trained. Note that we are handling this as a discrete classification problem. For this training phase, we utilize 20 objects per class, each object comprising 200 sampled points. Specifically using the PWGK with dimension 0, we observe satisfactory classification results for samples drawn from these spheres. We can achieve a confidence level of approximately 70% for each class and perfect classification accuracy.

We generate 17 spheres, each composed of 200 sampled points again, with radii incrementally increasing from 1 to 9 cm in steps of 0.5 cm, denoting a simple scaling of the sphere. Our hypothesis states that spheres with radii in between e.g. class 1 and 2 exhibit higher probability for classes 1 and 2 but a very low probability for class 3. This conjecture is confirmed, as illustrated in fig. 9. We observe that for radii in between 1 and 5, only the first two classes are assigned significant probabilities. Subsequently, for radii in between 5 and 9, the second and third classes demonstrate higher probabilities. In particular, the outcomes at radii of 1, 5, and 9 cm match our expectation as well. Similar results were obtained when training with the PD in dimension one. However, this behaviour is not

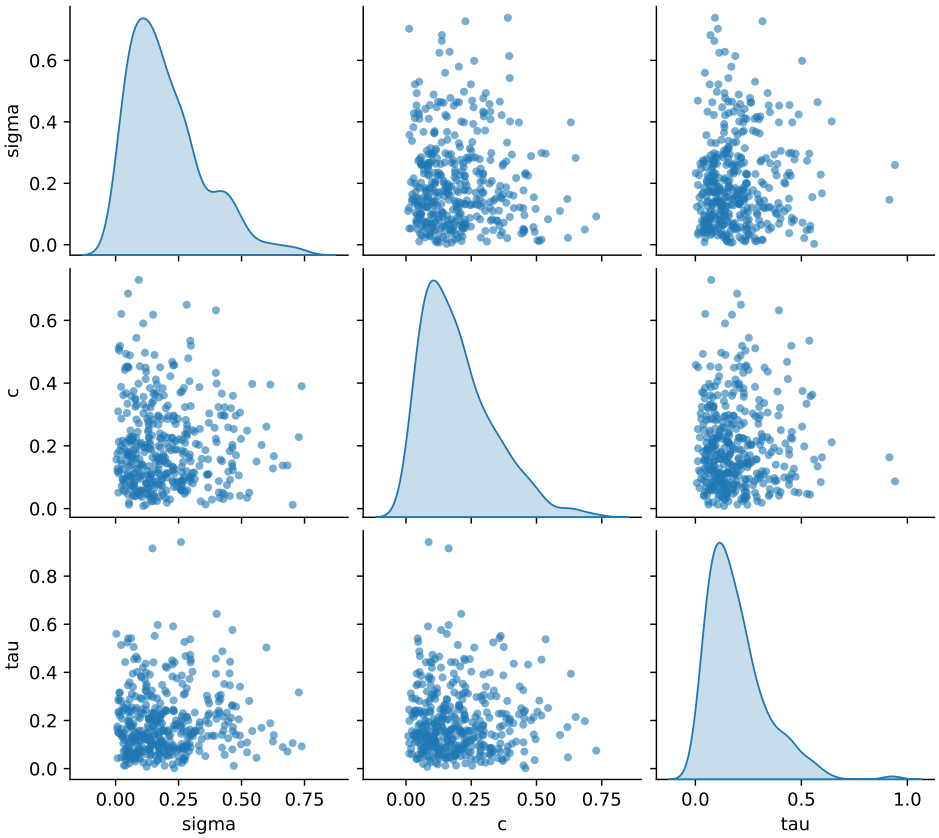


Figure 8: Pair plot for hyperparameters of PWGK when training torus vs. all other objects. For all parameter we can observe that the marginal distributions are unimodal and do not exhibit complex behaviour. Also, for no two pairs of hyperparameters a clear correlation pattern can be observed.

surprising when considering the persistence diagrams of the scaled spheres. The persistence diagrams are simply scaled as well, as long as the number of sampled points is kept fixed, see fig. 11. Comparing diagrams from spheres whose radii only differ slightly, and keeping in mind that we have many points in dimension zero and one, we notice that the points of those persistence diagrams still overlap a lot. The further the radii grow apart, the less the persistence diagrams overlap and the less similarity the kernel detects.

Interestingly, when training with the PSSK in dimension 2, we do not obtain this result, as can be seen in fig. 10. In the intermediate regions between these three classes, the kernel has proven to be unsuccessful in distinguishing the objects. However, we used $\sigma = 0.001$ as choice of the hyperparameter, a very small value. This scale factor makes spheres with only marginally different diameter appear very different. This is confirmed in fig. 11, where we observe that the persistence diagrams in dimension 2 in fact only consist of a single point, which matches our intuition as we have $H_2(S^2) \cong \mathbb{Z}$. When sampling objects from the same class, those points tend to be almost identical, as we did not perturb the sampled points with any noise. Those points in the 2nd dimensional persistence diagram originating from different classes, however, tend to be apart, and a low σ amplifies this behaviour when comparing the diagrams with the PSSK. This shows again that scale parameters play an important role in most kernels, and the quality and behaviour of the kernel is heavily dependent on selecting an appropriate scale.

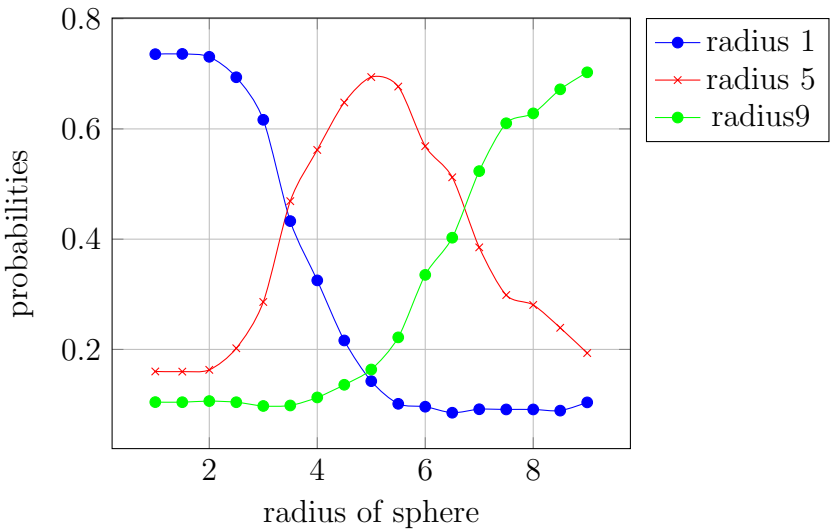


Figure 9: Probabilities of GP+PWGK, dimension 0, for each of the three classes which were used for training, tested with several spheres with increasing radius. Note that we report the full probability vector over all three classes, as we can observe that for each point on the x-axis, the corresponding y-values add up to one.

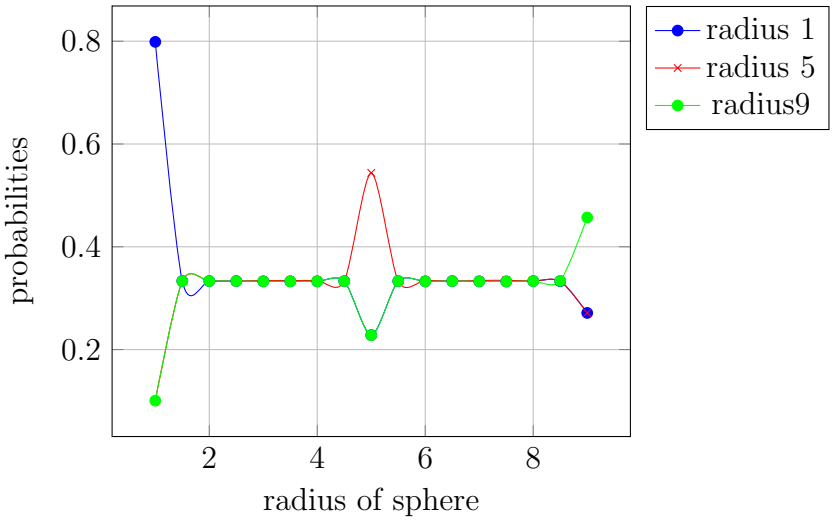
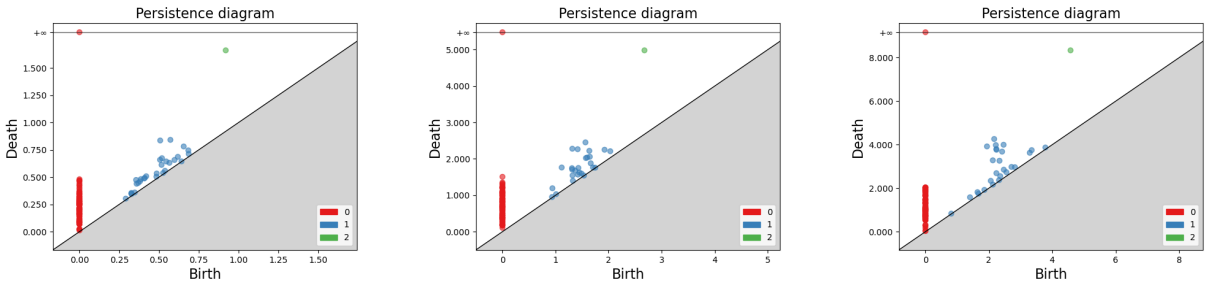


Figure 10: Returned probabilities of the GP with PSSK, dimension 2, for each of the three classes which were used for training, tested with several spheres with increasing radius. In the intervals $(1, 5)$ and $(5, 9)$ the probability is mostly $\frac{1}{3}$, showing that the GP does not detect any similarity. Note that we report the full probability vector over all three classes, which sum up to one for all test points.

5.5 Synthetic Data

We now turn to a slightly more complex data set that was investigated in [4]. It consists of six shape classes: a unit cube, a circle of diameter one, a sphere of diameter one, three clusters with centers randomly chosen in the unit cube, three clusters within three clusters (where the centers of the minor clusters are chosen as small random perturbations from the major cluster centers), and a torus with a major diameter of one and a minor diameter of one half. We produce 25 point clouds of 500 points sampled uniformly at random from each of the six shapes, and then add Gaussian noise. We compute persistence diagrams of these point clouds in dimension 0 and 1 using the Vietoris-Rips filtration. This gives 150 point clouds in total, and we evaluate our kernels with random 15/10 splits for each class



(a) Sphere of radius 1 (b) Sphere of radius 3 (c) Sphere of radius 5

Figure 11: Persistence Diagrams of spheres of radius 1, 3 and 5. Note that the diagrams only differ in their scale but not in their general structure. We can also observe that the diagrams of 0th and first dimension consist of many scattered points, whereas the second dimensional diagrams consists of a single point.

Kernel	Accuracy (Noise 0.05)	Accuracy (Noise 0.1)
PWGK, H_0	90.1%	94.8%
PWGK, H_1	100%	97.5%
PSSK H_0	76%	69%
PSSK H_1	100%	92.8%
PFK, H_0	95.7%	95.7%
PFK, H_1	97.3%	95.8%
SWK H_0	99.9%	98.7%
SWK H_1	100%	99%

Table 4: Accuracies on data set used in [4] for all four kernels, dimension 0 and one with two different levels of noise added.

and repeat this 50 times to obtain an average accuracy. We will search the optimal hyper-parameters using grid search, see table 4 for results of all kernels across dimension 0 and 1.

We can observe that for dimension 0, the PSSK performs particularly bad compared to the other three kernels, with the SWK performing particularly well. Interestingly, it performs just as well in dimension 0 compared to dimension 1, which is not the case for all three other kernels. This is a particularly nice behaviour as, by considering only 0-dimensional persistent homology, one could save some computation time if using the Vietoris-Rips filtration. On the other hand, the size of 0-dimensional peristence diagrams scales directly with the number of sampled points; therefore, the kernel evaluation becomes more costly. The increase of noise also affects each kernel differently, with some reducing their accuracy but with others being barely affected at all. Apart from the SWK, all kernels performed better using 1st-dimensional homology. With very little noise, all kernels except the PFK could perfectly learn all six objects. With an increase of noise, the SWK proves most robust compared to all other three kernels, with the PSSK performing particularly bad, which we could observe in both dimensions for the PSSK. This shows that not all kernels outperform persistence images, but the SWK does. It is worth mentioning that for using a noise level of 0.05 and considering first-dimensional homology, already 1 data point for each class as a training set suffices to reach accuracies of 0.99 with the SWK, PSSK and PFK kernel. The only difference is the average confidence/probability per correct class with which the GP classifies each point. These probabilities increase with the number of trained points, as can be observed in fig. 12. So an increase of number of trained points is somewhat reflected by an increase of the returned probabilities, although these probabilities differ for the kernels and do not seem directly comparable.

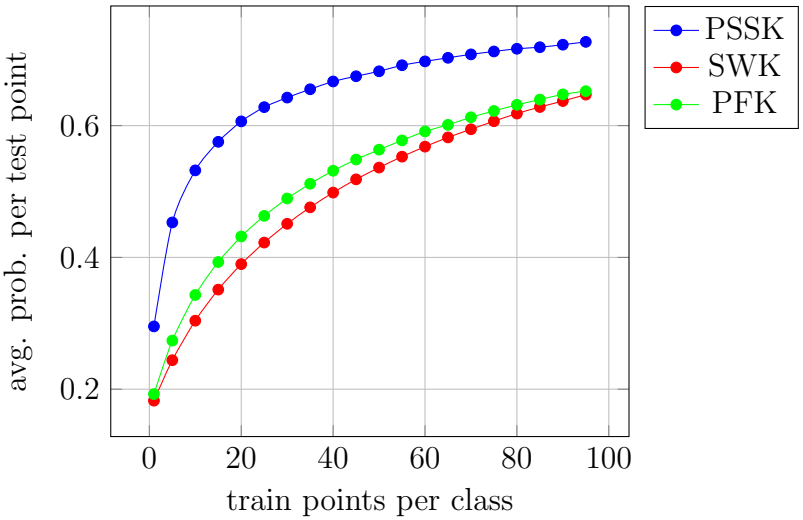


Figure 12: Average confidence of the GP over all correctly classified points, plotted against the number of trained points. For each number of trained points, we performed cross-validation and averaged the obtained confidence. Note that for these three kernels, accuracy was always at 1.0 for more than 20 train points and always higher than 0.98. We can observe a clear increase of the confidence with an increase of trained points.

5.6 Orbit Data

The orbit data set is another synthetic data set that has been used as a benchmark by several papers, including [8] and [20]. The latter performed a comparison of all 4 kernels on this set using support vector machines. The data set models orbits of a discrete dynamical system (up to some finite time), which exhibits chaotic behaviour and is sensitive to the choice of initial conditions. The initial conditions are chosen randomly within $[0, 1]^2$ and for starting values (x_0, y_0), the map is defined as follows:

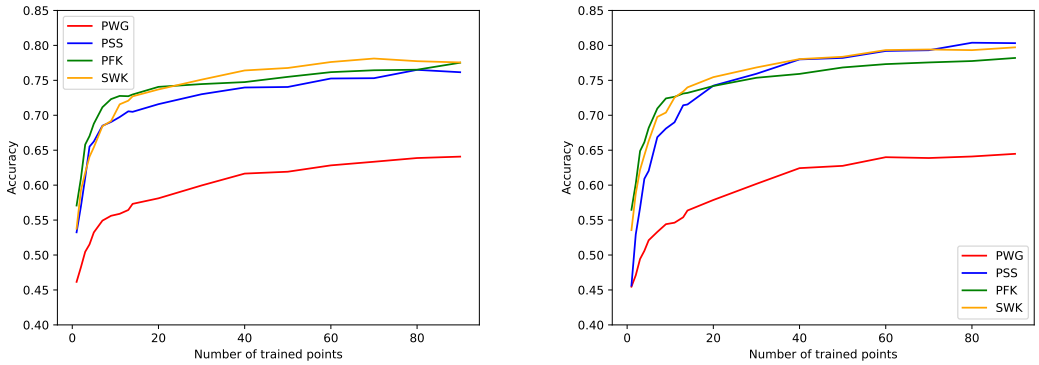
$$\begin{cases} x_{n+1} = x_n + r y_n (1 - y_n) & \mod 1 \\ y_{n+1} = y_n + r x_{n+1} (1 - x_{n+1}) & \mod 1 \end{cases}$$

given some parameter $r > 0$. Sample plots of all 5 considered parameters can be found in fig. 16, where we can identify clear topological similarities and dissimilarities between the point clouds by investigating the one-dimensional holes in the scatter plots. We follow the setup given in [20] and generate 100 point clouds per class with random initial conditions and iterate the map 1000 and 500 times respectively. We again generate a random 70/30 and 20/80 split and repeat this 100 times. We perform these four tests in order to highlight both the influence of the sampled points and the number of trained points on the performance of the model. The parameters of the kernels are chosen with grid search. Interestingly, for an increased number of sampled points and an increased number of points in the persistence diagram, the optimal σ for the PSSK an PFK tended to decrease. For the PFK, this consideration is very intuitive, as for an increasing number of points in the persistence diagram, the Gaussian distributions centered around the points tend to overlap more in case σ stays constant. This in turn results in more similar probability distributions for each diagram, which makes telling objects from different classes apart

more difficult. The results can be found in table 5.

First of all, we also report results for 0-dimensional persistence diagrams and can observe that they are inferior to the first dimension for all kernels and prove to be particularly bad for the PSSK. In the following sections we will therefore only restrict to first dimensional persistence diagrams. This is additionally justified as diagrams in the first dimension tend to be smaller than those of dimension 0, reducing the running time of the kernel evaluations. Considering the first dimension, we can observe that for the 70/30 split with 1000 points, all 4 kernels perform similar to those in [20]. Again, the SWK and PFK are better suited for categorizing the orbits compared to the PSSK and PWGK, with the SWK having the best accuracy. More interestingly, as could be observed already above, performance decreases only marginally if decreasing the number of trained points down to 20, in particular for the SWK and PFK with 1000 points per orbit. Interestingly, we can observe that the PSSK performs better when being trained with fewer points. When optimizing the likelihood of the PSSK, numerical problems arose as the optimized parameter got too small, causing problems with the positive definiteness of the Gram matrix. Optimizing with non-gradient-based methods like Golden Section Search also did not improve the performance. We also optimized the PWGK with Bayesian optimization for only 20 test points per class and 500 points per orbit (for practical computation time) and could improve the accuracy up to a staggering 0.83 for this specific case. This is surprising as in this case the PWGK performs even better than the SWK, which could only reach an accuracy of 0.78 in this case. This could be an indication that the PWGKs larger parameter space indeed allows for a more flexible adaption on more complex data sets. It also shows again that maximizing the likelihood is an important option for the PWGK. Finally, for this particular test case we could observe that the number of sampled points per train object should not be chosen too low. For the SWK and PFK, dropping the number of sampled points per object from 1000 to 500 with a fixed number of trained points per object of 70, reduced the accuracy by roughly 5%.

An obvious advantage of having a model that uses Gram matrices is the possibility of computing the Gram matrix for large data sets once, and then being able to assess the model's performance for various numbers of training points. Therefore, to further illustrate the effect of the number of trained points on the performance of a GP (and a SVM), we compute a Gram matrix for 100 point clouds for each class, resulting in a total of 500 point clouds, and then assessed the accuracy for various numbers of trained points per class between 1 and 90, see fig. 13. We note that both plots look almost identical, showing that for this data set, neither GP or SVM prove superior to the other. Already with one test point, all 4 kernels perform with reasonable accuracy of around 50% (using already tuned hyperparameters) and from around 10 points onward, all kernels except the PWGK only marginally improved. The PWGK does increase its accuracy for an increased number of training points. Also, the fact that the accuracies reach a plateau hints at the fact that persistent homology combined with an SVM or GP probably cannot, only perhaps with many more data points, explain the entire logic behind the data set. However, when only training with a very small number of points, for the GP there is always the option to use likelihood-maximization. This enables robust and convenient training. As described above, for the PWGK this could significantly improve performance even for only 20 test



(a) GP (b) SVM

Figure 13: Number of points used for training per class plotted against the accuracies for the orbit data set with 500 points per orbit. For each number of trained points per class, accuracy was obtained by cross-validating 50 times. We can observe that the accuracies increase roughly the same way when either using a SVM or a GP.

points and 500 iterations per orbit, becoming the best model for this selection of test points and iterations.

We again check how hell the KDE estimates perform for the PFK, and can observe that they do not quite, but almost perform as good as the best parameters, see fig. 15. Again, their choice should be done with great care as there is only an intuitive justification for this to work but we can empirically confirm that both estimates at least grasp the magnitude of the optimal hyperparameter and can potentially make fine tuning faster.

Kernel	Accuracy 70/1000	Accuracy 20/500	Accuracy 20/1000	Accuracy 70/500
PWG, H_0	62%	53.2%	56%	59.8%
PWG, H_1	$65.9 \pm 3.0\%$	$60.1 \pm 2.1\%$	$61.1 \pm 2.3\%$	$62.6 \pm 3.2\%$
Heat H_0	20%	20%	20%	20%
Heat H_1	$65.0 \pm 2.4\%$	$75 \pm 2.7\%$	$68.9 \pm 4.7\%$	$72.4 \pm 5.0\%$
PFK, H_0	63.9%	54.6%	59%	68.5%
PFK, H_1	$84.2 \pm 2.3\%$	$78 \pm 2.1\%$	$82.1 \pm 2.1\%$	$80.6 \pm 3.2\%$
SWK H_0	63.1%	54.6%	62.1%	61.3%
SWK H_1	$\mathbf{87.8 \pm 2.4}\%$	$\mathbf{78.1 \pm 1.5}\%$	$\mathbf{84.8 \pm 1.7}\%$	$\mathbf{82.6 \pm 2.6}\%$

Table 5: Accuracies for orbit data set with "points per class"/"size of point clouds" over dimension one and zero. We report only results of 1st dimensional homology with variances obtained via cross-validation as zero-dimensional persistence diagrams performed significantly worse.

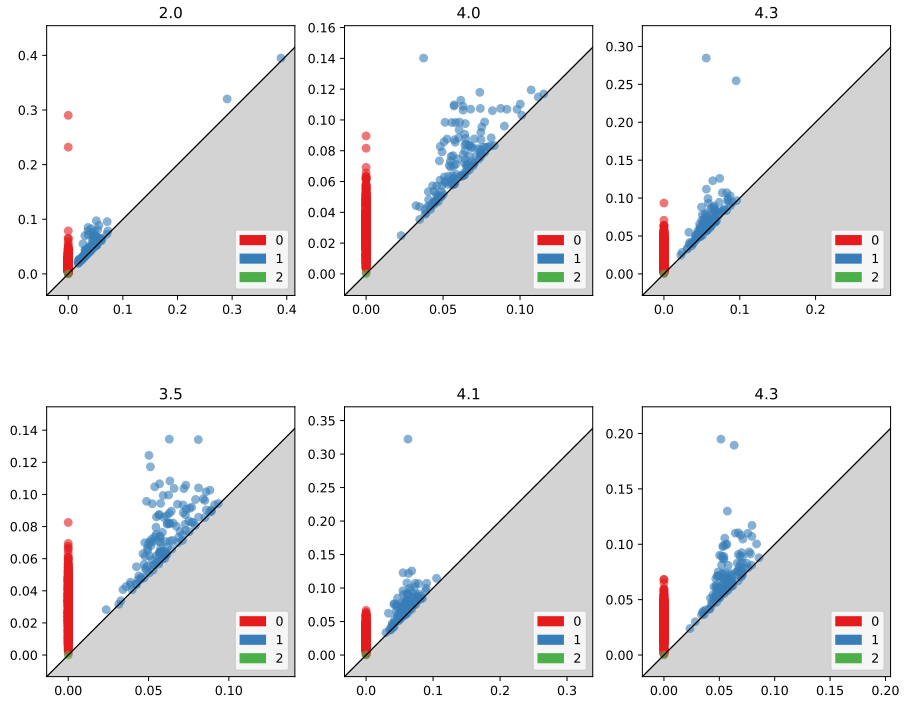


Figure 14: Persistence diagrams for different parameters of the Orbit data set obtained via Vietoris-Rips filtration. We can observe that the diagrams contain many points with varying persistence, sometimes with no distinct topological feature at all.

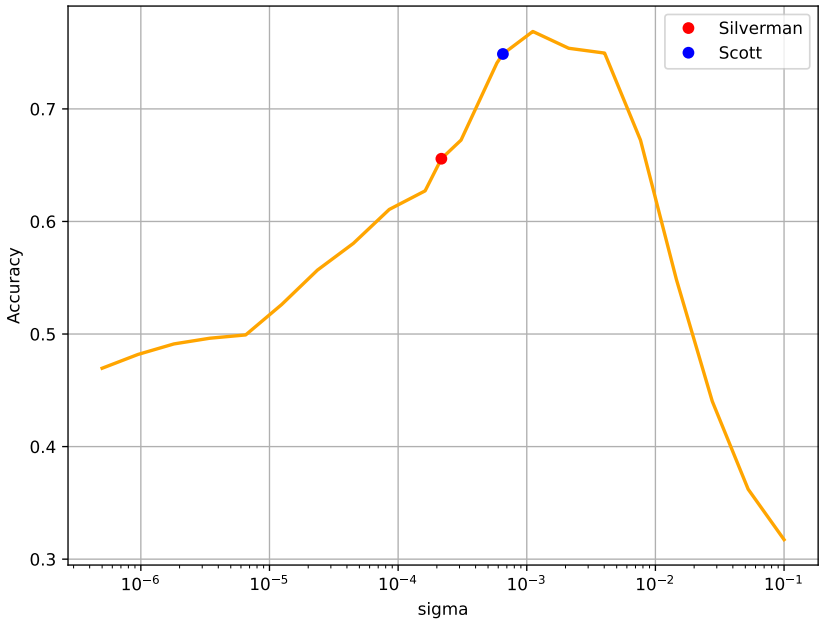
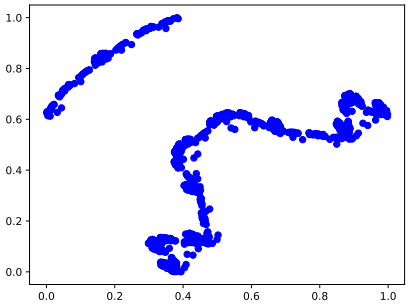
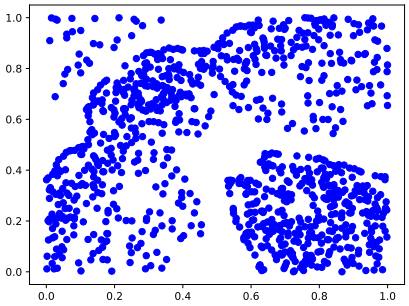


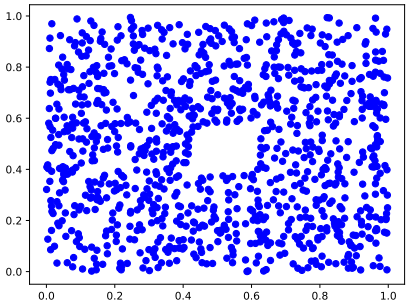
Figure 15: σ vs. accuracy for orbit data set for the PFK, plotted on a log-scale, highlighting the KDE estimates. We performed the experiment with 10 train points per class and computed accuracy for 20 test points per class.



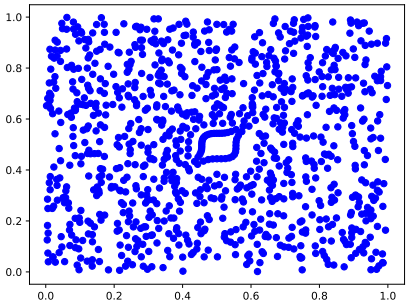
(a) $r = 2.0$



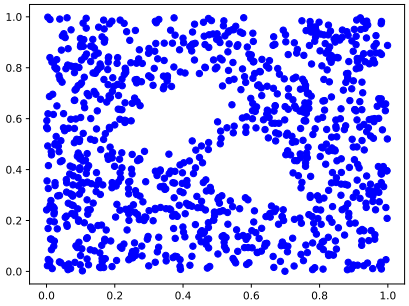
(b) $r = 2.0$



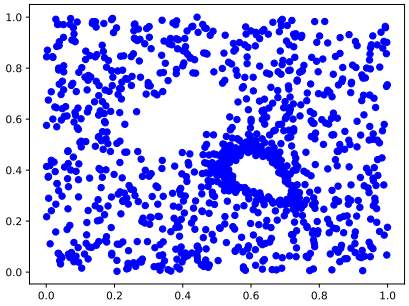
(c) $r = 4.0$



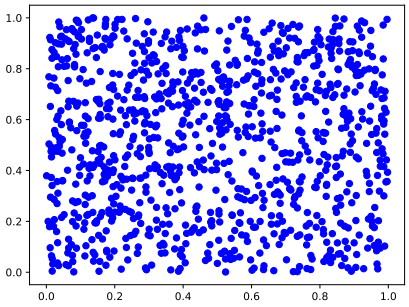
(d) $r = 4.0$



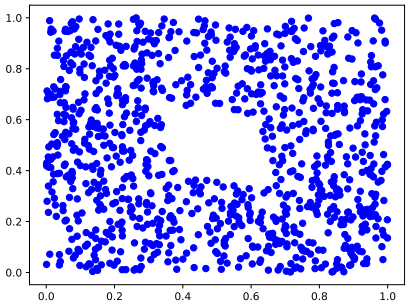
(e) $r = 4.3$



(f) $r = 4.3$



(g) $r = 3.5$



(h) $r = 4.1$

Figure 16: Scatter plots of orbit data sets for varying hyperparameters. Notice how data sets with the same hyperparameter can appear dissimilar while having clear topological similarities, compare for instance fig. 16c and fig. 16d or fig. 16e and fig. 16f.

6 Elaborate Tests

This sections aims at replicating and conducting experiments on real world data sets. We seek a direct comparison with previous experiments, which all relied on support vector machines for performing classification of point clouds with topological kernels. We will also test GPs on a common point cloud classification data set, ModelNet10, in order to stress test our model formulation on a large multi-class data set. Building on the experiments conducted in this chapter, we will also perform probability calibration in order to asses if the predictions of the model can be interpreted with respect to correct classification rates.

6.1 Contour Shape Classification

We replicate an experiment conducted in [20]. We extract the boundaries of a 10-class subset of the MPEG7 object shape data set. Each class has 20 samples. We randomly split the data into 70/30 sizes and repeat this process 50 times. The parameters were chosen with grid search for computational reasons. Note that we choose a data representation where the boundary of the shapes is very "regular," in the sense that it consists of points whose neighbours have a distance of either 1 or $\sqrt{2}$ (they are aligned on a discrete grid). This means that computing the Vietoris-Rips-filtration, all connected components merge either at 1 or $\sqrt{2}$. Also, for many objects, the persistence diagrams of the first dimension consist only of one point, see 17 for exemplary persistence diagrams.

Kernel	$noise = 0$	0.00005	0.0001	0.005
PSSK	86%	86%	86.1%	85.8%
PFK	-	-	62.2%	72.9%
SWK	79.4%	81.8%	82.2%	86.0%
PWGK	63.8%	48.8%	47%	49%

Table 6: Accuracy's for increasing noise levels added to the contour shape data set, using first dimensional homology. The PSSK performs best, whereas the PFK and PWGK perform particularly bad.

We again faced some difficulties with the computation of the Gram matrix of the PFK as it was not assessed as positive definite by the computer. Adding multiples of the identity matrix in order to stabilize the computation or removing negative eigenvalues were no good solutions. Surprisingly, all other kernels did not pose any difficulties. We resolved the problem of the PFK by perturbing the point clouds with Gaussian noise, respecting the magnitude of the data and ensuring that the original shape still was clearly visible. We also tried to identify single classes causing these problems by e.g. removing only one class from the testing process, but could not identify single classes causing this problem. We would like to note that we performed boundary extraction for the contour data shape originating from three different sources: from discrete grid data but also from jpeg and GIF files. However, no file format could fix the problems caused by the PFK.

The results are given in table 6. First of all, the SWK performs very well. But more strikingly, the PSSK performs just as well, outperforming the PFK used for a SVM. The authors of [20] claim that the PFK reaches an accuracy of 80%, whereas the PSSK reaches 86% when being used with a GP. This is particularly interesting as for a SVM, the PSSK performs worse than the PFK, with an accuracy of 73%. GPs therefore seem to sometimes bring complementary information compared to SVMs. We investigate the persistence diagrams produced during these experiments, see fig. 17 and fig. 18, and those of the orbit data set, see fig. 14. Firstly, we can clearly observe the added noise in the persistence diagrams, compare fig. 18 to fig. 17, but can also observe that the points of highest persistence stay unchanged, as expected. We can also observe that the persistence diagrams for the orbit data contain many more points, of which most are neither very close to the diagonal nor form points of very high persistence. We also recall the PSSK performed slightly inferior compared to the PFK for the orbit data set. We theorize that for diagrams with fewer points and many points of very low persistence and few points of very high persistence, the PSSK proves more efficient than the PFK. On the other hand, for persistence diagrams with a more chaotic form, the fisher kernel seems better suited. Analyzing the formula of the heat equation, we can see that it does a very good job of giving points of low persistence much less weight than those of high persistence. The PFK, on the other hand, places Gaussian distributions on every point, no matter the persistence, which can give too much probability mass to points of low persistence. However, when the persistence diagrams have a more complex form, like the ones in fig. 14, the PFK seems slightly better suited. Finally, we theorize that for some data sets, random perturbations can stabilize or even increase performance, as can also be read of table 6, for instance for the SWK. We observe that the PSSK shows the same performance for all perturbations, the PWGK showed worse performance, and the SWK even improved its performance. These perturbations can come as a remedy when, initially, the data is not well suited for kernel calculations.

6.2 Limitations PFK

We want to elaborate on the difficulties we faced when doing classification for the contour shape data set. We recall that the PFK puts 2D Gaussian distributions on each point of the persistence diagram (and its projection on the positive half-line in the first quadrant). The covariance matrix depends only on one parameter, σ, determining the variance in both dimensions. As for $X \sim N(0,1)$, we have that $\mathbb{P}\{|X| \geq t\} \leq 2e^{-t^2/2}$, we observe that the density of Gaussian distributions tends to zero very fast. In particular, for small choices of σ, this behaviour is amplified as we have if $X \sim N(0,\sigma^2)$, $\mathbb{P}\{|X| \geq t\} \leq 2e^{\frac{-t^2}{2\sigma^4}}$. From a practical perspective, this means that when choosing, e.g., $\sigma = 0.01$ as the standard deviation of a Gaussian distribution, the density outside of the interval $[-\frac{1}{2}, \frac{1}{2}]$ evaluates to zero on the computer when using double precision. So when using persistence diagrams where points lie further apart than length $\frac{1}{2}$ and we set $\sigma = 0.01$, we are missing crucial information for the computation of the kernel, which in theory, of course, always produces a positive definite matrix.
To practically stress test the limitation of the fisher kernel, we therefore conducted the following experiment: We randomly generated artificial persistence diagrams where points

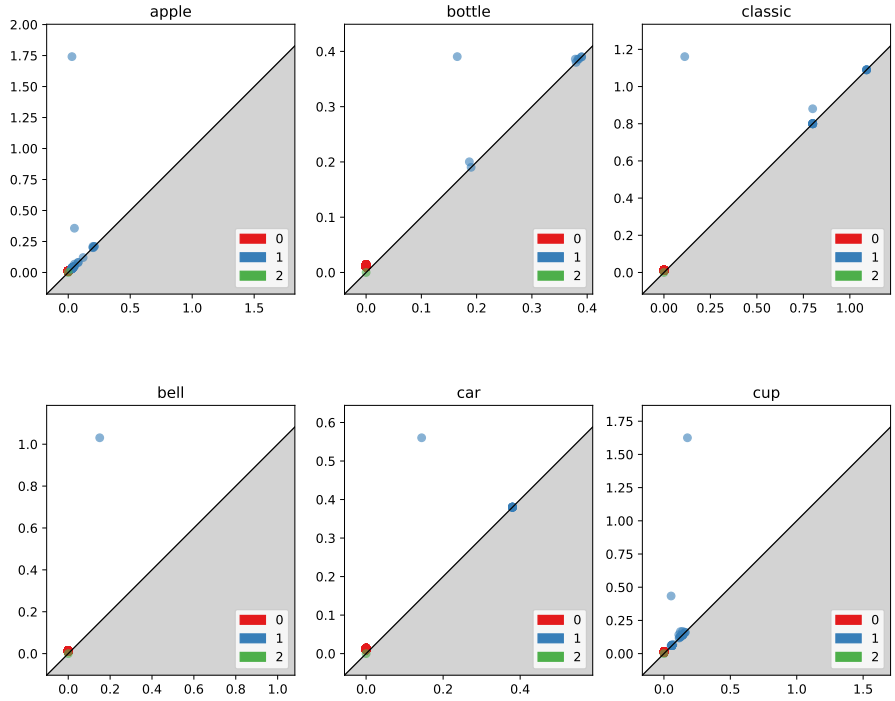


Figure 17: Exemplary persistence diagrams for various classes of the contour data set without added noise. Note how several diagrams contain only one or two points and no to little randomness seems present in the persistent diagrams, which were obtained from very regular shapes.

laid in the triangle spanned by the points $(0,0),(0,1),(1,1)$. Then we created Gram matrices for these diagrams for $\sigma = 0.01$ and checked them for positive definiteness. We did this both for one point and two points per diagram and grouped the persistence diagrams into triplets, resulting in Gram matrices in $\mathbb{R}^{3\times 3}$. For both the one-point and the two-point persistence diagrams, the fisher kernel produced non-positive-definite matrices. For the former case, 522 times out of 100,000; in the latter case, eight times out of 10,000. In the former case we additionally checked if using a value of $\sigma = 1.0$ changed the positive definiteness as we theorized that a small value causes the problems. It did so in 318 cases. However, we theorize that with more and more points per diagram, this effect can be neglected, but for data sets like the contour shape, where shapes were very regular and the first-dimensional persistence diagrams sometimes contained only one or two points, we assume that this behaviour explains the difficulties we faced.

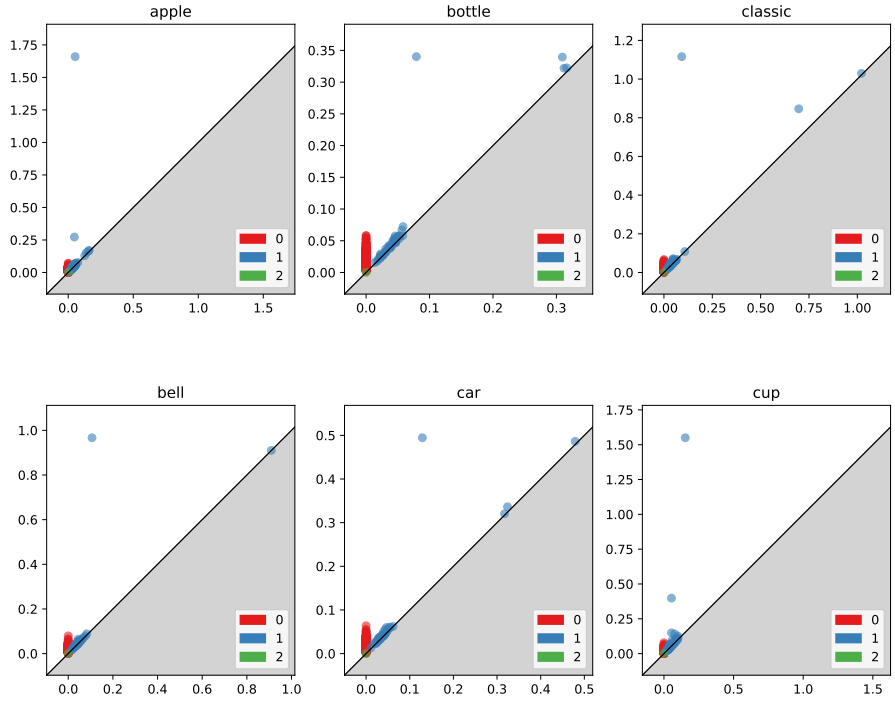


Figure 18: Exemplary persistence diagrams for various classes of the contour data set with added noise. Note how persistence diagrams which previously only contained one point (see fig. 17) have more points of low persistence.

6.3 Hemoglobin Classification

Hemoglobin classification is another test conducted in [20]. The PFK showed very high classification accuracies compared to all other three kernels when being used with an SVM. The original experiment, where topological methods were used for hemoglobin classification, was done in [6]. They introduced the so-called *molecular topological fingerprint* (MTF) as a feature vector constructed from persistent homology and used it for the input of an SVM. The MTF is given by a 13-dimensional vector whose elements consist of the persistence of some specific generators of persistent homology. The results from both [6] and [20] both seem very promising, reaching accuracies of more than 80% and up to 97%.

The experimental setup is as follows: The authors of [6] investigate 19 hemoglobin molecules from two classes, either R-Form or T-Form. There are 10 molecules of the former class and 9 of the latter. Also, they do not extract the full atomic structure as a 3-dimensional point cloud but only the carbon atom structure in order to reduce the size of the point clouds. Finally, they use alpha-complexes as filtration, which is what we are going to use in this case too. Interestingly, [19] and [20] use one atom less but do not specify which one they left out. Also, even though kernels must be defined on persistence

diagrams of a particular dimension (or as sums of them), both papers also do not specify which dimension they use for computing their kernels. We therefore follow the setup given in [6] and use both first- and second-dimensional persistence diagrams and select the ones which perform best. We obtain the atom structure from `https://www.rcsb.org/` and used standard python software to process .pdb files in order to extract the atomic structure of the carbon atoms as plain point clouds in $\mathbb{R}^3$. Finally, we perform 10×9 validation runs, where we train the GP on one data point from each class and evaluate its performance on the remaining data points. The experimental results can be found in table 7. Surprisingly, the obtained accuracies are all much worse than those obtained from an SVM. This could be due to a deficiency of the GP, but we want to point out that we were not able to fully reproduce and follow the experimental setup from [19] and [20]. Optimizing the likelihoods of PSSK and PWGK also does not improve accuracies for this test case.

	MTF	PSSK	PWGK	SWK	PFK
SVM	84.50	83.33	88.89	88.89	97.53
GP	-	68.5 ± 14 % (2)	70.1 ± 14 % (2)	67 ± 20 % (1)	77.2 ± 14 % (2)

Table 7: Accuracies for classification of taut and relaxed forms of hemoglobin. In parenthesis the dimension of the PD which achieves the best accuracy. We can not match the performance achieved in [19] with the GP.

6.4 3D Shape Segmentation

We want to replicate another experiment, conducted in [8], a classifier which assigns a point on some manifold in $\mathbb{R}^3$ a corresponding segment (or label). Their experiment in turn is based on the work in [10] and [9]. In the former, a common benchmark for evaluating segmentation algorithms in 3D was presented. The benchmark comprises a data set with 4,300 manually generated segmentations for 380 surface meshes of 19 different object categories, and they also include software for analyzing eleven geometric properties of segmentations and producing 4 quantitative metrics for comparison of segmentations. A novel approach, based on the topology of the underlying object, has been presented by Carrier et al. in [8]. They try to overcome the limitations of most shape descriptors, which focus on the local geometry of a point, and introduce a point descriptor capturing the global geometry of the object the considered point is lying on. Considering some compact smooth surface $\mathbb{X}$ in $\mathbb{R}^3$, they compute the local descriptor of a point $x \in \mathbb{X}$ as follows: They consider the evolution process of a geodesic ball centered at x, whose radius r grows from 0 to infinity. Along the way, they track the evolution of its topology, including its connected components (dimension 0), holes (dimension 1), and enclosed voids (dimension 2). As we are dealing with surfaces, the 0D topology is always trivial, whereas they claim that 2D topology has limited information. Therefore, they restrict to computing the holes (1D topology) only. For more details, we refer to [9]. They provide some software performing these computations for given point clouds, which can be found under `https://github.com/MathieuCarriere/perslocsig/tree/master`.

Our experimental setup now follows the one given in [8]. For each category, the goal is to design a classifier that can assign, to each point in the shape, a label that describes the relative location of that point in the shape. For instance, possible labels are, for the human category, head, torso, arm, etc. To train classifiers, we compute a persistence diagram per point using the geodesic distance function to this point, as explained above, and use 1-dimensional homology. For each category, the training set contains one hundredth of the points of the first five 3D shapes, and the test set contains one hundredth of the points of the remaining shapes in that category. Points in training and test sets are evenly sampled. Note also that [8] only uses 7 classes, to which we will restrict here as well. This results in a different number of segments for each object and a different number of train and test points for each object, see table 8. Obtaining optimal hyperparameters is, as usual, done by cross-validation on the train data for a grid of hyperparameters and then evaluated on the test set for the best performing hyperparameters. We omitted the PFK for this experiment as it has not shown a particularly high performance in previous experiments. Additionally, we tested if and how well maximizing the log-likelihood of the Gaussian process classifier via gradient descent can improve the accuracy of the PSSK. The resulting accuracies can be found in table 9.

In this experiment, the performance of the PSSK was better than the performance of the PWGK and the SWK, as it outperforms the other two kernels for all shapes. This is complementary to the results in [8], where the SWK was the best performing kernel, although PSSK and PWGK did not perform much worse. This shows us again that different kernel based models can favor different kernels. Despite the PFK having a very

TASK	TRAINING	TEST	LABELS
HUMAN	400	1500	8
AIRPLANE	300	980	4
ANT	364	1141	5
BIRD	257	832	4
FOURLEG	438	1097	6
OCTOPUs	334	1447	2
FISH	304	905	3

Table 8: Number of train and test points with number of labels for 3d segmentation tasks.

TASK	PSSK	PSSKOpt	PWGK	SWK
HUMAN	$\mathbf{71.9 \pm 1.3}$	**73.7**	50.2 ± 1.2	67.0 ± 1.9
AIRPLANE	$\mathbf{75.7 \pm 1.3}$	**72.7**	62.0 ± 2.5	64.0 ± 1.7
ANT	$\mathbf{87 \pm 0.9}$	**87.7**	63.1 ± 2.8	84.7 ± 1.6
BIRD	$\mathbf{77 \pm 1.5}$	**80.1**	65 ± 1.7	75 ± 1.8
FOURLEG	$\mathbf{65 \pm 1.8}$	**67.4**	58.4 ± 1.2	60.4 ± 1.8
OCTOPUS	$\mathbf{88.2 \pm 0.8}$	**87.5**	75.9 ± 1.1	84.2 ± 1.3
FISH	$\mathbf{84.7 \pm 0.9}$	**84.7**	82.4 ± 1.4	82.1 ± 1.1

Table 9: Accuracies for 3D Shape segmentation tests. We performed only one optimization and one subsequent test run for the optimized PSSK, achieving more or less the accuracies achieved via grid-serach.

nice geometric interpretation and the PWGK having more flexibility due to the high numbers of hyperparameters, PSSK and SWK proved to be the most easy and robust to train kernels so far. The infinite divisibility is an advantage when data sets are complex and when it is important to test many parameters via cross-validation. (Note however that this only applies to the case where the kernel computations are for more expensive than the matrix inversion of the kernel matrix. This is clearly the case here as all Gram matrices of size $n \times n$ have $n \leq 1500$ in this thesis, but the computation of such a Gram matrix with persistence diagrams having size $m \approx 1000$ results in a running time of $\mathcal{O}(n^2m^2)$, assuming quadratic running time for the kernel evaluations, as opposed to $\mathcal{O}(n^3)$ for matrix inversion.) And as mentioned before, the PSSK can be implemented particularly fast on a GPU, which makes it also practical to search for many parameters or even let an optimization procedure run. At least one of those two kernels showed very good performance for most previously conducted tests, indicating that restricting to those two kernels is a promising recipe for achieving the best performance while keeping computational costs comparably low. Another interesting observation is that optimizing the PSSK via maximization of the likelihood does not improve performance much, but always leads to more or less the same accuracy as found with grid search. On the one hand, values obtained via an optimization scheme can be preferred to those values that were obtained by grid search, as leaving out regions of high likelihood/accuracy is less likely in this case. On the other hand, this illustrates that choosing parameters via cross-validation, despite its primitive nature, is a valid procedure and does not necessarily lead to a worse performance. Also interesting to note is that the optimal parameters for all 7

objects are close to each other. This can reduce the time needed for finding an optimal parameter significantly when data sets can be assumed to not vary too much in their structure.

6.5 Probability Calibration

One of the main reasons for using a GP instead of a SVM, is its ability to return probabilities instead of black box classification decisions. In the following we want to elaborate this advantage by investigating how meaningful these probabilities are. For instance, for all test points with a confidence of around 80% for some positive class, do in fact 80% of those test points belong to the positive class and 20% to the negative class? One tool for answering this question are *calibration plots*. For a binary classifier they work as follows: After fitting a model on training data, one computes the probabilities for a fixed positive class on test data. Subsequently, after forming a partition of the unit interval into bins of equal size, one groups the train data according to their predicted probabilities into those bins. For all test points in one bin, the average confidence returned by the model is plotted as x-value and the average accuracy of the positive class is computed as the y-value. A model can be called well calibrated if all points lie close to the diagonal. Additionally, one can compute the so called Brier Score in order to quantify the quality of calibration instead of visualizing it. It is defined as follows:

$$BS = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - p_i)^2 \tag{6.1}$$

where p_i is the probability assigned to test point i, and y_i is the actual outcome, encoded as 0 for the negative class and 1 for the positive class. The Brier Score takes values in $[0, 1]$ and the closer this value is to 0, the better calibrated the model can be assessed.

Another question worth posing would be if for a badly calibrated model, the returned probabilities can be transformed in order to match the true probabilities. This would involve fitting a second model after fitting the original model. More specifically, we try to predict the conditional probability $\mathbb{P}(y_i = 1|p_i)$, where p_i is a given probability of the model and y_i and is test point. This process is called *calibration* and is typically done in two ways, either by *Platt scaling* or by applying *isotonic regression* to the probabilities. Platt scaling treats the probabilities as input features and applies logistic regression to estimate the probability of class membership based on these scores. Isotonic regression seeks a monotonic function that best fits the relationship between the ordered predicted scores and the observed frequencies of positive outcomes. The objective is to find a piecewise-constant, non-decreasing function that minimizes the distance between the predicted probabilities and the true probabilities of class membership. Both calibrated models need to be evaluated on unseen test data again. Extending calibration plots, the Brier Score and calibration to multi class tasks is done in a one-vs.-rest fashion. For more details, we refer to `https://scikit-learn.org/stable/modules/calibration.html`.

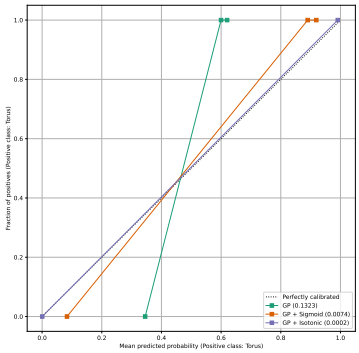
For all considered cases we will use the optimal parameters from previous experiments for fitting the models. For the binary classification cases, we present calibration curves for

the original model and calibration curves after applying both Platt scaling and isotonic regression. For three or more classes we will omit Platt scaling, both for readability and because isotonic regression achieved a better (lower) Brier score in all experiments. In particular, we want to find out if one model is calibrated superior to others and by how much calibrating a model improves its Brier Score and ability to return meaningful confidences.

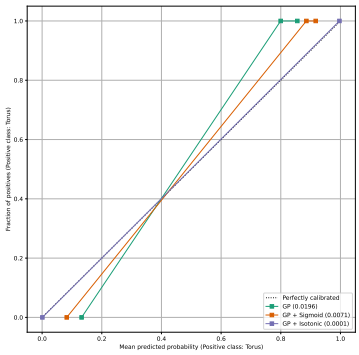
We start with a basic example of comparing two visually and topologically distinct objects, the sphere and the torus. We generate point clouds with 300 points and some added noise and train the GP with the PSSK, SWK and PWGK on 10 points per class and evaluate on 40 points per class. With all kernels we reach accuracies of 100%. Note that contrary to later experiments, objects sampled from the same class are also topologically and visually highly similar as they are sampled from them same, clearly defined manifolds. This results in confidences being almost the same for all objects of the same class, see fig. 19 for the resulting plots. This can also be observed by the fact that for all kernels, there are only two distinct points on the y-axis, 0 and 1 (denoting perfect accuracy) and two points on the x-axis, one for the correct positive class and one for the (correctly classified) negative class. For a perfectly calibrated model, we would expect confidences of close to 1.0. However, all three models can indeed be perfectly calibrated. We note that isotonic regression achieves a perfect calibration, whereas Platt scaling does not work quite as well.

We consider a more complex binary classification task, the Octopus data set from 3d mesh segmentation. We can observe more complex probability calibration plots, see fig. 20, indicating more complex data than in the previous experiment. Firstly, we observe that initially the PSSK and PWGK have calibration plots very close to the diagonal, whereas the SWK, despite equal accuracy, only distinguishes objects by a very small margin, as there are only two points on the x-axis, both close to 50%. The PSSK and PWGK however show a diverse distribution of confidences which all lie close to the diagonal, a sign for a well calibrated model. This is also reflected in a lower Brier score, compare 0.09 for the PSSK with 0.15 for the PWGK and 0.25 for the SWK. Interestingly, calibrating the PSSK and PWGK did not improve their Brier Score at all, whereas it was possible to reduce the SWK's Brier Score to at least match the one of the PWGK. Despite initially not being as well calibrated as the PSSK and PWGK, calibration improved the quality of the probabilities of the SWK significantly, making it also well suited for using its predictions as confidences. However, even after calibrating the models, the PSSK proved to be the best calibrated model.

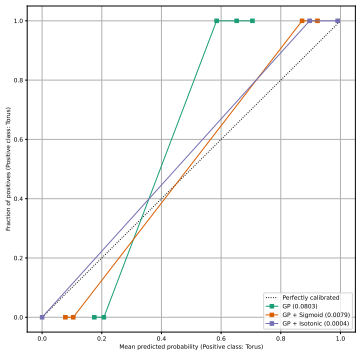
We increase the number of classes by one and inspect the Fish data set used for 3D mesh segmentation. The calibration curves were plotted as as one-vs.-rest classifiers for each class. We also recall that all three kernels reach similar accuracies of around 80% for this data set. We observe that among all three classes, the PSSK shows the best initial calibration again, followed by the PWGK and the SWK kernel, which can be seen both visually in the calibration curve and by comparing their respective Brier Scores, see fig. 21. Secondly, this order remains the same after calibrating all three model with isotonic regression. Interestingly, the PSSK calibration could only be improved slightly, calibrating the other two kernels however improved their Brier Scores to almost, but not



(a) PSSK

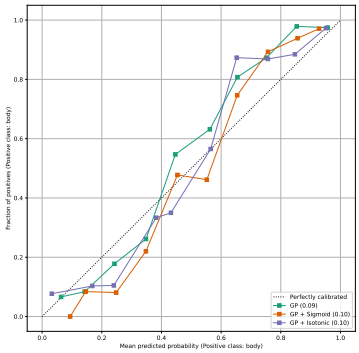


(b) PWGK

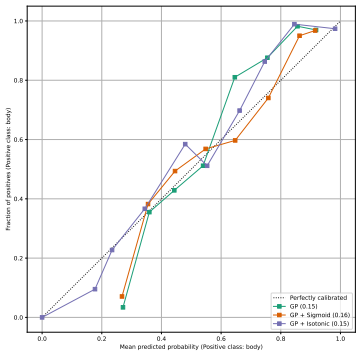


(c) SWK

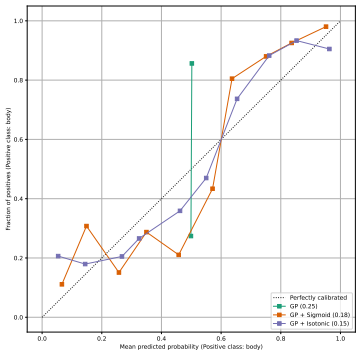
Figure 19: Calibration plots for sphere vs. torus. Note that objects from the same class are sampled without noise and thus appear very similar, thus being assigned the same probability by all three kernels. This results in very simple calibration plots for all three kernels.



(a) PSSK



(b) PWGK



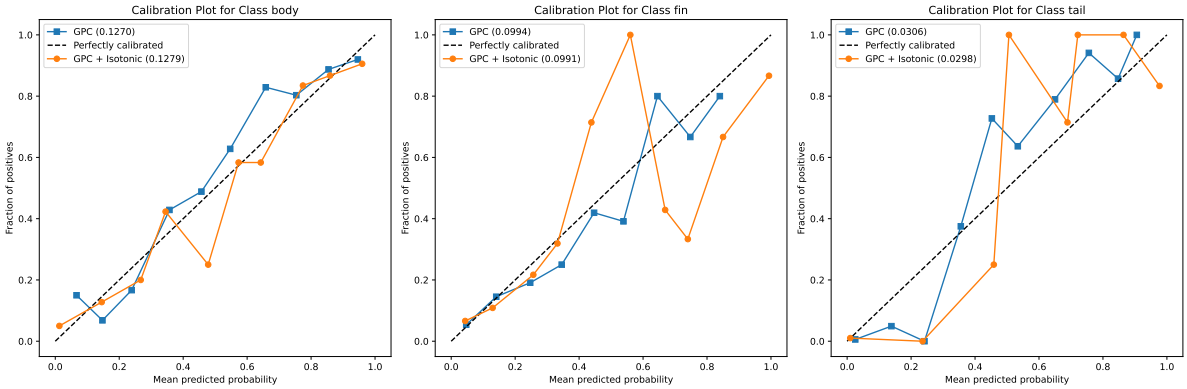
(c) SWK

Figure 20: Calibration plots for octopus data set from 3d mesh segmentation. The increased complexity of the data set yields a more complex calibration plot, compare to fig. 19. The PSSK is very well calibrated initially, without any need for calibration, whereas the SWK is not well calibrated at all in the beginning but can reach at least the Brier score of the PWGK when performing isotonic regression.

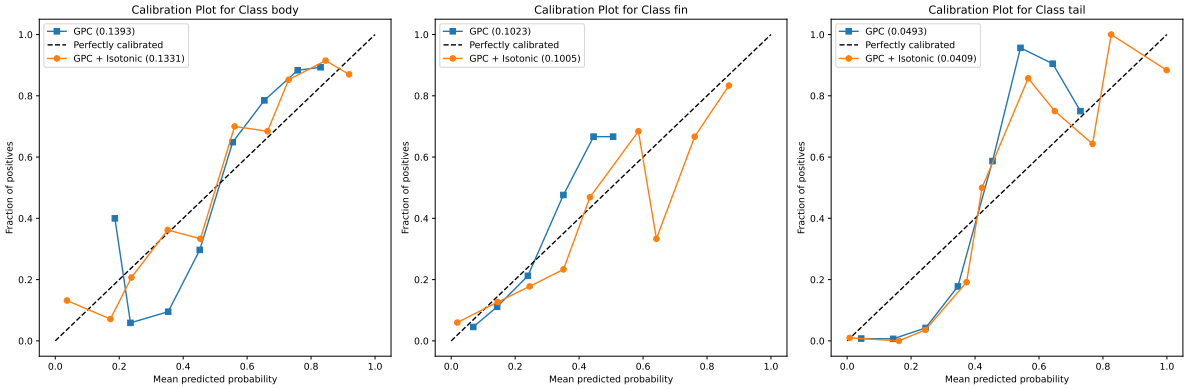
quite, the one of the PSSK. Again, despite being slightly inferior to the PWGK and PSSK, the SWK's calibration quality can be improved. Note that a calibration plot can appear highly non-monotonic and still have a comparably low Brier Score. This is because the calibration plot does not consider how many points are in each bin. For instance, a bin with only one point, which has high confidence, say 0.8, which is missclassified, does not change the Brier Score much but appears as a point far away from the diagonal, distorting the calibration plot. Of course such a behaviour could indeed be assessed as undesirable, in which case considering the calibration curve is more important than only the Brier Score.

We turn to another multi-class case, the synthetic data set with objects sphere, circle, torus, clusters, clusters within clusters and unit cube. Again, we generate point clouds with 300 points and added noise of $\sigma = 0.5$ and train the GP with PSS, SWK and PWGK on 10 points per class and evaluate on 40 points per class and use the first dimensional persistence diagrams to train the GP. Due to the low inter-class variance of the objects we do not expect many points on the x-axis, as we expect confidences of objects from the same class to be close. This is similar to the first calibration experiment. We also achieved accuracies of close to 100% for all kernels, which means that we also do not expect many distinct values on the y-axis, see fig. 22. Interestingly, in this example all three kernels exhibit a similar pattern. They classify most examples correctly, but with a very low confidence. Calibrating these models yields near perfect calibration for most objects.

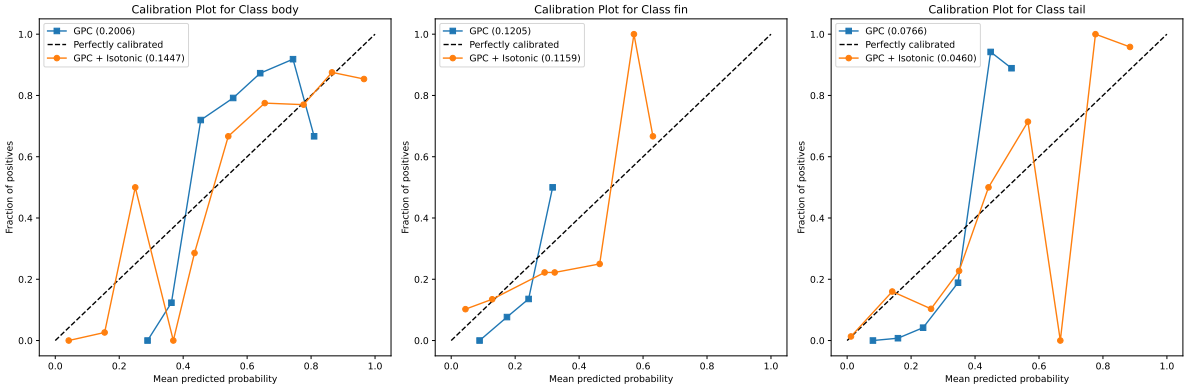
As a last example, confirming the above observations, we analyze calibration plots for the Bird data set from the mesh segmentation experiment, see fig. 23. Again, the PSSK is able to return the biggest range of confidences after initial training and also shows the best calibration among all three kernels. We can therefore conclude that in our experiments, the PSSK proved to be the best calibrated model of all kernels we investigated. Calibrating the other two models improved their respective Brier scores, in particular for the SWK, but they did not prove superior to the PSSK. This leaves us with a clear recommendation for using the PSSK when being interested in expected classification rates. As the PSSK proved to be among the best performing kernels so far, this is a particularly convenient result.



(a) PSSK

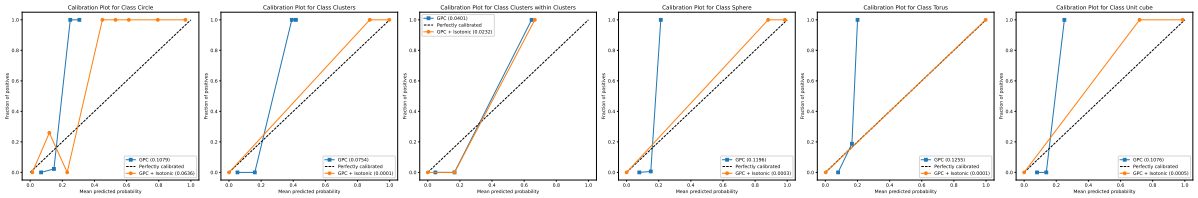


(b) PWGK

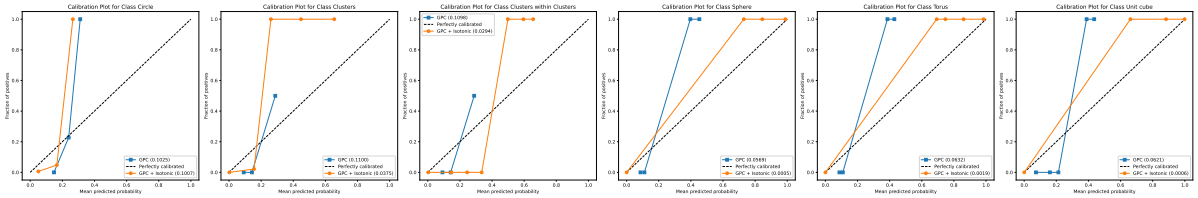


(c) SWK

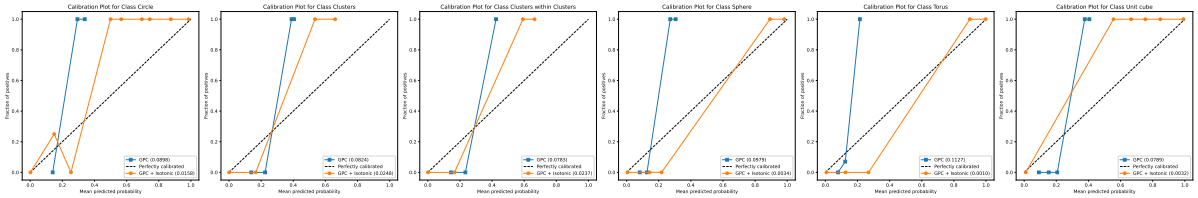
Figure 21: Calibration plots for fish data set from 3d mesh segmentation. Again, the PSSK appears to be best calibrated and while the Brier score of the other two models can be improved, they can not be better calibrated than the PSSK. Also visually the calibration curves of the PSSK look much closer to the diagonal than those of the other two kernels.



(a) PSSK

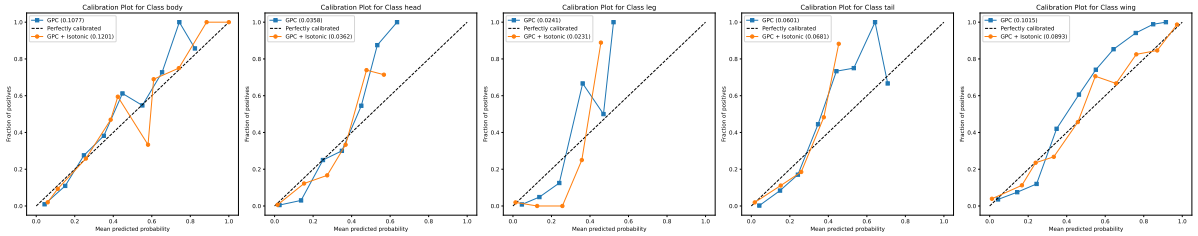


(b) PWGK

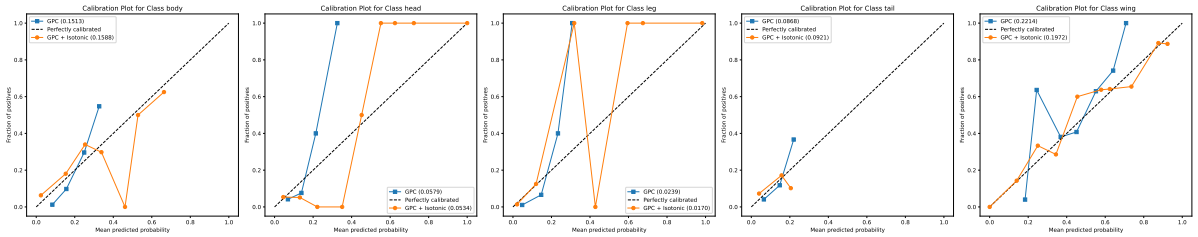


(c) SWK

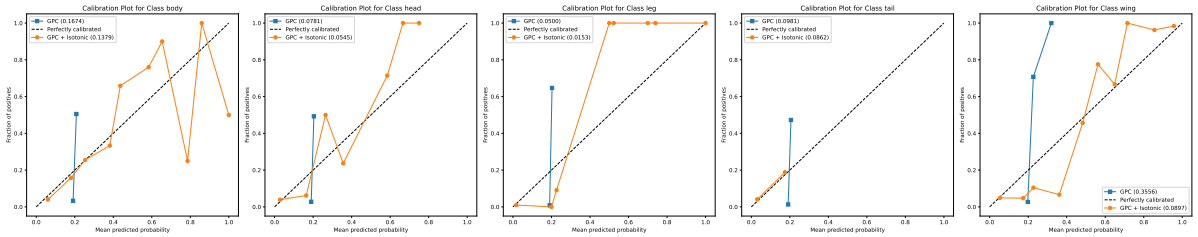
Figure 22: Calibration plots for the synthetic data set from [1]. Again, the PSSK appears to be best calibrated and while the Brier score of the other two models can be improved, they can not be better calibrated than the PSSK. Also visually the calibration curves of the PSSK look much closer to the diagonal than those of the other two kernels.



(a) PSSK



(b) PWG



(c) SWK

Figure 23: Calibration plots for bird data set from 3d mesh segmentation. Again, the PSSK appears to be best calibrated and also visually the calibration curves of the PSSK look much closer to the diagonal than those of the other two kernels. The Brier score of the PSSK is also better for all objects.

6.6 ModelNet10

We finally turn to a test set which has not been used in the context of topological kernels yet, the ModelNet10 dataset. It is a standard data set for benchmarking classification models, first introduced by We et al. [37]. The data set consists of 10 classes of standard household items (bed, night desk, table, toilet etc.). For each class, there are varying number of test and train objects with the number of test points varying between 50 and 100 and the number of train points varying between 100 and 900. Each data point is stored as a point cloud in an .off file. We note that for applying topological methods we are only interested in the vertices for computing persistence diagrams, not the edges and faces. However, we sub-sample points according to the respective face area in order to preserve the underlying topology as best as possible. Each .off file contains highly varying number of vertices, from 700 up to 30.000. We use the dataset with the aim of comparing SVMs and GPs on a much larger scale, contrary to the much smaller data sets used in the four kernel papers. We omit the Fisher kernel in the following test due to its slow running time and inferiority shown in the previous experiments. As always, the optimal hyperparameters for all kernels are obtained by cross validation. Note that for this particular example gradient based methods are computationally too expensive for GPs, even when paralleling Gram matrix calculations on a GPU. For the training of the GP there are as always several other hyperparameters to consider:

- Homology dimension: We have non-empty persistence diagrams in dimension 0, 1 and 2, as we have an object in $\mathbb{R}^3$. We choose to use first dimensional homology, as the second dimension is computationally too expensive for VR-complexes and 0th homology never proved superior to first homology in previous experiments.
- Sampled points: We subsample 600 and 1000 points per train and test point. The choice of these numbers is to enable fast computation. We will monitor whether increasing the number of sampled points enhances accuracy. Should there be no improvement, it may indicate that the our selection of sampled points is adequate.
- Number train points: We train with 50 and with 100 data points per class, e.g 500 or 1000 in total, and always test with 50 points per class. The idea is again to monitor whether increasing the number of sampled points enhances accuracy. Should there be no improvement, it may indicate that the our selection of sampled points is adequate for this particular experiment.

For the results with the GP, see fig. 24 and for the results with the SVM, see fig. 25. We inspect the results of the GP first. Despite varying the number of trained objects and the number of sampled points per point cloud, the overall accuracy seems not to be affected too much by these parameters when training with a GP. For all three kernels, we reach accuracies of around 40% when training with a GP, with the SWK being the best performing kernel by reaching 42% at best. We assess this performance as satisfactory as we are only using topological information. On such a large data set where computation of the Gram matrices becomes very expensive, the SWK displays its strengths as we could test many hyperparameters for it, contrary to the PSSK and PWGK.

Interestingly, using the same Gram matrices for a SVM, we reach superior performance, in particular for the PSSK which suddenly is the best performing kernel and also surpasses the SWK. It makes an improvement of about 10% compared to the GP and could reach an accuracy of almost 50% for 1000 sampled points per object and 100 data points per class used for training. Increasing the number of train points also improves performance for the SVM, combined with the PSSK and SWK. Therefore we can not rule out that an even better accuracy might be possible for the SVM if sampling even more points or using more data points for training. This does not seem to be the case for the GP. In our opinion, this is a clear indication that SVM's are better suited for classifying many classes simultaneously and when also having many train points available.

It is important to recall however, that we use a binary classifier for the GP instead of the true multi-class Laplace-approximation, which could limit performance. Initial tests on only two classes show very promising results for the GP classifier, with accuracies ranging between 0.8 and 0.9. For this reason we conduct the following experiment: For the setting with 600 samples per point cloud and 100 points per class used for training, we train the GP for all possible subsets of classes of the 10 class data set (of course not less than 2 classes). We compute both the average and the maximum of the accuracies per number of trained classes and plot these values from two classes up to 10 classes for all three kernels, see fig. 30. What we obtain is a significant decrease of performance with an increased number of classes. The GP indeed seems to be well suited for classifying complex objects with few classes, but cannot easily be generalized to a large number of classes. Also, in the case of little data, optimization of the likelihood is practical, making GPs even better suited for this setting compared to SVMs which do not have this property. As increasing the number of trained points makes optimizing the likelihood impractical anyway, we would rather recommend a SVM for performing a classification task similar to ModelNet10.

Finally, we provide confusion matrices for the predictions of the PSSK and SWK for both SVM and GP, see figs. 26 to 29. All combinations exhibit similar behaviour, as they mostly classify the same classes with high and low accuracies for the GP and SVM respectively. For instance, objects from the class "toilet" always have a very high accuracy, whereas the class "bathtub" has a very low accuracy. Exceptions for this are the classes "chair" and "table" for the PSSK (compare figs. 26 and 27) and "bed" for the SWK (compare figs. 28 and 29). This indicates that although sometimes GP and SVM slightly differ in their performance for fixed kernels, they do not fundamentally differ in their prediction when using the same kernel and seem to draw similar decision boundaries between the classes. We also suspect that this indicates a clear limitation of topological methods in general on this data set, as no combination of either a GP/SVM with the PSSK/SWK would yield complementary information, which could potentially be combined for better predictions.

	PSSK	PWGK	SWK
50	0.348	0.38	0.422
100	0.382	0.37	0.426

	PSSK	PWGK	SWK
50	0.374	0.354	0.406
100	0.362	0.372	0.41

Figure 24: Accuracies for 600 (left) and 1000 (right) sampled points per object for GP with both 50 and 100 train points per class.

	PSSK	PWGK	SWK
50	0.41	0.404	0.418
100	0.466	0.382	0.432

	PSSK	PWGK	SWK
50	0.46	0.36	0.384
100	0.498	0.386	0.432

Figure 25: Accuracies for 600 (left) and 1000 (right) sampled points per object for SVM with both 50 and 100 train points per class.

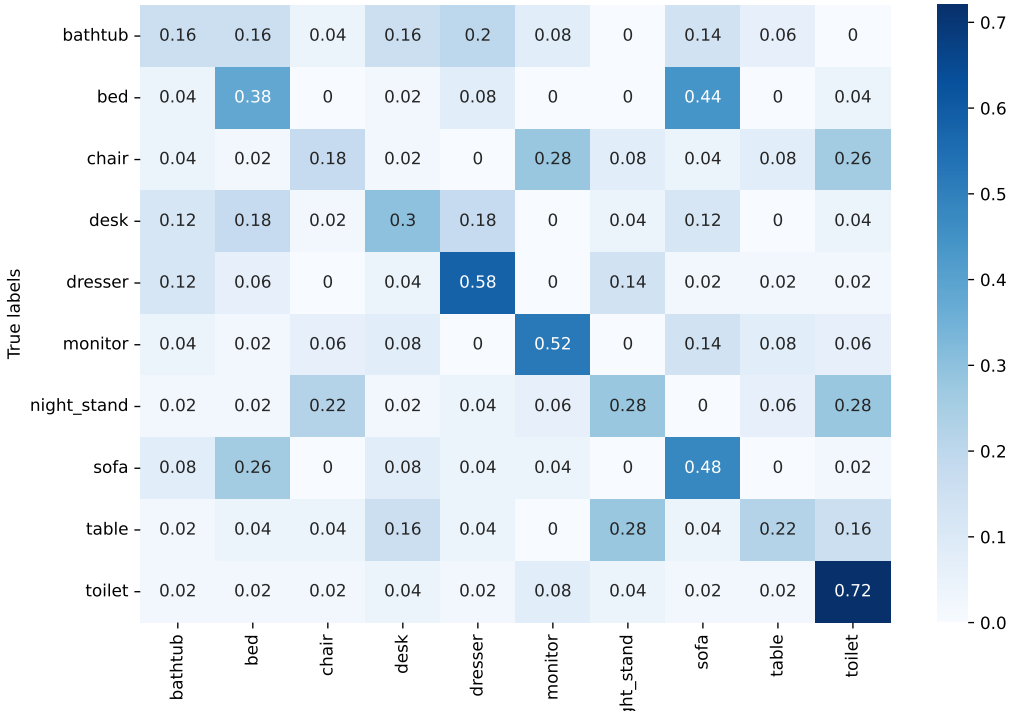


Figure 26: Confusion matrix with PSSK and GP.

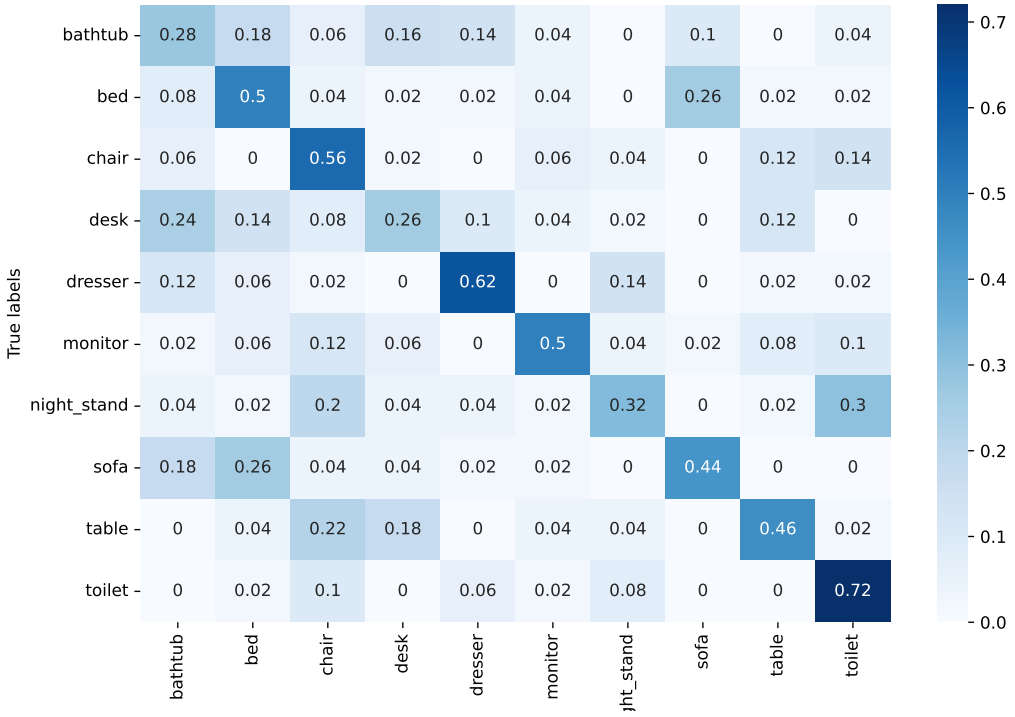


Figure 27: Confusion matrix with PSSK and SVM.

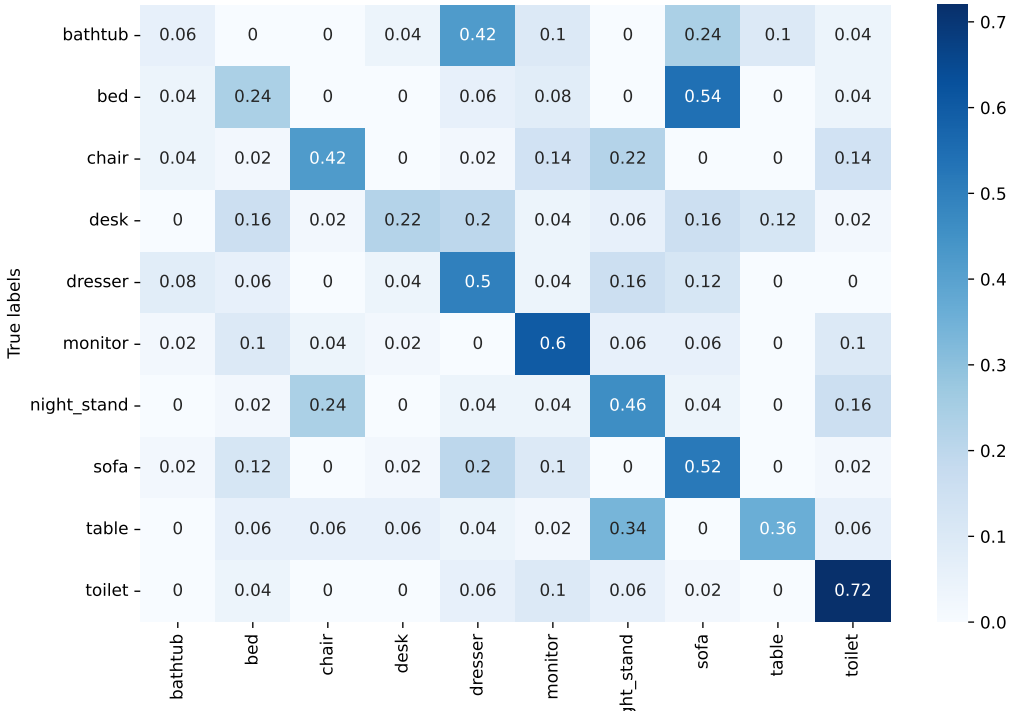


Figure 28: Confusion matrix with SWK and GP.

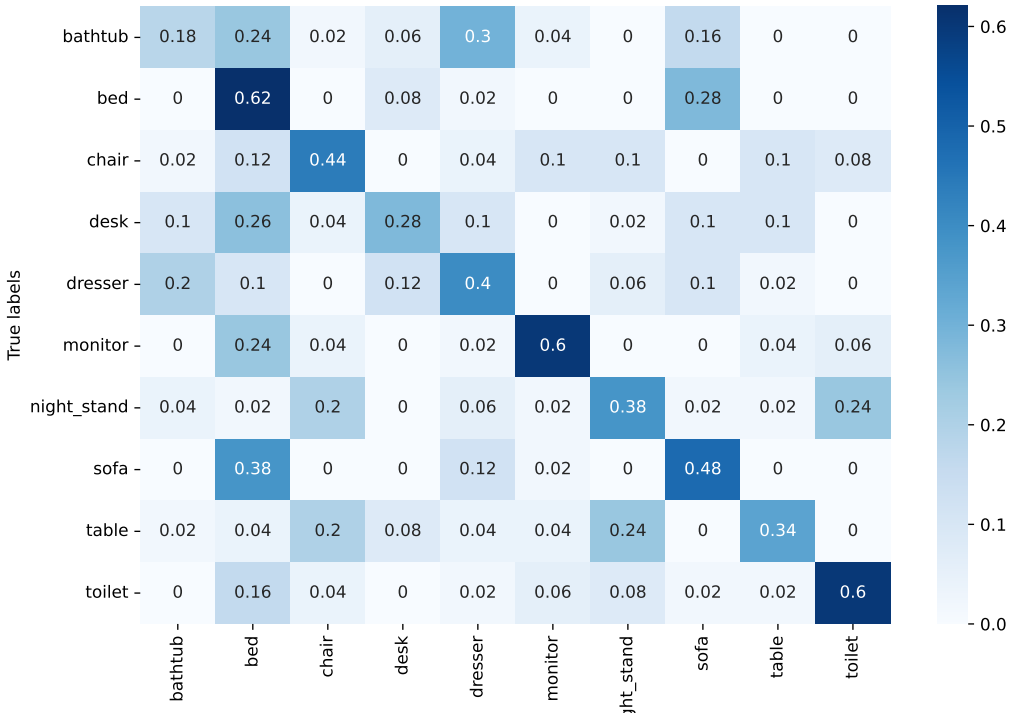
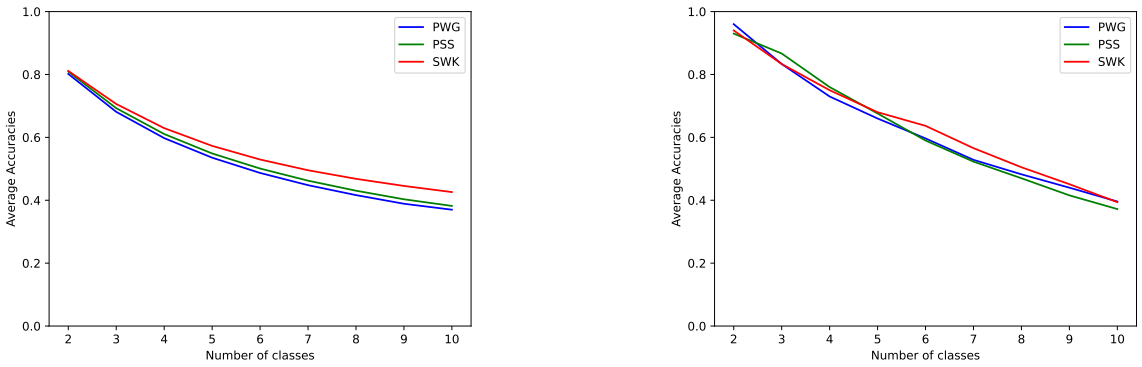


Figure 29: Confusion matrix with SWK and SVM.



(a) Average accuracies

(b) Maximal accuracies

Figure 30: Average and Maximal accuracies for the ModeltNet10 dataset, grouped by the number of trained classes.

7 Conclusion

In this thesis, we investigated how well topological kernels can be combined with Gaussian processes instead of e.g. support vector machines. Despite SVMs being the most common choice for performing classification tasks so far, the strengths of Gaussian processes made it a fascinating topic to explore. First of all, we wanted to compare the performance of GPs with many common benchmarks in TDA and in particular to support vector machines. However, we also tried to get a more detailed understanding. We set out to investigate how useful the returned probability vector of the GP is, a core advantage of GPs over SVMs. Secondly, we wanted to explore how to train this model effectively by e.g likelihood maximization, another advantage of GPs over SVMs. Finally, we wanted to explore if and how all kernels differ, both with respect to their performance and training methodology, in order to given final recommendations for which kernels are best suited for practical problems.

7.1 Summary and Contribution

We have detected many subtleties of GPs, serving as valuable insights for using this model in the future. We started on a very simple, yet illustrative test case by investigating all four kernels on five basic topological objects. We found that with simple cross-validation, GPs can use the PSSK, SWK and PFK to perfectly distinguish five simple objects. The PWGK however already proved more difficult to train but performed reasonable. We then added increasing levels of noise to the test data in order to investigate if this randomness can be read off the returned probability vector, which we found out not to be the case, illustating the robustnes of the model. On the same data set, we further investigated how to optimize all kernels. We found valid optimization schemes for all four kernels and pointed out their limitations. We can conclude that the SWK and PSSK seem most practical to optimize, even though PWGK and PFK still have unique properties speaking in their favor, like added flexibility and a nice geometric interpretation of the hyperparamter. Our implementation of the PSSK should also make experiments with the PSSK easier as the Gram matrix can be computed efficiently on a GPU. Employing MCMC simulations on the same data with the PWGK, we could show that we can also improve the performance of this kernel this way. Also, we could not find an indication that the PWGK has dependent parameters. We then set off to make a first, direct comparison of the performance to SVMs by investigating another synthetic data set and the orbit data set. We found that for these data sets, performance matches those from support vector machines. We then conducted tests on real world data by replicating experiments conducted in [19] and [20]. We could not match the results for the hemoglobin data set. For the contour shape data set, we could achieve superior performance by using the PSSK instead of the PFK. We also detected valuable insights about the limitations of the PFK when conducting these experiments. When replicating the mesh segmentation experiment conducted in [8], we could again achieve similar performance, yet not with the SWK but the PSSK, which achieved a particularly strong accuracy. These experiments led us to the important conclusion that different kernels perform differently when either being used with a GP or SVM, so its seems important to test several combinations instead of just one. We also conducted tests on the ModelNet10 data set. Considering that we used only topological

information, we could reach reasonable performance. However, comparing the results of the GP to those of a SVM showed that in a large multi-class setting with complex data, SVMs seem to be better suited. We noticed that for fewer classes, GPs still remain a powerful choice as they reached high accuracies in these cases. We finally investigated a technique called probability calibration, where we found ways to interpret the confidences of the GP on various test cases. The PSSK emerged as the best suited model for using the returned probabilities.

Summarizing all experiments, we showed empirically that GPs mostly, but not always, perform just as well as support vector machines and can therefore be seen as valuable addition, in particular when it is possible to perform likelihood maximization. Also, for almost all experiments, either the SWK or the PSSK emerged as the best performing kernel. As both kernels are also very fast to train, we recommend those two kernels as first choices for future experiments. However, PWGK and PFK still have interesting properties, making them also worth exploring if resources permit training them as well.

7.2 Limitations and Outlook

We identify several limitations and possibilities for future work. We relied only on one implementation for approximating the posterior distribution of the Gaussian process classifier, the Laplace approximation. Expectation propagation or other methods could also be investigated and may alter the results. We also note that we have not used a true multi-class classification model, which could also be a limitation and its utility should be further investigated. A serious limitation was also the size of persistence diagrams and the accompanying computational complexity. An even more efficient implementation of either the PWGK or PFK could therefore shed more light on the capabilities of topological kernels, in particular on larger data sets or increasingly larger persistence diagrams. It could also be interesting to investigate variances around the returned probability vectors. This would utilize the advantages of a GP even further as knowing this variance can be important for decision making. A topic we have not mentioned in this thesis was analyzing a regression task. Whilst we constructed a very simple experiment, we have not been able to come up with sensible explanations about the behaviour of the regression model. Finally, we have not made usage of the possibility of combining kernels via e.g addition. With this method, persistence diagrams of different dimensions or different kernels can be combined. This might lead to superior performance and would also be a fascinating topic to explore.

In conclusion, we hope that this thesis not only sheds light on the nuances, complexities and potential of topological kernels and Gaussian processes, but also equips researchers curios about these topics with valuable insights, useful for further elaboration on this topic and encouraging further exploration of the fascinating synergy of these two beautiful mathematical theories.

References

[1] Henry Adams et al. *Persistence Images: A Stable Vector Representation of Persistent Homology*. 2016. arXiv: 1507.06217 [cs.CG].

[2] Henry Adams et al. "Persistence images: A stable vector representation of persistent homology". In: *Journal of Machine Learning Research* 18.1 (2017), pp. 1–35.

[3] Ulrich Bauer. "Ripser: efficient computation of Vietoris-Rips persistence barcodes". In: *J. Appl. Comput. Topol.* 5.3 (2021), pp. 391–423. ISSN: 2367-1726. DOI: 10.1007/s41468-021-00071-5. URL: https://doi.org/10.1007/s41468-021-00071-5.

[4] Peter Bubenik. "Statistical Topological Data Analysis using Persistence Landscapes". In: *Journal of Machine Learning Research* 16 (2015). Ed. by David Dunson. Submitted 7/14; Published 1/15, pp. 77–102.

[5] Peter Bubenik. "Statistical topological data analysis using persistence landscapes". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 77–102.

[6] Zixuan Cang et al. *A topological approach for protein classification*. 2015. arXiv: 1510.00953 [q-bio.BM].

[7] Mathieu Carriere, Marco Cuturi, and Steve Oudot. "PersLay: A neural network layer for persistence diagrams and new graph topological signatures". In: *arXiv preprint arXiv:1904.09378* (2020).

[8] Mathieu Carrière, Marco Cuturi, and Steve Oudot. *Sliced Wasserstein Kernel for Persistence Diagrams*. 2017. arXiv: 1706.03358 [cs.CG].

[9] Mathieu Carrière, Maks Ovsjanikov, and Steve Oudot. *Stable Topological Signatures for Points on 3D Shapes*. 2015.

[10] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. "A Benchmark for 3D Mesh Segmentation". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (Aug. 2009).

[11] Vin De Silva and Robert Ghrist. "Coverage in sensor networks via persistent homology". In: *Algebraic & Geometric Topology* 7.1 (2007), pp. 339–358.

[12] Herbert Edelsbrunner. *A Short Course in Computational Geometry and Topology*. Berlin, Heidelberg: Springer, 2014. DOI: 10.1007/978-3-642-40176-5.

[13] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010. ISBN: 978-0-8218-4925-5.

[14] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[15] Felix Hensel, Michael Moor, and Bastian Rieck. "A Survey of Topological Machine Learning Methods". In: *Frontiers in Artificial Intelligence* 4 (2021). ISSN: 2624-8212. DOI: 10.3389/frai.2021.681108. URL: https://www.frontiersin.org/articles/10.3389/frai.2021.681108.

[16] Christoph Hofer et al. "Deep learning with topological signatures". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1633–1643.

[17] L Kanari et al. "Topological representations of complex networks for material connectivity". In: *Nature Communications* 9.1 (2018), pp. 1–13.

[18] Kwangho Kim et al. *PLLay: Efficient Topological Layer based on Persistence Landscapes*. 2021. arXiv: 2002.02778 [cs.LG].

[19] Genki Kusano, Kenji Fukumizu, and Yasuaki Hiraoka. *Persistence weighted Gaussian kernel for topological data analysis*. 2016. arXiv: 1601.01741 [math.AT].

[20] Tam Le and Makoto Yamada. *Persistence Fisher Kernel: A Riemannian Manifold Kernel for Persistence Diagrams*. 2018. arXiv: 1802.03569 [stat.ML].

[21] Felix Leibfried et al. *A Tutorial on Sparse Gaussian Processes and Variational Inference*. 2022. arXiv: 2012.13962 [cs.LG].

[22] Yongcheng Liu et al. "Relation-shape convolutional neural network for point cloud analysis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8895–8904.

[23] Michael Moor et al. *Topological Autoencoders*. 2021. arXiv: 1906.00722 [cs.LG].

[24] Jose A Perea et al. "SW1PerS: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data". In: *BMC bioinformatics* 16.1 (2015), p. 257.

[25] The GUDHI Project. *GUDHI - Geometry Understanding in Higher Dimensions*. http://gudhi.gforge.inria.fr/. Version 3.5.0. 2023.

[26] Charles R Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5099–5108.

[27] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 652–660.

[28] Julien Rabin et al. "Wasserstein Barycenter and Its Application to Texture Mixing". In: *Scale Space and Variational Methods in Computer Vision*. Ed. by Alfred M. Bruckstein et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 435–446. ISBN: 978-3-642-24785-9.

[29] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[30] Jan Reininghaus et al. *A Stable Multi-Scale Kernel for Topological Machine Learning*. 2014. arXiv: 1412.6821 [stat.ML].

[31] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 1992.

[32] B. W. Silverman. *Density estimation for statistics and data analysis*. CRC press, 1986.

[33] Gurjeet Singh et al. "Topological analysis of population activity in visual cortex". In: *Journal of vision* 8.8 (2008), pp. 11–11.

[34] Ann E. Sizemore et al. "Importance of the whole: topological data analysis for the network neuroscientist". In: *Network Neuroscience* 3.3 (2019), pp. 656–673.

[35] Nico Stucki et al. *Topologically faithful image segmentation via induced matching of persistence barcodes*. 2022. arXiv: 2211.15272 [cs.CV].

[36] Wee Peng Tay et al. “Topological data analysis of contagion maps for examining disease spread”. In: *Nature Communications* 11.1 (2020), pp. 1–9.

[37] Zhirong Wu et al. *3D ShapeNets: A Deep Representation for Volumetric Shapes*. 2015. arXiv: 1406.5670 [cs.CV].

[38] Yijia Yan et al. “Recurrent neural networks with topological design”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 5653–5660.

[39] Manzil Zaheer et al. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG].

[40] Qingkai Zhao and Yusu Wang. “Learning metrics for persistence-based summaries and applications for graph classification”. In: *arXiv preprint arXiv:1904.12085* (2019).